
# ARCHIOTECT: AN AI-POWERED KNOWLEDGE-DRIVEN ASSISTANT FOR IOT ARCHITECTURAL DESIGN SOLUTIONS

Fernando Novaes Ribeiro da Silva

Master's Dissertation presented to the Programa de Pós-Graduação em Engenharia de Sistemas e Computação, COPPE, from Universidade Federal do Rio de Janeiro, as part of the requirements necessary to obtain the title of master’s in systems and computer engineering.

Advisor: Guilherme Horta Travassos

RIO DE JANEIRO, RJ - BRAZIL
SEPTEMBER 2025

# ARCHIOTECT: AN AI-POWERED KNOWLEDGE-DRIVEN ASSISTANT FOR IOT ARCHITECTURAL DESIGN

Fernando Novaes Ribeiro da Silva

DISSERTATION SUBMITTED TO THE FACULTY OF THE ALBERTO LUIZ COIMBRA INSTITUTE FOR POSTGRADUATE STUDIES AND ENGINEERING RESEARCH AT THE FEDERAL UNIVERSITY OF RIO DE JANEIRO AS PART OF THE REQUIREMENTS FOR OBTAINING THE DEGREE OF MASTER OF SCIENCE IN SYSTEMS AND COMPUTER ENGINEERING.

Advisor: Guilherme Horta Travassos

Approved by: Prof. Cláudio Miceli de Farias

Profª. Regina Maria Maciel Braga

RIO DE JANEIRO, RJ - BRAZIL
SEPTEMBER 2025

da Silva, Fernando Novaes Ribeiro.

ArchIoTect: An AI-Powered Knowledge-Driven Assistant for IoT Architectural Design/ Fernando Novaes Ribeiro da Silva. – Rio de Janeiro: UFRJ/COPPE, 2025.

XI, 153 p.: il.; 29,7 cm.

Orientadores: Guilherme Horta Travassos

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2025.

References: p. 68-72.

1. Arquitetura de Sistemas de Software IoT. 2. Geração Aumentada Via Recuperação (Retrieval-Augmented Generation - RAG). 3. Inteligência Artificial Generativa. 4. Internet das Coisas. 5. Engenharia de Software Experimental. I. Travassos, Guilherme Horta *et al*. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

**Acknowledgments**

Reaching this moment, the conclusion of this Master's journey, is a milestone I carry with profound gratitude and a mix of emotions that only dedication to a purpose can evoke. This work is not merely the result of hours of study and research, but also a reflection of a path walked with the invaluable support of people and forces that propelled me forward.

First and foremost, I offer my most sincere thanks to the Blessing of God, for His infinite wisdom, strength, and guidance in all moments. His constant presence was the foundation upon which I built my resilience and my faith, allowing me to overcome challenges and find the clarity needed to move forward.

I am immensely grateful to my advisor, Professor Guilherme Horta Travassos, for being a fundamental guide in my academic journey. His love for his profession and the field he chose is inspiring and contagious, and his dedication to shaping me into a researcher is something I will carry forever. His unwavering patience, even in moments when we faltered or faced difficulties, was an essential pillar of support. Even during my move to another country, our objective remained steadfast. I am immensely grateful for his guidance, for believing in my potential, and for empowering me to tread the path of research with confidence and knowledge.

My deepest gratitude goes to my Family, my safe harbor and my greatest inspiration. The love, understanding, and unwavering support from my parents, Eva and Vanderlei, my brother Rafael, and especially my wife Gabriela and daughter Larissa, were the fuel that kept me moving. Your words of encouragement, your patience with my absences, and your celebration of every small achievement made this journey possible and infinitely more meaningful.

I would like to thank Professors Regina Maria Maciel Braga (DCC/UFJF) and Claudio Miceli de Farias (PESC/Coppe) for agreeing to participate in my defense committee and for offering valuable contributions.

This foundation of family support also extended to the colleagues and friends I met in the Experimental Software Engineering (ESE) Group. The journey was enriched by the company and support of each one of you. Special thanks to Hélvio, whose analysis

and comments during a critical publication moment were crucial for the success of an article fundamental to my defense. To Larissa, for her generosity in sharing her time and experience, driving the advancement of my research proposal. To Clinton, for the support and material that strengthened my preparation for the defense. To André, who was responsible for the deployment of the application, and whose technical support was instrumental in the validation of this work. And, in a unique way, to Bruno P. de Souza. Bruno, your contribution since the first semester has been immeasurable, marked by moral support, kindness, and a genuine desire to see others prosper. Your support was, without a doubt, one of the essential pillars that allowed me to become a researcher. Thank you very much, everyone!

I extend my thanks to CNPq for the crucial financial support, both to me and my program. This work is a testament to the transformative impact of this investment.

**Agradecimentos**

Chegar a este momento, a conclusão desta jornada de mestrado, é um marco que carrego com profunda gratidão e um misto de emoções que só a dedicação a um propósito pode evocar. Este trabalho não é apenas o resultado de horas de estudo e pesquisa, mas também o reflexo de um caminho trilhado com o apoio inestimável de pessoas e forças que me impulsionaram.

Primeiramente, elevo meu mais sincero agradecimento à Bênção de Deus, por Sua infinita sabedoria, força e guia em todos os momentos. Sua presença constante foi o alicerce sobre o qual construí minha resiliência e minha fé, permitindo-me superar os desafios e encontrar a clareza necessária para seguir em frente.

Agradeço imensamente ao meu orientador, Professor Guilherme Horta Travassos, por ter sido um guia fundamental em minha jornada acadêmica. Seu amor pela profissão e pela área de atuação que escolheu é inspirador e contagiante, e sua dedicação em me tornar um pesquisador é algo que levarei para sempre. Sua paciência inabalável, mesmo nos momentos em que falhamos ou enfrentamos dificuldades, foi um pilar de apoio essencial. Mesmo durante minha mudança de país, esteve firme em nosso objetivo. Sou imensamente grato por sua orientação, por acreditar em meu potencial e por me capacitar a trilhar o caminho da pesquisa com confiança e conhecimento.

Minha mais profunda gratidão à minha Família, meu porto seguro e minha maior inspiração. O amor, a compreensão e o apoio incondicional de sempre de meus pais, Eva e Vanderlei, meu irmão Rafael, e em especial, esposa Gabriela e filha Larissa, foram o combustível que me manteve em movimento. Suas palavras de encorajamento, a paciência com minhas ausências e a celebração de cada pequena conquista tornaram esta jornada possível e infinitamente mais significativa.

Aos professores Regina Maria Maciel Braga (DCC/UFJF) e Claudio Miceli de Farias (PESC/Coppe) por aceitarem em participar da minha banca e oferecerem valiosas contribuições.

Essa base de apoio familiar se estendeu também aos colegas e amigos que encontrei no Grupo de Engenharia de Software Experimental (ESE). A jornada foi enriquecida pela companhia e apoio de cada um. Um agradecimento especial ao Hélvio,

cuja análise e comentários em um momento crítico de publicação foram cruciais para o sucesso de um artigo fundamental para minha defesa. À Larissa, por sua generosidade em compartilhar tempo e experiência, impulsionando o avanço da minha proposta de pesquisa. Ao Clinton, pelo apoio e material que fortaleceram minha preparação para a defesa. Ao André, que foi o responsável pela implantação da aplicação, e cujo suporte técnico foi essencial para a validação deste trabalho. E, de maneira ímpar, a Bruno P. de Souza. Sua contribuição desde o primeiro período foi imensurável, marcada por um apoio moral, gentileza e um genuíno desejo de ver o próximo prosperar. Seu suporte foi, sem dúvida, um dos pilares essenciais que me permitiram ser um pesquisador. Muito obrigado a todos!

Estendo meus agradecimentos ao CNPq pelo crucial apoio financeiro, tanto a mim quanto ao meu programa. Este trabalho é uma prova do impacto transformador desse investimento.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

ARCHIOTECT: AN AI-POWERED KNOWLEDGE-DRIVEN ASSISTANT FOR IOT ARCHITECTURAL DESIGN

Fernando Novaes Ribeiro da Silva

September/2025

Advisor: Guilherme Horta Travassos

Department: Computer Science and Systems Engineering

The rapid expansion of the Internet of Things (IoT), amplified by the integration of Artificial Intelligence (AIoT), presents significant architectural complexities for software system design. A critical gap persists in providing architects with accessible, evidence-based guidance to navigate these challenges, often leading to suboptimal designs and project failures. This dissertation addresses this gap by investigating the core research question: "*What IoT application domains and characteristics of their software systems architectures influence Quality Requirements (QRs) and how this knowledge can be systematically organized and offered to support the decision-making in IoT software systems projects?*"

To answer this question, a Systematic Literature Review (SLR) was conducted, analyzing 37 primary studies to distill actionable architectural knowledge. The primary contribution of this work is twofold: first, the creation of a comprehensive and structured Knowledge Base of IoT architectural solutions; and second, the development of *ArchIoTec*, a novel decision-support tool. *ArchIoTec* provides a dual-modality interface, allowing users to explore the knowledge base through both hierarchical browsing and a

conversational AI assistant powered by a Retrieval-Augmented Generation (RAG) architecture. This AI grounds its responses exclusively in the curated knowledge base, ensuring domain-specific accuracy.

The tool's effectiveness, efficiency, and utility were validated through an evaluation involving realistic design scenarios tailored for software architects and engineers. The results demonstrate that *ArchIoTec* successfully provides relevant and actionable guidance. This research culminates in a tangible, knowledge-driven tool that bridges the gap between fragmented academic theory and industry practice, empowering architects to make more informed and effective design decisions for complex IoT systems.

# Index

# List of Figures

# List of Tables

# 1 Introduction

*This chapter elaborates on the motivation and contextual background that prompted this work. It further details the study's objectives and the methodological approach adopted, concluding with an overview of the dissertation's organization.*

## 1.1 Motivation and Context

The Internet of Things continued its expansive trajectory in 2024, with a notable increase in both the number of connected devices and overall enterprise spending (IoT Analytics, 2025). This growth, occurring within a period of economic adjustment and a burgeoning focus on AI's role within IoT, brings with it a fresh set of challenges and considerations for the design and development of the underlying software systems. As new connectivity options emerge, interoperability standards evolve, and regulatory landscapes adapt, the architectural decisions made during the design phase become increasingly critical (IoT Analytics, 2025). This chapter defines the central problem addressed by this research, stemming from contemporary IoT developments, and outlines the specific research questions that this dissertation aims to answer to enhance the architectural design process for IoT software systems.

The increasing integration of AI into IoT shifts the paradigm towards an "AIoT" (Artificial Intelligence of Things) (Global, 2021), further amplifying these architectural complexities. While AI offers unprecedented capabilities for data analysis, predictive maintenance, and autonomous decision-making within IoT (Soori et al., 2024), it also introduces new demands on system architecture regarding data pipelines, model deployment, computational resources, and explainability (Antoniadi et al., 2021). The shift from traditional IoT applications to more intelligent, data-driven systems need architectural patterns that can handle vast volumes of heterogeneous data, support real-time processing, ensure robust

security and privacy, and facilitate seamless integration across diverse hardware and software components (Raptis et al., 2019).

Despite the proliferation of IoT platforms and development tools, a significant challenge persists in providing architects and developers with adequate support for making informed architectural decisions early in the design lifecycle. Developer surveys consistently highlight difficulties in managing device diversity, integrating disparate systems, and ensuring the security and interoperability of end-to-end solutions (Eclipse Foundation, 2023). The lack of comprehensive, easily accessible, and context-aware architectural guidance can lead to suboptimal designs, increased development costs, longer time-to-market, and systems that fail to meet critical quality requirements such as performance, reliability, or maintainability (Bass et al., 2021). This gap is particularly acute given the rapid evolution of IoT technologies and the diverse application domains, from smart healthcare and industry 4.0 to smart cities and agriculture, each with unique architectural needs and constraints (Gubbi et al., 2013).

This dissertation is motivated by this need, aiming to bridge the gap between the vast body of available architectural knowledge and the practical challenges faced by IoT system designers.

This chapter will further define the central problem addressed by this research, which stems from the contemporary developments in IoT and the identified gaps in architectural decision support. Subsequently, this dissertation will outline the specific research questions it aims to address, to enhance the architectural design process for IoT software systems and ultimately contributing to the development of more robust, efficient, and successful IoT solutions.

## 1.2 Research Problem and Question

The Internet of Things (IoT) is rapidly expanding, with over 18 billion connected devices and significant investment by enterprises in 2024 (IoT Analytics, 2025). This growth underscores the critical role of robust software architecture in ensuring the success of complex IoT systems (Bass et al., 2021). However, designing these architectures presents considerable challenges due to

the inherent heterogeneity of IoT technologies, diverse application domains, and the increasing integration of Artificial Intelligence (AI) (Atzori et al., 2010; Soori et al., 2024).

Architects often struggle to select optimal architectural solutions that effectively meet specific Quality Requirements (QRs), such as security, performance, or scalability, tailored to the unique context of different IoT applications (Weyns, 2021). Developer surveys frequently highlight difficulties in managing this complexity and ensuring interoperability (Eclipse Foundation, 2023). This highlights a significant gap: a lack of structured knowledge and targeted decision support to guide architects in understanding how IoT application domain characteristics influence architectural choices for achieving the desired QRs.

The consequences of this gap include suboptimal system designs, increased development costs, and a higher risk of project failure, hindering the full realization of IoT's potential (Woods, 2018). Therefore, this research is motivated by the need to address the problem of insufficient architectural decision support in the complex and evolving IoT landscape. It aims to investigate the relationships between IoT application domains, their architectural solutions' characteristics, and QRs, to provide a foundation for more informed architectural design.

To build a robust knowledge base for an application supporting decision-making in the design phase of IoT software system development projects, this dissertation addresses the following research question: ***What IoT application domains and characteristics of their software systems architectures influence Quality Requirements (QRs) and how this knowledge can be systematically organized and offered to support the decision-making in IoT software systems projects?***

## 1.3 Objective

The primary objective of this work is to develop and evaluate a novel application designed to provide an intelligent decision support application during

the architectural design phase of Internet of Things (IoT) software systems. This overarching goal is decomposed into the following specific objectives:

1. **Design and implement a comprehensive Knowledge Base (KB)** dedicated to IoT software system architectures. This objective involves:
    - Systematically identifying, collecting, and structuring diverse architectural solutions, design principles, Quality Requirements (QRs) (e.g., security, performance, scalability, interoperability), and relevant technologies pertinent to IoT systems.
    - Establishing a formal schema for the KB to ensure consistent representation, semantic interoperability, and efficient querying of architectural knowledge.
    - Populating the KB with curated data from peer-reviewed literature, established reference architectures, industry best practices, and empirical studies.
2. **Develop an AI-based decision support tool** that leverages the Knowledge Base. This objective encompasses:
    - Designing algorithms that enable the AI-based module to process project-specific requirements (e.g., application domain, target QRs, resource constraints) as input.
    - Implementing functionalities within the AI-based module to query the KB, reason over the stored architectural knowledge, and generate context-aware architectural suggestions, trade-off analyses, or potential design flaw identifications.
    - Integrating the Knowledge Base and the AI-based Decision Support Module into a cohesive system involves ensuring seamless data flow and interaction between the KB and the AI-based module to facilitate effective decision support.

To evaluate the effectiveness and utility of the proposed framework in supporting architectural decision-making for IoT software systems. This objective will be pursued by:

- Defining appropriate evaluation metrics (e.g., quality of recommendations, reduction in design time, coverage of QRs, user satisfaction).
- Conducting experimental studies involving representative IoT software system design scenarios to assess the framework's performance and practical applicability.
- Gathering feedback from domain experts or software architects to observe the relevance and usefulness of the generated decision support.

By achieving these objectives, this research aims to contribute to a robust, knowledge-driven tool that empowers software architects and developers to make more informed, efficient, and effective decisions when designing complex IoT software system architectures, ultimately leading to higher quality and more successful IoT solutions.

## 1.4 Methodology

This study's primary aim is to explore and identify the Quality Requirements (QRs) commonly observed in Internet of Things (IoT) software system architectures, by comprehensively analyzing those identified in primary sources. The characterization of these IoT software systems, regarding application domains and other QRs (such as security, performance, maintainability, and compatibility), is conducted from the perspective of software engineering researchers, drawing upon existing knowledge in technical literature. To deepen understanding, the main research question was subdivided into five secondary questions (as detailed in Table 2).

The methodological approach adopted to achieve the proposed objectives consisted of a Systematic Literature Review (SLR), partially following the

guidelines for Literature Search (LS) proposed by Kuhrmann et al. (2017) and incorporating the Snowballing technique (Wöhlin, 2014). The process steps are detailed below.

The primary data source for identifying primary studies was the Scopus database (www.scopus.com). The articles' search and selection processes were conducted in the following phases:

- Initial Search and First Snowballing Cycle: An initial search was performed, using a search string inspired by the PICO format (detailed in Table 4). A temporal scope was considered, in line with the work of Alreshidi and Ahmad (2019), focusing on architectural design solutions that influence QRs in IoT software systems. A set of papers was selected as starting points for applying the Snowballing technique (one level backward and one level forward). As a result of this first Snowballing round, an initial set of articles was identified, of which two were deemed relevant and aligned with the objectives of this study.
- Refined Search and Second Snowballing Cycle: Subsequent searches and refinements were performed, incorporating search terms derived from related works (Alreshidi & Ahmad, 2019; Razzaq, 2020). A broader search was conducted in the Scopus database, and the result served as the basis for executing a new Snowballing cycle (one level backward and one level forward).
- String updated and Third Snowballing Cycle: To identify more recent articles, the search string was re-run in the Scopus database.
- Study Inclusion and Exclusion Criteria: Following a rigorous selection process, primary studies were included in this review if they specifically addressed architectural design solutions influencing Quality Requirements (QRs) in IoT software systems, were published from 2019 onwards, and directly aligned with the objectives and research questions of this study. Articles failing to meet these criteria were excluded from further analysis. After the final selection of primary

studies (resulting from the second Snowballing cycle), the relevant data will be extracted to answer the research questions.

The analysis was conducted qualitatively, seeking to identify patterns, architectural challenges, and relevant characteristics within the context of IoT software projects, as outlined by the five secondary research questions (Table 2).

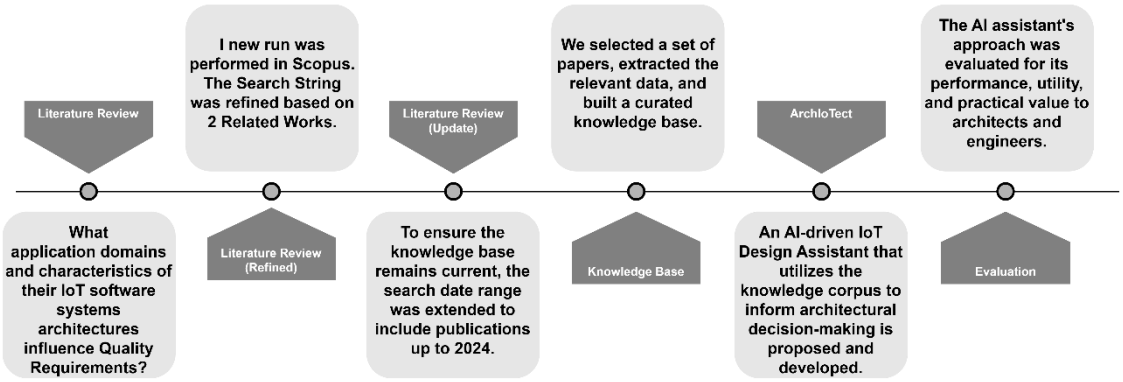



**Figure 1: Developing ArchIoTect: A Knowledge-First Approach.**

Figure 1 shows the research timeline, from the initial literature review to the final tool evaluation. This process was divided into the following key stages:

**Systematic Literature Review and Refinement:** The initial phase involved a broad literature review, followed by a systematic refinement process to distill the most relevant, up-to-date, and impactful architectural knowledge from academic and industry sources.

**Systematic Literature Review Update:** The search date range was extended to 2024 to ensure the inclusion of the most recent publications and maintain the currency of the knowledge base.

**Knowledge Base Construction:** This distilled information was then organized into a comprehensive knowledge base, structuring complex topics like communication protocols, security patterns, and data processing strategies into a coherent, actionable model.

**Tool Development - The ArchIoTect Assistant:** To make this knowledge actionable, we engineered ArchIoTect, the "IoT Architectural Design Assistant." This web application serves as an interactive interface to the knowledge base, offering both a conversational AI and a hierarchical browsing experience.

**Systematic Evaluation:** The final stage involved a thorough evaluation of the tool. By simulating real-world design challenges, we measured ArchIoTect's ability to provide effective, efficient, and user-friendly support to its target users, software architects, and engineers.

## 1.5 Publications

Throughout the realization of this work, two publications were produced :

- Silva, F., de Souza, B., & Werner, C. (2021). *Catálogo para Criação de Jogos Sérios para Sistemas Baseados em IoT*. In: Anais Estendidos do XX Simpósio Brasileiro de Jogos e Entretenimento Digital, (pp. 675-678). Porto Alegre: SBC. doi:10.5753/sbgames_estendido.2021.19705.
- Silva, F., Souza, B., & Travassos, G. (2024). *A Literature Study on Application Domains and IoT Software Systems Architectures Solutions Influencing Quality Requirements*. In: Anais do XXVII Congresso Ibero-Americano em Engenharia de Software, (pp. 181-195). Porto Alegre: SBC. doi:10.5753/cibse.2024.2844.

## 1.6 Text Organization

This dissertation is organized into four additional chapters, in addition to this first one, which describes the introduction, motivation, and context of the dissertation. The organization of this work follows the structure below:

***Chapter 2 – Theoretical Foundation:*** *Shows concepts directly related to what is proposed in this dissertation.*

***Chapter 3 – Literature Review:*** *Describes how the review was conducted.*

***Chapter 4 – Proposal of this Dissertation:*** *Presents the proposal, knowledge base, and the IoT Architectural Design Assistant.*

***Chapter 5 – Evaluating the Proposal of this Dissertation:*** *This chapter presents a TAM-based feasibility study of the ArchIoTect tool.*

***Chapter 6 – Final Considerations and Future Perspectives:*** *This section presents the key findings and contributions of this work, in addition to outlining future directions for this line of investigation.*

# 2 Theoretical Foundation

*The theoretical foundation for this dissertation is presented in this chapter. Key areas explored include the fundamentals of IoT software system architecture, the ISO/IEC 25010:2023 quality model, and the role of Large Language Models (LLMs), particularly within Retrieval Augmented Generation (RAG).*

## 2.1 Quality Requirements

Imagine you are building something complex, like a custom race car. You know it needs to do certain things – go fast, turn, stop. Those are its basic functions. However, just "going fast" is not enough, right? You also care about how well it does those things and other crucial aspects of its nature. This is where the 25010 concept of quality requirements, as guided by a framework such as ISO/IEC 25010 (ISO/IEC 25010, 2023), comes into play for software and systems.

So, a quality requirement, in this narrative, is akin to selecting one of the qualities from the master checklist (ISO/IEC 25010) and stating, "For my specific race car, this particular quality needs to be this good." For instance, the ISO checklist has a category called "Performance Efficiency." That is general. However, underneath it, there is "Time behavior." Now we are getting somewhere. A quality requirement for your race car, derived from this, would not just be "it needs to be fast." It would be something much more precise, like: "This car must be able to accelerate from 0 to 100 kilometers per hour in under 3 seconds." That is specific, measurable, and clearly defines a target for that aspect of performance. Similarly, the ISO checklist has "Reliability." A quality requirement would not just be "it should not break down." It might be: "The engine must be able to run at maximum RPM for at least four continuous hours without critical failure." For "Security," instead of "it should be hard to steal," a quality

requirement might be: "The car's ignition system must use encrypted key authentication that resists known brute-force attacks for at least 24 hours."

So, ISO/IEC 25010:2023 provides the map of all possible "goodness" attributes – like Performance Efficiency, Reliability, Security, Maintainability (how easy is it to fix or upgrade?), Flexibility (can it adapt to different tracks or conditions?), and even the newly added Safety (will it protect the driver in a crash?).

A quality requirement, then, is for you, the architect or designer, to look at that map and, for your specific project, plant a flag on certain attributes, saying, "Here, for this attribute, we need to achieve this specific level of quality." It is about translating those general quality concepts from the standard into concrete, verifiable goals that guide how you design, build, and test your software or system, ensuring it is not just functional, but truly excellent in the ways that matter most for its intended purpose. It is the promise you make about how well your creation will perform its duties and behave in the world.

## 2.2 IoT Software Systems Architecture

The design of robust and effective Internet of Things (IoT) software systems depends critically on a well-defined architecture that can manage the inherent complexity, heterogeneity, and scale of these interconnected environments. An IoT software architecture provides the blueprint for structuring the system, defining its components, their responsibilities, their interactions, and the overall data flow from sensor data acquisition to application-level insights and actions. Unlike traditional enterprise software architectures, IoT architectures must uniquely address challenges such as resource-constrained devices, diverse communication protocols, massive data volumes, and stringent requirements for security, reliability, and often real-time responsiveness (Gubbi et al. 2013; Atzori et al. 2010).

A common conceptualization of IoT architectures involves a layered approach, which helps in managing complexity by separating concerns. While specific implementations vary, a multi-layered model is frequently adopted,

typically encompassing layers for device interaction, network communication, data processing, and application services.

### *2.2.1 Foundational Layered Architectural Models*

A widely referenced architectural paradigm for IoT systems is the three-layer model, comprising the Perception (or Device) Layer, the Network Layer, and the Application Layer (Sethi and Sarangi, 2017; Khan et al., 2012).

- **Perception/Device Layer:** This foundational layer comprises the physical "things" – sensors, actuators, RFID tags, smart devices – responsible for interacting directly with the physical environment. Its primary functions include data acquisition from sensors (e.g., temperature, motion, location) and the execution of actions via actuators (e.g., controlling a valve, adjusting a thermostat). The heterogeneity of devices and communication protocols at this layer presents significant integration challenges (Gubbi et al. 2013).
- **Network/Connectivity Layer:** This layer is responsible for transmitting the data collected by the Perception Layer to data processing systems and for relaying commands from applications back to actuators. It encompasses a wide array of communication technologies, including short-range protocols (e.g., Bluetooth, Zigbee, Wi-Fi), long-range wide-area networks (LPWANs like LoRaWAN, Sigfox, NB-IoT), cellular networks (4G/5G), and wired connections (Al-Fuqaha et al., 2015; Sethi and Sarangi, 2017). Ensuring reliable, secure, and energy-efficient data transmission is a key concern for this layer.
- **Application Layer:** This is the topmost layer, responsible for delivering specific services and value to the end-users or other systems. It hosts IoT applications tailored to various domains, including smart cities, healthcare, and industrial automation. This layer processes and analyzes the data received from the network layer to provide insights, trigger actions, and present information through user interfaces or APIs (Khan et al., 2012).

To address more complex scenarios and the increasing need for intermediate data processing, more granular layered models have been proposed, such as five-layer architectures. These often introduce a Processing Layer (or Middleware Layer) between the Network and Application layers, and sometimes a Business Layer on top (Sethi and Sarangi, 2017; Wu et al., 2010).

- **Processing/Middleware Layer:** This layer is crucial for managing and processing the vast amounts of data generated by IoT devices before it reaches the application layer. Its functions include data filtering, aggregation, abstraction, semantic analysis, and often storage in databases. Middleware platforms play a vital role in providing device management, data normalization, and service discovery (Al-Fuqaha et al. 2015; Bandyopadhyay and Sen 2011).
- **Business Layer (Optional):** This layer manages the overall IoT system activities and services from a business perspective, including data analytics for business intelligence, process optimization, and decision-making based on the insights derived from the application layer (Sethi and Sarangi, 2017).

#### 2.2.1.1 Emerging Architectural Paradigms: Edge, Fog, and Cloud Computing

Traditional layered models are increasingly being augmented and sometimes reconfigured by distributed computing paradigms, such as Edge, Fog, and Cloud computing, which address specific IoT challenges, including latency, bandwidth, and data privacy (Shi et al. 2016; Bonomi et al. 2012).

- **Cloud Computing:** Serves as a centralized platform for extensive data storage, powerful data analytics, complex event processing, and scalable application hosting. Cloud platforms (e.g., AWS IoT, Azure IoT Hub, Google Cloud IoT) offer a comprehensive set of managed services that expedite IoT solution development (Botta et al. 2016). However, reliance solely on the cloud can introduce latency and bandwidth issues for time-sensitive applications and incur significant data transmission costs.

- **Fog Computing:** Proposed as an intermediate layer between edge devices and the cloud, Fog computing extends cloud capabilities closer to the data source. It consists of geographically distributed fog nodes (e.g., routers, gateways, local servers) that can perform localized data processing, analytics, storage, and control actions, thereby reducing latency, conserving network bandwidth, and enhancing responsiveness for critical applications (Bonomi et al. 2012; Chiang and Zhang, 2016).
- **Edge Computing:** Pushes computation, data storage, and application services even closer to the data sources, often directly onto the IoT devices themselves or local gateways. Edge computing is crucial for applications that require ultra-low latency, offline operation, and enhanced data privacy by processing sensitive data locally (Shi et al., 2016; Satyanarayanan, 2017). It also helps in reducing the volume of data transmitted to higher layers.

The interplay between these paradigms often results in hierarchical architectures (e.g., Edge-Fog-Cloud), where each tier handles tasks appropriate to its capabilities and proximity to the data source or end-user (Stojmenovic and Wen, 2014).

## 2.3 Large Language Model (LLM)

In recent years, Large Language Models (LLMs) have emerged as a pivotal technology within artificial intelligence, demonstrating remarkable capabilities in understanding, generating, and manipulating human language. These models, typically based on deep learning architectures such as the Transformer (Vaswani et al., 2017), are pre-trained on vast and diverse text corpora, enabling them to acquire extensive world knowledge and sophisticated linguistic patterns. Prominent examples, such as the GPT series (Brown et al., 2020; OpenAI, 2023), LLaMA (Touvron et al., 2023), and Gemini (Google, 2023), have demonstrated proficiency across a wide range of natural language processing (NLP) tasks, often achieving or surpassing human-level performance on various benchmarks.

The core strength of LLMs lies in their ability to perform in-context learning, where they can adapt to new tasks or generate specific types of output based on a few examples or instructions provided in a prompt, without requiring task-specific fine-tuning (Brown et al., 2020). This has opened new avenues for creating more intuitive and powerful human-computer interfaces, as well as for automating complex, knowledge-intensive processes.

Within the context of IoT software systems architecture, LLMs are beginning to play an increasingly significant role, particularly in areas that bridge human interaction with complex system data and control. One notable application is in Retrieval-Augmented Generation (RAG) systems (RAG) systems (Lewis et al., 2020; Gao et al., 2023).

Next, in this dissertation, we will consider how LLMs, particularly within RAG, can be leveraged to use structured knowledge about IoT software system architectures, quality requirements, and enabling technologies to provide decision support in IoT projects.

## 2.4 Retrieval Augmented Generation (RAG)

Imagine Large Language Models (LLMs) as incredibly smart and well-read scholars. They have read a vast library of books (their training data) and can discuss an enormous range of topics with impressive fluency. However, even the most brilliant scholar has limitations. Their knowledge is based on the books they have already read, so if new information comes out, or if you ask about a very niche, specialized topic not well-covered in their library, they might struggle. Sometimes, they might even try to "fill in the blanks" with information that sounds plausible but is incorrect, much like a scholar trying to bluff their way through a question. This is what the paper refers to as "hallucination."

Now, RAG is like giving that brilliant scholar a super-powered research assistant and an always up-to-date, specialized library.

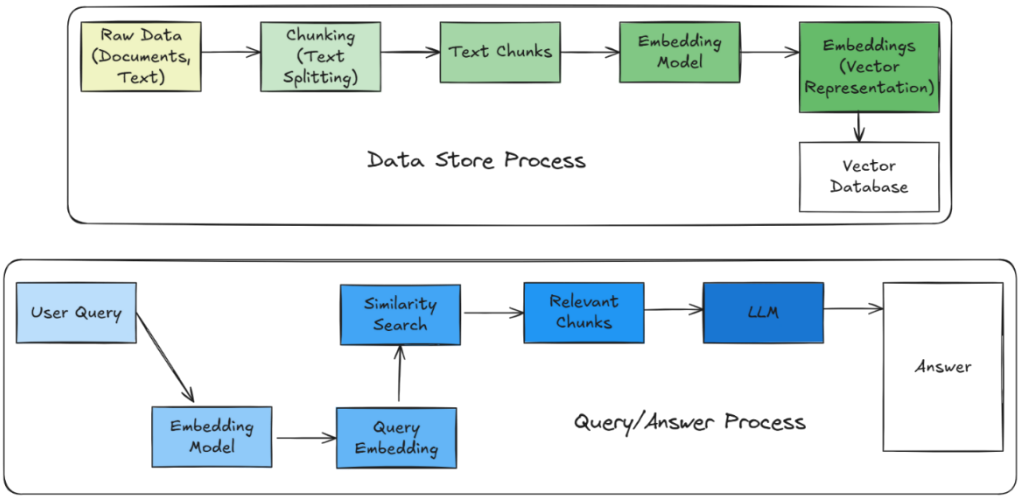



**Figure 2: Retrieval Augmented Generation Data Store and Query/Answer Process.**

RAG is a technique where an LLM's ability to generate text is enhanced by first looking up relevant information from an external source and using that information to inform its output (Jiang et al., 2023).

### *2.4.1 The Role of Prompt Engineering in RAG Systems*

Effective RAG systems rely heavily on prompt engineering, which is the art and science of crafting effective prompts to guide the LLM's behavior and elicit desired outputs (White et al., 2023). In a RAG context, the prompt typically instructs the LLM on how to utilize the retrieved context to answer the user's question. A well-designed prompt can significantly improve the quality, relevance, and factuality of the generated response.

The key components of a RAG prompt are:

- **Instruction:** Tells the LLM its task (e.g., "Answer the question based only on the provided context.").
- **Context Placeholder:** A designated section where the retrieved documents or text snippets will be inserted.
- **Question Placeholder:** Where the user's original query is placed.
- **Output Constraints (Optional):** Instructions on the desired format, length, or tone of the answer (e.g., "Provide a concise answer.", "If the context does not contain the answer, say 'I do not know.').

### *2.4.2 Leveraging Few-Shot Learning in RAG Prompts*

Few-shot learning is a prompt engineering technique where a few examples (shots) of desired input-output pairs are provided within the prompt itself (Brown et al., 2020). This helps the LLM better understand the task, the expected format, and the style of the desired response, especially for more complex or nuanced queries.

In a RAG system, a few-shot example can demonstrate how the LLM should synthesize an answer from the given context and a question.

By carefully engineering prompts, potentially incorporating few-shot examples, and leveraging frameworks, developers can build more robust, accurate, and contextually aware RAG systems that effectively harness the power of LLMs while grounding them in factual information.

**Table 1 - Few-shot RAG prompt example.**

| |
|---|
| You are a helpful AI assistant. Use the following pieces of retrieved context to answer the question. If you don't know the answer, simply say so. Don't try to make up an answer. Be concise and answer based *__only__* on the **provided context**. |
| ---<br>--- |
| **Context:** The LangChain framework provides modules for building applications powered by LLMs. Key components include Model I/O, Chains, and Agents. |
| **Question:** What are the key components of LangChain? |
| **Answer:** Key components of LangChain include Model I/O, Chains, and Agents. |
| --- |
| **Context:** Prompt engineering is crucial for RAG. It involves crafting prompts to guide the LLM. |
| **Question:** Why is prompt engineering important for RAG? |
| **Answer:** Prompt engineering is important for RAG because it involves crafting prompts to guide the LLM's behavior for better responses. |
| --- |
| **Context:** {retrieved_context} |

**Question:** {user_question}

**Answer:** {output}

## 2.5 Final Considerations about the Chapter

This chapter has established the theoretical framework for this dissertation. By examining the distinct yet interconnected concepts of IoT software systems architecture, the ISO/IEC 25010 quality model, and the capabilities of large language models, we have constructed a comprehensive framework for addressing the research problem. The discussion on IoT architectures highlighted the evolution from simple layered models to complex, distributed paradigms, such as Edge, Fog, and Cloud computing. This understanding is critical, as it defines the landscape of solutions that our proposed system must be able to represent and reason about.

The exploration of quality requirements, grounded in the ISO/IEC 25010 standard, provided a structured vocabulary for defining the non-functional "goodness" of a system. This framework is not merely theoretical; it serves as the primary mechanism for classifying and evaluating the architectural solutions within our knowledge base, enabling a more rigorous and standardized approach to design trade-offs.

Finally, the introduction of Large Language Models and, more specifically, the Retrieval-Augmented Generation (RAG) technique, provides the technological mechanism to make this knowledge actionable. RAG offers a powerful solution to the limitations of LLMs, such as knowledge cutoffs and hallucinations, by grounding their generative capabilities in a curated, factual knowledge base. The principles of prompt engineering and few-shot learning, as discussed, will be instrumental in designing the interaction between the user, the AI assistant, and the retrieved architectural data.

In essence, this chapter has laid out the "what" (the architectural knowledge and quality attributes) and the "how" (the RAG-based AI). The subsequent chapters of this dissertation will detail the practical implementation of

these concepts in the development and evaluation of the "IoT Architectural Design Assistant," demonstrating how this theoretical foundation is translated into a tangible research contribution.

# 3 Literature Study

*The present chapter aims to detail the methodology employed in conducting the literature study, as well as to present the quantitative results obtained, specifically the number of selected articles that will constitute the knowledge base for this research.*

## 3.1 Introduction

The Internet of Things (IoT) has emerged as a transformative technological paradigm, interconnecting the physical and digital worlds through a wide range of smart devices and communication networks. This proliferation of connected "things" has driven innovation across various sectors, including smart cities, precision agriculture, Industry 4.0, and personalized healthcare systems. At the core of the effectiveness and viability of such systems lies their software architecture – the fundamental structure that dictates how an IoT system's components interact, process data, and deliver value.

Designing software architecture for IoT software systems presents unique and multifaceted challenges (Gubbi et al., 2013; Al-Fuqaha et al., 2015). These systems are inherently complex, characterized by device heterogeneity, resource constraints (including energy, processing, and memory), scalability requirements for handling large volumes of data and devices, and the imperative need for security and privacy in often distributed and vulnerable environments (Atzori et al., 2010; Sethi & Sarangi, 2017). Additionally, the rapid evolution of IoT technologies and the diversity of application domains demand flexible and adaptable architectural approaches.

In this context, a systematic and comprehensive review of scientific and technical literature becomes fundamental. The primary objective of this chapter is to investigate the state-of-the-art in designing software architectures for IoT software systems. We will seek to identify the main proposed architectural patterns, the methodological approaches used for their design and evaluation,

prominent enabling technologies, and persistent challenges that still demand attention from the research community and industry. Particularly, this review will focus on architectures that prioritize quality requirements to compose a knowledge base.

The critical analysis of existing works will not only facilitate the consolidation of current knowledge but also identify gaps and opportunities for future contributions, thereby paving the way for the proposal of a knowledge base for designing resilient Internet of Things (IoT) architectures. It is expected that this review will provide a solid foundation for the development of the research presented in this dissertation, contributing to the advancement of knowledge in software engineering applied to Internet of Things software systems.

This research started with a Literature Search (LS) (Kuhrmann et al., 2017). We aimed to find articles published since 2019, based on the work of (Alreshidi and Ahmad, 2019). The search focused on technical literature regarding architectural design solutions that influence Quality Requirements (QRs) in IoT software systems. The central research question guiding this study is:

**"What application domains and characteristics of their IoT software systems architectures influence Quality Requirements (QRs)?"**

To answer the main research question, we divided it in 5 other research questions (Table 2). Our first search used a PICO-inspired search on Scopus (www.scopus.com) on December 22, 2022, and returned 130 articles. From these, we selected six to begin a Snowballing process (one level backward and forward) (Wöhlin, 2014). This gave us an initial group of papers, and two were highly relevant to our study's objectives. We then refined our search terms, incorporating insights from related works (Alreshidi & Ahmad, 2019; Razzaq, 2020). This led to a broader search in Scopus in July 2023, which produced 38 articles. We added four of these to our initial set, resulting in a total of ten papers. These ten papers were then used for another Snowballing trial (one level backward and forward), totalizing a final set of 28.

**Table 2: Research Questions.**

| | | | |
|---|---|---|---|
| **Research Questions** | **RQ1** | What are the application domains of IoT software systems? | |
| | **RQ2** | What are the proposed IoT software system architectures? | |
| | | **RQ2.1** | What are the characteristics of these IoT software system architectures? |
| | | **RQ2.2** | What are the QRs identified in these IoT software system architectures? |
| | | **RQ2.3** | How are these QRs worked out in these IoT software system architectures? |

To ensure the inclusion of the most recent publications and maintain the currency of the knowledge base, the search date range was extended. Consequently, the final search string (Table 3), but with the temporal constraint updated to **PUBYEAR > 2022 AND PUBYEAR <= 2024**, was re-executed on the Scopus database in December 2024. This updated search identified nine additional relevant articles.

**Table 3: Final search string performed in Scopus.**

| Search String (2024) |
|---|
| (software OR "Software Architect*") AND ("IoT" OR "Internet of Things") AND ("Quality Requirement" OR "Non-Functional" OR "Architectural Requirements") AND ("Architecture" OR "Architectural Elements" OR component OR design OR model OR framework) AND PUBYEAR > 2019 AND PUBYEAR <= 2024 |

These newly papers, along with the results from this latest search execution, then underwent a further cycle of Snowballing (one level backward and forward), followed by a comprehensive review of all extracted data. This final iterative process culminated in the selection of a total of 37 papers for inclusion in the knowledge base of this dissertation. The paper selection process involved first applying defined inclusion and exclusion criteria (Table 4). Subsequently, two

researchers thoroughly analyzed the chosen materials to address the research questions. This selection was then reviewed and confirmed by a final researcher.

**Table 4: Inclusion and exclusion criteria.**

| | | |
|---|---|---|
| **Criteria** | Inclusion | The paper must be in the context of IoT software systems. |
| | | The paper must report a primary study. |
| | | The paper must provide data to answer all the LS research questions. |
| | | The paper must be written in the English language. |
| | | The paper's publication date must be between 2020 and 2023 (later updated to 2024). |
| | Exclusion | Duplicate publication/self-plagiarism. |
| | | Register of proceedings. |
| | | Papers that are not peer-reviewed. |

The subsequent chart (Figure 3) illustrates the distribution of the selected articles according to their year of publication.

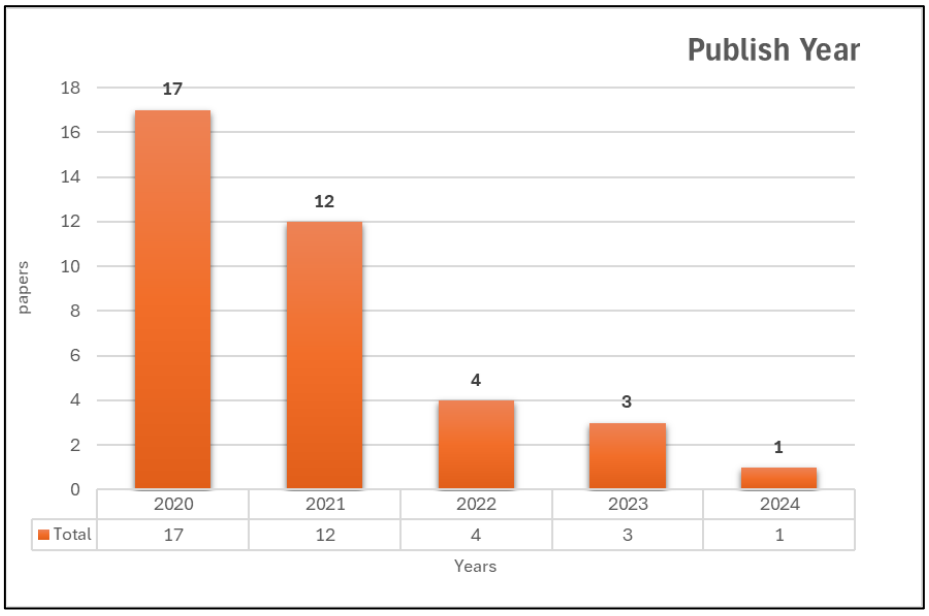



**Figure 3: Paper distribution by publishing year.**

## 3.2 Related Works

This current study highlights that choices about how to design Internet of Things (IoT) software systems need to be guided by up-to-date findings. The results from two important, related studies have a substantial impact on how this research is conducted.

First, Alreshidi and Ahmad (2019) focused on the difficulties in designing software for IoT settings. They pointed out how hard it is to create systems that can handle the wide variety, constant changes, and ability to grow that are typical of IoT systems. They also discussed the importance of incorporating security and privacy into the design of these systems. To address these design issues, their work proposed a basic model that utilizes cloud services and adheres to open standards.

Later, Razzaq (2020) carefully reviewed existing writings on software designs for IoT systems, focusing particularly on the use of microservice designs. Razzaq (2020) emphasized that IoT software systems must be capable of growth, flexibility, and the management of various types of data. The review examined older design patterns, such as n-tier and service-oriented designs, and highlighted their limitations when applied to IoT. After that, the study explored how microservice designs could provide benefits, such as improved organization into parts, flexibility, and the ability to integrate with other systems, to address the design challenges of IoT systems. The paper finishes by suggesting future research topics for using microservice designs in IoT.

In addition to these studies, it is also important to discuss IoT reference designs. This type of design provides a flexible plan for creating and testing systems, which is especially useful for Industrial IoT. It is not tied to specific technologies or rules, offering adaptable guidance on how networks, cloud services, and matching hardware should be structured. Experts from various fields typically recommend these designs to help transform industries using current technologies (Mirani et al., 2022).

To be more specific, reference designs show a basic layout for IoT software systems. They illustrate the main components, including hardware and software pieces, and how they connect to ensure an IoT-based software system functions correctly (Alreshidi and Ahmad, 2019). Typically, reference designs are created informally. However, using a planned design method is crucial to ensure they are of high quality, last a long time, and can be maintained. Even so, many reference designs do not become popular or last long after they are first released or published in journals or at science events (Nakagawa and Antonio, 2023).

## 3.3 Results

Upon completing this research, we identified various domains within IoT software systems, along with their principal architectural designs. These were characterized and linked to at least one Quality Requirement (QR) as defined by (ISO/IEC-25010 2023), such as security, performance/efficiency, or flexibility. We intended to generate a valuable Knowledge Base (presented in the next section), that supported informed decision-making for architectural choices during the creation of IoT software systems, particularly when prioritizing QRs. This involved classifying the identified IoT application areas, QRs, and software architectures. Subsequently, these architectural solutions were detailed, with an emphasis on the specific QRs they addressed. To conclude, we compiled the architectural elements relevant to each QR into distinct catalogs. Ultimately, the comprehensive set of findings will serve as input for an AI-driven application, designed to support informed decision-making throughout the design process of IoT software systems.

### *3.3.1 What are the application domains of IoT software systems?*

The study found varied terminology for similar IoT application domains in primary sources. To address this, a new classification of these domains was proposed, drawing on existing research. Additionally, 28 architectural solutions were identified, some specific to certain domains and others "Generic" for broader use.

### *3.3.2 What are the proposed IoT software systems architectures?*

The work identified a catalog of 37 proposed IoT software system solutions for diverse applications, including healthcare, Industry 4.0, smart cities, and smart farms. The catalog features a variety of architectural patterns, such as cloud-based systems, event-driven IoT, blockchain integration, geographic-based designs, layered structures, and fault-tolerant edge-computing frameworks. This diversity in design is intended to meet the specific challenges and quality requirements of each domain, illustrating the complexity of IoT software and the critical demand for customized solutions.

### *3.3.3 What are the QRs identified in these IoT software system architectures?*

The analysis of Quality Requirements (QRs) identified in the study highlights their role as critical design challenges in IoT software systems. Performance/Efficiency emerges as the most dominant QR, followed by Security and Flexibility. The prominence of security underscores the critical need for robust protective measures in IoT architecture—such as cryptography, certificates, blockchain, and layered security, particularly as comprehensive frameworks for managing and balancing multiple QRs are still emerging.

### *3.3.4 How are these QRs worked out in these IoT software system architectures?*

The architectural solutions investigated feature a diverse set of design elements to meet various quality requirements. These architectures are implemented using advanced technologies such as Kubernetes, Privacy-Preserving Searchable Encryption (PPSE), the FogBus framework, blockchain, Docker, and Software-Defined Networking (SDN). The use of these technologies provides a flexible and robust foundation for the design and implementation of IoT software systems.

### *3.3.5 What application domains and characteristics of their IoT software systems architectures influence Quality Requirements (QRs)?"*

The primary outcome of the study is a comprehensive Knowledge Base designed to support architectural decision-making for IoT systems. This Knowledge Base catalogs 37 distinct architectural solutions, systematically mapping their features to specific Quality Requirements (QRs). Ultimately, the research highlights the critical importance of the design phase, demonstrating that a thorough understanding of an application's domain, such as healthcare, is essential for selecting an architecture that can successfully meet its unique quality requirements.

### *3.3.6 Knowledge Base (KB)*

The preceding chapter embarked on a systematic exploration of the contemporary research landscape, meticulously reviewing and analyzing literature contributions pertinent to the architectural design of Internet of Things (IoT) software systems. That comprehensive Literature Review (Chapter 3) served not only to map the existing terrain of knowledge but also to identify a core set of impactful publications. These selected works provide a significant understanding of architectural patterns, quality attribute considerations, and enabling technologies within the IoT domain. From this rigorous selection process, a curated knowledge base emerged as the main result, serving as a foundational pillar for this research.

The primary objective here is to consolidate the collective insights embedded within these papers into a coherent and actionable repository of solutions knowledge. This endeavor moves beyond simple summarization, aiming to organize and categorize the extracted data in a manner that highlights shared characteristics, discerns trends, and makes complex architectural concepts more accessible and applicable for designers and researchers. The knowledge base constructed here is therefore a direct outcome of the systematic literature review process. It comprises systematically extracted data points covering several key facets of the selected articles. This includes the specific

architectural patterns, frameworks, or models proposed or analyzed within each study. Furthermore, the data captures the target IoT application domains (Atzori et al., 2010; Gubbi et al., 2013; Motta et al., 2019) (Figure 4), such as Smart Farm, Smart City, Industry 4.0, and Healthcare, for which these architectural solutions are intended.

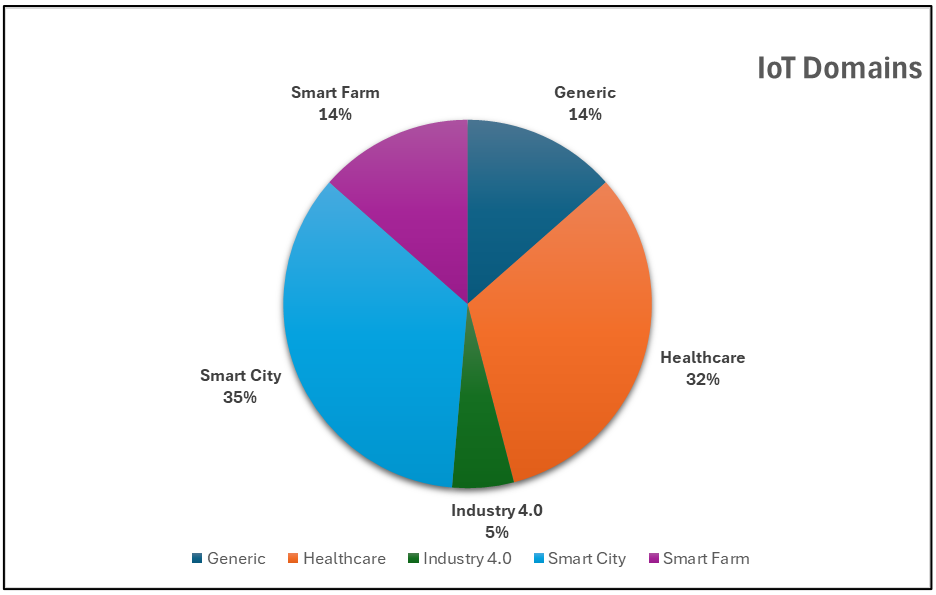



**Figure 4: Five IoT Domains found in the Literature Review.**

A crucial element of the extracted information pertains to the addressed Quality Requirements (QRs) – the non-functional, such as scalability, security, and reliability, often guided by frameworks like ISO/IEC 25010 (version 2023), that each solution aims to satisfy (Figure 5). The knowledge base also documents the enabling technologies and key design features highlighted as instrumental in realizing the proposed architectures and achieving desired quality attributes. Finally, to ensure traceability and facilitate further exploration, essential metadata from the source publications, including paper titles, authors, publication years, and access information such as DOIs or links, has been systematically recorded.

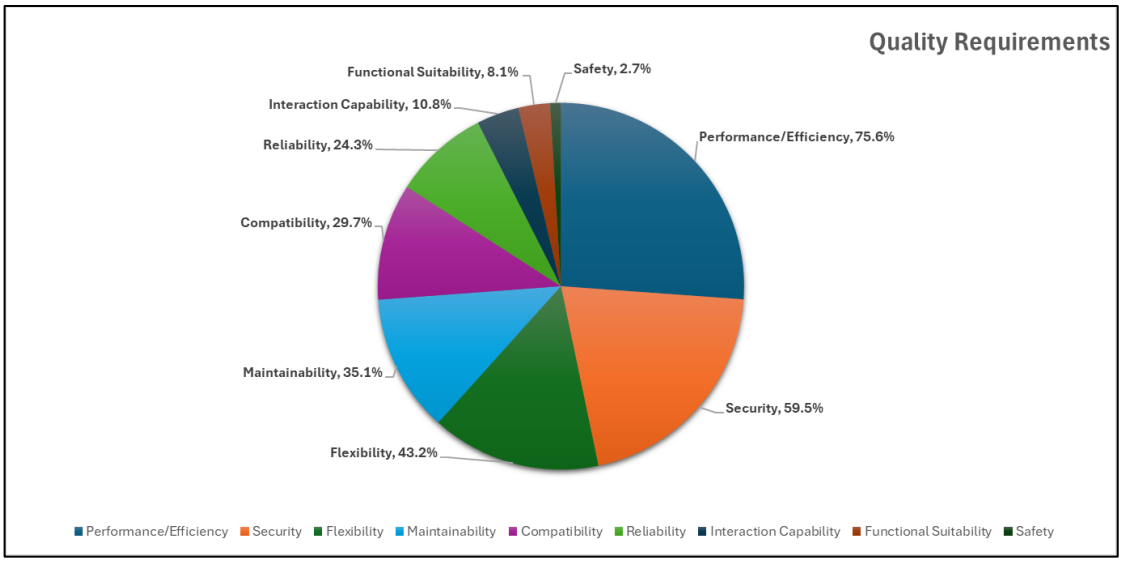



**Figure 5: How a solution addresses the Quality Requirement.**

The subsequent sections of this chapter will present this extracted information in a structured format. We will delve into a more detailed exposition of the individual architectural solutions cataloged, presenting each with its associated attributes as extracted from the source articles.

By consolidating and structuring this information, the ambition is to create a valuable resource that not only underpins the subsequent research and proposals within this dissertation but also serves as a reference point for practitioners and academics navigating the complex decision-making processes inherent in IoT system architecture design. This knowledge base forms the empirical bedrock upon which further analysis, pattern identification, and the development of design support mechanisms will be built.

In Appendix A – Extraction Data from Literature Review (Knowledge Base) all solutions extracted from the result (Knowledge base) of the literature review are presented and described.

#### 3.3.6.1 Knowledge Base Publication

The systematic literature review process described, along with the resulting knowledge base, forms the basis of a study that has been peer-reviewed and published. This work, which formally presents the mapping between IoT

application domains, architectural solutions, and their influence on quality requirements, is detailed in:

- Silva, F., Souza, B., & Travassos, G. (2024). **A Literature Study on Application Domains and IoT Software Systems Architectures Solutions Influencing Quality Requirements**. In: Anais do XXVII Congresso Ibero-Americano em Engenharia de Software, (pp. 181-195). Porto Alegre: SBC. doi:10.5753/cibse.2024.2844.

## 3.4 Final Considerations about the Chapter

This chapter provides a meticulous account of the methodological journey undertaken to construct the empirical foundation of this research. Through a rigorous and iterative process, combining a systematic search with the Snowballing technique, we have navigated the vast body of literature to address our central research questions. The application of strict inclusion and exclusion criteria, followed by a multi-researcher review process, ensured the quality and relevance of the selected works, culminating in a curated corpus of 37 seminal papers.

The quantitative results presented herein, such as the distribution of publications by year and the frequency of application domains and quality requirements, provide a clear snapshot of the current state-of-the-art in IoT software architecture. More importantly, the systematic extraction and analysis of data from these sources have led to the creation of the Knowledge Base, the primary outcome of this literature study.

This structured repository of information, which catalogs architectural solutions and maps them to specific application domains, quality requirements, and enabling technologies, is not an end in itself. Rather, it is the foundational asset that will drive the subsequent phases of this dissertation. The Knowledge Base provides the verified, context-rich data necessary to develop and train the AI-driven "IoT Architectural Design Assistant." The following chapter will now shift focus from knowledge acquisition to knowledge application, detailing the design

and implementation of this assistant and how it leverages the insights consolidated here.

# 4 Tool Proposal

*This chapter outlines the research proposal designed to address challenges in the architecture of IoT software systems. The core of this proposal comprises two interconnected components: the development of a structured Knowledge Base of architectural solutions derived from the literature review, and the creation of an AI-driven IoT Design Assistant that interacts with and leverages this knowledge for design support.*

## 4.1 Introduction

Addressing the complexities inherent in designing effective Internet of Things (IoT) software systems requires robust, evidence-based support. Building upon the theoretical foundations and literature review, this chapter outlines a research proposal aimed at providing such support. We propose the creation of a systematically compiled Knowledge Base of IoT architectural solutions (see section 3.3.6) and, critically, the development of an AI-powered IoT Design Assistant that utilizes this knowledge base to guide architects and researchers in their design endeavors.

## 4.2 A Knowledge-Driven Decision Support Tool for IoT Architectural Design

The preceding sections have detailed the construction and composition of a comprehensive knowledge base, systematically derived from an extensive review of contemporary literature on Internet of Things (IoT) software system architectures. This curated repository encapsulates valuable insights into architectural patterns, quality attribute considerations, and enabling technologies. However, to fully leverage this wealth of information and translate it into practical design guidance, a more interactive and assistive mechanism is beneficial.

This section introduces the IoT Architectural Design Assistant, a novel decision support tool developed as a key contribution of this research. Built upon the foundational knowledge base presented earlier, this tool is specifically designed to aid software architects, engineers, and researchers in navigating the complex decision-making processes inherent in IoT architectural design.

The primary objective of the assistant is to provide a structured and evidence-based approach to selecting appropriate architectural solutions and technologies that align with specific project requirements and desired quality attributes. By interacting with the tool, users can input their project context, prioritize quality requirements, and explore potential architectural strategies drawn directly from the curated knowledge. The following subsections will detail the architecture of this decision support tool, its core functionalities, and the underlying mechanisms that connect it to the knowledge base. This tool aims to bridge the gap between theoretical architectural knowledge and its practical application, empowering stakeholders to make more informed and effective design choices for their IoT software systems.

### *4.2.1 ArchIoTect: IoT Architectural Design Assistant*

As an outcome and an instrumental component of the research presented herein, a web application titled the "IoT Architectural Design Assistant" was developed. This application is engineered to function as an interactive interface, enabling software architects, engineers, and researchers to effectively engage with a curated knowledge base on Internet of Things (IoT) system architectures. The system's development was driven by the objective of systematically translating knowledge derived from peer-reviewed literature and established industry practices into actionable architectural guidance and a collaborative resource.

### *4.2.2 Main Requirements*

This module serves as the primary user interface for architects and researchers to interact with and manage the IoT architectural knowledge base. The functional and non-functional requirements are presented.

#### 4.2.2.1 Functional Requirements

- **FR-MR01** - The application shall display the knowledge base content in a hierarchical (tree-like) structure.
    - **FR-MR01.1** - The application shall allow the user to select and change the root node for the hierarchical presentation of the knowledge base.
    - **FR-MR01.2** - The application shall provide users with options to filter the displayed knowledge base content.
    - **FR-MR01.3** - The application shall provide filtering options, including, at a minimum: Architecture (Pattern/Name), IoT Domain, Quality Requirement, and Technology.
- **FR-MR01.4** – The Filtering option shall permit “criteria”, where a combination of options must be possible.
- **FR-MR02** - When a knowledge base item (e.g., an architectural solution, a specific architecture, an IoT domain, a quality requirement, a technology) is selected, the application shall display a detailed description and associated attributes for that item (source paper details, description, target domain, addressed QRs, and technologies used).
- **FR-MR03** - The application shall provide an administrative interface for managing the knowledge base content.
    - **FR-MR03.1** - Administrative users shall be able to perform CRUD (Create, Read, Update, Delete) operations on all primary entities within the knowledge base, including but not limited to: Architectural Solutions, Architectural Patterns/Names, IoT Domains, Quality Requirements, Technologies, and Paper References.
- **FR-MR04** - The application shall implement role-based access control to restrict knowledge base management functionalities to authorized administrative users.

#### 4.2.2.2 Non-functional Requirements

- **NFR-MR01** - The IoT Architectural Design Assistant shall communicate with the AI Assistant module via its defined APIs to submit user queries and receive synthesized architectural guidance.

### *4.2.3 Technological Implementation*

As shown in Figure 4, IoT Architectural Design Assistant is implemented utilizing a contemporary technology stack selected to ensure robustness, maintainability, and a responsive user experience. The backend infrastructure was developed using the Spring Boot framework, version 3.4.2, leveraging its capabilities for building enterprise-grade applications within the Java ecosystem. The core backend logic was implemented in Java (version 21), chosen for its modern language features and long-term support.

The user interface (UI), through which architects and researchers interact with the system, was constructed using Vaadin (version 24) (Vaadin, 2024), a Java-based web application framework. This selection facilitates a unified development approach, allowing for UI construction using Java, thereby aligning frontend and backend development paradigms. Vaadin's component-centric model supports the creation of rich, interactive interfaces necessary for presenting, navigating, and potentially contributing to complex architectural information.

For data persistence, the application employs an embedded H2 Database Engine (H2 Database Engine, 2023). H2 offers a lightweight, SQL-compliant relational database solution that manages the application's underlying knowledge base, user interaction data, and potentially researcher contributions.

The entire application is containerized using Docker, ensuring environmental consistency, simplifying deployment, and enhancing portability for use by the intended audience of architects and researchers across diverse platforms.

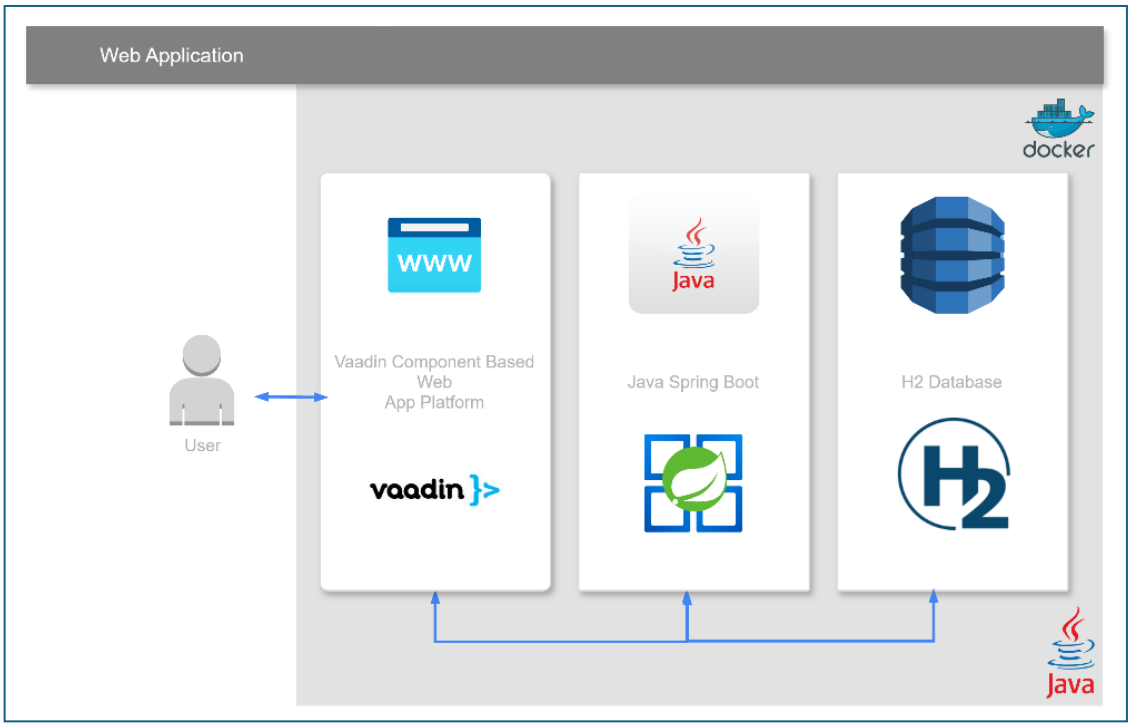



**Figure 6 - IoT Architectural Design Assistant Architecture.**

### *4.2.4 AI Assistant Module Architecture*

Complementing the web-based "IoT Architectural Design Assistant" (described in Section 4.2.1), an AI-powered assistant module has been developed to provide intelligent query processing and generative architectural guidance. This AI assistant functions as a backend service, programmatically accessible via Application Programming Interfaces (APIs), and is designed to interpret user queries related to IoT software system architecture, retrieve relevant information from the curated knowledge base, and synthesize coherent, context-aware responses.

### *4.2.5 Main Requirements*

This module provides intelligent query processing and generative guidance, accessed via APIs by the IoT Architectural Design Assistant.

#### 4.2.5.1 Functional Requirements

- **FR-MR01 -** The AI module shall retrieve relevant documents from the knowledge base based on a user query.
- **FR-MR02 -** The AI module shall synthesize a textual answer using the retrieved documents and the user query.

#### 4.2.5.2 Non-Functional Requirements

- **NFR-MR01 -** The AI Assistant shall expose a set of well-defined APIs (e.g., RESTful) for interaction with the IoT Architectural Design Assistant web application.
- **NFR-MR02 -** The AI module shall provide answers only based on knowledge base data.
- **NFR-MR03 -** The AI module shall not, in any circumstance, search or access external sources.
- **NFR-MR03** - The “Temperature” parameter for the AI module shall be set to 0.2.
- **NFR-MR04 –** The AI module shall be configurable to utilize either a locally hosted Large Language Model (LLM) or an online LLM service with minimal code modifications.

### *4.2.6 Technological Implementation and RAG Architecture*

As in Figure 7, the AI assistant is developed primarily in Python, version 3.12.9, leveraging its extensive ecosystem for machine learning and natural language processing. The core of its functionality is built upon a Retrieval Augmented Generation (RAG) architecture. This RAG approach enables the assistant to ground its responses in the information contained within the specialized IoT architectural knowledge base, mitigating the risk of hallucination and ensuring domain-specific relevance.

Inter-component communication between the primary "IoT Architectural Design Assistant" web application and this AI module is facilitated through a set

of well-defined APIs. These APIs are implemented using the Flask micro-framework (Pallets Projects, 2024), chosen for its lightweight nature and suitability for developing RESTful services in Python.

The knowledge ingestion and retrieval pipeline utilizes LangChain (LangChain Team, 2025), an open-source framework for developing applications powered by large language models. LangChain is employed for several critical tasks:

- **Document Processing:** It facilitates the loading and preprocessing of textual data from the curated knowledge base (derived from the 37 selected articles as detailed in Chapter 3) and managed via the H2 database of the primary web application.
- **Text Chunking:** Sophisticated text splitting strategies within LangChain are used to divide the knowledge base content into semantically coherent chunks, optimized for effective embedding and retrieval.
- **Embedding Generation:** Numerical vector representations (embeddings) of these text chunks are generated using an embedding. These embeddings capture the semantic meaning of the text.
- **Vector Storage and Retrieval:** The generated embeddings, along with their corresponding text chunks, are stored and indexed in a ChromaDB vector database (ChromaDB Team. 2024). ChromaDB is utilized for its efficiency in performing similarity searches, enabling the RAG system to retrieve the most relevant chunks of information from the knowledge base based on the semantic content of a user's query.
- **Prompt Templates:** LangChain's PromptTemplate class is essential for dynamic prompt construction. It allows developers to define prompt structures with placeholders for context, questions, and few-shot examples, which are then formatted with the actual data at runtime.

Google’s Gemini Large Language Model (LLM) was chosen for this implementation to orchestrate the generative component of the RAG architecture. Upon receiving a user query (via the Flask API) and the relevant context retrieved

from ChromaDB by LangChain, the Gemini model is prompted to synthesize a comprehensive and contextual appropriate response, adhering to the structured output formats and constraints defined for the IoT Architectural Design Assistant.

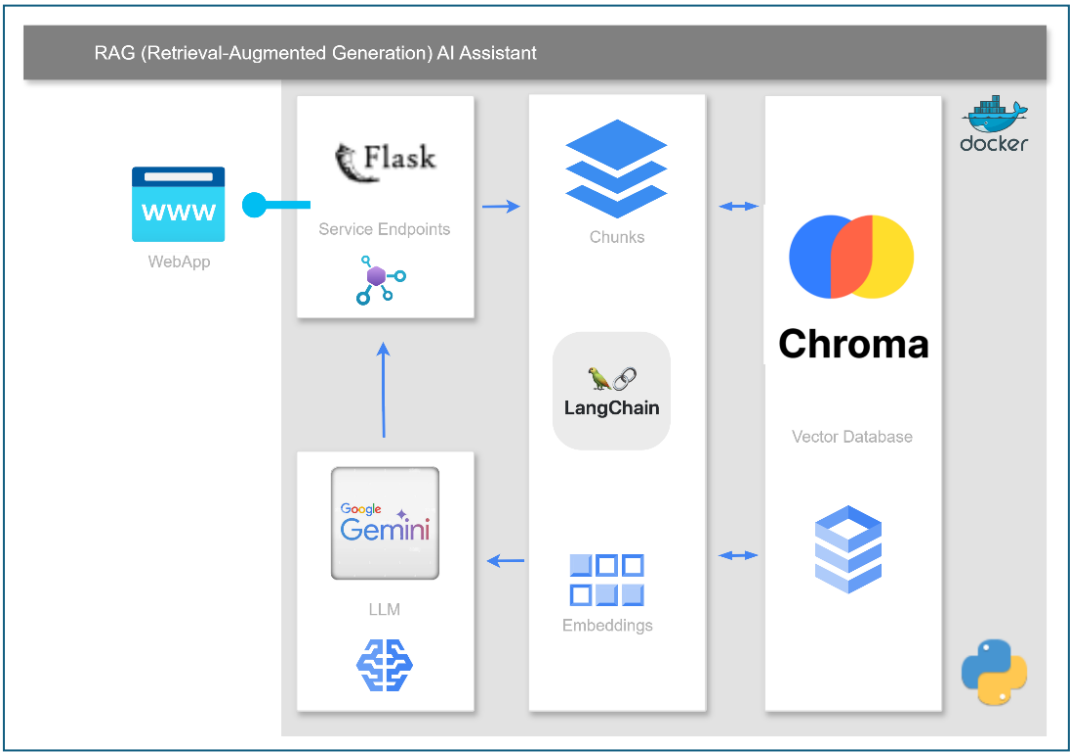



**Figure 7 - AI Assistant Architecture.**

#### 4.2.6.1 Role in the Ecosystem and Interaction Flow

When a user (architect or researcher) poses a query through the "IoT Architectural Design Assistant" web interface, the query is routed to the AI assistant's Flask API. The AI assistant then:

- Processes the query and uses LangChain to retrieve the most relevant document chunks from the ChromaDB vector store.
- Constructs a detailed prompt, combining the user's query with the retrieved contextual information.
- Sends this prompt to Google Gemini LLM for response generation.
- Receives the LLM's response and returns it to the web application for presentation to the user.

- This AI assistant, therefore, acts as the intelligent engine that transforms the static knowledge base into a dynamic and interactive resource, capable of providing nuanced and context-specific architectural guidance for IoT system design.

This AI assistant, therefore, acts as the intelligent engine that transforms the static knowledge base into a dynamic and interactive resource, capable of providing nuanced and context-specific architectural guidance for IoT system design.

## 4.3 User Interface Overview

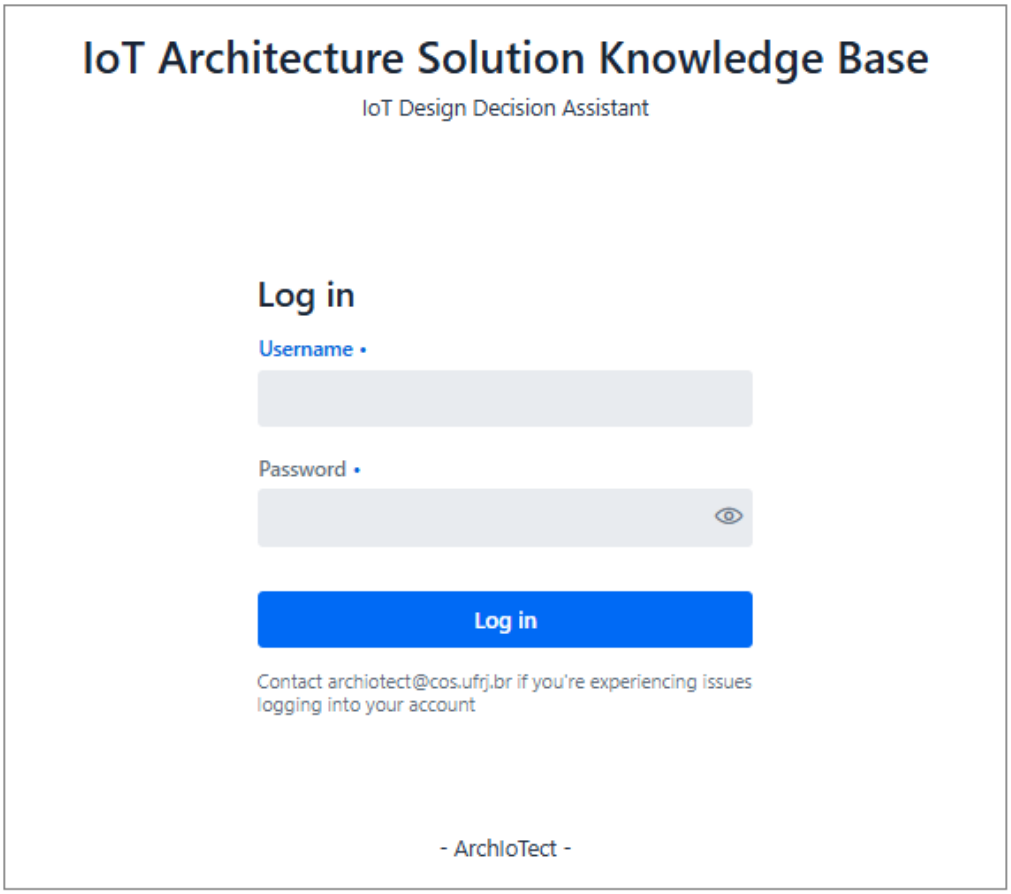



**Figure 8 - The Login page of the application.**

Figure 8 displays the application's login page. This screen serves as the primary entry point, requiring users to enter their registered username and

password to access the system's functionalities. Upon successful authentication, users are directed to the main home page.

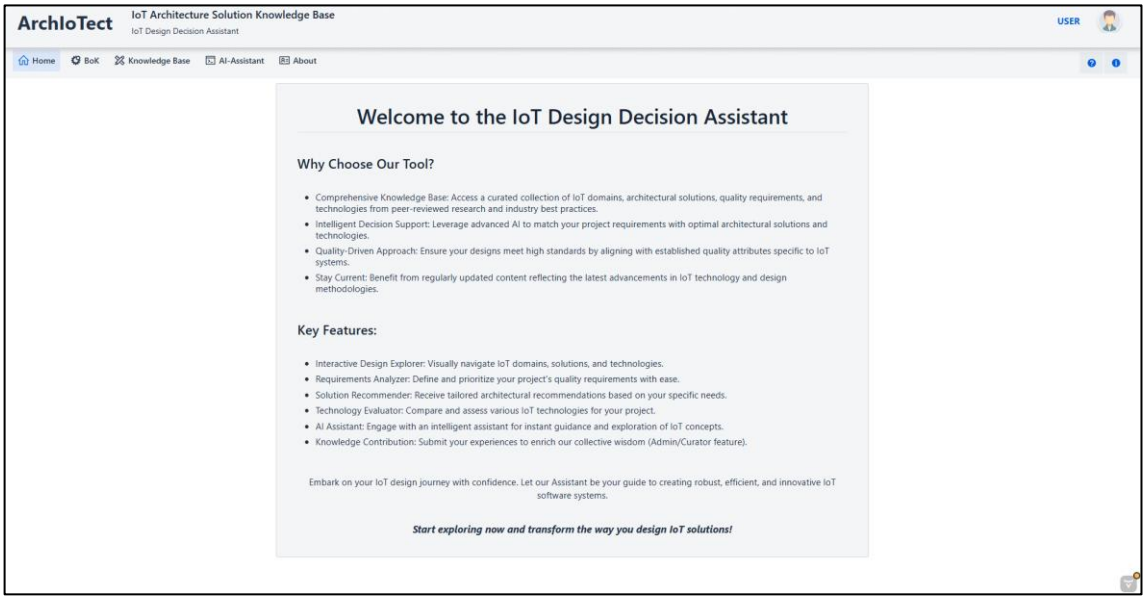



**Figure 9 - The Home page of the application.**

The home page interface, as shown in Figure 6, is the initial screen displayed to authenticated users. It integrates the primary navigation structure, allowing access to all system modules. The menu system employs role-based access control, dynamically rendering menu options based on the permissions associated with the authenticated user's profile, thereby restricting access to unauthorized functionalities (e.g., Knowledge Manager, User Manager, App Config).

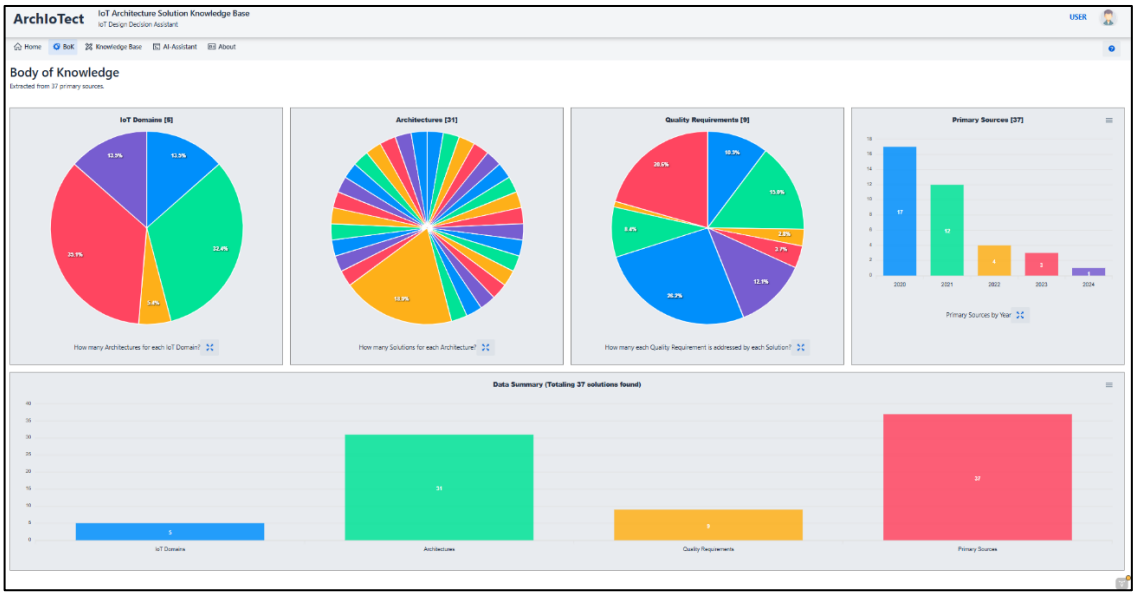



**Figure 10 - The Body of Knowledge of the KB.**

The interface for presenting knowledge base statistics, depicted in **Figure *10*** and titled Body of Knowledge, aggregates and displays quantitative data about the knowledge repository. This includes, but is not limited to, IoT Domains and QR code findings, total records, and classification breakdowns. These statistics serve to characterize the knowledge base and support its management and utilization.

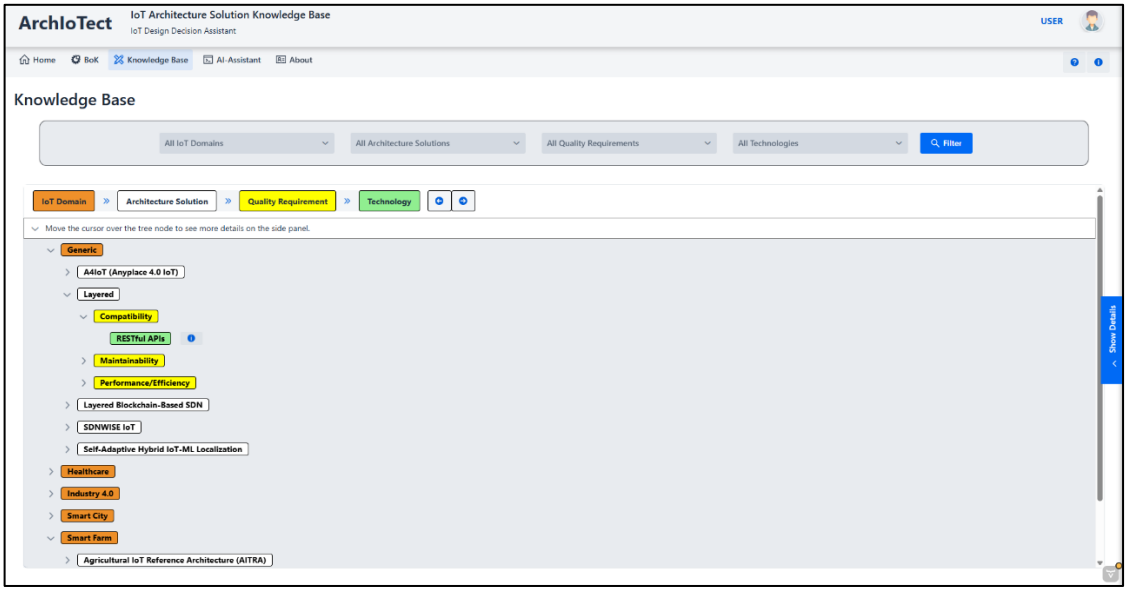



**Figure 11 - The Knowledge Base Visualization Page.**

The 'Knowledge Base Visualization' interface, depicted in Figure 11, facilitates an in-depth exploration of the underlying data. Users can navigate the static representation of the knowledge, delve into the technical and reference details of its hierarchical structure (Figure 12 and Figure 13), and apply filters to focus on specific subsets of information. A key interactive feature enables users to re-root the hierarchical display, allowing them to examine data relationships from various starting points.

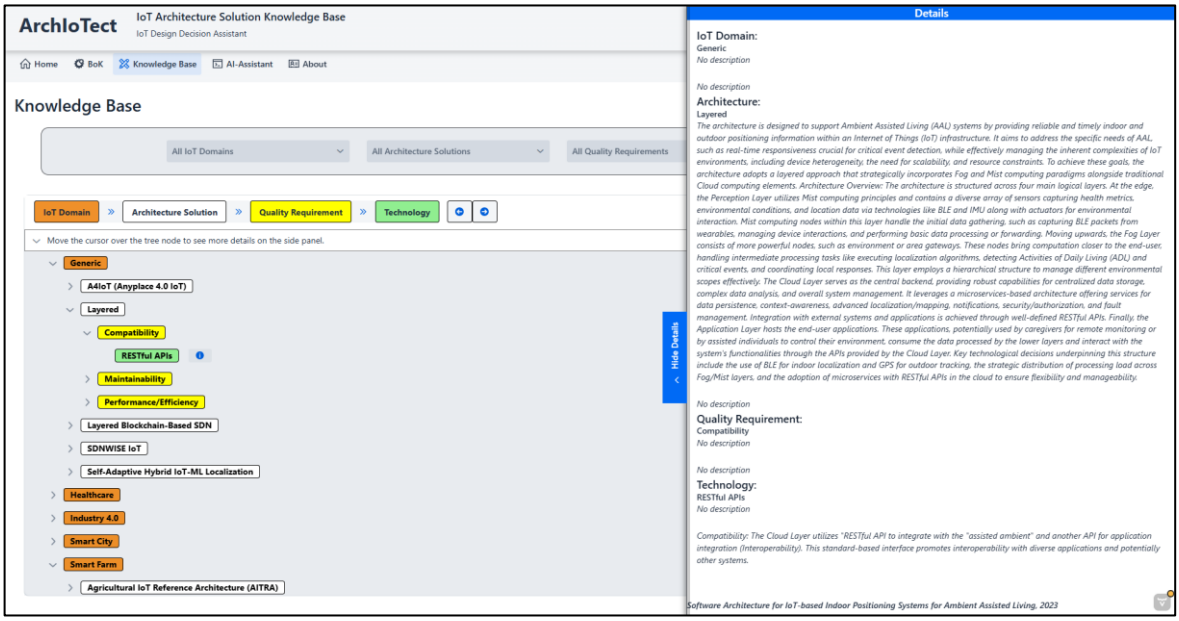



**Figure 12 - Showing details about the data node.**

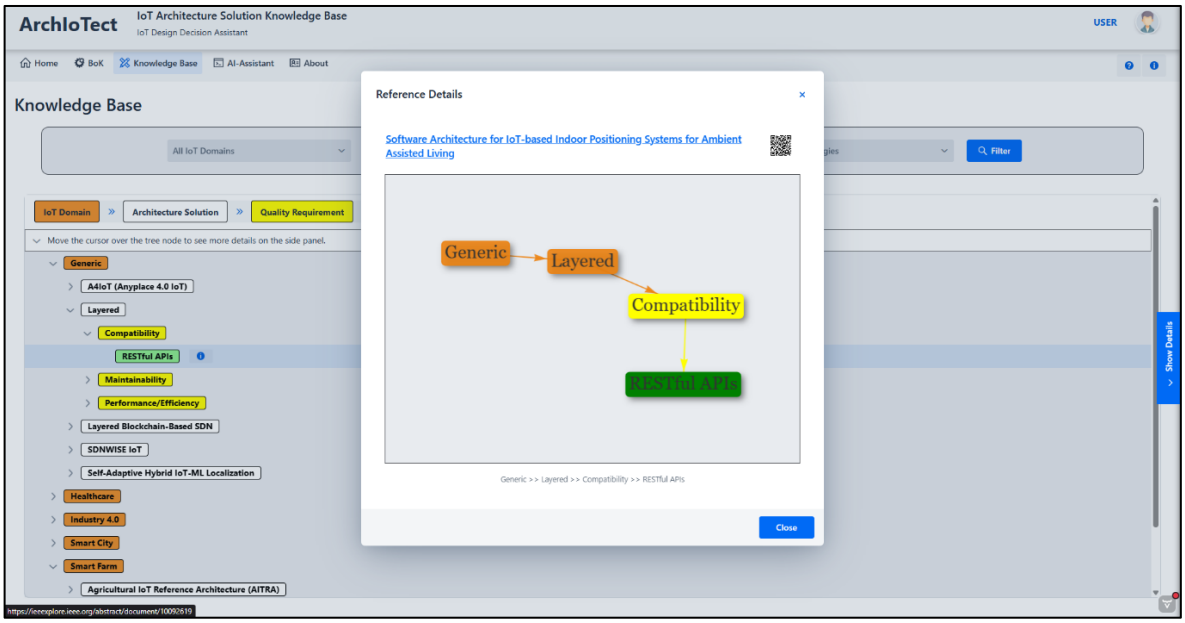



**Figure 13 - Showing reference details about the data node.**

The 'AI Assistant' page, shown in Figure 14, enables users to engage with the system's AI-powered capabilities. Through a chat-like interface, users can ask questions about IoT architectures. The AI assistant then utilizes the underlying knowledge base to understand the query, locate relevant documents, and generate informative textual responses, providing intelligent decision support.

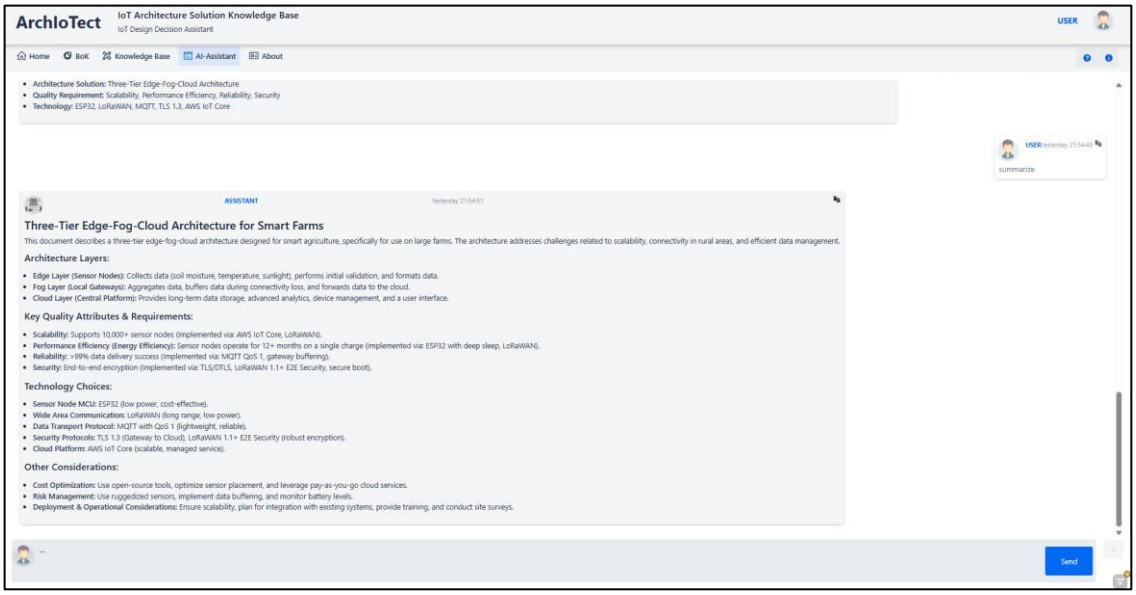



**Figure 14 – AI-based assistant.**

The 'Knowledge Base Manager,' shown in Figure 15, is the administrative interface for overseeing the system's knowledge repository. Authorized

administrators can use this section to manage all aspects of the knowledge base content, including adding new entries, modifying existing information, and removing outdated or incorrect data across various categories like IoT domains, architectures, QRs, and paper references.

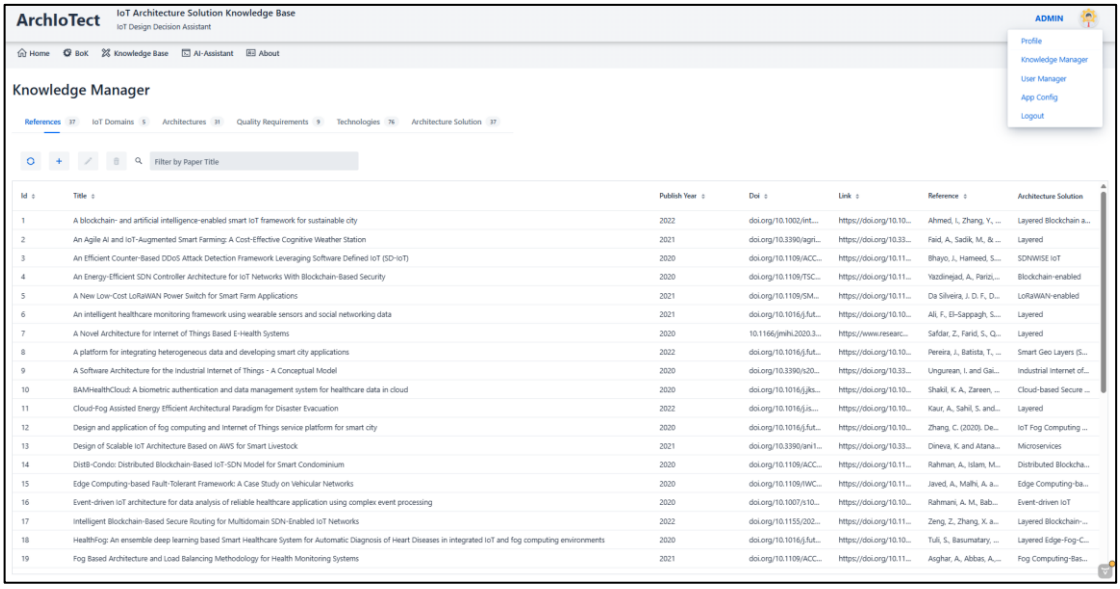



**Figure 15 - Knowledge Base manager page.**

## 4.4 Final Considerations about the Chapter

This chapter details the design and implementation of the proposed research artifact, the "IoT Architectural Design Assistant." We have transitioned from the theoretical concepts presented in the preceding chapters to the tangible construction of a dual-component system, designed to address the challenges of IoT architectural design.

The architecture presented is deliberately decoupled, comprising two distinct yet interconnected modules. The primary web application, built with a robust Java and Spring Boot Framework, serves as the user-facing portal for knowledge exploration, visualization, and management. Its detailed functional requirements ensure a structured and navigable interface to the curated Knowledge Base.

Complementing this, the AI Assistant module, developed using Python's rich data science ecosystem, functions as the intelligent engine. By implementing

a sophisticated Retrieval-Augmented Generation (RAG) architecture, this module transforms the static Knowledge Base into a dynamic resource, capable of synthesizing information and providing context-aware answers to complex design queries. The non-functional requirements for this module, particularly those concerning data grounding and model temperature, were defined to ensure the reliability and factuality of its responses.

In essence, this chapter outlines the complete technical blueprint of the solution. It has defined the system's functionalities, its technological underpinnings, and the interaction flow between its components. With the system's architecture and functionality now clearly established, the subsequent chapter will shift focus from implementation to validation, detailing the methodological approach designed to evaluate the tool's effectiveness, efficiency, and overall utility in supporting architectural decision-making.

# 5 Evaluating the First Version of ArchIoTect

*This chapter presents a feasibility study of the ArchIoTect tool based on the Technology Acceptance Model (TAM) to assess its perceived usefulness and ease of use. The findings provide preliminary evidence of the tool's viability and suggest a strong potential for positive user acceptance.*

## 5.1 Introduction

Assessing the potential success of a new software tool requires a robust theoretical foundation. As established in software engineering research, experimental evaluation is crucial for validating new approaches. To this end, this chapter employs the Technology Acceptance Model (TAM), originally proposed by Davis (1989), to conduct a feasibility study on the *ArchIoTect* tool. The TAM provides a proven framework for this, focusing on two critical indicators of user adoption: perceived usefulness and perceived ease of use. By applying this model, this study aims to verify whether *ArchIoTect* achieves its main objectives and gather empirical evidence to support its feasibility and acceptance by potential users.

## 5.2 Feasibility Plan

The criticality of architectural definition in software engineering is further emphasized in the field of IoT software systems, which are characterized by their autonomy and complexity in terms of both hardware and interaction. The *ArchIoTect* tool was developed to help with this tough decision-making task. Therefore, this chapter presents a feasibility study to evaluate the first version of *ArchIoTect*, based on the influential Technology Acceptance Model (TAM) proposed by Davis (1989). The objective is to verify whether the tool is perceived as useful and easy to use, thus providing evidence of its feasibility and potential for acceptance by the development community.

### *5.2.1 Planning*

At this stage, the study design was prepared, encompassing all artifacts for participants and researchers (Appendix A). This included a Consent Form, a participant characterization form, and a detailed description of the study's problem and its respective working scenarios.

The object of study is the *ArchIoTect* tool, which aims to support architectural decision-making in IoT software system projects. The intention is to evaluate and discuss the tool's feasibility. Therefore, it is hoped to answer the following question:

*"Is the ArchIoTect tool a feasible solution for providing decision support to software engineering professionals?"*

Following the recommendation of Barcelos and Travassos (2006), who emphasize the importance of experimental studies in academic environments as an initial step to validate new technologies, this study was conducted with students in the second semester of 2025 of a graduate program in Software Engineering. This approach aims to reduce the risk of introducing immature technologies into the industrial environment.

### *5.2.2 Execution*

The *ArchIoTect* tool evaluation was conducted with a group of 16 participants, all of whom were graduate students in software engineering and professionals working in the IT field, with varying levels of experience and in different roles. This group was carefully selected to ensure a representative sample of potential users with varied technical knowledge, allowing for evaluation of the tool from multiple perspectives. The study execution was structured in three sequential and well-defined phases: preparation, task execution, and data collection.

#### 5.2.2.1 Phase 1: Preparation and Instruction of Participants

Initially, to ensure that all participants had a basic and homogeneous level of knowledge about the tool's operation, a detailed explanatory video was made

available. This video demonstrated the main features of *ArchIoTect*, including navigation through the knowledge base, use of the filter system, interaction with the AI assistant, and visualization of architectural solution details. The objective of this stage was to mitigate biases arising from the learning curve and allow participants to focus on evaluating the tool's effectiveness, usability, and potential for adoption, rather than on the initial exploration of its features.

#### 5.2.2.2 Phase 2: Task Execution with Challenge Scenarios

After watching the video, participants were presented with three challenge scenarios based on real problems in the IoT domain. Each scenario described a software system to be designed, with its respective main functional and non-functional requirements. Each participant was instructed to choose one of the three scenarios, the one with which they had the greatest affinity or interest, and then use the *ArchIoTect* tool to propose an architectural solution that met the demands of the chosen challenge.

This stage represented the practical use of the tool. During this phase, participants were able to freely explore *ArchIoTect's* features, such as searching the hierarchical knowledge base and consulting the AI assistant, to analyze, compare, and select the architectures they considered most appropriate or combine features from different solutions to build a proposal. The goal was to simulate a real decision-making process, where the tool would serve as the main technical support.

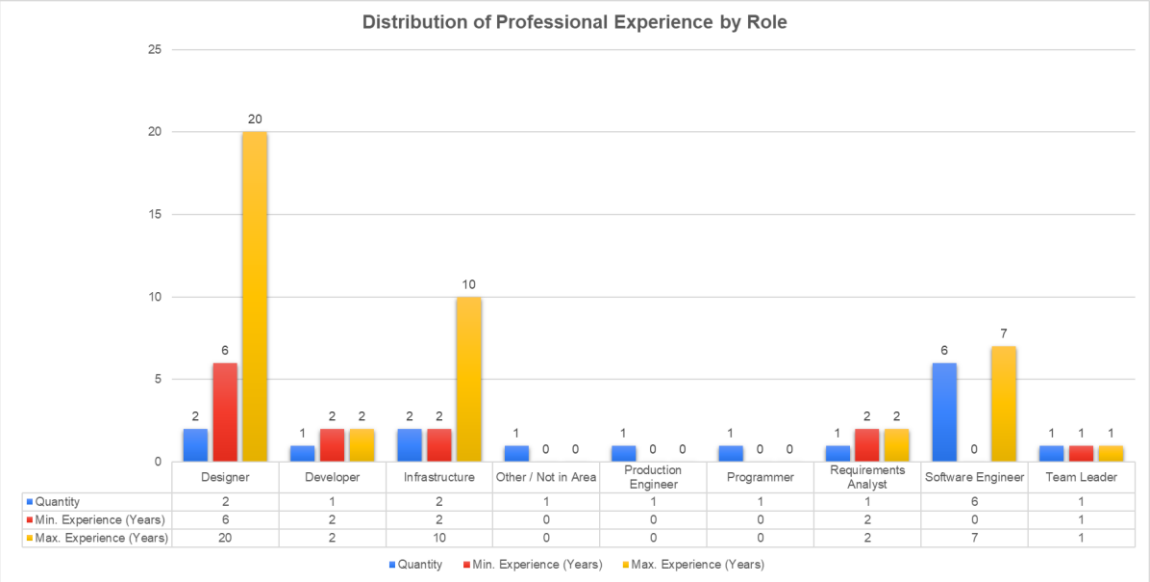
#### 5.2.2.3 Phase 3: Post-Validation Data Collection

Upon completing the scenario resolution, each participant was directed to fill in a post-validation questionnaire. This instrument, structured based on the TAM model, was designed to collect quantitative data (through a Likert scale) and qualitative data (through open-ended questions) to assess the tool's perceived usefulness, perceived ease of use, and behavioral intention to use. In addition, the questionnaire included a space for participants to provide feedback, criticism, and suggestions for improvement.

The process was designed to simulate a complete cycle of learning and use, from introduction to the tool to its application to a practical problem, thereby providing a rich and contextualized database for the feasibility analysis of ArchIoTect.

### *5.2.3 Characterization Results*

All the participants signed the consent form and agreed to participate. From the data collected through the characterization form, it was possible to establish a general profile for each participant, the details of which are consolidated in Figure 16.




| | Designer | Developer | Infrastructure | Other / Not in Area | Production Engineer | Programmer | Requirements Analyst | Software Engineer | Team Leader |
|---|---|---|---|---|---|---|---|---|---|
| Quantity | 2 | 1 | 2 | 1 | 1 | 1 | 1 | 6 | 1 |
| Min. Experience (Years) | 6 | 2 | 2 | 0 | 0 | 0 | 2 | 0 | 1 |
| Max. Experience (Years) | 20 | 2 | 10 | 0 | 0 | 0 | 2 | 7 | 1 |

**Figure 16 - Years of Professional Experience by Role. This chart illustrates the experience profile of the 16 study participants.**

The professional experience of the 16 participants was diverse, covering a wide range of software development roles and seniority levels. The largest group consisted of Software Engineers (six participants), whose experience ranged from 0 to seven years, with a median of two years. The Designers (two participants) represented the most senior members, with experience ranging from six to 20 years (median: 13). The Infrastructure group (two participants) had experience between two and ten years (median: six years). The remaining roles were represented by single participants, including a Developer (2 years), a Requirements Analyst (2 years), a Team Leader (1 year), and several participants

with no experience in roles such as Production Engineer, Programmer, and Others.

In addition to their professional roles, the participants' prior knowledge of IoT software systems was also varied (Figure 17). The most common level of experience, reported by half of the participants (eight out of 16), was a foundational knowledge acquired through self-study, such as readings and lectures. A significant portion of the group (four participants) reported having no prior knowledge at all, while the remainder had more direct experience through practical projects (three participants) or formal courses (one participant).

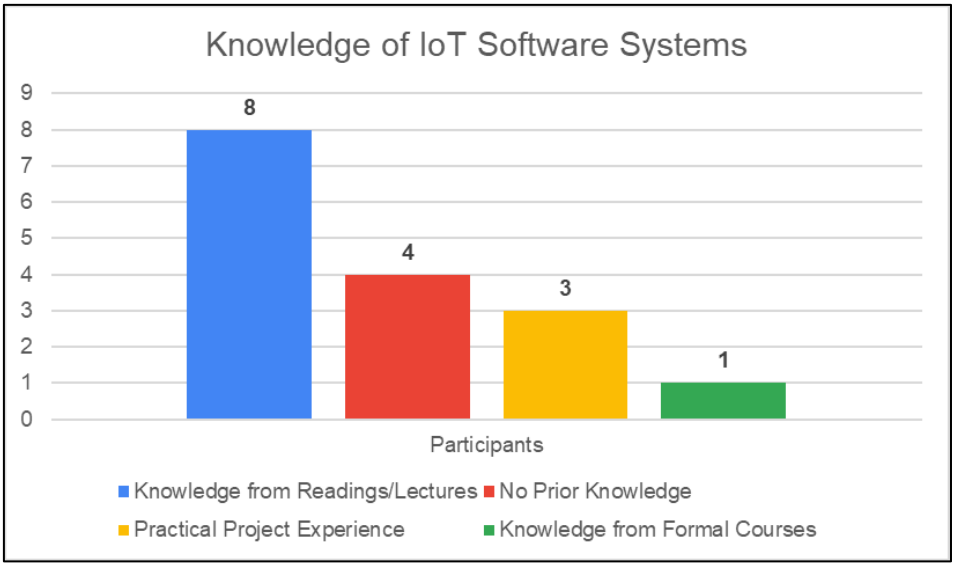



**Figure 17: Participants' knowledge of IoT software systems.**

### *5.2.4 Quantitative analysis of user perceptions*

A quantitative analysis of the *ArchIoTect* tool was conducted to objectively measure user perceptions following a practical evaluation task. Data was collected from a diverse group of 16 IT professionals, all of whom were also postgraduate students in software engineering. The questionnaire was administered via *Google Forms*, whose linear scale format guided the use of a 10-point range. Consequently, a 10-point Likert scale was employed, where one meant "*Completely disagree*" and ten meant "*Completely agree*". Self-assessment of software architecture experience was also measured on a scale of one to ten, from "*Totally unfamiliar*" to "*I am an expert*".

Table 5 presents a detailed breakdown of the quantitative data collected from the user survey (N=16). For each of the four questionnaire items, the table displays the raw response data, the frequency distribution of those responses, and the resulting mode. Furthermore, it maps each item to its corresponding Technology Acceptance Model (TAM) construct, providing a transparent foundation for the analysis discussed in this section.

**Table 5: Mapping Questions to TAM Constructs and Summary of Quantitative Results.**

| **The *ArchIoTect* tool interface is intuitive and easy to use.** | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **TAM Construct** | Perceived Ease of Use (PEOU) | | | | | | | | | | | | | | | |
| **Data (N=16)** | 7 | 10 | 10 | 7 | 9 | 7 | 6 | 7 | 8 | 5 | 7 | 10 | 9 | 7 | 8 | 10 |
| **Frequency** | 5:1 | | 6:1 | | **7:6** | | | 8:2 | | | 9:2 | | | 10:4 | | |
| **I am satisfied with the overall quality of the recommendations provided by the *ArchIoTect* tool.** | | | | | | | | | | | | | | | | |
| **TAM Construct** | Perceived Usefulness (PU) | | | | | | | | | | | | | | | |
| **Data (N=16)** | 9 | 9 | 8 | 9 | 8 | 9 | 10 | 7 | 8 | 8 | 8 | 10 | 9 | 5 | 8 | 7 |
| **Frequency** | 5:1 | | | 7:2 | | | **8:6** | | | 9:5 | | | 10:2 | | | |
| **The *ArchIoTect* tool met my expectations for assisting with architectural decisions.** | | | | | | | | | | | | | | | | |
| **TAM Construct** | Perceived Usefulness (PU) | | | | | | | | | | | | | | | |

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Data (N=16)** | 8 | 10 | 8 | 8 | 9 | 10 | 8 | 7 | 5 | 8 | 9 | 10 | 8 | 6 | 9 | 8 |
| **Frequency** | 5:1 | | | 6:1 | | | 7:1 | | **8:7** | | | 9:3 | | | 10:3 | |
| **The *ArchIoTect* tool should be recommended to other professionals working with IoT software system architectures.** | | | | | | | | | | | | | | | | |
| **TAM Construct** | Behavioral Intention (BI) | | | | | | | | | | | | | | | |
| **Data (N=16)** | 9 | 10 | 9 | 7 | 9 | 10 | 10 | 7 | 6 | 10 | 8 | 10 | 10 | 10 | 9 | 9 |
| **Frequency** | 6:1 | | | 7:2 | | | 8:1 | | | 9:5 | | | **10:7** | | | |

The aggregate results, presented in Figure 18, reveal a consistently positive reception of the tool across all measured constructs.

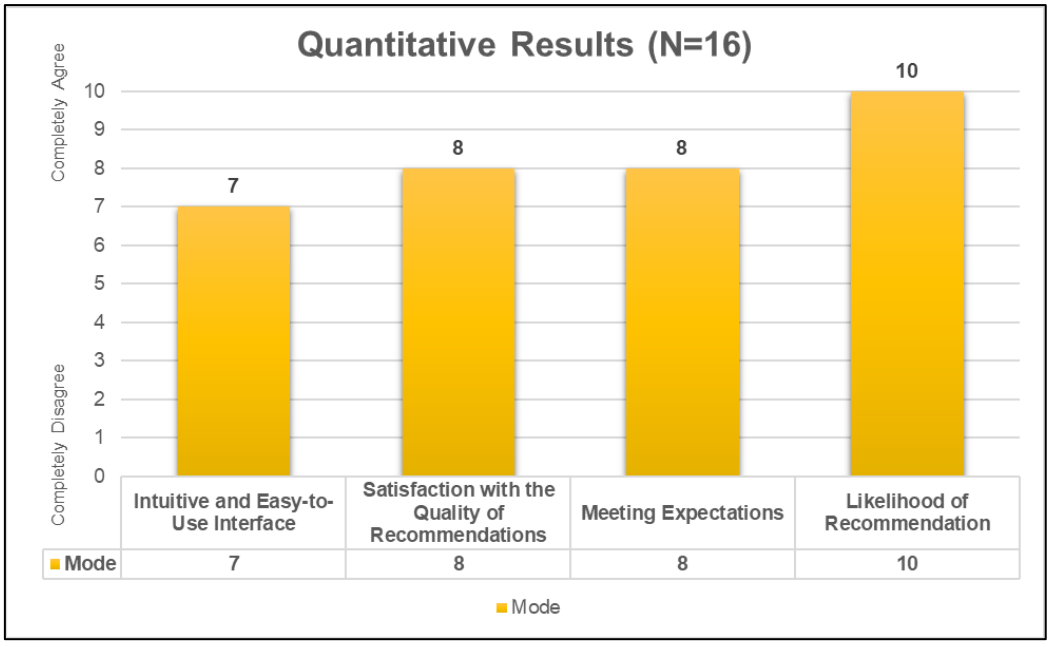



**Figure 18: Quantitative Results (Mode, N=16).**

The perception of the tool's interface and ease of use obtained a mode of 7 (87,5% of participants marked at least 7). This result indicates that the most frequent response from participants was one of moderate agreement that *ArchIoTect* is intuitive and easy to operate, a crucial factor for Perceived Ease of Use (PEOU) in the TAM model. While this score is positive, it also suggests that the user experience was not uniformly seamless for all participants, which corroborates the feedback that pointed to specific challenges in the filter system.

Regarding satisfaction with the quality of recommendations, the tool achieved a mode of 8 (93,8% of participants marked at least 7). This demonstrates a consensus among participants, as agreement was the most frequent response. This indicates that the content and architectural solutions provided by *ArchIoTect* are considered high-quality and reliable, a fundamental pillar for the tool's viability, as they validate its main value proposition.

Similarly, the tool scored a mode of 8 (87,5% of participants marked at least 7) on the metric of meeting expectations for assistance in architectural decisions. This result is an indicator of Perceived Usefulness (PU), demonstrating that the most common user experience was that *ArchIoTect* successfully fulfilled its promise of being an effective resource during the execution of a practical task. Participants not only considered it theoretically useful but also confirmed its value in a real-world scenario.

Finally, the metric with the most significant result was the likelihood of recommendation, which achieved a mode of 10 (93,8% of participants marked at least 7). This signifies that the single most frequent response was "*Completely agree*", a positive indicator of overall satisfaction and Behavioral Intention (BI). Such a definitive score suggests that participants would not only adopt the tool for their use but also can act as its advocates, recommending it to other professionals in the field.

In summary, the quantitative analysis paints a clear picture of a successful evaluation. The *ArchIoTect* tool is perceived by its target audience as useful and high-quality, generating a strong intention to use and recommend it. The data

provides a solid basis for affirming the tool's viability from the perspective of user acceptance.

### *5.2.5 Qualitative Results*

The qualitative data, collected from open-ended questions in the post-validation questionnaire, provide crucial context for the quantitative scores. Analysis of this feedback reveals several key themes relating to the strengths and weaknesses of the tool and the ideal workflow, as perceived by the 16 participants.

#### 5.2.5.1 A Complementary, Two-Tool System

The most significant conclusion from the qualitative analysis is that participants do not perceive the AI Assistant and Hierarchical Knowledge Base as competing features. Instead, there was a strong consensus in seeing them as two parts of a single, more powerful and complete workflow. When asked to choose the most effective mode, the group was almost evenly split, with six participants favoring the AI Assistant, four favoring the Knowledge Base, and a notable group of six explicitly stating that the ideal strategy involves using both in combination (Figure 17).

This sentiment was best articulated by one participant who described the ideal workflow as "*starting with the assistant for initial guidance, then using the Knowledge Base as supporting material.*" This highlights a clear two-step process: discovery followed by validation. Another user reinforced this by stating: "*AI served for initial research that was then further explored by navigating the knowledge base to explore alternatives and validate the AI's presentation.*"

Consequently, qualitative data indicate that users naturally assigned distinct roles to each component based on their strengths. The AI Assistant was valued as a tool for speed, discovery, and initial ideation, while the Knowledge Base was established as the essential resource for validation, accuracy, deep analysis, and trust.

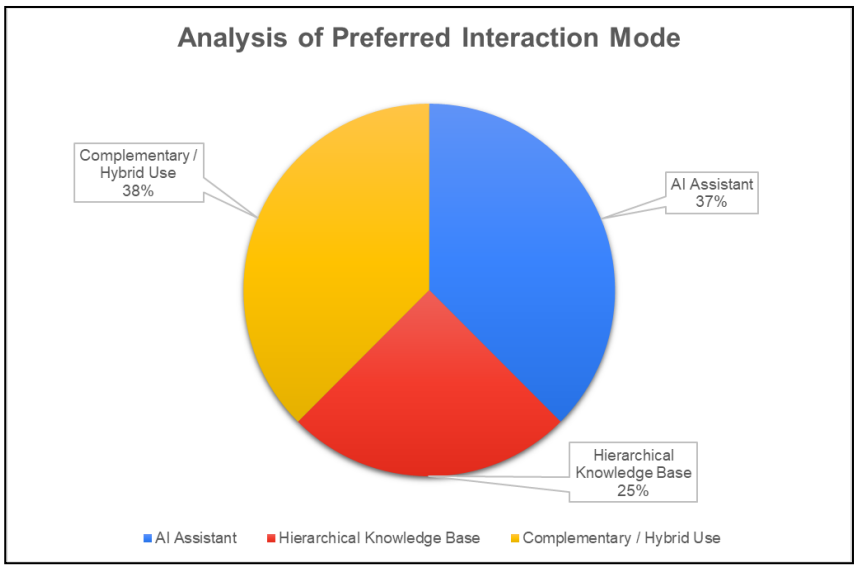



**Figure 19 - Analysis of Preferred Interaction Mode: AI vs. Knowledge Base vs. Hybrid.**

#### 5.2.5.2 Key strengths outlined for each component

Participants' comments provided clear justification for the roles they assigned to each module of the tool.

**The AI assistant:** valued for speed, simplicity, and synthesis.

AI was consistently praised for its efficiency and ease of use, especially for users less familiar with the domain. It was described as "more pleasant and easier to use" and capable of providing "*clearer and more straightforward solutions for someone with less experience in software architecture.*" Its main value lies in its ability to process complex queries and synthesize information. As one participant noted, "*with AI, the information came summarized and together, so comparison became much easier and more efficient*", a task described as "time-consuming" when done manually with filters.

**The Hierarchical Knowledge Base:** the source of trust, control, and accuracy.

In contrast, the Knowledge Base was consistently positioned as the definitive and reliable component. Its structured nature provided a sense of control and confidence that AI lacked. One user stated clearly: "*In the end, I*

*trusted the tree-based knowledge base more.*" This trust stems from its traceability and the user's ability to apply their own knowledge. For example, one participant valued the fact that the knowledge base allowed them to use their "*tacit knowledge not present in the AI assistant.*" In addition, it was praised for its accuracy and depth, with users noting that they could "*easily find an architecture model that met all requirements*" and that its descriptions explained how a system could achieve a specific quality, not just that it could.

#### 5.2.5.3 Weaknesses and opportunities for improvement

Qualitative feedback was key to identifying specific points that explain why quantitative scores, although high, were not perfect. These weaknesses represent clear opportunities for future development.

- **Significant usability challenges in the filter system:** This was by far the most criticized aspect of the tool. Feedback pointed to several key issues that detracted from the user experience:
    - **Lack of multiple selection:** A major source of frustration was the fact that "*it is not possible to search for more than one technology or requirement at a time.*"
    - **Filter reset:** Users found the workflow inefficient because "*when you redo a selection, the filter is completely cleared, forcing the user to fill in all the fields again.*"
- **Panel to show details layout:** The "Show details" panel was described as poorly positioned, "*covering part of the filter menu*" which forced a tedious cycle of hiding and showing the panel to adjust the filters.
- **Lack of Trust and Cohesion in the AI Assistant:** While valued for speed, the AI's credibility was undermined by several factors:
    - **Inconsistency with the Knowledge Base:** The most damaging issue was the AI providing answers that were

"*incoherent with the database*" which, as one user stated, "*made me insecure about the AI assistant tool*".

- **Lack of traceability:** Users reported "*difficulty finding the models suggested by the assistant in the hierarchical knowledge base*" disrupting the desired workflow of using AI for discovery and the knowledge base for validation.
- **Lack of refinement:** It was also noted that the AI occasionally "*switched between Portuguese and English*" further reducing its perceived professionalism and reliability.

- **User suggestions for future improvements:** Participants provided constructive suggestions, the most common being the need for closer integration between the two main components, so that they "*work more cooperatively, rather than appearing to be two competing features.*" Other important suggestions included expanding the knowledge base, adding a "*feature to include new knowledge*," and creating a "*summary area of my choices*" to help track the design process.

### *5.2.6 Evolving ArchIoTect Based on User Feedback*

The previous sections presented a comprehensive evaluation of the *ArchIoTect* tool by synthesizing quantitative data and qualitative feedback from target users. The insights gathered from this analysis provided a clear and actionable roadmap for refinement. This section details the subsequent evolution of the tool, a development phase driven directly by the key findings of the user evaluation.

Based on extensive user feedback, the tool's filter system underwent a major evolution. The original design was criticized for lacking a multi-select option and for an inefficient workflow where filters would reset after each selection. To resolve these issues, the system was enhanced to allow for multiple, simultaneous selections of technologies and requirements. Additionally, the filter state is now preserved during use, creating a more intuitive and efficient

experience by eliminating the need for users to repeatedly re-enter their criteria (Figure 20).

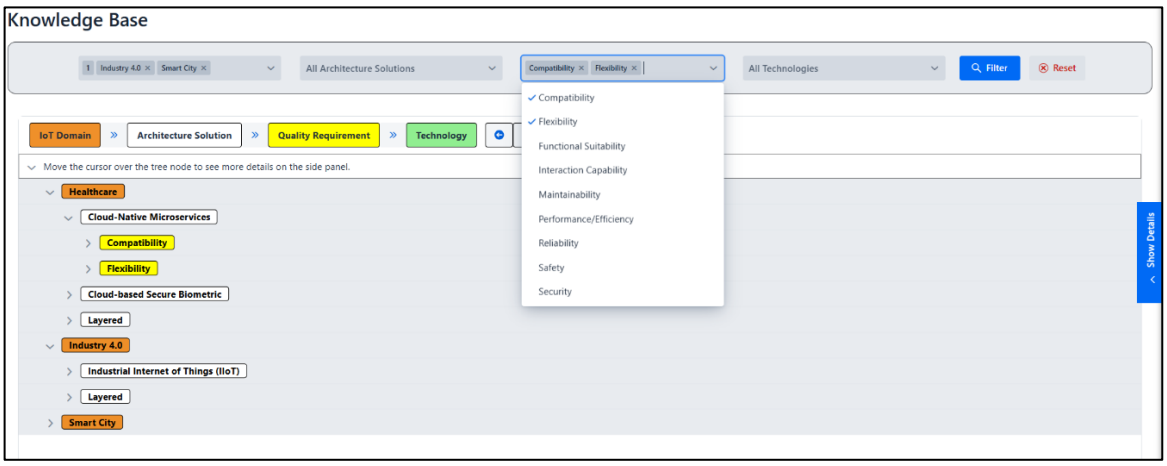



**Figure 20: Enhancements to the Filtering Feature.**

To address user feedback regarding the AI Assistant's lack of trust and traceability, key enhancements were implemented. Although all provided suggestions are followed by references, the primary solution was to integrate actionable "Filter Options" into the AI's answers (Figure 21). These pre-configured filters enable users to instantly access the AI's architectural suggestions within the main knowledge base, establishing a direct and reliable link that was previously missing. Furthermore, all AI responses were standardized to English to ensure professionalism and consistency, resolving the reported issues of incoherence and improving the overall reliability of the assistant.

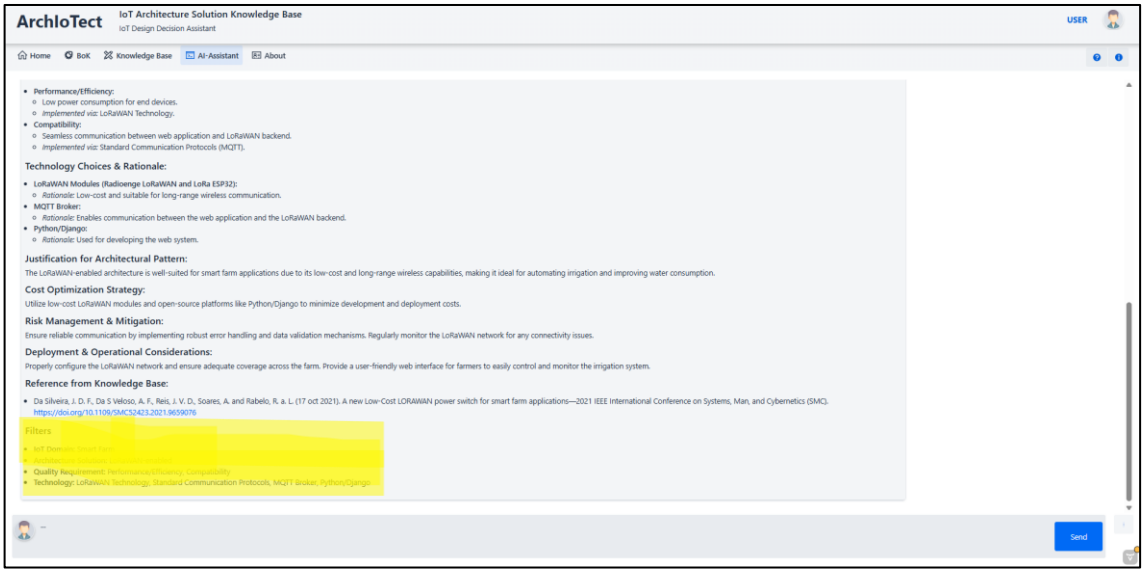



**Figure 21: Filter option for Knowledge Base use.**

To improve the overall user experience and simplify onboarding, a new guided tour feature was implemented (Figure 22). This tour provides users with step-by-step guidance on navigating the tool, exploring its features, and understanding its core functionalities through interactive tips and highlights. This addition is designed to reduce initial confusion and help users become proficient with *ArchIoTect* more quickly.

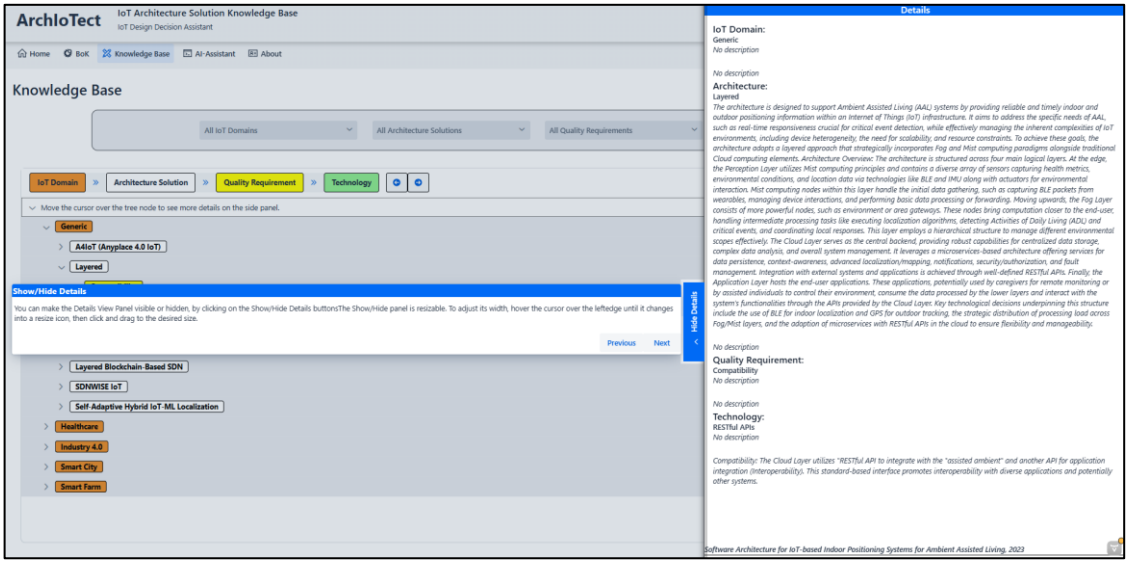



**Figure 22: An interactive guided tour.**

Finally, in response to direct user feedback gathered during the qualitative analysis, significant user experience enhancement was implemented.

Participants noted that the plain text presented in the "Details Tab" was difficult to read and lacked clear structure, as illustrated in Figure 22. To address this specific point of friction, the component where users enter the description text was upgraded to a rich text editor (Figure 23). This enhancement empowers users to structure their content with essential formatting tools such as headings, lists, and emphasis (bold, italics), significantly improving the clarity and comprehensibility of the text. The tangible improvement in the user interface is demonstrated in the before-and-after comparison in Figure 24.

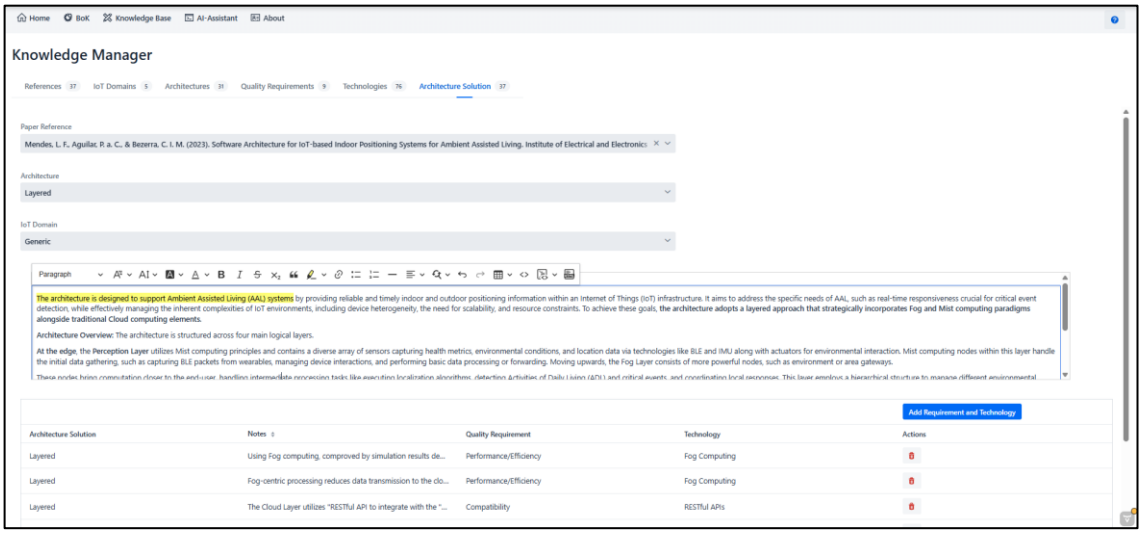



**Figure 23: The Rich Text field.**

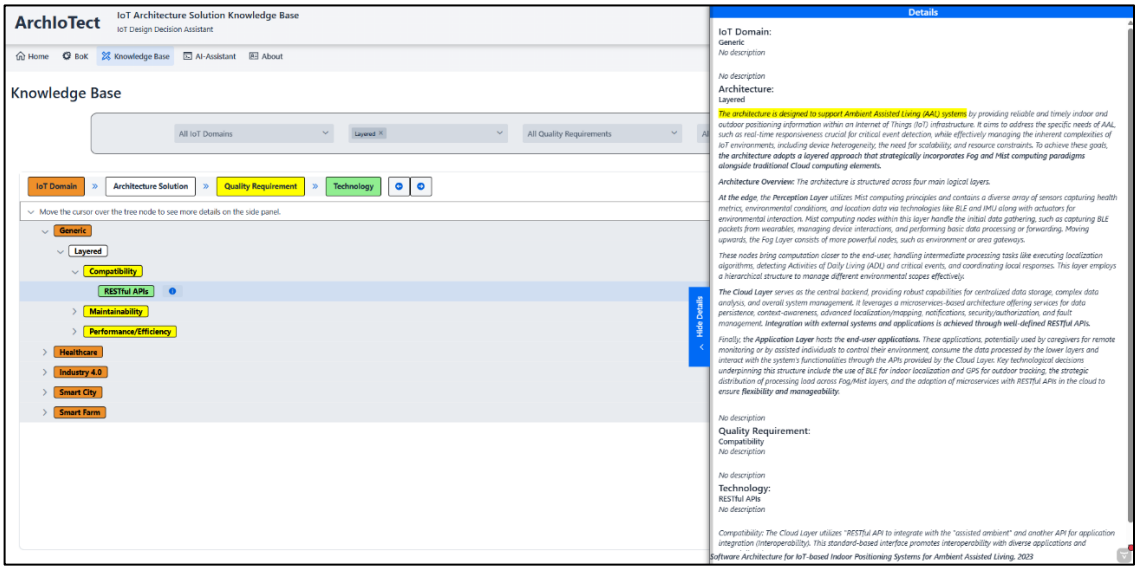



**Figure 24: The Description in the Details Tab After the Rich Text Editor Upgrade.**

An essential component of any empirical investigation in software engineering is a self-critical examination of factors that could compromise its findings. This chapter is dedicated to a transparent discussion of the potential threats to the validity of the *ArchIoTect* feasibility study and its inherent limitations. By acknowledging these factors, we do not diminish the study's outcomes; rather, we frame them appropriately, allowing for a more nuanced interpretation and providing a clear path for future research. The analysis is structured around four primary categories of validity: construct, internal, external, and conclusion.

#### 5.2.6.1 Threats to Construct Validity

Construct validity concerns the alignment between the study's measurements and its underlying theoretical concepts.

**Reliance on a Single Method:** The assessment of user perception depended entirely on a post-validation questionnaire. While the Technology Acceptance Model (TAM) is a well-established framework, the lack of complementary data collection techniques, such as direct user observation or in-depth follow-up interviews, may restrict the full understanding of how users genuinely interact with the tool.

**Evaluation Apprehension:** Given that the tool was developed within an academic context, there is a possibility that participants, themselves postgraduate students, might have felt inclined to provide socially desirable or overly positive responses to support the researchers' work. This threat was mitigated by ensuring participant anonymity and explicitly stating that critical and honest feedback was vital for the tool's improvement.

#### 5.2.6.2 Threats to Internal Validity

Internal validity addresses the confidence in the causal link between the intervention (using the tool) and the observed outcomes, ensuring that extraneous variables did not influence them.

**Heterogeneity of Participant Experience:** A significant threat emerges from the diverse professional backgrounds of the 16 participants. Despite all being IT professionals, their varying degrees of expertise in software architecture

and IoT could have heavily skewed their perceptions. An architect with decades of experience might view the tool's guidance as rudimentary, while a less experienced developer could find it highly insightful. Consequently, their ratings of usefulness and ease of use may reflect their prior knowledge as much as the tool's intrinsic qualities.

**The Learning Effect:** The study's design limited tool usage to a single session, focusing on a single scenario. Although an introductory video was provided to establish a baseline understanding, it is plausible that some reported usability challenges stem from an incomplete learning curve rather than fundamental design flaws. A single interaction may not be sufficient to achieve full proficiency.

#### 5.2.6.3 Threats to External Validity

External validity pertains to the extent to which the study's findings can be generalized to different people, settings, or conditions.

**Participant Representativeness:** The primary challenge to generalizing these findings is the specific profile of the participants. As postgraduate software engineering students, they may be inherently more open to novel academic tools and methodologies compared to industry practitioners at large, who are not engaged in a research environment. As such, the high level of acceptance observed may not be fully representative of the broader software industry.

**Task Representativeness:** The evaluation was based on three challenge scenarios (Appendix F). While designed to be realistic, these tasks do not encompass the full spectrum of complexity and scale found in real-world IoT projects. The tool's perceived effectiveness may vary significantly in projects of much larger or smaller scope, or in different IoT application domains.

**Tool Maturity:** The findings of this evaluation are intrinsically bound to the first version of *ArchIoTect*. The results and conclusions reflect the specific feature set and interface of this initial prototype. Subsequent versions with expanded capabilities or a refined user interface could produce markedly different evaluation outcomes.

#### 5.2.6.4 Threats to Conclusion Validity

Conclusion validity relates to the robustness of the statistical inferences drawn from the collected data.

**Limited Sample Size:** The results of this analysis are inherently constrained by the small sample size of 16 participants. While this number is adequate for a feasibility study aimed at identifying qualitative themes and quantitative trends, it is insufficient for making definitive statistical claims with a high confidence level. Therefore, the results should be interpreted as indicators of feasibility rather than as statistically conclusive proof.

## 5.3 Final Considerations about the Chapter

The evaluation of the first version of the *ArchIoTect* tool, detailed in this chapter, provides strong evidence supporting its viability. The study, conducted with 16 postgraduate software engineering students and structured using the Technology Acceptance Model (TAM), provided affirmative answers to the central research question regarding the tool's effectiveness and feasibility.

The quantitative results showed a strong positive consensus, with the most frequent response being an 8 for Perceived Usefulness and a 7 for Perceived Ease of Use. This culminated in an exceptional finding for the likelihood of recommendation, where the mode was 10, indicating "*Complete Agreement*" was the most common answer.

These figures indicate that users consider the tool valuable, intuitive, and worthy of adoption into their workflows.

The qualitative analysis revealed the ideal usage pattern: a complementary system where the AI Assistant is used for rapid discovery and the Hierarchical Knowledge Base is used for validation and in-depth analysis, ensuring confidence.

However, the study also identified areas for improvement that are critical. The primary weaknesses highlighted were the poor usability of the filter system

and the need to improve the traceability between the AI Assistant's responses and the core knowledge base.

Despite the study's limitations, such as the sample size and participant profile, the results support *ArchIoTect's* value proposition. The next steps should prioritize improving the usability of the filters and strengthening the integration and reliability of the AI Assistant to consolidate the tool as an indispensable resource for decision-making in IoT software architecture.

# 6 Final Considerations and Future Perspectives

*This chapter provides a comprehensive conclusion to the research. The study is structured first to present its final considerations and contributions. Next, it addresses the research limitations to contextualize the findings and, finally, describes the prospects for future work.*

## 6.1 Final Considerations

This dissertation confronted the escalating complexity of designing software architectures for Internet of Things software systems. Recognizing that the reuse of proven solutions is crucial for effective engineering, this research pursued a dual objective: first, to investigate the state-of-the-art in the field systematically, and second, to embody that knowledge in a practical, intelligent tool to support architectural decision-making.

The initial phase of this research, the systematic literature review, culminated in a structured Knowledge Base that revealed significant trends in current architectural practice. A prominent observation is the frequent adoption of cloud-based processing to satisfy performance and energy efficiency criteria. Similarly, the persistence of layered architectural patterns underscores their value in managing complexity and ensuring maintainability. Most notably, Security emerged as the dominant non-functional concern, with technologies such as blockchain, cryptography, and Software-Defined Networking (SDN) being commonly employed as mitigation strategies.

These findings, however, were not merely an academic exercise. They formed the empirical foundation for the primary artifact of this research: the "IoT Architectural Design Assistant." The subsequent evaluation of this tool was designed to assess its real-world utility in supporting architectural decision-making.

The subsequent evaluation of this tool was designed to assess its real-world utility in supporting architectural decision-making. To this end, a feasibility study was conducted with 16 postgraduate students of Software Engineering, using the Technology Acceptance Model (TAM) as its theoretical framework. The quantitative results demonstrated a highly successful reception, as presented in Table 6. This positive sentiment was underscored by an exceptional mode of 10 for the likelihood of recommendation, indicating a strong intention for future adoption.

**Table 6: Summary table, combining the TAM constructs with the questions and their calculated modes.**

| TAM Variable (Construct) | Questionnaire Item | Percentage of at least moderate agreement (7) | Mode |
|---|---|---|---|
| **Perceived Ease of Use (PEOU)** | The *ArchloTect* tool interface is intuitive and easy to use. | **87,5%** | 7 **-moderate agreement (7)**. |
| **Perceived Usefulness (PU)** | I am satisfied with the overall quality of the recommendations provided by the *ArchloTect* tool. | **93.8%** | 8 **-agreement (8)**. |
| **Perceived Usefulness (PU)** | The *ArchloTect* tool met my expectations for assisting with architectural decisions. | **87.5%** | 8- **agreement (8)**. |
| **Behavioral Intention (BI)** | The *ArchloTect* tool should be recommended to other professionals working with IoT software system architectures. | **93,8%** | 10 - **complete agreement (10)**. |

Qualitatively, the study's most significant finding was the emergence of a preferred hybrid workflow. Participants valued the AI Assistant for its speed and ability to generate ideas and compose answers, but consistently turned to the

Hierarchical Knowledge Base for its perceived reliability, traceability, and depth, using it to validate the AI's suggestions. While the evaluation confirmed the tool's core value, it also highlighted critical areas for enhancement, primarily the usability of the filter system and the need to improve the AI's consistency to build user trust. Ultimately, the evaluation provides robust evidence of the tool's feasibility and its potential to become an indispensable resource for software architects. Collectively, the insights from the Knowledge Base analysis and the empirical results from the tool's evaluation affirm the value of a knowledge-driven, AI-assisted approach to IoT architecture.

Collectively, the insights from the Knowledge Base analysis and the empirical results from the tool's evaluation affirm the value of a knowledge-driven, AI-assisted approach to IoT architecture. These combined outcomes form the basis for the specific contributions detailed in the following section, offering a robust solution that bridges the gap between theoretical knowledge and the practical challenges faced by system designers.

## 6.2 Contributions

This research makes several distinct contributions to the field of software engineering, specifically in the domain of Internet of Things (IoT) software system design.

First and foremost, this work presents a systematically curated and structured knowledge base of IoT architectural solutions. By conducting a rigorous systematic literature review, we have addressed the significant challenge of information fragmentation, where critical design knowledge is often scattered across numerous academic papers and technical documents. This consolidated repository provides a verified and centralized source of information on architectural solutions, quality requirements, and enabling technologies, which serves as a valuable resource.

Second, we deliver a tangible software artifact, the "IoT Architectural Design Assistant," an interactive web application that makes this knowledge base accessible and actionable. The tool's dual-modality interface is a key contribution,

catering to different user workflows. It allows for both a structured, hierarchical exploration of the knowledge base for users who prefer methodical discovery and a dynamic, conversational query system for those seeking immediate, targeted answers.

Finally, the integration of a generative AI module for intelligent guidance represents a novel application of LLMs to the architectural design process. Unlike simple search-and-retrieval systems, our AI assistant is capable of synthesizing information from multiple sources within the knowledge base to provide coherent, context-aware responses to complex design queries. This moves beyond mere information access to offer a form of generative design support.

Collectively, these contributions form a bridge between theoretical academic research and the practical challenges faced by software architects and engineers, providing a robust tool to enhance decision-making and reduce the complexity of designing modern IoT software systems.

## 6.3 Research Limitations

To provide a transparent account of this work, several limitations that frame the scope and applicability of our findings must be acknowledged. These constraints also serve to identify clear pathways for future inquiry.

Initially, the knowledge base represents a snapshot in time, with its underlying literature search having a cutoff date of late 2024. Given the dynamic nature of the IoT field, where standards and technologies evolve at a rapid pace, the dataset of the tool may not capture the most recent innovations that emerged beyond this timeframe.

Furthermore, the search methodology, while systematic, has inherent constraints. The specificity of our search queries and the scope of the indexed databases mean that some relevant literature may have been unintentionally omitted. As a result, the dataset should be viewed as a representative, yet not exhaustive, collection of architectural knowledge rather than a complete census of all published work.

Consequently, caution should be exercised when extrapolating the findings. The architectural trends and solutions identified by the assistant are valid within the scope of our dataset but may not be broadly generalizable to all specialized IoT sub-domains or emerging application areas that were not well-represented in the selected literature.

Finally, the evaluation of the tool was conducted within a simulated academic context, not in an operational industrial environment. Therefore, its "*in-the-wild*" performance, including its scalability under enterprise-level loads, its ease of integration with existing development pipelines, and its practical adoption by engineering teams, remains an open question. Validating the assistant's real-world efficacy is a critical next step.

These limitations do not diminish the study's contributions but rather define its boundaries and highlight promising directions for subsequent research.

## 6.4 Future Perspectives

Future perspectives for the lifecycle and evolution of ArchIoTect include:

- **Continuous expansion and curation of the knowledge base**, through the systematic incorporation of new research, emerging architectural patterns, and the investigation of a model for community contributions, in order to ensure its ongoing relevance and comprehensiveness.
- **Enhancement of the User Interface and User Experience (UI/UX)**, based on conducting new usability studies to optimize the navigation and visualization of complex architectural information.
- **Evolution of the Retrieval-Augmented Generation (RAG) architecture**, through investigating more advanced retrieval strategies (e.g., hybrid or graph-based search) and refining the model's capacity to synthesize complex and comparative architectural analyses.

- **Proposing and evaluating new AI functionalities**, such as a proactive design recommendation system and a mechanism for real-time validation of architectural decisions, using the knowledge base as a reference.
- **Investigating the applicability of the *ArchIoTect* paradigm in other phases of the software architecture lifecycle**, such as architectural evaluation, technical debt analysis, and system evolution planning.
- **Finally, we posit that the foundational concepts of *ArchIoTect* can be extended beyond system design.** Subsequent research will explore the adaptation of this AI-assisted, knowledge-driven approach to support other critical architectural activities, including formal architectural evaluation and strategic system evolution.

# References


AL-FUQAHA, A., GUIZANI, M., MOHAMMADI, M., et al., 2015, "Internet of Things: A survey on enabling technologies, protocols, and applications", *IEEE Communications Surveys & Tutorials*, v. 17, n. 4, pp. 2347–2376. Available at: https://doi.org/10.1109/comst.2015.2444095.

ALRESHIDI, A., AHMAD, A., 2019, "Architecting software for the Internet of Things-based systems", *Future Internet*, v. 11, n. 7, p. 153.

ANTONIADI, A. M., DU, Y., GUENDOUZ, Y., et al., 2021, "Current challenges and future opportunities for XAI in machine learning-based clinical decision support systems: a systematic review", *Applied Sciences*, v. 11, n. 11, p. 5088.

ATZORI, L., IERA, A., MORABITO, G., 2010, "The Internet of Things: A survey", *Computer Networks*, v. 54, n. 15, pp. 2787–2805. Available at: https://doi.org/10.1016/j.comnet.2010.05.010.

BANDYOPADHYAY, D., SEN, J., 2011, "Internet of Things: Applications and Challenges in Technology and standardization", *Wireless Personal Communications*, v. 58, n. 1, pp. 49–69. Available at: https://doi.org/10.1007/s11277-011-0288-5.

BARCELOS, R. F., TRAVASSOS, G. H., 2006, "ARQCheck: uma abordagem para inspeção de documentos arquiteturais baseada em checklist". In: *Anais do Simpósio Brasileiro de Qualidade de Software (SBQS)*, pp. 174–188. Available at: https://doi.org/10.5753/sbqs.2006.15608.

BASS, L., CLEMENTS, P., KAZMAN, R., 2021, *Software Architecture in Practice*. 4th ed. Addison-Wesley Professional.

BONOMI, F., MILITO, R., ZHU, J., et al., 2012, "Fog computing and its role in the internet of things". In: *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*. Available at: https://dl.acm.org/doi/10.1145/2342509.2342513.

BOTTA, A., DE DONATO, W., PERSICO, V., et al., 2015, "Integration of Cloud computing and Internet of Things: A survey", *Future Generation Computer Systems*, v. 56, pp. 684–700. Available at: https://doi.org/10.1016/j.future.2015.09.021.

BROWN, T. B., MANN, B., RYDER, N., et al., 2020, "Language Models are Few-Shot Learners". In: *Neural Information Processing Systems*, v. 33, pp. 1877–1901. Available at: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

CHIANG, M., ZHANG, T., 2016, "Fog and IoT: An Overview of Research Opportunities", *IEEE Internet of Things Journal*, v. 3, n. 6, pp. 854–864. Available at: https://doi.org/10.1109/jiot.2016.2584538.

CHRISTIAN, C., 2025, *IoT 2024 in review: The 10 most relevant IoT developments of the year*. IoT Analytics, Jan. 15. Available at: https://iot-analytics.com/iot-2024-review/.

CHROMA TEAM, 2025, *Chroma - The AI-native open-source embedding database*. Available at: https://www.trychroma.com/. Accessed on: 02 May 2025.

DAVIS, F. D., 1989, "Perceived usefulness, perceived ease of use, and user acceptance of information technology", *MIS Quarterly*, v. 13, n. 3, p. 319. Available at: https://doi.org/10.2307/249008.

ECLIPSE FOUNDATION, 2023, *IoT & Edge Developer Survey 2023 Report*. Available at: https://outreach.eclipse.foundation/iot-edge-developer-survey-2023.

GAO, Y., XIONG, Y., GAO, X., et al., 2023, "Retrieval-Augmented Generation for Large Language Models: A survey". *arXiv (Cornell University)*. Available at: https://doi.org/10.48550/arxiv.2312.10997.

GLOBAL, B., 2021, *Entenda o que é o conceito e como funciona!*. Bosch No Brasil. Available at: https://www.bosch.com.br/noticias-e-historias/aiot/. Accessed on: 10 June 2025.

GUBBI, J., BUYYA, R., MARUSIC, S., et al., 2013, "Internet of Things (IoT): A vision, architectural elements, and future directions", *Future Generation Computer Systems*, v. 29, n. 7, pp. 1645–1660. Available at: https://doi.org/10.1016/j.future.2013.01.010.

H2 DATABASE ENGINE, 2023, *H2 Database Engine (Version 2.2.220)*. Available at: https://www.h2database.com/html/download.html.

ISO/IEC-25010, 2023, *ISO/IEC 25010—Systems and Software Quality Requirements and Evaluation (SQuaRE)—System and software quality models*. ISO.org. Available at: https://www.iso.org/obp/ui/en/#iso:std:iso-iec:25010:ed-2:v1:en.

JIANG, Z., XU, F., GAO, L., et al., 2023, "Active Retrieval Augmented generation". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Available at: https://doi.org/10.18653/v1/2023.emnlp-main.495.

KHAN, R., KHAN, S. U., ZAHEER, R., et al., 2012, "Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges". In: *Proceedings of the 10th International Conference on Frontiers of Information Technology*. Available at: https://doi.org/10.1109/fit.2012.53.

KUHRMANN, M., MÉNDEZ, D., DANEVA, M., 2017, "On the pragmatic design of literature studies in software engineering: an experience-based guideline", *ESE*, v. 22, n. 6, pp. 2852–2891.

LANGCHAIN TEAM, 2025, *LangChain Documentation*. Available at: https://python.langchain.com/.

LEWIS, P. S. H., PEREZ, E., PIKTUS, A., et al., 2020, "Retrieval-Augmented Generation for Knowledge-Intensive NLP tasks". In: *Neural Information Processing Systems*, v. 33, pp. 9459–9474. Available at: https://proceedings.neurips.cc/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.

MIRANI, A. A., VELASCO-HERNANDEZ, G., AWASTHI, A., et al., 2022, "Key Challenges and Emerging Technologies in Industrial IoT architectures: A review", *Sensors*, v. 22, n. 15, p. 5836.

MOTTA, R. C., SILVA, V., TRAVASSOS, G. H., 2019, "Towards a more in-depth understanding of the IoT Paradigm and its challenges", *JSERD*, v. 7, p. 3. Available at: http://dx.doi.org/10.5753/jserd.2019.14.

NAKAGAWA, E., ANTONIO, P. (eds), 2023, *Reference architectures for critical domains: Industrial Uses and Impacts*. 1st ed. Springer Cham.

OPENAI, 2023, "GPT-4 Technical Report", *arXiv preprint arXiv:2303.08774*. Available at: https://doi.org/10.48550/arXiv.2303.08774.

PALLETS PROJECTS, 2025, *Flask: A web framework for Python*. Available at: https://flask.palletsprojects.com/. Accessed on: 13 May 2025.

RAZZAQ, A., 2020, "A Systematic Review of software architectures for IoT systems and future direction to adopting a microservices architecture", *SN Computer Science*, v. 1, n. 6.

SATYANARAYANAN, M., 2017, "The emergence of edge computing", *Computer*, v. 50, n. 1, pp. 30–39. Available at: https://doi.org/10.1109/mc.2017.9.

SETHI, P., SARANGI, S. R., 2017, "Internet of Things: architectures, protocols, and applications", *Journal of Electrical and Computer Engineering*, 2017, pp. 1–25. Available at: https://doi.org/10.1155/2017/9324035.

SHI, W., CAO, J., ZHANG, Q., et al., 2016, "Edge computing: vision and challenges", *IEEE Internet of Things Journal*, v. 3, n. 5, pp. 637–646. Available at: https://10.1109/jiot.2016.2579198.

SOORI, M., et al., 2024, "AI-Based Decision Support Systems in Industry 4.0, A Review", *Journal of Economy and Technology*. Available at: https://doi.org/10.1016/j.ject.2024.08.005.

STOJMENOVIC, I., WEN, S., 2014, "The Fog Computing Paradigm: Scenarios and security issues", *Annals of Computer Science and Information Systems*, v. 2, pp. 1–8. Available at: https://doi.org/10.15439/2014f503.

TEAM, G., ANIL, R., BORGEAUD, S., et al., 2023, "Gemini: a family of highly capable multimodal models", *arXiv (Cornell University)*. Available at: https://doi.org/10.48550/arxiv.2312.11805.

TOUVRON, H., MARTIN, L., STONE, K., et al., 2023, "Llama 2: Open foundation and Fine-Tuned chat models", *arXiv (Cornell University)*. Available at: https://doi.org/10.48550/arxiv.2307.09288.

VAADIN LTD., 2024, *Vaadin Platform Documentation*. Available at: https://vaadin.com/docs/latest/.

WEYNS, D., 2021, *An Introduction to Software Architecture*. 2nd ed. MIT Press.

WHITE, J., FU, Q., HAYS, S., et al., 2023, "A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT", *arXiv preprint arXiv:2302.11382*.

WÖHLIN, C., 2014, "Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering". In: *Proceedings of EASE 14*, pp. 1–10.

WOODS, E., 2018, *Software Systems Architecture*. 2nd ed. Addison-Wesley.

WU, N. M., LU, N. T.-J., LING, N. F.-Y., et al., 2010, "Research on the architecture of Internet of Things", *Advanced Materials Research*. Available at: https://doi.org/10.1109/icacte.2010.5579493.

# Appendix A – Extraction Data from Literature Review (Knowledge Base)

| KBRef-01 | |
|---|---|
| **IoT Domain** | Smart City |
| **Architecture** | Layered Blockchain and AI-enabled |
| It is a multi-layered security approach. We combine blockchain and AI to provide strong data integrity and smart threat detection. IoT device authentication protocols are used to verify the identity of devices, and data encryption protects sensitive information. Privacy techniques, like anonymization and data minimization, are used to comply with data protection rules. Also, real-time threat detection and response mechanisms are included to proactively address security threats.<br><br>It proposes a seven-layer blockchain architecture designed for IoT environments:<br><br>Physical Layer: Collecting data from the environment using sensors and devices.<br>Data Layer: Securely storing and managing data using a distributed ledger, Merkle trees, and asymmetric encryption.<br>Network Layer: Facilitating communication between nodes through peer-to-peer networks.<br>Consensus Layer: Ensuring data integrity and agreement through various consensus mechanisms.<br>Incentive Layer: Rewarding nodes for validating blocks, encouraging network participation.<br>Smart Contract Layer: Automating tasks and interactions with smart contracts.<br>Application Layer: Enabling user interaction and application development through APIs and user interfaces.<br>To address the limitations of traditional AI and blockchain use in IoT, we propose a distributed architecture that includes edge/fog/cloud computing. This framework has physical, communication, blockchain, and application layers:<br><br>Data Processing: Data is collected from IoT devices, processed at gateways and fog devices, and then verified and stored in the blockchain.<br>AI-Enhanced Cloud: Cloud-based AI algorithms analyze and process data, enabling intelligent decision-making.<br>Security and Privacy: Encrypted data storage and smart contract-based authentication ensure data security and user privacy.<br>Hybrid Design: A hybrid approach, distributing tasks between IoT, cloud, and blockchain, optimizes computational efficiency.<br>Smart Contracts for Authentication: Smart contracts facilitate user authentication, data format verification, and reward point management.<br><br>Authentication Keys: Users are provided with authentication keys to access personalized data.<br>Secure Communication Channels: Secure communication channels are established based on system security parameters.<br>In conclusion, combining blockchain, AI, and cloud computing is a good solution for developing secure, efficient, and sustainable smart city IoT applications. The proposed distributed architecture addresses the limitations of these technologies, offering a scalable and robust framework for data-intensive environments. Future research will focus on testing this architecture in real-world smart city deployments, exploring the optimization of consensus mechanisms, and developing adaptive AI algorithms for dynamic IoT environments. This paper provides a basic framework for the next generation of smart city infrastructure, emphasizing security (Confidentiality). | |
| **Quality Requirements** | |
| **Security** | Confidentiality is archived using Blockchain is to enhance confidentiality by providing immutable and auditable records of data transactions. |
| **KBRef-02** | |
| **IoT Domain** | Healthcare |

| | |
|---|---|
| **Architecture** | Layered |
| It is a multi-layered framework for monitoring and managing diabetes and abnormal blood pressure in patients. The architecture integrates data from diverse sources, including wearable sensors, medical records, and social media platforms, to provide a holistic view of patient health. This data is collected, stored on a scalable cloud-based system utilizing technologies like Amazon S3, Hadoop, and HBase, and then analyzed using advanced machine learning techniques, specifically Bi-LSTM and ontology-based approaches, to classify health risks and predict potential complications. The resulting insights are presented to physicians to aid in treatment decisions and deliver personalized healthcare recommendations to patients. The framework emphasizes leveraging big data analytics to improve patient outcomes and proactively manage chronic conditions. While technologically sophisticated, the practical implementation challenges (e.g., social media data privacy, real-world performance) and quantifiable benefits require further investigation. | |
| **Quality Requirements** | |
| **Flexibility** | Utilizes Amazon S3 for data storage, which is inherently scalable (Flexibility) due to its distributed nature and ability to handle large volumes of data. S3's bucket system allows for efficient organization and retrieval of patient data. |
| **Security** | Cloud-based Infrastructure (Amazon S3), which offers robust security features including access controls, data encryption, and compliance certifications (HIPAA eligibility if configured properly). This provides a foundation for secure data storage (Confidentiality and Integrity). |
| **Performance/Efficiency** | A big data analytics engine is proposed for the analysis of real-world big data. It is used to accurately handle healthcare data containing inconsistencies and that have missing values, noise, different formats, a large size, and high dimensionality. In addition, it is utilized to improve the quality of data processing and to save time (Time Behaviour and Resource Utilization). |
| **KBRef-03** | |
| **IoT Domain** | Healthcare |
| **Architecture** | Fog Computing-Based Three-Tier |

| | |
|---|---|
| It is a three-tier architecture for real-time health monitoring using fog computing.<br><br>IoT Layer (Tier 1): Comprises sensors attached to the patient (monitoring body temperature, heart rate, pulse rate) which transmit data via the patient's smartphone. This layer acts as the data source.<br><br>Fog Layer (Tier 2): Consists of fog nodes co-located with Base Stations (BSs). This intermediate layer sits close to the end-users (IoT devices) to perform initial data processing and analysis. It determines the patient's health status and sends results back to the smartphone for immediate feedback. It also forwards results to the cloud. This layer is crucial for achieving low latency.<br><br>Cloud Layer (Tier 3): The top layer, primarily providing large-scale, permanent storage via data centers. It connects to the fog layer through a proxy server and stores the processed health status results for long-term record-keeping and potential future retrieval.<br><br>The system utilizes an Application Model with three distinct modules:<br><br>Client Module: Resides on the patient's smartphone, providing the user interface and collecting sensor data.<br><br>Processing Module: Located on the fog nodes, responsible for analyzing the sensor data and generating health status results.<br><br>Storage Module: Integrated into the cloud server for permanent storage of the processed results.<br><br>A key feature is the proposed Load Balancing Scheme (LBS) operating at the Fog Layer. It aims to minimize overall system latency (communication and computing) by dynamically distributing the workload (IoT device connections and processing tasks) among different BSs/fog nodes, especially in areas with overlapping BS coverage. | |
| **Quality Requirements** | |
| **Performance/Efficiency** | The Load Balancing Scheme (LBS) optimizes performance by managing both communication and computing loads to minimize delays and network usage (Resource Utilization). |
| **Maintainability** | The three-tier structure and the modular application design (Client, Processing, Storage) promote separation of concerns. This modularity allows individual components or layers to be modified or updated with potentially reduced impact on the rest of the system. |
| **KBRef-04** | |
| **IoT Domain** | Healthcare |
| **Architecture** | Fog Based Efficient |

It is a multi-tiered architecture designed to address latency, network bandwidth, and security challenges inherent in purely cloud-based IoT healthcare systems. The core concept involves leveraging Fog Computing nodes positioned closer to the data sources (IoT devices and Body Sensor Networks - BSNs) to perform initial data processing, analysis, and management, thereby reducing reliance on distant cloud servers for time-sensitive operations.

1.Tiers:

1.1.Edge Tier: Consists of IoT devices and BSNs collecting patient data.
1.2.Fog Tier: Geographically distributed Fog Nodes acting as intermediaries. These nodes host Virtual Machines (VMs) dedicated to specific tasks (BSN data processing, health record management, clinical document processing, user identity management). They perform significant computation and short-term storage. A Proxy Server may mediate between Fog and Cloud.
1.3.Cloud Tier: Centralized Cloud Server for long-term data storage and potentially more complex, less time-sensitive analysis.

2.Key Pattern: Edge/Fog Computing pattern combined with Service/Module partitioning using Virtual Machines within the Fog nodes.

3.Data Flow: Data flows from Edge devices -> Gateways -> Fog Nodes (for processing/analysis/short-term storage/authentication) -> Proxy Server -> Cloud Server (for long-term storage). Users (Patients, Medical Staff) interact primarily with the Fog Nodes to access/upload data and utilize system features.

Data flows from sensors/devices to the Fog Nodes for immediate processing and potential action, with less time-sensitive data or aggregated results forwarded to the cloud. Users interact mainly through the Fog layer for access and data management.

| **Quality Requirements** | |
|---|---|
| **Performance/Efficiency** | A primary goal is latency and network load (Bandwidth) reduction by processing data closer to the source (Fog) (Time Behaviour and Resource Utilization). |
| **Security** | Achieved via authentication, role-based access, and processing sensitive data locally on Fog nodes (Confidentiality). |

| **KBRef-05** | |
|---|---|
| **IoT Domain** | Healthcare |
| **Architecture** | Cloud-Native Microservices |

The proposed system employs a Microservices architecture that runs on Cloud platform. It decomposes the application into small, independent services, each dedicated to a specific business capability and designed for loose coupling. These services are deployed onto a major Cloud Platform (such as AWS, Azure, or GCP), leveraging managed cloud services for infrastructure needs.

For deployment and runtime management, services are packaged as Docker containers and orchestrated using Kubernetes, enabling automated scaling, deployment, and management. Communication between services and clients utilizes both synchronous methods (like RESTful APIs or gRPC, exposed via an API Gateway) for direct interactions, and asynchronous, event-driven communication via a Message Broker to enhance decoupling, resilience, and handle background processing.

Data management follows a Polyglot Persistence strategy, where each microservice owns its data and selects the most suitable database technology for its specific requirements. A central API Gateway serves as the single entry point for external clients, managing routing, security aspects like authentication/authorization, and rate limiting. Finally, the architecture incorporates robust Observability through centralized logging, metrics collection, and distributed tracing, and relies on fully automated CI/CD pipelines for streamlined and reliable deployments of each microservice.

| **Quality Requirements** |
|---|

| | |
|---|---|
| **Flexibility** | Containerization with Docker abstracts the application and its dependencies from the underlying OS. Kubernetes provides a consistent orchestration layer across different environments, reducing vendor lock-in at the orchestration level (Adaptability). |
| **Compatibility** | Use of standard protocols like HTTP/REST, gRPC, and AMQP/Kafka protocol for messaging ensures interoperability between services and potentially with external systems. API Gateway provides a stable, documented external interface. |
| **Maintainability** | The microservices architecture inherently enforces modularity by breaking the system into small, focused services with well-defined APIs. Each service has its own codebase and deployment pipeline. |
| **KBRef-06** | |
| **IoT Domain** | Smart City |
| **Architecture** | Unicorn ChargeUp |
| The Unicorn ChargeUp Architecture is a cloud-native, microservices-based system designed for the e-mobility domain. It comprises two main user-facing applications (ChargeUp ESP for drivers and ChargeUp CPO for operators) built upon a set of core microservices (Main, CPR, Reporting, Communication Log, Gateway, Customer & Contract Management for CPO; Main, Portal for ESP). This application layer leverages the foundational Unicorn Architecture (https://www.unicorn.com), which provides frameworks for GUI (uuHi), IoT integration (uuTi), application server logic (uuAppServer), and cloud deployment/management (uuCloud), along with common infrastructure services (uuOidc, uuMessageBroker, uuAsyncJob, uuPaymentGateway). The architecture emphasizes scalability, reliability, security, and interoperability through standardized protocols and a modular design deployed on cloud infrastructure (initially Azure, designed for portability). | |
| **Quality Requirements** | |
| **Reliability** | In summary, the Unicorn Cloud Framework (uuCloud) framework achieves high availability for the Unicorn ChargeUp solution by orchestrating a runtime environment built on isolation (containing failures), redundancy (NodeSets), automatic failover (redirecting traffic from failed nodes), elastic scalability (handling load and providing failover capacity), statelessness (enabling rapid recovery), and automation (reducing errors). These elements work together synergistically to meet the demanding 99.97% availability target. |
| **Performance/Efficiency** | In essence, the Unicorn Cloud Framework (uuCloud) framework provides the orchestration, automation, and management layer that enables the Unicorn ChargeUp architecture to leverage cloud-native principles (microservices, containers, elasticity) effectively. This directly translates into achieving high performance efficiency through robust scalability, optimized resource utilization, responsive time-behaviour under load, and the ability to meet defined requirements. |
| **Flexibility** | Stateless Application Servers (uuAppServer): Enabling Horizontal Scaling<br><br>The uuAppServer instances are designed to be stateless regarding user sessions. State is managed externally (e.g., JWT tokens, shared session store).<br><br>Flexibility Achieved:<br>Deployment Simplicity: Any server instance can handle any request, simplifying deployment and load balancing configurations. You don't need "sticky sessions."<br>Resilience: If one instance fails, requests can be seamlessly routed to another healthy instance without losing user session data (assuming external state management).<br>Scalability Link: This is the primary enabler for horizontal scalability (scaling out). Because instances are interchangeable, you can easily add or remove them behind a load balancer to match demand. This flexibility in adding/removing instances is scalability. |

| | |
|---|---|
| **Security** | Unicorn Cloud Framework (uuCloud) framework provides strong, explicit security foundations through its built-in components and patterns, particularly for: Authentication (uuIdM), Authorization (uuAA),and Input Validation (DTO Schemas).<br><br>1.Authentication (uuIdM): Verifies who the user is. It uses the dedicated uuIdentityManagement component and JWTs (JSON Web Tokens). The framework automatically validates these tokens on incoming requests to confirm user identity.<br><br>2.Authorization (uuAA): Determines what the user is allowed to do. It relies on Profiles (grouping permissions) defined by developers. Authorization checks must be explicitly implemented within the application's command/query logic, using framework patterns to verify user permissions against their assigned profiles.<br><br>3.Input Validation (DTO Schemas): Ensures data integrity and prevents malformed input. The framework mandates Data Transfer Objects (DTOs) with defined schemas for all inputs. It automatically validates incoming data against these schemas before executing application logic, rejecting invalid requests.<br><br>All three are explicitly designed features of the uuCloud framework, providing foundational security controls. |
| **KBRef-07** | |
| **IoT Domain** | Generic |
| **Architecture** | SDNWISE IoT |

The SDNWISE IoT architecture is structured into two primary components: the Control Plane and the Data Plane. Its main aim is to integrate Software Defined Networking (SDN) principles into the Internet of Things (IoT) to improve management, security, and flexibility.

1. Data Plane (Infrastructure Layer): This layer handles the actual forwarding of data packets.

Sensor Nodes (IoT Nodes): These are low-powered, 6LowPAN-based devices deployed in the environment. They are easily compromised and can be exploited by attackers to generate malicious traffic for DDoS attacks.

Sensor OpenFlow Switch (SOFS): A customized OpenFlow-based switch that forwards IoT traffic. It is less intelligent, acting primarily as a forwarding device. However, in this architecture, it is programmed to:

Report traffic load to the controller.

Trigger alerts to the attack receiver component in the SDNWISE controller when the traffic load reaches a predefined threshold.

Reset the threshold counter upon receiving a message from the attack receiver component (indicating the reported traffic was not malicious).

2. Control Plane: This layer provides centralized control and management of the network.

SDNWISE Controller: This is the "brain" of the SDN-based IoT network. Its core functions include:

Defining network policies.

Managing the control of the SDN-based IoT network.

Exposing APIs that enable developers to create applications for the IoT network.

IoT Controller: Acts as a mediator or translator between the SDNWISE controller and the SD-IoT network.

Receives traffic from the SDNWISE controller and converts it into a format understandable by the SD-IoT network.

Performs the reverse conversion (from SD-IoT format to SDNWISE controller format).

Crucial for enabling heterogeneous network support in the future (connecting multiple types of IoT networks to SDNWISE).

Attack Detection and Mitigation:

Counter-based DDoS Attack Detection (C-DAD) Application:

Monitors and analyzes IoT traffic based on counter values to detect DDoS attacks, anomalies, and threats.

Comprises different algorithms that leverage counter variables for detection.

Attack Mitigation Module:

Receives reports of malicious traffic from the C-DAD module.

Performs countermeasure actions on the identified malicious flows.

Sub-modules coordinate to mitigate attacks using SDN security features.

Malicious Flow Entry: Adds entries for malicious flows in the network to block them.

Malicious Node Removal: Uses the SDNWISE controller (via network graphic APIs) to remove malicious nodes (those generating the malicious traffic) from the SD-IoT network.

In summary: The SDNWISE architecture strives to leverage SDN principles to create a more manageable, secure, and flexible IoT network. Key aspects include the separation of control and data planes, centralized control via the SDNWISE controller, specialized IoT controllers for translation, and dedicated modules for DDoS attack detection and mitigation.

| **Quality Requirements** | |
|---|---|
| **Security** | This module specifically targets DDoS attacks, a common threat to IoT networks. It monitors traffic patterns and uses counter-based algorithms to detect anomalous behavior that could indicate an attack. |
| **KBRef-08** | |
| **IoT Domain** | Smart City |
| **Architecture** | Blockchain-Based Zero Trust on the Edge |

The proposed "Blockchain-based Zero Trust on the Edge" architecture is conceived as a distributed system meticulously designed to enforce fine-grained, continuously verified access control for Internet of Things (IoT) devices and users within environments like smart cities. It uniquely leverages blockchain technology to provide an immutable ledger for logging requests and verifying trust. The system is structured around three primary groups of components: Zero Trust Architecture (ZTA) components, blockchain components, and the IoT/user components interacting with the system. Adopting a hybrid architectural style, it blends Microservices, Component-Based design, and specific ZTA patterns with Blockchain integration, with the implementation explicitly following a "microservice manner".

At its core are the ZTA components responsible for policy enforcement and validation. The Policy Enforcement Point (PEP) acts as the crucial gateway, receiving incoming requests from client-side components – an Angular-based Analyser for administrators and a Python-based Client for users and devices. The PEP forwards these requests to the Policy Administrator (PA) for validation and, upon approval, interacts with the relevant Persistence Managers (PMs) to grant access to resources. The PA orchestrates the complex validation process by sending requests to multiple Policy Engines (PEs). It utilizes a consensus algorithm (specifically, a majority

vote) based on the responses from the PEs to determine the final decision. The PA also generates the necessary access tokens for PMs and triggers the immutable logging of requests and decisions onto the blockchain, adeptly handling both synchronous and asynchronous request flows differently. Multiple instances of the PE execute the core Trust Algorithm (TA), which performs multifaceted security checks based on identity (provided by the AS), the operational environment (using data from OSV), request parameters (validated by PC), and historical behavior patterns (queried from the BC-P-MON).

Supporting this core validation logic are several specialized components: the Authentication Service (AS) supplies vital information on known users/actors, including IDs, access rights, and network addresses (read-only for the TA); the Operating System Vulnerability (OSV) component provides data on known OS vulnerabilities for environmental risk assessment; the Parameter Checker (PC) validates the syntactic and semantic correctness of incoming request parameters; and the Blockchain Peer Monitoring (BC-P-MON) component, operating with a blockchain peer's identity, enables the TA to query the historical request data stored immutably on the blockchain for behavioral analysis.

Data and resource management are handled by dedicated Persistence Managers (PMs), each responsible for a specific resource type, such as the AUTH-PM managing authentication data via the AS-DB. Access to these PMs is strictly controlled through valid access tokens issued by the PA. Complementing the PMs is the Blockchain Peer Logging (BC-P-LOG) component, which also uses a blockchain peer identity, tasked by the PA with logging the details of incoming requests and their final validation outcomes onto the blockchain ledger, ensuring immutability.

The blockchain infrastructure itself is implemented as a permissioned blockchain using Hyperledger Fabric (HLF). In the described Proof-of-Concept (PoC), this setup includes a single organization, a single orderer node, three peers (though only one actively submits transactions in the PoC), a single channel, and one chaincode designed for storing and retrieving actor request history. The fundamental role of the blockchain is to provide a secure, tamper-proof, distributed ledger for this request history, which is critical for the TA's behavioral analysis and for logging access decisions for auditability and trust verification.

Interacting with the system are the end-users, IoT devices, and administrative entities (like a public service), connecting via the aforementioned Client and Analyser components. Communication within the architecture predominantly uses synchronous REST APIs for interactions between the various ZTA microservices. For specific asynchronous communication needs, such as notifications from the PA to the PEP, Redis is employed as a message broker.

Finally, the deployment strategy involves packaging all system components (except end-user tools) as Docker containers, orchestrated using Docker Compose. The core ZTA components are implemented as Java/Spring Boot applications, while the client-facing components utilize Angular and Python.

| **Quality Requirements** | |
|---|---|
| **Security** | The core focus. Achieved via: ZTA principles (least privilege, continuous verification), multi-factor checks (identity, environment, usage, behavior), blockchain for immutable request history/logging (integrity, non-repudiation, accountability support), cryptographic identity verification (authenticity). |
| **Performance/Efficiency** | Redis as a message broker supports Pub/Sub and queuing patterns for real-time (Time Behaviour) communication between systems. It provides efficient message delivery with low-latency, leveraging its in-memory data store. Redis Streams also enable event-driven architectures through time-ordered logs for asynchronous message processing. |
| **Maintainability** | The system is explicitly broken down into fine-grained microservices with distinct responsibilities (Modularity), facilitating independent development, deployment, and maintenance. |

| | |
|---|---|
| **Flexibility** | Scalability is explicitly discussed and evaluated, particularly concerning the impact of adding more Policy Engines (PEs). Potential for scaling PEs is noted, but performance impacts are measured.<br><br>Use of Docker containers for all components (except end-user tools) simplifies deployment and adaptation across different host environments supporting Docker (Installability and Adaptability). |
| **KBRef-09** | |
| **IoT Domain** | Healthcare |
| **Architecture** | Adaptive Fog-Cloud IoHT (Internet of Health Things) |
| It is a heterogeneous cloud-assisted communication framework specifically designed for Internet of Health Things (IoHT) applications. The architecture is structured into four distinct layers to efficiently manage diverse healthcare data types and processing requirements:<br><br>Data Collection Layer: This foundational layer interfaces with various healthcare devices (sensors, medical instruments, etc.) to gather both real-time (or near real-time) data (e.g., vital signs) and non-real-time big data (e.g., Electronic Health Records (eHR), medical images like MRI). Based on the data's characteristics and processing needs, it forwards the data upwards to either the Fog or Cloud layer.<br><br>Fog Layer: Positioned closer to the data sources, this layer is crucial for handling time-critical data. It performs initial processing tasks like filtering, aggregation, compression, rule-based preprocessing, and intermediate analytics. This reduces latency for urgent actions (e.g., detecting high blood pressure fluctuations needing immediate attention) and lessens the load on the cloud, improving overall QoS and bandwidth utilization. It also provides local storage.<br><br>Cloud Layer: This layer serves as the central hub for heavy computation and long-term storage. It receives preprocessed data from the Fog layer and also directly ingests large-volume, non-time-critical data (like eHR or high-resolution MRI images) from the Data Collection layer. The Cloud performs advanced analytics using data mining, machine learning, and complex algorithms on the aggregated heterogeneous data to extract deep insights.<br><br>Application Layer: The topmost layer provides user interfaces for various stakeholders (patients, doctors, researchers) to access the processed information, alerts, and healthcare applications derived from the underlying layers.<br><br>To manage data flow, resource allocation, and load balancing effectively across these layers, particularly for optimizing QoS, the framework utilizes Software Defined Networking (SDN). SDN decouples the control plane from the data plane, enabling centralized/distributed control over routing, scheduling, and resource allocation through network virtualization, adapting dynamically to different application needs. | |
| **Quality Requirements** | |
| **Performance/Efficiency** | The architecture explicitly aims to improve performance. Low latency for real-time data is achieved via the Fog layer. High throughput for big data analytics is handled by the Cloud. Dynamic allocation of the resources with SDN is used for efficient resource allocation and load balancing to optimize these metrics (latency, throughput, response time, bandwidth usage, E2E delay) (Resource Allocation). |
| **KBRef-10** | |
| **IoT Domain** | Smart Farm |
| **Architecture** | LoRaWAN-enabled |

The system architecture is designed as a low-cost, long-range wireless switching system specifically for intelligent agricultural applications, with a primary focus on automating irrigation to reduce labor and improve water consumption.

The core components of the architecture are:

End Devices (IoT Devices): These are the LoRaWAN Power Switch units. They consist of a board with:
LoRaWAN modules (specifically Radioenge LoRaWAN and LoRa ESP32 modules are tested).
Relays to control power to external devices (e.g., solenoid valves for irrigation, up to 220V).
Power supplies (5V for the board, 12V/24V for actuators like solenoid valves, using a hi-link mini-transformer and a step-down transformer). Status LEDs. Manual override switches.

LoRaWAN Gateway (GW): This device receives transmissions from the End Devices via LoRaWAN.
Network Server: The gateway forwards data to a Network Server (The Things Network platform is mentioned).

Application Server/Web System: The Network Server forwards messages to the correct application. In this case, it's a web system (developed with Python/Django) hosted on a UFPI server. This web system includes:
An MQTT Broker for communication with the IoT Hub (which seems to be part of the LoRaWAN network infrastructure, likely the gateway or network server).

A front-end for the farmer to remotely control the irrigation valves via a cell phone or computer.
The communication flow is: Farmer interacts with Web System -> Web System sends command via MQTT -> LoRaWAN Network Server -> LoRaWAN Gateway -> LoRaWAN End Device (Power Switch) -> Actuator (Solenoid Valve). Data from End Devices (e.g., status) would flow in the reverse direction.

| **Quality Requirements** | |
|---|---|
| **Performance/Efficiency** | It is achieved by the inherent low-power (Resource utilization) nature of LoRaWAN technology used in the end devices. |
| **Compatibility** | Use of MQTT allows the web application to communicate with the LoRaWAN backend and other subsystems (Interoperability). |
| **KBRef-11** | |
| **IoT Domain** | Smart Farm |
| **Architecture** | Microservices |

The proposed architecture is built using AWS serverless services. It aims to remotely monitor livestock, providing insights into their health and environment. The architecture is divided into several key frames, each leveraging specific AWS services:

AWS IoT Core: Manages and secures IoT devices, handling device connection, communication (MQTT), shadowing, authentication, and remote management.

Lambda: Serves as a versatile compute service, converting video frames, storing them in S3, transmitting them to Rekognition, and updating DynamoDB metadata.

Data Recognition: Uses AWS Rekognition to analyze video frames, identifying animals, people, and objects, detecting abnormal behavior, and sending alerts via Amazon Pinpoint.

Streaming Data: Employs Kinesis Data Streams for real-time data ingestion from IoT devices. Kinesis Data Firehose scales, groups, compresses, transforms, and encrypts data before storing it in S3.

Data Stores: Uses purpose-built databases like DynamoDB and Redshift to store events, deliver microservices, and generate operational dashboards accessible through AWS AppSync.

Data Processing: Utilizes AWS Glue for ETL (Extract, Transform, Load) processes, preparing data for analysis. Data Lake: Leverages Amazon S3 for storing both raw and processed data, enabling decoupled compute and storage using Amazon EMR for scalable data processing.

Logging: Uses Amazon CloudWatch for collecting metrics, logs, and audits, setting alarms, and triggering scaling operations.

Machine Learning: Employs Amazon SageMaker for building, training, and deploying machine learning models. It uses boosted decision trees for health prediction and linear regression for milk production forecasting. Edge models on AWS IoT Greengrass optimize battery power consumption.

Analytics: Uses Amazon Athena for querying data stored in S3 and Amazon QuickSight for creating scalable, ML-powered business intelligence dashboards.

Presentation: Uses Amazon Route 53, Elastic Beanstalk, and Elastic Load Balancer to provide a scalable and accessible web application.

User Identities: Secures access to the system using Amazon Cognito User and Identity Pools, providing features like MFA, compromised credential checks, and account takeover protection.

The system comprises three types of participants: Livestock IoT Devices (collecting sensor data), Cameras (monitoring animals), and IoT Edge (acting as a mediator with AWS IoT Greengrass core).

Data ingestion is critical, using Kinesis Data Streams and Firehose to handle high throughput and payload sizes up to 24KB per request.

Overall, the architecture is designed to be scalable, resilient, and cost-effective by using AWS serverless services.

| **Quality Requirements** | |
|---|---|
| **Flexibility** | The use of AWS serverless services (Lambda, Kinesis, S3, DynamoDB, etc.) inherently provides scalability (Flexibility). These services automatically scale up or down based on demand, ensuring the system can handle fluctuating workloads and increasing data volumes. Elastic Load Balancer distributes traffic across multiple EC2 instances. |
| **Maintainability** | The use of Amazon CloudWatch for logging, monitoring, and auditing directly supports analyzability. These tools provide insights into system behavior, making it easier to diagnose problems and assess the impact of changes. |

**KBRef-12**

| IoT Domain | Smart Farm |
|---|---|
| **Architecture** | Agricultural IoT Reference Architecture (AITRA) |
| The Agricultural IoT Reference Architecture (AITRA) is proposed as a standardized, comprehensive framework designed to guide the development of diverse smart agriculture solutions. Its primary goal is to offer templates and reusable components that streamline the creation of customized applications tailored to specific agricultural fields, while ensuring interoperability between these solutions and external systems.<br><br>AITRA presents its structure through two main views:<br><br>Layered System Architecture: A conceptual model emphasizing modularity and separation of concerns. It includes layers for Devices, Transport, Services (e.g., intelligence, visualization), Information & Data Management, and Applications. Users can interact via a rich GUI layer, either connecting directly to devices via Transport services or leveraging pre-built Application and Service layer modules. Crucially, all layers can utilize the Information & Data Management layer's services.<br><br>Three-Tier System Architecture: An implementation-focused model dividing the system into distributed hardware tiers:<br><br>Device Tier: Comprises IoT Devices (direct cloud connectivity) and Gateways (connecting local, potentially non-IP devices like sensors, actuators, drones, robots, embedded systems). Gateways can function as Edge Gateways, performing local processing (filtering, aggregation, analytics, control) to improve real-time response and reduce bandwidth usage. Key gateway components include Data Transport, Synchronization, Filtering/Aggregation, Device Management, Offload Analytics/Controls, and an Application (App) layer for abstraction and control.<br><br>Cloud Tier: Hosts the core platform logic, structured loosely rather than strictly layered. It includes Transport, Service, Information/Data Management, and Application layers, facilitating communication and providing core functionalities, data handling, and intelligence.<br><br>Business Tier: Represents the end-users (developers, companies, governments, individuals) and their applications, interacting with the Cloud Tier primarily through its GUI layer and development tools (like the design wizard shown in the example).<br><br>Communication between tiers relies heavily on messaging protocols (specifically a publish/subscribe model with a defined topic tree structure like AgriIoTFM, EconAndFIData, FieldAccount) and web browser-server interactions. Standardized frame formats are defined for configuration, data transmission, commands, and events/notifications to ensure consistent data exchange.<br><br>A key feature highlighted is the user-friendly development process, exemplified by a scenario where a user builds a farm management solution using drag-and-drop GUI components and pre-built modules for planning, monitoring, and control, drastically reducing development time and complexity. AITRA aims to be vendor-neutral, secure, reliable, and scalable, supporting standardized descriptions of local network topologies and providing dedicated telemetry, management, and manipulation capabilities. | |
| **Quality Requirements** | |
| **Compatibility** | Publish/Subscribe to standard messaging pattern. Focuses on interoperability through standardization, enabling communication and data exchange between diverse internal and external components/systems. |

| | |
|---|---|
| **Security** | Imagine AITRA as a secure community for smart farming devices and applications.<br><br>Getting Past the Gate (Authentication): Before a new device (like a sensor gateway) can even join this community and start talking, it needs to prove it belongs there. AITRA uses something called "code protection." Think of this like a secret handshake or a unique ID card the device shows at the gate during setup. This makes sure only genuine, authorized devices get registered, preventing random or malicious devices from joining.<br><br>Talking Securely (Secure Communication): Once inside, when a device sends its valuable data (like soil moisture readings) up to the cloud, or when the cloud sends commands back (like "start irrigation"), AITRA wants this conversation to be private. The text says devices "securely send its data." This means the messages are likely sent through a protected tunnel (using encryption, like HTTPS for websites or TLS for messaging). This stops others from easily listening in on the conversation or tampering with the messages as they travel across the internet.<br><br>Knowing Who Can Do What (Authorization & Access Control): Inside the community, not everyone needs to know everything or be able to control everything. AITRA organizes information into specific channels or "topics" (like different mailboxes). An application interested only in temperature data will subscribe only to the temperature mailbox for a specific field. It won't see the irrigation commands unless it's allowed to. Likewise, when an application tries to send a command (like telling a specific tractor to move), the system needs to check: "Is this application allowed to give orders to that tractor?" This ensures only authorized applications can access specific data or control specific devices.<br><br>So, in simple terms, AITRA checks who you are before letting you in (authentication), protects your conversations (secure communication), and controls what information you can see and what actions you're allowed to take (authorization and access control). It combines these steps to provide overall "security support" for the smart farm. |
| **Maintainability** | Achieved through clear separation of concerns via layered and tiered structures (Modularity), distinct functional modules within tiers (e.g., Filtering, Management, Analytics), promoting easier updates, replacements, and understanding. |
| **KBRef-13** | |
| **IoT Domain** | Smart Farm |
| **Architecture** | Layered |

This text describes a multi-layered system designed for an agro-weather station, focusing on data collection, transmission, presentation, management, and orchestration. Here's a summary:

1. Perception Layer (Data Collection):

Uses interconnected sensors to collect environmental data (temperature, humidity, pressure, solar radiation).
Nodes operate in hybrid mode, collecting data locally or transmitting wirelessly.
Employs an AI-based LSTM model for weather forecasting.
Nodes feature hybrid power (solar/AC) and energy-efficient firmware with adaptive communication.
Utilizes change point detection to adjust measurement frequency.
Includes agents for performance monitoring and alerts.
Nodes are designed for portability, scalability, and interoperability.
2. Transmission Layer (Data Transfer):

Ensures reliable communication between system elements.
Transports data from perception to presentation layers wirelessly or locally.
Supports various wireless standards (WIFI, NRF24L01, Bluetooth) and is open-source for future technologies (2G/3G/4G, Lora, LPWAN, Zigbee, Sigfox).
Offers connection-oriented and connection-less transmission modes.
Supports wired communication as well.
3. Presentation Layer (Data Visualization):

Presents formatted data through a user-friendly GUI compatible with various devices.
Supports the creation and enhancement of the AI-based weather forecasting model.
Prioritizes data visibility, accessibility, ease of use, system attractiveness, and system alerts.
Uses lightweight protocols (HTTP, MQTT) for energy efficiency.
4. Management Layer (System Control):

Provides real-time centralized monitoring of the base station and network elements.
Collects critical parameters (processor load, RAM usage, traffic, network performance, station health).
Allows remote configuration and monitoring of alert thresholds.
Accessible through a VPN tunnel or local network.
5. Middleware Layer (System Orchestration):

Acts as an orchestrator, creating interfaces between different layers.
Facilitates fast deployment of perception layer elements.
Includes a database for parameter management, cloud computing for data and model training, and a decision-making entity.

The agro-weather station's system architecture, emphasizing interoperability and efficiency. It opts for Docker containerization over virtualization for system abstraction, due to its advantages in scalability and performance. The system is built on open-source technologies (Debian OS, Docker) to reduce costs and allow customization. It employs a hybrid centralized/distributed design for device connection, ensuring portability and compatibility. Wireless nodes use standardized protocols (MQTT) for plug-and-play interoperability. The GUI prioritizes usability and data visibility. AI-powered features, leveraging Google Colab for resource-efficient background analysis, support informed decision-making. A multi-agent system (MAS) approach using Docker enhances modularity and maintainability, enabling efficient service upgrades via a SUS (Start, Upgrade, Stop) approach with secure script management.

| **Quality Requirements** | |
|---|---|
| **Compatibility** | Docker achieves interoperability (Compatibility) by creating standardized, portable containers for each system service, minimizing dependency conflicts. This allows seamless communication between modules using open-source technologies and standardized protocols like MQTT, ensuring compatibility across diverse deployment environments. |

| | |
|---|---|
| **Flexibility** | Docker contributes to scalability (Flexibility) in this system by:<br>Independent Service Scaling: Individual services within Docker containers can be scaled up or down based on demand, without affecting other components.<br>Orchestration with Kubernetes: Docker integrates with Kubernetes, enabling automated scaling and management of containerized services across multiple nodes. |
| **Maintainability** | Docker enhances maintainability (Modularity, Modifiability, Testability, and Reusability) in this system through:<br>Modular Updates: Containerized services enable isolated updates and fixes, reducing the risk of system-wide disruptions.<br>Simplified Troubleshooting: Consistent environments within containers streamline debugging and issue resolution.<br>Version Control: Docker images allow for versioning and rollback capabilities, simplifying software management.<br>Automated Deployment: Docker facilitates automated deployments, reducing manual intervention and potential errors. |
| **Performance/Efficiency** | Docker contributes to Performance/Efficiency in this system by:<br>Resource Optimization: Docker's efficient resource utilization allows for denser deployments, enabling the system to handle increased workloads. |
| **Interaction Capability** | Operability is archived by remote access to the deployed platform and its components (Web app and SSH). |
| **KBRef-14** | |
| **IoT Domain** | Smart City |
| **Architecture** | Secure Publish-Subscribe |

The architecture facilitates the transmission of vehicle distress signals to emergency services using a publish-subscribe model, specifically designed to function even in environments with limited traditional network connectivity.

It employs a hybrid communication approach:

1.Edge Communication (Vehicle-to-Infrastructure): Uses MQTT-SN over Zigbee for low-power, short-range communication.

1.1.ClientApp (Publisher): Runs on the vehicle, broadcasts distress messages (containing VIN, location) via Zigbee.

1.2.ForwardApp (Forwarder): Optional component, potentially on other vehicles or static units, receives MQTT-SN messages via Zigbee and relays them towards the gateway.

1.3.Road-side Infrastructure (MQTT-SN Gateway): Receives MQTT-SN messages via Zigbee from ClientApps or ForwardApps.

2.Backend Communication (Infrastructure-to-Services): Uses MQTT over TLS for secure communication over standard IP networks.

2.1.The Gateway translates MQTT-SN messages to MQTT and forwards them securely (via TLS) to the broker.

2.2.Control Centre (MQTT Broker): Central component that manages subscriptions and routes messages based on topics (e.g., distress signals) to appropriate subscribers.

| | |
|---|---|
| 2.3.Emergency Services (Subscribers): Applications used by emergency responders, subscribing to relevant topics on the MQTT Broker to receive distress signals.<br><br>The core function is asynchronous distress signal dissemination from a vehicle (ClientApp) potentially via intermediate forwarders (ForwardApp) to a gateway, then through a central broker to subscribed emergency services. The architecture explicitly addresses the "weakest link" identified in similar systems – the client-to-gateway communication – by incorporating MQTT-SN forwarders and defining requirements for this segment. | |
| **Quality Requirements** | |
| **Security** | Confidentiality: The architecture requires payload data encryption (SR3) throughout its journey, ensuring only authorized subscribers can read it. It also mandates secure channels (TLS) for communication segments involving the Gateway, Broker, and Subscribers.<br><br>Privacy: The architecture requires mechanisms like digital signatures, MACs, or hashing to allow subscribers to verify that neither the message payload nor the header has been tampered with during transit. TLS provides integrity for backend communication segments.<br><br>Authenticity: The architecture mandates verification of the message origin by the subscriber and requires the Broker to verify the identity of publishers and subscribers before allowing communication (Authenticity).<br><br>Non-Repudiation: The proposed use of digital signatures for authentication and integrity inherently provides non-repudiation capabilities if implemented correctly. |
| **Performance/Efficiency** | The architecture specifically selects MQTT-SN and Zigbee for the edge communication due to their lightweight nature and suitability for devices with "restricted power and memory capabilities" (Resource Utilization). |
| **KBRef-15** | |
| **IoT Domain** | Healthcare |
| **Architecture** | Reliable Mobile Healthcare |
| The proposed architecture presents a three-layered system (Device/IoT, Fog, Cloud) for pervasive healthcare monitoring. A key innovation is the division of the Fog layer into two sub-layers (Fog Node and Smart Gateway) to better distribute responsibilities and enhance specific quality attributes. The architecture aims explicitly to address non-functional requirements like interoperability, reliability, availability, and response time, leveraging best practices from IoT, Fog, and Cloud paradigms. Its design has been formally modeled (GTS, 4+1 views) and evaluated (Model Checking, ATAM). | |
| **Quality Requirements** | |
| **Performance/Efficiency** | Performance Efficiency is a key driver for the Fog layer design. The architecture aims to reduce latency and overhead (Time Behaviour and Resource Utilization), by:<br><br>Performing time-sensitive analysis and real-time decision making in the Fog layer (specifically Smart Gateway), closer to the user, reducing cloud round-trip time. Dividing Fog responsibilities: Fog Nodes handle basic connectivity/status, while Smart Gateways handle heavier local processing, storage, and scheduling, reducing overhead on individual nodes. Local processing minimizes delays for critical events. Scheduling optimizes resource usage for efficient task completion. |
| **Reliability** | Availability are central to this architecture, achieved largely through its innovative two-sub-layer Fog design that effectively distributes load. Crucially, local processing at the Smart Gateway level provides fault tolerance, enabling critical functions like emergency detection even without internet access. Workflow scheduling further ensures reliable performance under load, while integrated data consistency mechanisms maintain trustworthy information, especially vital during patient mobility. |

| KBRef-16 | |
|---|---|
| **IoT Domain** | Smart Farm |
| **Architecture** | Layered |

It is a six-layer architecture designed to structure the technological infrastructure for smart indoor farms. This architecture aims to provide a framework for collecting data from the physical environment, transmitting it for processing, analyzing it to derive insights and make decisions, and presenting information and control capabilities to users and business stakeholders. It conceptualizes the flow from physical devices to business value, leveraging modern technologies to create controlled, intelligent farming environments.

The layers are:

AgriSense Layer: The physical layer containing sensors (for soil moisture, temp, humidity, pH, cameras, etc.), actuators (motors, relays), micro-controllers, and tags (RFID) deployed on the farm to monitor the environment and plant status and perform actions.

Connectivity Layer: Responsible for routing data from the AgriSense layer upwards. It uses various communication technologies (Zigbee, Bluetooth, IEEE 802.x standards, GSM/GPRS) and network equipment (gateways, routers) along with web service protocols (SOAP, REST) for application communication.

Intermediate Layer: Acts as a bridge, facilitating communication between devices (AgriSense) and higher layers. It uses data protocols like MQTT, CoAP, and XMPP-IoT for bidirectional exchange, performs short-term data handling, aggregation, and protocol translation to enhance interoperability.

Core Data Handling Layer: The central processing and storage hub. It utilizes technologies like Cloud Computing, Big Data (e.g., HDFS for storage), databases, and Machine Learning/AI for long-term data analysis, decision-making (e.g., yield prediction, health monitoring, resource optimization), and management services.

Farmer Experience Layer: The user-facing layer that delivers services, information, predictions, and analysis reports to farmers via interfaces like mobile apps or customized web applications, enabling remote monitoring and control and potentially collecting user feedback.

Agri-Business Layer: Focuses on the overall system management, economic viability, and stakeholder integration. It defines profit models, manages services across layers, utilizes data analysis tools (AgBiz Logic, TOA-MD), and drives concepts like the Agricultural Value Chain (AVC).

This layered approach aims to structure the complex interactions within a smart indoor farm, leveraging technologies like IoT, WSN, Cloud Computing, Big Data, and AI to enable precision agriculture and move towards Agriculture 4.0 concepts within a controlled indoor setting.

**Quality Requirements**

| | |
|---|---|
| **Performance/Efficiency** | Addresses performance through efficient communication protocols, fast cloud processing, optimization algorithms for resource utilization (energy, water, nutrients), and energy-saving strategies (Green IoT).<br><br>Storing all that data reliably is crucial. What if one computer in the cloud fails? That's where HDFS (Hadoop Distributed File System) often steps in within the cloud environment. Think of HDFS as a super-smart, ultra-reliable filing system designed for huge datasets. It automatically breaks down the incoming data streams and copies pieces across many different computers in the cloud. If one computer goes down, the data isn't lost because copies exist elsewhere, ensuring the farm's historical records are safe and always available for analysis – it provides efficient distributed storage with built-in safety nets.<br>With all this data safely stored and accessible in the cloud brain, how does the farm make smart decisions? How does it decide the absolute best lighting schedule to maximize growth while minimizing electricity cost, or the perfect fertilizer mix based on sensor readings and crop type? This requires complex calculations and balancing competing goals. That's the role of Optimization Algorithms (like PSO, MOEAs, GA, LP). These are sophisticated mathematical tools, like expert planners running in the cloud. They sift through all the possibilities and constraints (water available, energy cost, desired yield) to find the optimal strategy or plan – the most efficient way to allocate resources or schedule actions.<br>While optimization algorithms handle the big-picture planning, something needs to manage the immediate, second-to-second adjustments. How do you keep the temperature exactly at 22°C or the humidity precisely at 65%? That's where PID (Proportional-Integral-Derivative) and Fuzzy Controllers act like highly responsive thermostats. A PID controller constantly measures the current state (e.g., temperature), compares it to the target, and makes precise, calculated adjustments to the heater or cooler to minimize the error quickly and smoothly. Fuzzy controllers are similar but excel when dealing with less precise inputs or rules, mimicking human-like reasoning (e.g., "if it's getting a bit warm and slightly humid, reduce heating a little"). They keep the environment stable based on the targets set, perhaps by the optimization algorithms.<br>Finally, underlying all of this is a growing concern for sustainability and energy use. Green IoT isn't a single technology, but an approach or philosophy applied throughout the system. It means choosing those energy-sipping Zigbee protocols, designing algorithms that explicitly consider energy consumption as a factor to minimize, putting devices to sleep when inactive, and potentially optimizing data transmission schedules. It's about making the entire smart farm operation as energy-efficient and environmentally friendly as possible, from the smallest sensor to the cloud data centers. |
| **Maintainability** | Layered architecture is explicitly structured into distinct layers with specific responsibilities (sensing, connectivity, data handling, etc.), promoting modularity and separation of concerns. |
| **Compatibility** | Layered architecture acknowledges interoperability challenges with diverse components. The Intermediate Layer explicitly aims to improve it via protocol translation. Proposes a software ecosystem as a potential solution. Uses standard protocols. |
| **Flexibility** | Layered architecture allows customization (apps, ecosystems). Mentions adapting models from other domains. The layered structure inherently supports adapting or replacing components within layers for different needs or technologies (Adaptability). |
| **KBRef-17** | |
| **IoT Domain** | Smart City |
| **Architecture** | Edge Computing-based Fault Tolerant Framework |

This describes a proposed framework designed for fault-tolerant edge computing in IoT environments, particularly focusing on smart mobility applications. The architecture emphasizes dynamic recovery from node, network, and data failures while leveraging edge processing for low latency and reduced bandwidth consumption, all within the constraints of limited edge resources managed via software containers.

The framework is structured into four distinct layers:

The foundation is the Device Layer, comprising various sensors, computing devices, and information systems responsible for collecting vast amounts of real-time data directly at the network edge. This layer supports on-device computation and can integrate external processing modules for simpler sensors.

Next, the Communication Layer establishes a unified publish/subscribe pipeline. It uses O-MI and O-DF messaging standards to enable peer-to-peer communication, publish real-time data, and crucially, replicate this data locally among edge devices. This local replication ensures data availability and overcomes network faults.

Overseeing resilience, the Management Layer employs the Kubernetes framework to orchestrate the deployment and operation of the data processing pipeline. Its key function is to provide node-level fault tolerance by automatically rescheduling failed processing tasks onto other available nodes within the edge cluster, transparently handling hardware failures.

Finally, the Application Layer hosts the end-user IoT applications, such as vehicular network-based smart mobility services. These applications leverage the fault tolerance, low latency, and efficient data handling provided by the underlying layers.

In essence, the framework provides a robust, layered approach using specific standards (O-MI/O-DF) and orchestration tools (Kubernetes) to build resilient IoT systems that can handle failures gracefully while optimizing performance and resource usage through edge processing and local data replication.

| **Quality Requirements** | |
|---|---|
| **Reliability** | The architecture is designed to automatically handle failures (Fault Tolerance). Node failures are managed by rescheduling processing tasks on other available nodes. Network and data failures are addressed by replicating data locally among devices in a cluster using a publish/subscribe mechanism. The management layer itself is designed to avoid single points of failure.<br>Kubernetes is used in the Management Layer to orchestrate processing stages, reschedule failed tasks (node fault tolerance), and ensure high availability. |
| **Performance/Efficiency** | The proposed architecture significantly addresses Performance Efficiency by strategically placing computation closer to the data sources in an edge-centric environment. This approach directly tackles Time Behavior, minimizing the physical distance data travels and thus ensuring low latency for critical processing tasks. Furthermore, it optimizes Resource Utilization by drastically reducing the amount of network bandwidth consumed, as raw data doesn't need extensive transmission. Recognizing the often limited resources available at the edge, the framework employs software containers (implicitly managed by Kubernetes) to efficiently package and manage applications, ensuring effective use of the available computational power and memory within those constraints (Resource Utilization). |
| **Flexibility** | The Management Layer uses Kubernetes to "orchestrate the placement of processing stages." Kubernetes is inherently designed to manage applications across a cluster of nodes. While the text highlights its use for rescheduling failed tasks (fault tolerance), the same orchestration capabilities allow Kubernetes to scale applications horizontally by deploying more instances (pods) of processing stages across available nodes as load increases. It can also manage the addition of new nodes to the cluster to increase overall capacity (Scalability). |
| **KBRef-18** | |
| **IoT Domain** | Smart City |
| **Architecture** | Layered |

| | |
|---|---|
| A layered architecture for a panic attack-based disaster evacuation system. The architecture is divided into three major layers:<br><br>Data Acquisition Layer: Collects data from various IoT sensors, categorized into:<br><br>Health-related attributes (e.g., heart rate, chest pain) using ECG, piezoelectric, and other sensors.<br>Environmental influent attributes (e.g., temperature, smoke) using temperature, pressure, infrared, and other sensors.<br>Location attributes (coordinates) of users and disaster sites.<br><br>Energy Efficient Event Classification Fog Layer: Processes data from the Data Acquisition Layer to classify the health state of users (Fine or Not Fine) while conserving energy. It contains:<br><br>Event Classification Layer: Uses a Fuzzy K-Nearest Neighbor algorithm to determine the user's health status.<br>Energy Conservation Layer: Consists of Spatio-temporal Analysis-based Data Selection and Dimensionality Reduction sublayers to minimize data transmission energy consumption.<br>Components: Sensor Microcontroller Unit (SMU) on a mobile phone synchronizes data and performs initial event classification. An Intelligent Gateway Node (IGN) at a regional server adds further energy efficiency by reducing data dimensions before sending it to the cloud.<br><br>Temporal Health Prediction and Geographic Mapping Cloud Layer: Predicts future health states and prioritizes evacuation efforts. It includes:<br><br>Temporal Health Prediction Layer: Forecasts future panic attacks using past and current health data.<br>Geographic Analysis Layer: Determines evacuation priorities for individuals and regions based on predicted health status and location, categorizing individuals as Extremely, Moderate, Mildly panicked, or normal.<br>The architecture prioritizes accurate data acquisition, energy-efficient processing at the fog layer, and predictive analysis at the cloud layer to optimize disaster evacuation strategies. | |
| **Quality Requirements** | |
| **Performance/Efficiency** | Energy efficiency (Resource Utilization) is achieved by selectively choosing and reducing the amount of data (via spatio-temporal analysis and dimensionality reduction) that needs to be transmitted from the resource-constrained fog devices (SMUs and sensors) to the cloud. This is accomplished through intelligent data selection, dimensionality reduction, and hierarchical processing within the fog layer. By minimizing data transmission, architecture aims to extend the battery life of the fog devices and sensors, which is crucial in disaster scenarios where power may be limited. |
| **KBRef-19** | |
| **IoT Domain** | Generic |
| **Architecture** | Self-Adaptive Hybrid IoT-ML Localization |

| |
|---|
| The proposed "Self-Adaptive Hybrid IoT-ML Localization" architecture is designed as a distributed system featuring two primary, loosely coupled components connected via middleware: an Internet-of-Things (IoT) Platform and a Localization Module. Its architectural style is considered Hybrid, integrating a Layered approach within the IoT platform (spanning Hardware, Data, Business, Display, and UI layers), likely leveraging Microservices for implementation as indicated by its use of the Spring Boot framework, and adopting a Service-Oriented/Component-Based structure for the overall separation between the platform and the localization logic.<br><br>The IoT Platform, deployed on a remote cloud server like Alibaba Cloud, handles device provisioning, management, and automation. It ingests raw sensor data (RSSI) from Bluetooth gateways using the MQTT protocol, performs initial data processing like parsing and filtering within its Business Logic layer (potentially interacting with an MS SQL database for user/device data), and manages data visualization through its upper layers.<br><br>The distinct Localization Module is responsible for the core positioning calculations. It employs a novel LSTM-based deep learning model combined with an MLP to accurately estimate distance from RSSI, moving beyond traditional propagation models. Position is then determined using trilateration techniques enhanced with self-adaptive mechanisms like "elastic radius intersecting" and "multiple weighted centroid localization" to ensure robustness against inaccurate measurements. Finally, it utilizes a self-adaptive Kalman Filter (specifically UKF) to smooth the calculated trajectory, improving overall stability and accuracy over time. This module runs as a separate component, acting as a client in the gRPC communication setup.<br><br>Communication between the IoT Platform (server) and the Localization Module (client) is handled by gRPC, ensuring loose coupling by exchanging only necessary processed RSSI data and receiving back calculated coordinates. Data management involves handling raw RSSI streams, processed data exchanged via MQTT and gRPC, likely persistent storage for platform management, and the crucial offline collection of ground-truth data for training the LSTM distance estimator. Key technologies underpinning this architecture include BLE 5.0, Spring Boot, MQTT, gRPC, LSTM, MLP, and Kalman Filters (UKF). |

| **Quality Requirements** | |
|---|---|
| **Functional Suitability** | Refining position estimates using weighted centroids and Kalman filtering (Functional Correctness). |
| **Maintainability** | The system is explicitly divided into two main components (Modularity), with distinct responsibilities (IoT platform vs. Localization algorithms), connected via a defined interface (gRPC). The IoT platform itself uses a multi-layer structure and microservices. |
| **Compatibility** | Uses standard protocols (MQTT, gRPC) for communication between different system parts (hardware-platform, platform-module), enabling components to interact across potential language/environment differences, interoperability. |
| **Performance/Efficiency** | Loose coupling via gRPC ensures only necessary data is exchanged, minimizing overhead compared to tighter integration and preventing components from negatively impacting each other's resource usage. |
| **Interaction Capability** | Provides a Web UI (Thymeleaf - a modern server-side Java template engine for both web and standalone environments [thymeleaf.og]) for users to view estimated positions, trajectories, and system status. Allows basic control via settings toggles. |
| **KBRef-20** | |
| **IoT Domain** | Smart City |
| **Architecture** | Cloud-Edge EV Smart Charging |

| The proposed architecture describes an IoT system designed to facilitate user-accepted smart charging for Electric Vehicles (EVs). It aims to make the charging process tangible and controllable for the user while enabling optimization based on factors like electricity price and battery health. The system comprises a user-facing web app (Smart Charging Wizard with UI and Optimization module), a backend session manager (CSH), a communication broker (MQTT), and a Programmable Logic Controller (PLC) interfacing directly with the charging station. This architecture explicitly and implicitly addresses several key quality attributes. | |
|---|---|
| **Quality Requirements** | |
| **Interaction Capability** | Interaction Capability (Operability) is a primary focus, driven by the need for user acceptance. This is achieved through the dedicated Smart Charging Wizard UI, which provides a simple, interactive, web-based interface (built with Streamlit) for users to easily set charging parameters (SOC, times), visualize the optimized charging plan (power profile, SOC evolution), monitor the live charging process, and start/stop/adjust sessions. This transparency makes the complex optimization tangible and controllable for the user. |
| **Compatibility** | The architecture uses standard protocols (HTTP, MQTT, JSON, UDP, IEC 61851) for communication between diverse components (web app, cloud services, PLC, station). The PLC API aims for manufacturer independence (Interoperability). |
| **Reliability** | The CSH is designed for continuous availability using containerization and orchestration (Docker, Kubernetes), supporting resilience and access from multiple points. Its independence prevents single points of failure tied directly to the Wizard. |
| **Maintainability** | The generic Programmable Logic Controller (PLC) API is designed to be reusable across different charging station types. |

| | |
|---|---|
| **Safety** | Safety (Risk Identification) is explicitly addressed in the described architecture. The design incorporates multiple mechanisms primarily implemented at the Programmable Logic Controller (PLC) level (safety limits, current limit checks, message validation, adherence to IEC 61851) and within the optimization logic (battery operating constraints).<br><br>In essence, the system achieves safety through active prevention and adherence to standards, primarily managed by the PLC and the optimization logic:<br><br>1.By Enforcing Hard Limits (PLC Safety & Current Limits): The PLC acts as a vigilant guard. It constantly checks if the requested charging power or current exceeds the predefined maximum safe levels for the station, the EV, or the electrical circuit. If a command asks for too much, the PLC blocks or reduces it.<br>How it mitigates: This directly prevents overheating of wires, components (in the station or EV), and potentially the building's wiring, thus mitigating fire and electrical shock risks (Health & Safety). It also prevents physical damage to the expensive charging equipment and the EV's battery/charger due to electrical stress (Economic Risk).<br><br>2.By Validating Instructions (PLC Message Validation): Before acting on any command from the cloud, the PLC checks if the instruction makes sense and is correctly formatted. If it receives a garbled or illogical request, it rejects it.<br>How it mitigates: This prevents the system from performing unpredictable or dangerous actions based on faulty data, safeguarding against potential electrical hazards (Health & Safety) and preventing operations that could damage equipment (Economic Risk).<br><br>3.By Following Established Rules (IEC 61851 Standard): The system uses a well-defined industry standard for the charging communication itself. This standard includes built-in safety handshakes and checks (like verifying a proper connection before power flows).<br>How it mitigates: This ensures fundamental electrical safety protocols are followed, reducing risks like energizing an improperly connected cable (Health & Safety) and ensuring compatible, non-damaging interaction between the car and station (Economic Risk).<br><br>4.By Respecting Component Capabilities (Battery Constraints in Optimization): The smart charging plan itself is designed not to push the EV battery beyond its safe operating conditions (temperature, charge level, power input).<br>How it mitigates: This avoids stressing the battery, which reduces the risk of dangerous internal failures like thermal runaway (Health & Safety) and prevents accelerated degradation or damage, preserving the battery's lifespan and value (Economic Risk).<br><br>In short, the architecture uses the PLC as a local safety enforcer and relies on standardized protocols and intelligent planning to keep the charging process within safe physical and electrical boundaries, protecting both people and equipment. |
| **KBRef-21** | |
| **IoT Domain** | Generic |
| **Architecture** | Layered |

The architecture is designed to support Ambient Assisted Living (AAL) systems by providing reliable and timely indoor and outdoor positioning information within an Internet of Things (IoT) infrastructure. It aims to address the specific needs of AAL, such as real-time responsiveness crucial for critical event detection, while effectively managing the inherent complexities of IoT environments, including device heterogeneity, the need for scalability, and resource constraints. To achieve these goals, the architecture adopts a layered approach that strategically incorporates Fog and Mist computing paradigms alongside traditional Cloud computing elements.

Architecture Overview:

The architecture is structured across four main logical layers. At the edge, the Perception Layer utilizes Mist computing principles and contains a diverse array of sensors capturing health metrics, environmental conditions, and location data via technologies like BLE and IMU along with actuators for environmental interaction. Mist computing nodes within this layer handle the initial data gathering, such as capturing BLE packets from wearables, managing device interactions, and performing basic data processing or forwarding.

Moving upwards, the Fog Layer consists of more powerful nodes, such as environment or area gateways. These nodes bring computation closer to the end-user, handling intermediate processing tasks like executing localization algorithms, detecting Activities of Daily Living (ADL) and critical events, and coordinating local responses. This layer employs a hierarchical structure to manage different environmental scopes effectively.

The Cloud Layer serves as the central backend, providing robust capabilities for centralized data storage, complex data analysis, and overall system management. It leverages a microservices-based architecture offering services for data persistence, context-awareness, advanced localization/mapping, notifications, security/authorization, and fault management. Integration with external systems and applications is achieved through well-defined RESTful APIs.

Finally, the Application Layer hosts the end-user applications. These applications, potentially used by caregivers for remote monitoring or by assisted individuals to control their environment, consume the data processed by the lower layers and interact with the system's functionalities through the APIs provided by the Cloud Layer. Key technological decisions underpinning this structure include the use of BLE for indoor localization and GPS for outdoor tracking, the strategic distribution of processing load across Fog/Mist layers, and the adoption of microservices with RESTful APIs in the cloud to ensure flexibility and manageability.

| **Quality Requirements** | |
|---|---|
| **Performance/Efficiency** | Using Fog computing, comproved by simulation results demonstrate a significant reduction in latency (Time Behaviour), making it suitable for "real-time applications".<br><br>Fog-centric processing reduces data transmission to the cloud, resulting in a "significant decrease in network usage". This leads to a "substantial reduction in cloud processing costs". A "slight reduction" in cloud server energy consumption was also observed (Resource Utilization). |
| **Compatibility** | The Cloud Layer utilizes "RESTful API to integrate with the "assisted ambient" and another API for application integration (Interoperability). This standard-based interface promotes interoperability with diverse applications and potentially other systems. |
| **Maintainability** | The architecture employs a clear layered structure, promoting modularity. The use of RESTful APIs significantly enhances maintainability by allowing individual components to be modified independently. |
| **KBRef-22** | |
| **IoT Domain** | Generic |
| **Architecture** | A4IoT (Anyplace 4.0 IoT) |

The A4IoT architecture is presented as a novel, open-source solution designed specifically for IoT localization within diverse "smart spaces" like factories, hospitals, and ships. Recognizing the limitations of satellite-based positioning indoors, A4IoT utilizes signal fingerprinting principles to integrate and manage a wide array of localization technologies (including Wi-Fi, BLE, Cellular, UWB, and Computer Vision) under a unified framework. Its core purpose is to provide accurate (targeting room-level, ≈ 2m) localization data, running efficiently on edge devices (from Raspberry Pi up to Datacenters) and integrating seamlessly with existing software ecosystems via Web 2.0 endpoints. It supports crowdsourcing for data collection and aims to address real-world industrial requirements, such as those from smart factories.

Architecture Overview:

A4IoT is structured into distinct frontend and backend components:

1.Backend:
1.1.Server: Built using the Play framework, it contains the core application logic, exposes a RESTful API for interaction (crowdsourcing, queries), handles OAuth 2.0 authentication, performs floorplan tiling, and provides visual analytics capabilities.
1.2.Data Store: Manages different types of data using specialized stores: a Distributed Filesystem (DFS, e.g., GlusterFS) for raw data/files, a Time-series store (specifically InfluxDB) for IoT sensor readings and tracking data, and a Document store (initially Couchbase, migrating towards MongoDB) for JSON objects and potentially replacing some spatial views.

2.Frontend:
2.1.Web Applications: Includes modules like Architect (for floorplan design/POI management), Viewer (search/navigation engine), and Analytics (visualization dashboard, FMS - Fig. 6), built with HTML5, CSS3, and AngularJS.
2.2.Library: Provides core functionalities reusable by clients. Key libraries are anyplace-core (generic Java/Gradle library wrapping API endpoints) and anyplace-android (Android-specific version handling permissions, background tasks, caching).
2.3.IoT Clients: Reference implementations and tools enabling deployment on various platforms (Linux, macOS, Windows, Android, RobotOS) using the provided libraries, including a Command Line Interface (CLI).

Key technologies and design decisions include fingerprinting (primarily RSSI), edge computing deployment (Raspberry Pi to cluster), containerization via Docker (Sec IV), crowdsourcing support, RESTful APIs, use of specialized databases (InfluxDB, GlusterFS, MongoDB), and modular client libraries.

| **Quality Requirements** | |
|---|---|
| **Performance/Efficiency** | With spatio temporal store (InfluxDB) we can now consume high-volume input streams to provide real-time IoT tracking (Time-behaviour). |
| **Reliability** | Our production environment uses a 3-node cluster with two replicas, for both the database engines and the DFS (Distributed Filesystem). This setup allows for a full operation with just a single node active at any time.<br><br>HAProxy provides protection from attacks like DDoS and supports availability. |
| **Security** | All communication is encrypted even in internal networks, as A4IoT image automatically creates and uses SSL certificates (Confidentiality). |
| **Flexibility** | It utilizes Docker to enable the deployment of single-node or multi-node configurations, regardless of any dependencies or the underlying OS (Adaptability). |
| **KBRef-23** | |
| **IoT Domain** | Smart City |

| **Architecture** | Smart Geo Layers (SGeoL) |
|---|---|
| SGeoL is designed as a modular, open, and scalable platform to simplify smart city application development. It uses a distributed middleware architecture, built on open standards like NGSI-LD and RESTful APIs, to ensure interoperability and ease of integration. This architecture prioritizes data management by supporting diverse data formats and utilizing specialized databases for context, geographic, and semantic information.<br><br>Key characteristics include:<br><br>Interoperability: Seamless integration with external systems through open protocols and standardized APIs.<br>Data Heterogeneity: Ability to handle and integrate data from various sources and formats.<br>Modular Design: Independent components for core functionalities like security, IoT management, and data analysis.<br>Scalability: Designed for cloud deployment, with future plans for Kubernetes to enhance self-scalability.<br>Security: Robust security measures through OAuth and role-based access control.<br>Data Analysis: Real-time and batch processing capabilities for deriving insights from urban data.<br>IoT Integration: Simplified connection and management of IoT devices.<br>Essentially, SGeoL provides a comprehensive, flexible, and secure framework for building smart city applications by abstracting complexities and leveraging powerful middleware services. | |
| **Quality Requirements** | |
| **Security** | Security in the SGeoL architecture is achieved through a combination of authentication, authorization, and access control mechanisms, leveraging existing middleware services and implementing a dedicated Security Manager component. Here's a breakdown:<br><br>Authentication (Identity Manager):<br><br>SGeoL utilizes the Identity Manager component, which is realized by FIWARE's Keyrock, to handle user and application authentication.<br>This component manages user credentials, issues access tokens (using OAuth), and validates these tokens when requests are made to SGeoL APIs.<br>When a request is sent to SGeoL, the Security Manager extracts the access token from the HTTP header and forwards it to the Identity Manager for verification. If the token is invalid, access is denied.<br>Authorization (Authorization PDP):<br><br>Once a user or application is authenticated, the Authorization PDP component (FIWARE's AuthzForce) comes into play.<br>This component manages access policies that define what actions authenticated users or applications are allowed to perform.<br>The Security Manager component queries the Authorization PDP to determine if the user or application has the necessary permissions for the requested operation. If authorized, the request proceeds; otherwise, access is denied.<br>Security Manager Component:<br><br>This component acts as a gatekeeper, intercepting all requests to the SGeoL APIs.<br>It enforces the security model, which is based on roles, access policies, and the OAuth protocol.<br>It handles the interaction between the Identity Manager and Authorization PDP, ensuring that only authenticated and authorized entities can access and manipulate SGeoL data.<br>This component effectively centralizes the security enforcement.<br>Role-Based Access Control:<br><br>The system uses roles to define permissions, which allows for granular control over data access. |

| | |
|---|---|
| | In essence, SGeoL's security is a two-step process: first, verifying the identity of the requester (authentication), and then, checking if they have the right permissions to perform the requested action (authorization). This is all managed by the Security Manager, which uses the underlaying middleware services. |
| **Flexibility** | SGeoL achieves scalability (Flexibility) primarily through its distributed architecture and planned Kubernetes container (Docker) orchestration. Individual components are designed for independent scaling, leveraging cloud infrastructure (OpenStack currently, transitioning to Kubernetes). Kubernetes will automate deployment, scaling, and fault management of containerized services, enhancing self-scalability and resource optimization. This approach ensures the platform can adapt to varying loads efficiently. |
| **Performance/Efficiency** | Containerization (Docker) and orchestration with Kubernetes: Docker containerization simplifies deployment and resource management, while the planned Kubernetes integration will further optimize resource utilization and automate scaling, improving efficiency. |
| **Compatibility** | The platform exposes its functionalities through high-level RESTful APIs, which are widely used and well-understood. This allows developers to easily access and utilize SGeoL's services from various programming languages and platforms (Interoperability). |
| **KBRef-24** | |
| **IoT Domain** | Smart City |
| **Architecture** | Distributed Blockchain-SDN Based |

The proposed architecture is built using AWS serverless services. It aims to remotely monitor livestock, providing insights into their health and environment. The architecture is divided into several key frames, each leveraging specific AWS services:

AWS IoT Core: Manages and secures IoT devices, handling device connection, communication (MQTT), shadowing, authentication, and remote management.

Lambda: Serves as a versatile compute service, converting video frames, storing them in S3, transmitting them to Rekognition, and updating DynamoDB metadata.

Data Recognition: Uses AWS Rekognition to analyze video frames, identifying animals, people, and objects, detecting abnormal behavior, and sending alerts via Amazon Pinpoint.

Streaming Data: Employs Kinesis Data Streams for real-time data ingestion from IoT devices. Kinesis Data Firehose scales, groups, compresses, transforms, and encrypts data before storing it in S3.

Data Stores: Uses purpose-built databases like DynamoDB and Redshift to store events, deliver microservices, and generate operational dashboards accessible through AWS AppSync.

Data Processing: Utilizes AWS Glue for ETL (Extract, Transform, Load) processes, preparing data for analysis. Data Lake: Leverages Amazon S3 for storing both raw and processed data, enabling decoupled compute and storage using Amazon EMR for scalable data processing.

Logging: Uses Amazon CloudWatch for collecting metrics, logs, and audits, setting alarms, and triggering scaling operations.

Machine Learning: Employs Amazon SageMaker for building, training, and deploying machine learning models. It uses boosted decision trees for health prediction and linear regression for milk production forecasting. Edge models on AWS IoT Greengrass optimize battery power consumption.

Analytics: Uses Amazon Athena for querying data stored in S3 and Amazon QuickSight for creating scalable, ML-powered business intelligence dashboards.

Presentation: Uses Amazon Route 53, Elastic Beanstalk, and Elastic Load Balancer to provide a scalable and accessible web application.

User Identities: Secures access to the system using Amazon Cognito User and Identity Pools, providing features like MFA, compromised credential checks, and account takeover protection.

The system comprises three types of participants: Livestock IoT Devices (collecting sensor data), Cameras (monitoring animals), and IoT Edge (acting as a mediator with AWS IoT Greengrass core).

Data ingestion is critical, using Kinesis Data Streams and Firehose to handle high throughput and payload sizes up to 24KB per request.

Overall, the architecture is designed to be scalable, resilient, and cost-effective by using AWS serverless services.

| **Quality Requirements** | |
|---|---|
| **Performance/Efficiency** | The DistB-Condo architecture incorporates features to enhance performance efficiency, notably energy savings via cluster head selection, optimized network control and filtering via SDN, and potentially faster processing and load balancing via NFV. However, the integration of Blockchain, while crucial for security, introduces significant performance trade-offs in terms of latency (time behavior), resource consumption (CPU, storage, bandwidth), and transaction throughput limits (capacity). The overall performance efficiency depends heavily on the specific implementation details and the balance struck between security requirements and performance needs. |

| | |
|---|---|
| **Security** | Blockchain adds a robust layer of data-centric security, ensuring integrity, confidentiality, non-repudiation, and authenticity of transactions via its distributed ledger, cryptographic linking, consensus mechanisms, and smart contract validation. |
| **KBRef-25** | |
| **IoT Domain** | Healthcare |
| **Architecture** | Event-driven IoT |
| This architecture proposes a layered approach for IoT-aware systems, focusing on context interpretation, event processing, and service execution.<br><br>The Context Layer defines the system's understanding of the current situation. It gathers data from heterogeneous sources (user profiles, sensors), represented hierarchically using XML profiles with weighted components.<br><br>This raw data undergoes filtration to remove noise and errors. Specific filters are applied to derive measurable parameters (like heart rate from continuous signals). The resulting context model is then generated and broadcast for decision-making.<br><br>The Event Layer processes streams of events based on changes detected in the context layer, utilizing Complex Event Processing (CEP). It supports both offline processing (using stored data) and online processing (using CEP for real-time analysis). Key components convert context changes into events, guarantee complete processing (considering event priorities for performance), use a CEP engine for complex event analysis, and trigger real-time responses via an Event Trigger component.<br><br>The Service Layer is activated by the Event Trigger. It manages the execution of appropriate services based on detected events. A Service Manager determines if a service is simple or composite and queues requests. A Service Coordinator selects the appropriate service for execution, potentially suspending others. A Service Execution unit runs the chosen service, while a Logger unit records any execution failures and reports them back to the manager. | |
| **Quality Requirements** | |
| **Performance/Efficiency** | The architecture aims for timely responses by processing events as they happen ("online," "real-time") using specialized techniques (CEP, in-memory processing) and by prioritizing important events to ensure they are handled promptly (Time Behaviour). |
| **Functional Suitability** | The system strives for correctness by applying Event processing logic (within the Event Processing component from CEP) and filters to raw data to remove noise/errors and derive accurate values (e.g., heart rate). It aims for completeness by ensuring that all relevant events are processed before triggering actions. |
| **Reliability** | Reliability, defined as the timely and consistent reception of valid data packets, is primarily demonstrated through verification and testing, rather than specific resilience features described here. This involves validating incoming packets against protocols, measuring inter-packet timing to confirm low delay, and checking for packet loss (TCP-IP protocol and Client/Server architecture for data communication). Ultimately, the claim of reliability relies on experimental results indicating acceptable performance under the tested conditions. |
| **KBRef-26** | |
| **IoT Domain** | Smart City |
| **Architecture** | SmartGC |

| | |
|---|---|
| The SmartGC architecture is presented as a layered system designed to test, deploy, and manage garbage collection routes within a smart city context, leveraging the Artificial Transportation Systems (ATS) concept as its underlying methodology. It explicitly defines functionalities and considers quality attributes crucial for its stakeholders (regulators, operators, app providers, users). The architecture integrates data from various sources, processes it using semantic technologies and simulation, and provides interfaces for monitoring, decision support, and application development, often leveraging components from the FIWARE platform for implementation. | |
| **Quality Requirements** | |
| **Performance/Efficiency** | Performance Efficiency is addressed through architectural choices supporting responsiveness and efficient resource use. Real-time capabilities are crucial (monitoring, communication, data processing). This is supported by NoSQL databases for handling large data volumes, indexed RDF stores for fast queries, stream processing capabilities within the microservice-based Execution Manager, and publish-subscribe mechanisms (like FIWARE Orion). Simulation aims to find optimized (efficient) routes.<br><br>FIWARE provides core components like Orion and IoT Agents (IDAS), and security GEs (mentioned later in the text like Keyrock, AuthZForce, PEP Proxy). It then relies on integrating with standard NoSQL databases (like MongoDB), processing engines (Stream/Batch), semantic stores (RDF Stores), and domain-specific tools (ATS/Simulation) to build complete solutions, often facilitated by custom middleware (SGEOL) or architectural patterns (Microservices). |
| **Interaction Capability** | Interaction Capability is addressed through dedicated user interface components and specific agent roles. The Dashboard component provides visualizations for monitoring and decision support. The Open Data Publisher (CKAN) offers a user-friendly portal. The test-bed aims for easy visual interpretation. The Tutor Agent specifically addresses operator training (Learnability). FIWARE Wirecloud facilitates building usable web interfaces (widgets/mashups). |
| **Security** | Security is primarily addressed through robust Access Control. A dedicated Security Control component intercepts API requests, evaluates policies based on roles/permissions, and enforces access decisions. The implementation explicitly leverages FIWARE security GEs (FIWARE Platform): Keyrock for identity management and authentication, AuthZForce for authorization policy management (using XACML), and PEP Proxy as the enforcement point (Authenticity and Confidentiality). |
| **Flexibility** | Flexibility is addressed through Scalability. Scalability is explicitly mentioned as a benefit of modularity and supported by technology choices like NoSQL databases and a microservice-based Execution Manager designed to handle large user/data loads. The InserSCSimulator used in implementation is noted for "massive scalability". |
| **Maintainability** | Maintainability is supported by the explicit choice of a layered architecture to enhance modifiability. Components are designed to be modular and self-contained. The Execution Manager uses a microservice architecture, further promoting modularity. Using FIWARE leverages potentially well-maintained, modular components. |

| | |
|---|---|
| **Compatibility** | FIWARE achieves interoperability using two main components working together with the NGSI standard language:<br><br>FIWARE IoT Agents (like IDAS): Act as translators at the edge, converting data from diverse IoT devices (each with its own protocol) into the standard NGSI format. This provides interoperable IoT device integration, as the rest of the system only needs to understand NGSI, not specific device protocols.<br><br>FIWARE Orion Context Broker: Serves as the central hub that manages this standardized NGSI data. It stores the real-time status (context) of entities and allows applications to query or subscribe to updates, also using the standard NGSI interface. This ensures interoperable context data management, enabling different applications to seamlessly share and react to real-time information. |
| **KBRef-27** | |
| **IoT Domain** | Healthcare |
| **Architecture** | Layered |
| It is a five-layer architecture for an IoT-based e-health system, prioritizing security, privacy, and real-world applicability. It incorporates blockchain for secure data management, utilizing both public and private mechanisms to ensure data integrity and authorized access. A fog layer is introduced to enhance performance by processing sensitive patient data at the network edge, minimizing latency. The architecture is designed to align with existing healthcare practices in Pakistan, facilitating remote patient monitoring through sensor data transmitted to the cloud. The inclusion of an e-governance layer emphasizes regulatory oversight and interoperability between e-health systems. The sensing layer gathers patient data, the application layer provides user interfaces and real-time alerts, and the transmission layer securely transfers data to storage. The system aims to improve healthcare in emerging countries like Pakistan, while addressing the security challenges inherent in IoT through blockchain integration. | |
| **Quality Requirements** | |
| **Performance/Efficiency** | Connecting IoT devices at the network's edge, through fog computing, enhances e-health system performance by: reducing latency, minimizing network congestion, distributing processing loads, enabling real-time data analysis, and optimizing bandwidth usage (Time Behaviour and Resource Utilization). |
| **Security** | Blockchain is used as a distributed ledger to ensure data integrity and immutability (Integrity). Data is stored in blocks with timestamps and cryptographic signatures, making it tamper-proof (Non-repudiation).<br>Both public and private blockchain mechanisms are employed, controlling data access and ensuring only authorized participants can view or modify information through cryptographic keys and consensus (Confidentiality). This distributed approach eliminates reliance on a central authority, enhancing security against single points of failure (Authenticity). |
| **KBRef-28** | |
| **IoT Domain** | Healthcare |
| **Architecture** | Cloud-based Secure Biometric |

1. Healthcare Cloud (Core):

General Concept: This is the central cloud-based environment provided by a third-party provider. It offers the infrastructure (servers, storage, networking) and platform services needed to run the BamHealthCloud system.

Health Data Store (HDMS):

Purpose: Manages all patient-related information. This encompasses electronic health records (EHRs), lab results, imaging studies, medical history, medications, and billing information. It ensures efficient storage, retrieval, and updates of this data.

Layers:

Client Level: This is the user interface layer. It provides access to the data for doctors, nurses, administrators, and potentially patients (with appropriate access controls). This layer would likely be implemented using web-based or mobile applications.

Health Service Provider Level:

Administrative Level: Manages user accounts, permissions, resource allocation, and overall system configuration.
Security Level: Where the Biometric Authentication Agent (BAA) operates, enforcing access control policies.

Data Center Level: Deals with the physical storage and management of the healthcare data. It ensures data durability, backups, and disaster recovery. It also includes functionalities for data indexing and optimization for efficient querying. This layer would leverage cloud storage services provided by the cloud provider (e.g., AWS S3, Azure Blob Storage).

Security Manager:

Role: This component is responsible for enforcing all security policies within the system. It integrates with the BAA to authenticate users and authorize access to specific data and functions.

Functions: In addition to user authentication, the Security Manager likely handles other security tasks, such as:

Data Encryption: Ensuring that sensitive data is encrypted both in transit and at rest.
Access Control Policies: Defining and enforcing fine-grained access control rules (e.g., allowing a doctor to view a patient's records but not modify billing information).

Auditing: Logging all access attempts and data modifications for security monitoring and compliance purposes.

Intrusion Detection: Monitoring the system for suspicious activity and taking appropriate action (e.g., blocking an IP address after multiple failed login attempts).

2. Biometric Authentication Agent (BAA) - Signature Dynamics:

Feature Extraction:

Details: The BAA captures dynamic characteristics of a signature, which are much harder to forge than static features (like the visual appearance of the signature). Key dynamic features include:
X and Y Coordinates: The trajectory of the pen across the signature area.

Velocity: The speed of the pen at different points in the signature.
Time:* Total time taken to complete the signature.
Pen Angle: The angle of the pen relative to the writing surface.
Pen-Ups and Pen-Downs:* The number of times the pen is lifted off the writing surface.
Acceleration: The rate of change of velocity.

Hardware:* Digitizing tablets provide precise capture of these features. Smartphones equipped with signature capture software are a more convenient (and potentially less secure) alternative.

Template Creation:

Process: The extracted features are preprocessed and organized into a template. Preprocessing might involve normalization (scaling features to a common range) and noise reduction. The template represents a mathematical model of the user's signature dynamics.

Neural Network Model:

Training: The templates are used to train a neural network. The neural network learns to distinguish between genuine signatures and forgeries. Different types of neural networks could be used, but the text mentions a feedforward neural network with Resilient Backpropagation.

Storage: The trained neural network is stored securely within the cloud.

Verification:

Process: When a user attempts to authenticate, their signature is captured, features are extracted, and these features are fed into the trained neural network. The network outputs a score or probability indicating the likelihood that the signature is genuine.

Matching: This score is compared to a threshold. If the score exceeds the threshold, the user is authenticated. The threshold value may be adjusted based on the user's priority level.

3. Enrollment and Authentication Processes:

Enrollment (Phase I):

Quality Check: The "SigQuality checker" is a crucial step. It ensures that the signature samples are consistent, complete, and free from excessive noise. This improves the accuracy of the biometric system.

Multiple Samples: The system likely requires multiple signature samples during enrollment to capture the natural variations in a person's signature.

Authentication (Phase II):

Threshold Tuning: As mentioned, the authentication threshold is adjusted based on user priority. High-priority users (e.g., doctors with access to sensitive patient data) have a higher threshold, requiring a more precise signature match.

4. Priority-Based Access Control:

Purpose: To implement a fine-grained security model, limiting access to data based on user roles and responsibilities.

Priority Levels (1-4):
Example Mapping (from Table 2, though specifics would vary):
Priority 4: Head Doctors/Senior Administrators (Full access)
Priority 3: Registered Doctors (Access to patient records)
Priority 2: Nurses/Technicians (Limited access to specific patient data)
Priority 1: Regular Staff/Clerical Staff (Limited access to administrative data)
Algorithm 1 (ALGOHealthSecurityCheck): This algorithm sets the authentication threshold based on the user's priority. It likely performs a lookup in a table or applies a formula to determine the appropriate threshold for a given priority level.

Algorithm 2 (ALGOHealthAuthentication): This algorithm performs the actual authentication process, incorporating the threshold value set by ALGOHealthSecurityCheck.

5. MapReduce and Parallel Processing:

Hadoop Framework: Hadoop provides a distributed storage and processing platform.
MapReduce: A programming model for parallel processing of large datasets.
Application:
Signature Template Training: Training the neural network with a large dataset of signature templates can be computationally intensive. MapReduce allows this training to be distributed across multiple machines, significantly speeding up the process.
Covariance Calculation: The algorithm mentions calculating covariance on the input data sample. Covariance calculations are common in statistical analysis and can be parallelized efficiently using MapReduce.
Parallel Authentication: Potentially, the verification process (matching the signature features against the template) can also be parallelized to some extent, allowing multiple authentication requests to be processed concurrently.

| Quality Requirements | |
|---|---|
| **Flexibility** | Parallelized MapReduce programming model is a key phrase that captures how BamHealthCloud can efficiently process large volumes of healthcare data by distributing the workload across multiple computing nodes, providing scalability, cost-effectiveness, and fault tolerance. It's a technical detail that underlines the overall scalability of the architecture. |
| **Performance/Efficiency** | Parallelized MapReduce programming model is a key phrase that captures how BamHealthCloud can efficiently process large volumes of healthcare data by distributing the workload across multiple computing nodes, providing scalability, cost-effectiveness, and fault tolerance. It's a technical detail that underlines the overall scalability of the architecture. |
| **Reliability** | MapReduce framework contributes to achieving availability in BamHealthCloud, although it's more accurate to say it enhances availability rather than solely achieving it. Here's why and how:<br>Replication of Data: Hadoop, the typical implementation of the MapReduce framework, inherently involves the replication of data across multiple nodes in the cluster. This means that if one node fails, the data is still available on other nodes. This data replication is a core mechanism for ensuring availability.<br>Fault-Tolerant Task Execution: If a task fails on one node (due to hardware failure, software error, etc.), the MapReduce framework automatically retries the task on another available node. This ensures that the processing continues even if some nodes experience problems.<br>Automatic Failover: In a well-configured Hadoop cluster, there are mechanisms for automatic failover. If a critical component (like the NameNode, which manages the file system) fails, a standby component can take over its role, minimizing downtime.<br>Distributed Nature: The distributed nature of MapReduce means that the workload is spread across multiple nodes. This reduces the impact of a single node failure on the overall system. |
| **Security** | The architecture employs biometric authentication using dynamic signature analysis (Confidentiality, Integrity, and Authentication). A user's signature is captured (pen strokes, speed, pressure) and used to train a neural network. This model is stored securely in the cloud. During login, the user provides a signature, which is compared to the stored model. Access is granted only if the signature matches, with higher security thresholds for users with greater data access privileges. This method enhances security by verifying identity based on unique behavioral characteristics. |
| **KBRef-29** | |

| | |
|---|---|
| **IoT Domain** | Healthcare |
| **Architecture** | Edge-Secured Healthcare |
| The proposed SHS architecture explicitly targets improvements in Security (Confidentiality, Privacy, Access Control) and Performance Efficiency (Latency, Time Behaviour, Resource Utilization) compared to traditional cloud-based healthcare systems. It achieves this by introducing a dedicated Edge Computing layer that handles crucial tasks like PPSE encryption and initial processing closer to the user. The architecture also implicitly supports Maintainability through its modular, layered design. Key technologies enabling these qualities include the Edge Computing paradigm itself, the specific PPSE technique, and a detailed, policy-based Access Control mechanism. | |
| **Quality Requirements** | |
| **Security** | Security, particularly Confidentiality and Access Control, is a core focus. Confidentiality and privacy are achieved by encrypting PHI at the Edge layer using a Privacy-Preserving Searchable Encryption (PPSE) technique before storage. This allows searching over encrypted data without full decryption, limiting exposure. Access Control is implemented via a dedicated module on the data server (using components like PEP, PDP, PIP, PAP, AMs – indicative of an Attribute-Based Access Control or similar policy-driven system) to strictly enforce policies and prevent unauthorized users from accessing PHI. |
| **Performance/Efficiency** | Performance Efficiency is a key driver for adopting the edge architecture. By placing an Edge Computing layer closer to the data source (patients/sensors), the system significantly reduces network latency and data transfer times compared to direct cloud interaction, crucial for real-time monitoring and decision-making. This local processing also demonstrably reduces power and energy consumption. |
| **Maintainability** | Maintainability is supported through Modularity. The architecture (Layered) is clearly divided into layers (IoT/Sensor, Edge, Cloud/Server) and functional modules (Encryption, Access Control). Within these modules, distinct components are identified (e.g., Edge Gateway, Edge Server, Database Manager, AC, KG, QP, PEP, PDP). This separation of concerns likely makes the system easier to understand, modify, and test, as changes within one module or layer should have a relatively contained impact. |
| **KBRef-30** | |
| **IoT Domain** | Smart City |
| **Architecture** | SAPPARCHI (Smart City) |
| The Sapparchi architecture explicitly prioritizes Scalability (Flexibility) and considers Maintainability and Security by leveraging microservices, serverless concepts, and platform-provided mechanisms. Reliability, Compatibility, and Portability are implicitly supported through design choices like load balancing, monitoring, message queuing, standardized communication protocols (HTTP, AMQP via RabbitMQ), and containerization (Docker). The use of distinct components and technologies like Nginx, RabbitMQ, Docker, MongoDB, and Redis underpins the achievement of these quality attributes. | |
| **Quality Requirements** | |
| **Flexibility** | Flexibility, particularly the Scalability sub-characteristic, is a primary architectural driver. It's achieved horizontally by dynamically increasing or decreasing instances of serverless Actions and Microservices deployed in Docker containers across distributed worker nodes. This allows the system to adapt its capacity to varying loads. The use of microservices also enhances Modifiability (Maintainability), allowing parts of the system to be changed or adapted more easily. The architecture's design for deployment across Cloud, Fog, and Edge tiers demonstrates Adaptability. |

| | |
|---|---|
| **Performance/Efficiency** | Performance Efficiency, particularly Time Behaviour, is addressed through architectural patterns that promote responsiveness and throughput. Asynchronous request handling via the API Gateway and message queuing (RabbitMQ) prevents blocking and allows the system to handle incoming requests efficiently. Parallel execution of Actions on worker nodes and load balancing via Nginx further enhance throughput and response times underload. Distributing tasks across Cloud, Fog, and Edge tiers implies optimizing Resource Utilization based on computational capacity. |
| **Maintainability** | Maintainability is supported by adopting a microservices architectural style and a component-based design (Manager Service, Executors, Data Service, etc.). This promotes Modularity, allowing components/microservices/actions to be developed, deployed, and updated independently. The granularity of Actions, Microservices, and Services allows for easier modification and evolution (Modifiability). The Monitor Service aids Analysability by providing insights into node execution. |
| **KBRef-31** | |
| **IoT Domain** | Industry 4.0 |
| **Architecture** | Layered |
| It is a layered software architecture for an IoT system designed to monitor environmental conditions in industrial settings. This system utilizes open-source software and hardware components. The architecture is structured into four distinct layers, as shown in a deployment diagram (Figure 2, not provided here but referenced):<br><br>Perception Layer: Contains the sensing elements that gather environmental data.<br><br>Transport Layer: Handles the communication of data from the perception layer upwards.<br><br>Middleware Layer: Processes and potentially aggregates data received from the transport layer. Modifications in the perception layer (e.g., adding sensors) necessitate changes here.<br><br>Application Layer: Provides end-user services and interfaces, consuming data processed by the middleware layer.<br><br>The core design philosophy emphasizes a layered approach with loosely coupled components to manage the complexity arising from diverse hardware and software elements in dynamic industrial environments. | |
| **Quality Requirements** | |
| **Maintainability** | Maintainability is achieved primarily through layered architecture, which promotes loose coupling between components in different layers. This modular design makes it easier to understand and modify (Modifiability) the system, as changes tend to have a contained and predictable impact, propagating logically through the layers. |
| **Security** | Security is addressed by focusing on verifying the identity of devices sending data (Authenticity) and protecting the data during transmission and storage (Confidentiality) via encryption. The architecture relies on the security mechanisms provided by communication protocols (MQTT, CoAP) and requires securing the core infrastructure components. |
| **Reliability** | System Availability (a key aspect of Reliability) is supported by using a modular architecture with clear component relationships. This design, combined with the selection of open-source solutions (potentially offering community support and transparency), aims to simplify maintenance, speed up repairs (Recoverability), and reduce the likelihood or impact of failures (Fault Tolerance), thereby ensuring continuous operation. |

| | |
|---|---|
| **Flexibility** | Scalability is explicitly addressed by designing the system to handle increases in processing load, data storage, and the number of connected devices/users. This is achieved through the inherent modularity of the layered architecture, allowing individual layers or components to be scaled independently. Specific technology choices and component designs across all layers were made with these scalability dimensions in mind. |
| **KBRef-32** | |
| **IoT Domain** | Healthcare |
| **Architecture** | Layered Edge-Fog-Cloud Integrated |
| The architecture consists of several hardware and software components. Hardware includes a Body Area Sensor Network (medical, activity, environment sensors) collecting patient data, Gateway devices (like mobile phones) acting as initial fog nodes to forward data, FogBus Modules (Broker nodes for managing tasks and security, Worker nodes like Raspberry Pis for performing computations), and Cloud Data Centers used for overflow processing or handling large datasets.<br><br>Software components handle the data flow and analysis. Data undergoes filtering and pre-processing (including dimensionality reduction with PCA/SPIHT and encryption with SVD) before being fed into a Deep Learning Module. This module uses trained neural networks for diagnosis (predicting heart disease presence). An Ensembling Module combines predictions from multiple models (using bagging/voting) running on different worker nodes to improve accuracy. A Resource Manager within the Broker node, featuring a workload manager and an arbitration module, handles job queuing and intelligently decides whether tasks should be processed by the Broker itself, a Fog Worker node, or offloaded to the Cloud Data Center, aiming for load balancing and optimal performance.<br><br>The system follows a Master-Slave topology within a Local Area Network (LAN), where the Broker node (Master) controls Worker nodes (Slaves). Communication between edge/fog devices uses FogBus, while interaction with the cloud uses Aneka. Gateways initiate requests to the Broker, which determines the processing location (Broker, Worker, or Cloud via Broker). An Android application serves as the gateway interface, communicating via HTTP REST APIs. The overall goal is to provide a robust, efficient, and accurate diagnostic service by integrating IoT, edge, fog, and cloud computing. | |
| **Quality Requirements** | |
| **Functional Suitability** | Accuracy in diagnosing heart disease is a primary functional goal, achieved by using deep learning models and further enhanced by employing an ensemble method (bagging/voting) that combines results from multiple models to improve the final prediction (Functional Correctness). |
| **Performance Efficiency** | FogBus Framework. The architecture aims for fast processing and low response times by leveraging fog computing (processing closer to the source) and edge resources. The Resource Manager's arbitration module dynamically allocates tasks to Broker, Worker, or Cloud based on load and task requirements to optimize performance. Cloud offloading handles heavy loads quickly, though potentially increasing latency. Real-time analysis is a stated goal (Time Behaviour). |
| **Security** | FogBus Framework. A dedicated Security Management module within the Broker (FogBus Broker) node is responsible for securing communication channels and protecting data integrity against unauthorized access or tampering. Data is also encrypted during pre-processing (using Singular Value Decomposition (SVD) for encryption) (Integrity and Confidentiality). |
| **KBRef-33** | |
| **IoT Domain** | Industry 4.0 |
| **Architecture** | Industrial Internet of Things (IIoT) |

A layered software architecture for Industrial Internet of Things (IIoT) deployment, specifically designed to address the challenges of integrating diverse industrial fieldbuses, handling real-time constraints, and leveraging the benefits of fog/edge computing. The architecture prioritizes modularity, scalability, and interoperability, acknowledging the machine-centric nature of IIoT compared to the consumer-focused IoT.
Key Architectural Features:

Layered Structure: The architecture is organized into four distinct layers:

a. Sensing/Things Layer: This layer encompasses the physical devices, sensors, actuators, and PLCs connected to various industrial fieldbuses (e.g., CANOpen, Modbus, Profibus). It focuses on the hardware aspect of fieldbus connections. Software modules at the upper level handle the configuration of fieldbuses and connected devices.

b. Data Provider Layer: This layer acts as an intermediary, acquiring data from fieldbuses, storing it in buffers, and transmitting it to the fog layer. It also handles sending data back to fieldbuses. The layer includes drivers for each supported fieldbus, abstracting the specific details of each network and creating a unified address space. This contributes significantly to interoperability. Real-time requirements are addressed by leveraging SoCs with specialized co-processors for time-critical communications.

c. Fog/Edge Computing Layer: This is a crucial layer that enables local data processing, analysis, and interaction between "things." It implements a publisher-subscriber paradigm, where "things" (physical and virtual) publish their values, and other "things" subscribe to them. Virtual things can process data from physical things, enabling complex local logic and decision-making. The Data Distribution Service (DDS) middleware for real-time systems is proposed for communication between the things to ensure interoperability between the IIoT systems.

d. Applications/Services Layer: This layer provides the platform for developing industry-specific applications, such as remote monitoring and control (SCADA), HMIs, report generation, and data visualization. These applications can subscribe to data from the fog layer and publish commands back to the industrial environment. Web servers can be hosted on the fog nodes to visualise data and provide remote configuration.

Fog/Edge/Gateway Nodes: The architecture emphasizes the use of fog/edge/gateway nodes, which combine the data provider and fog computing layers. These nodes are implemented on computing systems with sufficient processing power and peripherals to connect to industrial networks. The nodes can connect to multiple fieldbuses, enabling data aggregation and distributed processing close to the source.
Fieldbus Integration: A core goal is to seamlessly integrate various fieldbuses. The data provider layer's drivers and the unified address space abstraction hide the complexities of different fieldbus protocols. The architecture supports "plug and play" integration of fieldbuses through device description languages.

Real-Time Capabilities: The architecture is designed to address the real-time requirements of industrial environments. This is achieved through:

a. SoC-based Implementation: Utilizing SoCs with specialized co-processors for real-time communication with fieldbuses, separating real-time tasks from less critical operations.

b. DDS Middleware: The use of DDS middleware with its quality of service (QoS) levels to guarantee data availability, delivery, and timeliness.

c. Prioritization: Implementing acquisition cycles where data is updated periodically and prioritized based on criticality.
Virtual Environment: The architecture incorporates a virtual environment where "things" (physical and virtual) can interact and exchange data. This allows for simulation, testing, and advanced control strategies.

Security: Security is addressed at multiple layers. Fieldbus security relies on existing mechanisms and restricted physical access. Higher layers implement security measures like encryption of configuration files, restricted remote access, and OS security updates. The DDS protocol with the latest security specifications provides authentication, access control, logging, data tagging, cryptography, and certificates.

| Quality Requirements | |
|---|---|
| **Compatibility** | Data Distribution Service (DDS) with Publish-Subscribe provides the core communication layer. It enables loosely coupled communication (Interoperability) between "things," allowing them to exchange information without direct connections. |
| **Flexibility** | "Things" communicate by publishing data and subscribing to data, rather than being tightly coupled through direct point-to-point connections. This decoupling allows new "things" to be added to the system, or existing "things" to change their data sources, without requiring modifications to other components. The architecture is said to be versatile, being able to be used in a wide range of industrial applications. |
| **Performance/Efficiency** | DDS leverages its Quality of Service (QoS) policies to guarantee specific delivery characteristics for data. These QoS policies allow developers to fine-tune how data is transmitted, ensuring that critical information arrives on time and with the required level of reliability (Time behaviour and Resource Utilization). |
| **Security** | DDS Security: Employs the latest DDS Security Specification, offering comprehensive security features:<br><br>Authentication: Verifying the identity of communicating entities to prevent impersonation.<br><br>Access Control: Enforcing policies to restrict access to data and services based on user roles and permissions.<br>Logging: Auditing system events to detect and investigate security breaches.<br><br>Data Tagging: Assigning security labels to data to control its dissemination and usage.<br><br>Cryptography: Using encryption and digital signatures to protect the confidentiality and integrity of data in transit and at rest.<br><br>Certificates: Leveraging digital certificates for secure authentication and key exchange. |
| **[KBRef-34]** | |
| **IoT Domain** | Healthcare |
| **Architecture** | Secure NDN-Edge Healthcare |
| The SHNIE (Secure Healthcare data communication framework integrating NDN-based IoT with Edge cloud) architecture aims to provide secure and efficient medical data delivery for healthcare IoT systems, specifically addressing latency, cost, and the resource limitations of IoT devices. It employs a hierarchical, three-layer structure (Patient, Edge Cloud, User) integrating IoT device clustering, Edge computing, and Named Data Networking (NDN) principles. | |
| **Quality Requirements** | |
| **Security** | Security is a primary focus, implemented through multiple Named Data Networking (NDN)-adapted strategies. Confidentiality is achieved by using hash names (ciphertexts) within NDN's FIB and PIT tables and transmitting ciphertexts of names, provider IDs, and the medical data itself, preventing disclosure even if traffic is intercepted (addressing eavesdropping).<br>Integrity and Authenticity are addressed using digital signatures on data packets to |

| | |
|---|---|
| | prevent tampering (false data) and ensure data originates from a legitimate source and is retrieved by authorized users (preventing illegal acquisition/spoofing). |
| **Performance/Efficiency** | Performance efficiency is a key goal, addressed by leveraging Named Data Networking (NDN) features and edge computing (Time Behaviour and Resource Utilization).<br><br>Latency and cost are reduced through:<br>1) NDN's in-network caching implemented on edge devices, bringing data closer to users;<br>2) NDN's request aggregation, allowing multiple users requesting the same data to be served via a single retrieval process, reducing redundant traffic and server load;<br>3) Utilizing edge devices for caching overcomes the storage/computation resource limitations of IoT devices. Custom caching and delivery algorithms further optimize this. |

| **[KBRef-35]** | |
|---|---|
| **IoT Domain** | Smart City |
| **Architecture** | Blockchain-enabled |

This proposed architecture aims to enhance security and reduce energy consumption in IoT networks by integrating blockchain technology with Software Defined Networking (SDN) in a clustered structure.

Key Components & Concepts:

Clustered SDN Architecture (SDN Domains): The network is divided into clusters called SDN domains, each managed by an SDN controller acting as a cluster head. This structure improves efficiency in large networks.

Blockchain Integration: SDN controllers are interconnected through a peer-to-peer (P2P) network utilizing blockchain technology for secure communication.

Public Blockchain: A public blockchain connects the SDN controllers (cluster heads). Adding a new SDN controller (and its associated IoT devices) to the network is treated like adding a new block to the chain. This provides a shared history of transactions between controllers. The use of a clustered architecture mitigates the computational power requirements associated with public blockchains. A Proof-of-Work (POW) mechanism is not needed to add new controllers due to SDN controller managing authentication

Private Blockchain: A private blockchain is implemented within each SDN domain, between the SDN controller and its connected IoT devices. This manages transactions and enforces energy efficiency policies for IoT devices. Access to the private blockchain requires invitation and authentication.

Secure Access Control: The public and private blockchains are used to provide secure access control for IoT devices and their data. Each IoT device has a unique public and private key pair after being authenticated by the controller of its initial SDN domain. This key pair is used for secure communications.

IoT Device Migration: IoT devices can migrate between SDN domains to avoid excessive energy consumption or delays. The process involves requesting membership from the new cluster head, which verifies the device's identity using the public blockchain and retrieves the device's public key from its previous cluster head. The transation is registered in the public blockchain

Decentralization and Security: The distributed, P2P nature of the blockchain enhances security by eliminating single points of failure (SPOFs). The architecture creates a secure and comparable design in the proposed architecture. The P2P connection among the IoT devices can be observed inside the SDN domain.

Benefits Highlighted:

Enhanced security for communication between IoT devices.

Reduced energy consumption among IoT devices.

Improved network efficiency through the cluster structure.

Elimination of single points of failure due to the decentralized nature of the blockchain.

Secure access control for IoT devices and data.

In summary, the proposed architecture combines the centralized control and flexibility of SDN with the security and decentralization of blockchain to create a more robust, efficient, and secure IoT network. The use of both public and private blockchains addresses different security and management needs within the network.

This architecture prioritizes security and energy efficiency in IoT environments by leveraging SDN and blockchain technology. It tackles the energy-intensive nature of traditional blockchains by replacing Proof-of-Work (PoW) with a lightweight, distributed trust-based authentication system managed by SDN controllers.

Security is enhanced through:

Distributed Trust: Validating blocks using the SDN controller and its distributed trust algorithm, ensuring data integrity without heavy computation.

Blacklisting: Malicious or selfish nodes are identified and blacklisted in a public blockchain, preventing them from re-registering in other domains.

Public/Private Key Infrastructure: Secure communication is ensured through the use of public and private key pairs for IoT devices.

Energy efficiency is achieved through:

PoW Elimination: Replacing PoW with distributed trust authentication drastically reduces energy consumption associated with block creation.

Energy-Aware Routing: The routing protocol considers the energy levels of IoT devices, optimizing data transfer paths to minimize energy expenditure.

SDN Controller Management: The SDN controller monitors device energy levels and can facilitate device migration to other domains when energy is low, further extending network lifetime.

In essence, the architecture creates a secure IoT environment while significantly reducing energy consumption by replacing computationally expensive tasks with efficient and secure authentication methods and energy-aware network management.

| **Quality Requirements** | |
|---|---|
| **Security** | Distributed Trust: Validating blocks using the SDN controller and its distributed trust algorithm, ensuring data integrity without heavy computation.<br><br>Blacklisting: Malicious or selfish nodes are identified and blacklisted in a public blockchain, preventing them from re-registering in other domains.<br><br>Public/Private Key Infrastructure: Secure communication is ensured through the use of public and private key pairs for IoT devices. |

| | |
|---|---|
| **Performance/Efficiency** | PoW Elimination: Replacing PoW with distributed trust authentication drastically reduces energy consumption associated with block creation.<br><br>Energy-Aware Routing: The routing protocol considers the energy levels of IoT devices, optimizing data transfer paths to minimize energy expenditure.<br><br>SDN Controller Management: The SDN controller monitors device energy levels and can facilitate device migration to other domains when energy is low, further extending network lifetime. |
| **[KBRef-36]** | |
| **IoT Domain** | Generic |
| **Architecture** | Layered Blockchain-Based SDN |

This architecture proposes a secure routing solution for multi-controller SDN-enabled IoT networks, addressing privacy concerns associated with sharing detailed network information. Instead of precise topologies, controllers generate and share abstract topologies, which hide internal network details using virtual links. These abstract topologies are stored securely and immutably on a Blockchain managed via a smart contract.

The architecture consists of four layers: Forwarding, Control, Application, and Blockchain. The Control Layer uses LLDP (with extensions) for link discovery and configures flow tables in the Forwarding Layer. The Application Layer is responsible for generating the abstract topology from the precise view and interacting with the blockchain.

A key feature is the verification process for submitted abstract topologies. When a controller uploads its topology via gRPC to the smart contract, other controllers in the Application Layer validate it by sending ICMP probing packets to check the existence of advertised edge switches/links. These controllers then vote via the smart contract. Only if a majority confirms the topology's correctness is it permanently stored on the blockchain; otherwise, it's discarded.

This validation process feeds into a Reputation Mechanism. Controllers calculate a local reputation for peers based on the correctness of their submissions (using Bayesian estimation). These local reputations are sent to the smart contract, which calculates a global reputation (a weighted mean, considering time decay) stored on the blockchain.

When routing is needed, controllers retrieve the verified abstract topologies and global reputations from the blockchain. They compute paths that are not only shortest (under constraints) but also reliable, avoiding domains managed by controllers with low reputations. This ensures secure and trustworthy routing across multiple SDN domains while preserving network privacy.

| **Quality Requirements** | |
|---|---|
| **Security** | Privacy/Confidentiality is achieved by using abstract topologies instead of precise ones, hiding internal network structure. Integrity of the stored topology information is ensured by the immutable nature of the Blockchain and the |

| | |
|---|---|
| | verification/voting mechanism that filters out mistaken data before final storage. Secure routing and prevention of data leakage/packet loss are achieved by computing paths using only these verified, trustworthy topologies and by avoiding domains with low reputations (indicating potential maliciousness or unreliability). |
| **Reliability** | Reliability is achieved by ensuring the correctness and trustworthiness of the topology information used for routing. A verification process (ICMP probing + majority voting via smart contract) filters out incorrect ("mistaken") topologies, enhancing accuracy. The Reputation Mechanism quantifies the historical reliability of each controller's shared information using Bayesian estimation locally and a weighted, time-decayed global score on the blockchain. Path computation explicitly uses this reputation data to select reliable paths. |
| **[KBRef-37]** | |
| **IoT Domain** | Smart City |
| **Architecture** | IoT Fog Computing Based |

A layered fog computing architecture designed to balance data processing efficiency, low latency, and user privacy in the Internet of Things (IoT).

Key Features of the Architecture:

Fog Node Core Network: Uses a Software Defined Network (SDN) to manage fog nodes, separating control and data planes for flexible resource allocation and network virtualization. This is connected via high-capacity fibers to base stations/routing devices.

Cloud Integration: Fog nodes connect to the cloud, allowing for leveraging cloud computing power when local fog nodes are insufficient. Tradeoff: Cloud processing introduces communication delay.

Privacy-Preserving Data Handling: Addresses privacy concerns by introducing proxy virtual processors (VMs) connected to each type of user's IoT devices. These proxy VMs:

Classify and analyze data before transmitting it further.

Remove personal privacy information before forwarding the data to application VMs.

Provide semantic models to allow application VMs to access needed information without sensitive data.

Dynamic Deployment of Proxy VMs: Proxy VMs can be statically deployed near fog nodes for stationary IoT devices (e.g., smart home sensors). For mobile devices (e.g., smartphones), proxy VMs can be partially static and partially mobile to minimize network load and latency.

Application VM Deployment Schemes: Two options:

Local Deployment: Application VMs deployed within fog nodes process data from local proxy VMs (e.g., parking applications).

Remote Deployment: Application VMs deployed in the cloud process data from proxy VMs across multiple fog nodes when a broader view is needed (e.g., intelligent transportation).

Example Application: Intelligent Video Surveillance: Employs background modeling and convolutional neural networks to detect and analyze behaviors, raising alarms when necessary. This is balanced with privacy protection.

Core Goal:

The primary goal is to provide a flexible and privacy-aware fog computing architecture that can efficiently process IoT data while minimizing latency and protecting user privacy by removing sensitive information at the edge of the network. The SDN controlled network and the dynamic and strategic deployment of virtual processors contribute to this goal.

| **Quality Requirements** | |
|---|---|
| **Flexibility** | Software Defined Networking (SDN):<br><br>Dynamic Resource Allocation: SDN allows for flexible allocation of network resources on demand. As the number of IoT devices or the volume of data increases, the SDN controller can dynamically adjust network bandwidth, routing paths, and other parameters to accommodate the increased load.<br><br>Centralized Control: The centralized controller in SDN simplifies network management and allows for efficient scaling of the network infrastructure.<br><br>Network Virtualization: SDN enables network virtualization, which allows multiple virtual networks to coexist on the same physical infrastructure. This can improve resource utilization and scalability (Flexibility). |
| **Performance/Efficiency** | The architecture optimizes performance by distributing processing resources closer to the data source (Resource Utilization), employing high-speed network connections, utilizing SDN for intelligent network management, performing data pre-processing at the edge, and strategically deploying application VMs based on the application's latency requirements. |
| **Security** | Proxy Virtual Processors (VMs) are a core component of this architecture, designed to address privacy (Confidentiality) concerns and improve efficiency in IoT data processing.<br><br>Privacy Protection: Their primary goal is to remove personally identifiable information (PII) from data before it's transmitted to application VMs. This is crucial for protecting user privacy in a world where IoT devices are constantly collecting personal data.<br><br>Data Classification and Analysis: They classify incoming data from IoT devices based on its type and perform initial analysis or pre-processing. This reduces the burden on application VMs and allows them to focus on specific tasks. |

| | |
|---|---|
| | Semantic Modeling: They provide semantic models that allow application VMs to access the information they need without exposing them to raw, sensitive data. In essence, they translate raw data into a more abstract and privacy-preserving representation. |

# Appendix B - Extraction Data References (Knowledge Base References)

[KBRef-01] AHMED, I., ZHANG, Y., JEON, G., *et al.*, 2022, "A blockchain- and artificial intelligence-enabled smart IoT framework for sustainable city", *International Journal of Intelligent Systems*, v. 37, n. 9, pp. 6493–6507.

[KBRef-02] ALI, F., EL–SAPPAGH, S., ISLAM, S. M. R., *et al.*, 2021, "An intelligent healthcare monitoring framework using wearable sensors and social networking data", *Future Generation Computer Systems*, v. 114, pp. 23–43.

[KBRef-03] ASGHAR, A., ABBAS, A., KHATTAK, H. A., *et al.*, 2021, "FOG based architecture and load balancing methodology for health monitoring systems", *IEEE Access*, v. 9, pp. 96189–96200. Available at: https://doi.org/10.1109/access.2021.3094033.

[KBRef-04] AWAISI, K. S., HUSSAIN, S., AHMED, M., *et al.*, 2020, "Leveraging IoT and fog computing in healthcare systems", *IEEE Internet of Things Magazine*, v. 3, n. 2, pp. 52–56. Available at: https://doi.org/10.1109/iotm.0001.1900096.

[KBRef-05] BARBOSA, P., FIGUEIREDO, A., SOUTO, S., *et al.*, 2020, "An Open Source Software Architecture and Ready-To-Use Components for Health IoT". In: *Proceedings of the Institute of Electrical and Electronics Engineers Inc.*, pp. 374–379. Available at: https://doi.org/10.1109/cbms49503.2020.00077.

[KBRef-06] BERÁNEK, M., FEUERLICHT, G., KUČERA, O., *et al.*, 2023, "An Architecture for a Large-Scale IoT e-Mobility Solution". In: *Proceedings of Science and Technology Publications, Lda*, pp. 733–740. Available at: https://doi.org/10.5220/0011827100003467.

[KBRef-07] BHAYO, J., HAMEED, S., SHAH, S. A., 2020, "An efficient Counter-Based DDOS attack detection framework leveraging software defined IoT (SD-IoT)", *IEEE Access*, v. 8, pp. 221612–221631. Available at: https://doi.org/10.1109/access.2020.3043082.

[KBRef-08] BICER, C., MURTURI, I., DONTA, P. K., *et al.*, 2023, "Blockchain-Based zero trust on the edge". In: *Proceedings of the 2021 International Conference on Computational Science and Computational Intelligence (CSCI)*, pp. 1006–1013. Available at: https://doi.org/10.1109/csci62032.2023.00167.

[KBRef-09] CHUDHARY, R., SHARMA, S., 2021, "Fog-cloud assisted framework for Heterogeneous Internet of Healthcare Things", *Procedia Computer Science*, v. 184, pp. 194–201.

[KBRef-10] DA SILVEIRA, J. D. F., DA S VELOSO, A. F., REIS, J. V. D., *et al.*, 2021, "A new Low-Cost LORAWAN power switch for smart farm applications". In: *Proceedings of the 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*.

[KBRef-11] DINEVA, K., ATANASOVA, T., 2021, "Design of scalable IoT architecture based on AWS for smart livestock", *Animals*, v. 11, n. 9, p. 2697.

[KBRef-12] EL-BASIONI, B. M. M., EL-KADER, S. M. A., 2020, "Laying the foundations for an IoT reference architecture for the agricultural application domain", *IEEE Access*, v. 8, pp. 190194–190230.

[KBRef-13] FAID, A., SADIK, M., SABIR, E., 2021, "An agile AI and IoT-Augmented Smart Farming: a Cost-Effective Cognitive Weather Station", *Agriculture*, v. 12, n. 1, p. 35. Available at: https://doi.org/10.3390/agriculture12010035.

[KBRef-14] GUPTA, H., NAYAK, A., 2024, "Publish Subscribe System Security Requirement: A case study for V2V communication", *IEEE Open Journal of the Computer Society*, v. 5, pp. 389–405. Available at: https://doi.org/10.1109/ojcs.2024.3442921.

[KBRef-15] HAJVALI, M., ADABI, S., REZAEE, A., *et al.*, 2021, "Software architecture for IoT-based healthcare systems with cloud/fog service model", *Cluster Computing*, v. 25, n. 1, pp. 91–118.

[KBRef-16] HATI, A. J., SINGH, R. R., 2021, "Smart Indoor Farms: Leveraging technological advancements to power a sustainable agricultural revolution", *AgriEngineering*, v. 3, n. 4, pp. 728–767.

[KBRef-17] JAVED, A., MALHI, A., FRÄMLING, K., 2020, "Edge Computing-based Fault-Tolerant Framework: A Case Study on Vehicular Networks". In: *Proceedings of the International Wireless Communications and Mobile Computing (IWCMC)*.

[KBRef-18] KAUR, A., SAHIL, S., SOOD, S. K., 2022, "Cloud-FOG assisted Energy Efficient Architectural Paradigm for disaster evacuation", *Information Systems*, v. 107, p. 101732.

[KBRef-19] LI, Z., CAO, J., LIU, X., *et al.*, 2020, "A Self-Adaptive Bluetooth Indoor Localization System using LSTM-based Distance Estimator". In: *Proceedings of the Institute of Electrical and Electronics Engineers Inc.*, pp. 1–9. Available at: https://doi.org/10.1109/icccn49398.2020.9209674.

[KBRef-20] MEISENBACHER, S., SCHWENK, K., GALENZOWSKI, J., *et al.*, 2021, "A lightweight user interface for smart charging of electric vehicles: a Real-World application". In: *Proceedings of the 2021 9th International Conference on Smart Grid and Clean Energy Technologies (ICSGCE)*, pp. 57–61. Available at: https://doi.org/10.1109/icsgce52779.2021.9621604.

[KBRef-21] MENDES, L. F., AGUILAR, P. A. C., BEZERRA, C. I. M., 2023, "Software Architecture for IoT-based Indoor Positioning Systems for Ambient Assisted Living". In: *Proceedings of the Institute of Electrical and Electronics Engineers Inc.*, pp. 93–104. Available at: https://doi.org/10.1109/icsa56044.2023.00017.

[KBRef-22] MPEIS, P., ROUSSEL, T., KUMAR, M., *et al.*, 2020, "The Anyplace 4.0 IoT Localization Architecture". In: *Proceedings of the Institute of Electrical and Electronics Engineers Inc.*, pp. 218–225. Available at: https://doi.org/10.1109/mdm48529.2020.00045.

[KBRef-23] PEREIRA, J., BATISTA, T., CAVALCANTE, E., *et al.*, 2022, "A platform for integrating heterogeneous data and developing smart city applications", *Future Generation Computer Systems*, v. 128, pp. 552–566.

[KBRef-24] RAHMAN, A., ISLAM, MD. J., RAHMAN, Z., *et al.*, 2020, "DISTB-ConDo: Distributed Blockchain-Based IoT-SDN model for smart condominium", *IEEE Access*, v. 8, pp. 209594–209609.

[KBRef-25] RAHMANI, A. M., BABAEI, Z., SOURI, A., 2020, "Event-driven IoT architecture for data analysis of reliable healthcare applications using complex event processing", *Cluster Computing*, v. 24, n. 2, pp. 1347–1360.

[KBRef-26] RAMALHO, M. S., ROSSETTI, R. J. F., CACHO, N., *et al.*, 2020, "SmartGC: a software architecture for garbage collection in smart cities", *International Journal of Bio-inspired Computation*, v. 16, n. 2, p. 79.

[KBRef-27] SAFDAR, Z., FARID, S., QADIR, M., *et al.*, 2020, "A novel architecture for the Internet of Things based E-Health systems", *Journal of Medical Imaging and Health Informatics*, v. 10, n. 10, pp. 2378–2388.

[KBRef-28] SHAKIL, K. A., ZAREEN, F. J., ALAM, M., *et al.*, 2020, "BAMHealthCloud: A biometric authentication and data management system for healthcare data in the cloud", *Journal of King Saud University - Computer and Information Sciences*, v. 32, n. 1, pp. 57–64.

[KBRef-29] SINGH, A., CHATTERJEE, K., 2021, "Securing smart healthcare system with edge computing", *Computers & Security*, v. 108, p. 102353. Available at: https://doi.org/10.1016/j.cose.2021.102353.

[KBRef-30] SOUZA, A., CACHO, N., BATISTA, T., *et al.*, 2022, "SAPPARCHI: an Osmotic Platform to Execute Scalable Applications on Smart City Environments". In: *Proceedings of the IEEE 15th International Conference on Cloud Computing (CLOUD)*.

[KBRef-31] STOJANOV, Ž., DOBRILOVIĆ, D., 2021, "Software architecture quality attributes of a layered sensor-based IoT system (short paper)". In: *Proceedings of the Workshop Information Technologies: Algorithms, Models, Systems (ITAMS)*, pp. 66–74.

[KBRef-32] TULI, S., BASUMATARY, N., GILL, S. S., *et al.*, 2020, "HealthFog: An ensemble deep learning-based Smart Healthcare System for Automatic Diagnosis of Heart Diseases in integrated IoT and fog computing environments", *Future Generation Computer Systems*, v. 104, pp. 187–200.

[KBRef-33] UNGUREAN, I., GAITAN, N. C., 2020, "A software architecture for the Industrial Internet of Things - A Conceptual model", *Sensors*, v. 20, n. 19, p. 5603.

[KBRef-34] WANG, X., CAI, S., 2020, "Secure healthcare monitoring framework integrating NDN-based IoT with edge cloud", *Future Generation Computer Systems*, v. 112, pp. 320–329.

[KBRef-35] YAZDINEJAD, A., PARIZI, R. M., DEHGHANTANHA, A., *et al.*, 2020, "An Energy-Efficient SDN controller architecture for IoT networks with Blockchain-Based security", *IEEE Transactions on Services Computing*, v. 13, n. 4, pp. 625–638.

[KBRef-36] ZENG, Z., ZHANG, X., XIA, Z., 2022, "Intelligent Blockchain-Based secure routing for multidomain SDN-Enabled IoT networks", *Wireless Communications and Mobile Computing*, v. 2022, pp. 1–10.

[KBRef-37] ZHANG, C., 2020, "Design and application of fog computing and Internet of Things service platform for smart city", *Future Generation Computer Systems*, v. 112, pp. 630–640.

# Appendix C – Feasibility Study Protocol

**UNIVERSIDADE FEDERAL DO RIO DE JANEIRO**

**PROGRAMA DE ENGENHARIA DE SISTEMAS DA COMPUTAÇÃO**

**ENGENHARIA DE SOFTWARE EXPERIMENTAL**

## 1. IDENTIFICATION

**Title**: ArchIoTect

**Technical Area:** Software Engineering

**Authors**: Fernando Novaes Ribeiro da Silva, Bruno Pedraça de Souza, and Guilherme Horta Travassos

**Affiliation**: COPPE/UFRJ

**Local**: Rio de Janeiro

**Date**: July 14 to 25

## 2. CHARACTERIZATION

An applied feasibility study will be carried out in a tool to support decision making on the architecture of an IoT software system.

**Type**: Feasibility study.

**Domain**: Experimental Software Engineering – Software System Architecture.

**Language**: Portuguese (Brazilian).

**Partners**: Federal University of Rio de Janeiro – PESC/COPPE.

**Expected Execution**: Second Half of 2025

**Glossary of Terms:**

Software Engineering (ES);

Internet of Things (IoT);

## 3. INTRODUCTION

Software engineering faces a growing challenge with the rise of Internet of Things (IoT) systems. These new systems, characterized by their autonomy and complex interaction between software, hardware, and the physical world, make the early stages of development, such as requirements analysis and architecture definition, more critical than ever. An inadequate architectural decision in this heterogeneous environment can lead to catastrophic failures.

To mitigate these risks and provide structured support to professionals, the ArchIoTect tool was designed to guide decision-making about the architecture of IoT software systems. The feasibility of this proposal will be investigated through a study using the

Technology Acceptance Model (TAM). Proposed by Davis (1989), the TAM model will allow us to assess whether ArchIoTect is perceived as useful and easy to use, which are determining factors for its acceptance and practical success.

## 4. DEFINITION OF THE EXPERIMENTAL STUDY

**Object of Study:** ArchIoTect.

**Global Objective:** This work aims to conduct a feasibility study with graduate students in software engineering and professionals in the field to verify whether the ArchIoTect tool is capable of supporting architectural decision-making in IoT systems.

**Specific Objectives**

- **Analyze**: ArchIoTect
- **for the purpose of**: characterizing
- **with respect to**: the feasibility of ArchIoTect, observed in terms of being considered useful, easy to use, and feasible.
- **From the point of view of**: of researchers in ES.
- **In the Context of**: Use of the ArchIoTect tool to solve a proposed IoT scenario for architectural decision with graduate students in software engineering and professionals in the area.

**Questions and Metrics**

**Is the ArchIoTect tool interface intuitive and easy to use?**

*This question aims to verify, in the context of designing an IoT software system architecture, if the tool is easy to use.*

**Am I satisfied with the overall quality of the recommendations provided by the ArchIoTect tool?**

*This question aims to verify, in the context of designing an IoT software system architecture, if the recommendations based on user needs are reliable.*

**Did the ArchIoTect tool meet my expectations for assisting with architectural decisions?**

*This question aims to verify, in the context of designing an IoT software system architecture, if the tool is considered useful for making architectural decisions.*

**Should the ArchIoTect tool be recommended to other professionals who work with IoT software system architectures?**

*This question aims to verify, in the context of designing an IoT software system architecture, if the use of the tool is viable for professionals in the field.*

## 5. PLANNING

**Variable Selection**

**Dependent variables:** user experience.

**Independent variables:** area of expertise and experience of the participants in the area.

We did not define a hypothesis, since this study had a small sample. Therefore, the application of statistical tests would not be adequate.

**Selection of Participants**

**Participant Selection Criteria**: convenience sample.

**Required Experience**: students are taking the Software Engineering course or are professionals in the technology area.

**Probabilistic Sampling Techniques**: not applicable.

**Non-Probabilistic Sampling Techniques**: not applicable.

**Resources:**

**Software**: ArchIoTect.

**Hardware**: Computer with internet access.

**Questionnaires**: a questionnaire for qualitative data collection (after having used the tool), a term of agreement in study participation, and characterization of the participants.

**Experiment Design**

**Objects: scenario-based architectural decision.**

**Measurements**: user experience.

- **Techniques**: Use of the tool for architectural decision making.

**Instrumentation**

**Description of Instrumentation**: In this part of the research, the questions were prepared for the participants to answer the questionnaire. The following form will be used to obtain/collect data: a questionnaire after using the tool.

**Support for Quantitative Analysis**: will not be necessary.

**Support for Qualitative Analysis**: ad hoc analysis.

**Observation Criteria**: not applicable.

**Artifacts (Questionnaires, Procedures, etc.):** questionnaire.

**Analysis Engines**

**Criteria for Elimination of *Outliers:*** not applicable.

**6. TRAINING**

**Definition of Training and Procedures**

**Applicators**: by the researcher.

**Participants**: graduate students and IT professionals.

**Procedures**: classes on IoT, software engineering, explanatory video of the use of the tool, and user manual.

**7. ENFORCEMENT PROCEDURES**

**Definition of Execution of the Experimental Study:** A feasibility study will be carried out with graduate students of software engineering courses. An explanatory video is delivered to prepare users for the tool's use. After that, the evaluation questionnaire is

made available with three scenarios for architectural decision making, and after using the tool based on the chosen scenario, the questionnaire is filled out.

**Artifacts (Instructions, Documents, etc):**

**Informed Consent Form**: for the support and anonymity of students.

**Characterization of participants**: to capture the professional experience of students.

**Questionnaire**: to collect feedback from participants after using the ArchIoTect tool.

**8. EVALUATION OF THE PLAN**

**Objectives**: reviewed by authors.

**Participants**: researchers.

**Execution Procedures:** 15 days (following this execution plan);

**Artifacts Used**: ArchIoTect tool.

**Artifacts Generated (Lessons Learned, Suggestions for Modification of the Plan):** questionnaire of participants' perception after using the tool;

**9. COST PLANNING**

**Costs of the Experimental Study**

**Planning Costs:** not applicable.

**Plan itself:** not applicable.

**Instrumentation**: not applicable.

**Training Material**: explanatory video.

**Plan Evaluation**: not applicable.

**Execution Costs**

**Displacements**: not applicable.

**Training**: explanatory video of the use of the tool.

**Human Resources**: graduate students and researchers.

**Material Resources**: forms, computers, software.

**Analysis Costs**: not applicable.

**Packaging Costs**: not applicable.

# Appendix D – Term of Consent

Dear Sir/Madam,

I declare that I am over 18 years of age and I agree to participate in this study conducted by the Software Engineering team at the Systems and Computer Engineering Program (PESC/COPPE). This study aims to analyze the use of a computational tool to support decision-making in defining software architectures specific to IoT software systems. The results of the study will contribute to understanding how to improve the quality of IoT software systems, as well as the technologies, specifically software architecture, that can be utilized. Your participation is not mandatory; however, if you wish to participate, please read the terms available in this document and express your agreement.

**1) Procedure**

A computational support tool will be used to assist in architectural decision-making for IoT software systems. The focus of observation is the computational tool, not the participant. The participant will never be identified in the data. After collecting the study data, any reference to the participant will be removed and will not be used at any point during the analysis or presentation of the results. Therefore, your agreement to participate in this study implies your permission for the researchers to use your profile characterization data, the study results, and your responses to questions about the computational support tool.

**2) Handling of Potential Risks and Discomforts**

During data collection, your privacy and anonymity are guaranteed. The data collected during this study is intended strictly for research activities related to the techniques being studied. The entire procedure is in full compliance with the Lei *Geral de Proteção dos Dados* (LGPD).

**3) Benefits and Costs**

This study will contribute important results to research in the general areas of the Internet of Things (IoT), Software Engineering, and Software Architecture. You will not incur any expense or burden from your participation in the study, nor

will you receive any kind of reimbursement or gratification for your participation, other than the knowledge acquired in the application of the tool and the specific software architectures of IoT software systems.

**4) Confidentiality of the Research**

All information collected in this study is confidential and anonymous, except in cases where explicit authorization is requested for such a purpose.

**5) Participation**

I understand that I am participating voluntarily, solely to contribute to the advancement and development of software technology. You have the right to decline participation or to withdraw from this study at any time, without penalty.

By proceeding with this form, you are agreeing to the presented consent form and agree to participate voluntarily in this research.

The researchers responsible for the study can provide any clarification about it, as well as answer any questions. Please contact them at the following emails:

**Researchers:**

Fernando Novaes Ribeiro da Silva (COPPE/UFRJ): *fernandonrs@cos.ufrj.br*

Bruno Pedraça de Souza (COPPE/UFRJ): *bpsouza@cos.ufrj.br*

**Advisor:**

Guilherme Horta Travassos (COPPE/UFRJ): *ght@cos.ufrj.br*

**6) Declaration of Consent**

Do you agree to participate in the evaluation and accept the terms cited above?

☐ Yes

# Appendix E – Participant Profile

Dear Participant,

This form is designed to assess your level of familiarity with various aspects of software architecture and IoT.

All data collected will be handled with complete anonymity, and your responses cannot be used to identify you.

**1 - What is your area of expertise in Software Projects?**

( ) Software Architect

( ) Software Engineer

( ) Team Leader

( ) Infrastructure

( ) Others

**2 - How many years of experience do you have in this role?**

**3 - Knowledge of IoT software systems. Regarding your level of knowledge about IoT software systems, please mark the alternative that best applies to your answer.**

( ) None

( ) I have a basic understanding of IoT software systems from readings and lectures

( ) I have studied IoT software systems in formal courses.

( ) I have hands-on experience with IoT software systems through projects in university and/or industry.

**4 - Knowledge of software architecture. In relation to your level of knowledge about software architecture, mark the alternative that best**

| | | |
|---|---|---|
| Completely unknown | ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5 ( ) 6 ( ) 7 ( ) 8 ( ) 9 ( ) 10 | I am an expert |

**applies to your answer.**

# Appendix F – Post-Inspection Questionnaire

### Post-Inspection Questionnaire of the ArchIoTect Application

**Using the tool: ArchIoTect**

A new software system for the Internet of Things (IoT) needs to be designed, and you, as a **software architect or engineer**, are responsible.

Every decision you make now — about the communication protocol, the security standard, the data architecture — will have a lasting impact on the project's performance, cost, and success. Information is scattered, technology evolves rapidly, and the pressure to make an appropriate choice for the software system's architecture is immense.

It is at this point that the tool should be used as a specialized assistant that can help, or not, in decision-making.

**Your task is to use the tool to identify an architectural solution for an IoT software system.** Therefore, choose 1 (one) of the scenarios you feel most comfortable with below and provide a possible architectural solution for the IoT software system.

**Scenarios:**

***Scenario 1 - "Greenfield" Project - Smart Cities***

**Context**: You have been tasked with initiating a new project related to an energy consumption monitoring system for smart homes ("Smart City Energy Monitoring"). The initial requirements are vague, but management expects the system to be scalable (Flexibility) (to support thousands of homes in the future) and reliable (Reliability). You need to propose an initial architecture and don't know where to start.

***Scenario 2 - "SmartTruck" Project - Logistics Fleets***

**Context**: Your team is developing a logistics fleet management system. They need real-time communication (Performance/Efficiency) to track vehicles. A senior developer suggested using WebSockets, as the team already has

experience with it. Another suggested MQTT, which they heard is the "standard for IoT." You need to make a decision based on evidence and justify the choice to the team and the project manager.

***Scenario 3 - "Wearable" Project - Healthcare***

**Context**: You are a project manager with a technical background. Your team is developing a wearable device for health monitoring. Data privacy (Security) is the highest legal and ethical priority, followed by long-term system maintainability (Maintainability). The team is undecided between a Monolithic Layered architecture and a Microservices architecture.

**Important Notice:**

Please Read Before Proceeding

At this moment, your task is to use the ArchIoTect tool to solve the project scenario you have chosen.

We ask that you do not proceed to the evaluation form until you have fully completed your analysis and made an architectural decision using the tool.

Your evaluation should reflect the complete usage experience, from initial exploration to the formulation of a solution.

Only after completing your architectural project task, proceed to the questions.

We appreciate your attention to this crucial detail for the validity of our research.

**Beginning of the Questionnaire to Evaluate the Use of ArchIoTect: A Tool to Support Decision-Making on the Architecture of an IoT Software System**

**Instructions:**

Answer each question in the following sections by selecting an option from 0 (Strongly Disagree) to 10 (Strongly Agree).

**Click Next for us to begin.**

Before we proceed, please tell us which scenario you used for the tool evaluation:

( ) Scenario 1 - "Greenfield" Project - Smart Cities

( ) Scenario 2 - "SmartTruck" Project - Logistics Fleets

( ) Scenario 3 - "Wearable" Project – Healthcare

**1 - Comparative Evaluation: Knowledge Base vs. AI Assistant**

Query modes for finding an architectural solution:

- **Static Mode:** Navigate through the hierarchical knowledge base (Knowledge Base menu).
- **Dynamic Mode:** Interact with the AI assistant (AI Assistant menu).

Based on your experience, which of the modes proved to be more *effective* and *efficient* in supporting your decision-making?

( ) Hierarchical Knowledge Base

( ) AI Assistant

( ) One mode complements the other

**2 - Justify your previous answer, considering factors such as time, accuracy, and confidence in the result.**

**User Satisfaction:**

**User satisfaction** is one of the main indicators of **efficiency/effectiveness** in software, as it reflects how well the system meets the expectations and needs of the end user. This assessment considers **Ease of Use (Usability), Suitability to User Needs,** and **Overall User Experience (UX)**.

**4 – The ArchIoTect tool's interface is intuitive and easy to use.**

| Completely Disagree | ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5( ) 6 ( ) 7 ( ) 8 ( ) 9 ( ) 10 | Completely Agree |
|---|---|---|

**5 – I am satisfied with the overall quality of the recommendations provided by the tool ArchIoTect.**

| Completely Disagree | ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5( ) 6 ( ) 7 ( ) 8 ( ) 9 ( ) 10 | Completely Agree |
|---|---|---|

**6 - The tool ArchIoTect met my expectations for helping with architectural decisions.**

| Completely Disagree | ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5( ) 6 ( ) 7 ( ) 8 ( ) 9 ( ) 10 | Completely Agree |
|---|---|---|

**7 - This tool should be recommended to other professionals working with IoT software system architectures.**

| Completely Disagree | ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5( ) 6 ( ) 7 ( ) 8 ( ) 9 ( ) 10 | Completely Agree |
|---|---|---|

**Open Questions:**

**8 - What would you suggest to make the tool more useful in the context of IoT systems?**

**9 - Was there any aspect in which the tool did not meet your needs? If so, please explain.**