



University of  
Zurich<sup>UZH</sup>

# **EvoStar: A Modular Integration Platform for a Smart Building Privacy Analysis Framework**

*Markus Senn  
Zurich, Switzerland  
Student ID: 18-755-447*

Supervisor: Katharina O.E. Mueller, Daria Schumm, Prof. Dr.  
Burkhard Stiller


Date of Submission: May 20, 2025



# Declaration of Independence

I hereby declare that I have composed this work independently and without the use of any aids other than those declared (including generative AI such as ChatGPT). I am aware that I take full responsibility for the scientific character of the submitted text myself, even if AI aids were used and declared (after written confirmation by the supervising professor). All passages taken verbatim or in sense from published or unpublished writings are identified as such. The work has not yet been submitted in the same or similar form or in excerpts as part of another examination.

Zürich,



---

Signature of student



# Abstract

Bluetooth Low Energy (BLE) hat eine Vielzahl neuer Technologien hervorgebracht. In jeder neuen Nische wie dieser, muss geprüft werden, ob diese neue Technologie missbraucht werden kann und, falls ja, wie dies verhindert werden kann. Gegenwärtig sind die Gegenmassnahmen und Rahmenbedingungen, mit denen diese Bedingungen zuverlässig überprüft werden können, noch nicht vollständig verfügbar. In Anbetracht dessen wird in dieser Arbeit versucht, mehrere kleinere Programme zur Sicherheitsevaluierung und -entdeckung miteinander zu verbinden, um die Vorteile dieser Programme zu nutzen und eine erweiterbare Plattform zur Sicherheitsevaluierung zu schaffen. Ziel ist es, eine Plattform zu schaffen, die von den Kombinationen der Dateneingaben aus den verschiedenen Programmen profitiert und diese auf sinnvolle Weise nutzt. Die Plattform wird anhand der Standards der Einzelanwendungen und einer voraussichtlichen Nichtverschlechterung der Datenverarbeitung bei gleichzeitiger Ausführung mehrerer Anwendungen bewertet.

Bluetooth Low Energy (BLE) enabled a wide range of new technologies to emerge. In any new niche like this, there exists a need to check if this new technology can be abused and, if yes, how it can be prevented. Currently, the countermeasures and frameworks to verify these conditions reliably, are not yet fully available. Considering this, this work tries to interconnect multiple smaller security evaluation and discovery programs to take advantage of them and create an extensible security evaluation platform. The goal is to have a platform that profits off of combinations of data inputs from the different programs and utilizes them in a meaningful way. The platform will be evaluated by the metrics of the standalone applications and by a projected non-degradation of data handling when running multiple applications at once.

# Acknowledgments

I would like to express my gratitude to my supervisor, Katharina O. E. Mueller, for her support and guidance throughout the development of this thesis. Her insightful feedback, openness to discussion, and continued encouragement made this experience both enriching and rewarding. I am also thankful for the opportunity to work within the Communication Systems Research Group (CSG) at the Department of Informatics, University of Zurich. Special thanks go to my study friends, who always encouraged me to keep going and were always good for a much needed laugh.





# Contents

<b>Declaration of Independence</b>	<b>i</b>
<b>Abstract</b>	<b>iii</b>
<b>Acknowledgments</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Thesis Goals . . . . .	2
1.3 Methodology . . . . .	2
1.4 Thesis Outline . . . . .	3
<b>2 Fundamentals</b>	<b>5</b>
2.1 Background . . . . .	5
2.1.1 Bluetooth Low Energy . . . . .	6
2.1.2 UWB . . . . .	7
2.1.3 TCP / UDP . . . . .	9
2.1.4 Privacy Requirements . . . . .	11
2.2 Related Work . . . . .	14
2.2.1 Privacy Evaluation Frameworks . . . . .	14
2.2.2 Specialized Tools and NLP . . . . .	16
2.2.3 Research Gap . . . . .	17

<b>3</b>	<b>Design</b>	<b>19</b>
3.1	Requirements Engineering . . . . .	19
3.2	Relevant Data Formats and Technologies . . . . .	20
3.2.1	Privacy Requirements . . . . .	21
3.2.2	HomeScout . . . . .	21
3.2.3	ZigBee Packet Sniffer . . . . .	22
3.2.4	IoT Network Topology . . . . .	22
3.2.5	High-traffic Dataset Generation . . . . .	22
3.3	Platform Specifics Choice . . . . .	22
3.3.1	Web-based vs Desktop Application . . . . .	23
3.3.2	Language Selection . . . . .	23
3.3.3	Similar existing Solutions . . . . .	24
3.4	Projected Class Diagram . . . . .	25
3.5	Test Suite . . . . .	26
<b>4</b>	<b>Implementation</b>	<b>27</b>
4.1	Compose . . . . .	27
4.2	Packaging . . . . .	28
4.2.1	Windows . . . . .	29
4.2.2	MacOS . . . . .	30
4.2.3	Linux . . . . .	30
4.3	Containerization . . . . .	31
4.4	Core Components . . . . .	33
4.4.1	Views . . . . .	35
4.4.2	Lifecycle Manager . . . . .	36
4.4.3	Data Integration Hub . . . . .	37
4.4.4	Container Manager . . . . .	38
4.4.5	Container . . . . .	38

<i>CONTENTS</i>	ix
4.4.6 Plugin Manager . . . . .	39
4.4.7 PodmanService . . . . .	41
4.4.8 Adjusted Projects . . . . .	41
4.5 Testing . . . . .	43
4.5.1 UI . . . . .	43
4.5.2 Logic . . . . .	43
<b>5 Evaluation</b>	<b>45</b>
5.1 Benchmarks . . . . .	45
5.2 Comparison . . . . .	46
5.3 Analysis . . . . .	47
<b>6 Final Considerations</b>	<b>51</b>
6.1 Summary . . . . .	51
6.2 Challenges and Accomplishments . . . . .	51
6.3 Conclusions . . . . .	52
6.4 Future Work . . . . .	53
<b>Bibliography</b>	<b>55</b>
<b>Abbreviations</b>	<b>61</b>
<b>List of Figures</b>	<b>62</b>
<b>List of Tables</b>	<b>63</b>
<b>List of Listings</b>	<b>65</b>

**A Contents of the Repository 69**

A.1 README . . . . . 69

A.2 Source code . . . . . 69

A.3 Evaluations . . . . . 69

    A.3.1 File . . . . . 69

    A.3.2 Results raw . . . . . 70

    A.3.3 Results evaluated . . . . . 71

A.4 Instructions . . . . . 71

# Chapter 1

## Introduction

Since the official integration of Bluetooth Low Energy (BLE) into the Bluetooth 4.0 standard at the end of 2009, it has been rapidly growing as a staple technology for close proximity, low energy consumption solutions. Bluetooth is one of the fastest growing tech markets and it is estimated that, by the end of 2028, there will be 7.5 billion devices shipped annually[1]. BLE has had a major impact on several industries like healthcare, where it enabled the development of remote patient monitoring systems and wearable medical devices that continuously track vital signs with minimal power usage. In industrial automation it facilitates asset tracking and process monitoring that allow companies to optimize their operations on a scale that has not been seen before[2].

The low-energy demand and cost-effectiveness has been especially useful for the Internet of Things (IoT) sphere, like smart homes and tracking devices [3]. From smart locks and lighting to thermostats and security systems, BLE facilitates seamless communication between devices, creating interconnected ecosystems that can be managed through smartphones alone. It is estimated that by 2029, household penetration from smart home devices in the United States will be 99% [4].

Along with these advances came several security concerns that are tied to the wireless nature of BLE communication. This makes it susceptible to various attacks like eavesdropping, man-in-the-middle attacks and unauthorized access, if proper security measures are not implemented[5]. Bluetooth as a technology is constantly evolving and tries to respond to these vulnerabilities but when one loophole is closed, another may emerge as a result of newly added capabilities [6]. Current gaps include the need for standardized security protocols that can be universally adopted across different manufacturers and device types. Additionally, there is an ongoing challenge in balancing the trade-off between energy efficiency and strong encryption measures [7] [8].

### 1.1 Motivation

In accordance with the needs of the Communications Systems Group (CSG) to further their research in this field, this thesis aims to consolidate 5 existing projects into one

usable application. The goal is to effectively utilize their respective resources and outputs to analyze compliance with privacy requirements.

These 5 projects entail the following topics:

- BLE tracking device detection for Android [9]
- ZigBee packet sniffing and security analysis [10]
- IoT network topology detection and visualization [11]
- AirTag Privacy Concern analysis [12]
- Dataset generation for high-traffic environment personal tracker identification [13]

Given the scope of a regular bachelor's thesis, there is not enough time to fully adapt to all capabilities of all projects, especially real-time analysis. Thus, this work will limit itself to offline data that was gathered beforehand first and aim for real-time analysis functionality if there is sufficient time. There are more than the 5 mentioned projects that have been developed at CSG, but due to the aforementioned time constraint, this application will not be able to integrate all of them yet. Instead, the focus lies on building a platform that is easily extensible to allow further development with this technology.

## 1.2 Thesis Goals

This thesis aims to build the foundation for a platform that will draw out the potential of each project involved and enhance the products by setting their results into a predefined context given by the privacy concern project. The underlying projects were not created with this kind of streamlining in mind and thus need to be adapted for their new purpose. A broader understanding of the whole field is necessary to determine the correct data types to use and what is to be achieved with them.

Ensuring a smooth execution of the projects as a standalone on the application, but also while running in parallel is the baseline for this thesis' benchmark.

## 1.3 Methodology

The application is developed in increments. Starting with a skeleton that builds the basis for hosting the different projects, the number of projects on the platform is gradually increased while ensuring that the performance is stable.

## 1.4 Thesis Outline

Chapter 2 provides the necessary background knowledge for the underlying projects and the security aspect. Some context to related works will be given.

Chapter 3 will feature the requirements engineering artifacts necessary for this application and the resulting design decisions that will have been made, according to the prevalent standards.

Chapter 4 follows the implementation part where the development of the application prototype is executed. It explains all the decisions and changes to the planning that were done.

Chapter 5 presents the results of an evaluation phase, according to benchmarks that have been set in the outline of the thesis' declaration. Deviations from the benchmarks are properly explained.

Chapter 6 concludes the work and expresses ways to further contribute to the application in the future.





# Chapter 2

## Fundamentals

A detailed record of every project is already present with their corresponding theses, but the overarching challenges and technologies involved are characterized in this section. It is necessary to see the bigger picture for this work and where it wants to go to in the future. The modern connected ecosystem relies on a variety of communication protocols that address different needs. Among these, BLE, Ultra-Wideband (UWB), and Transmission Control Protocol (TCP)/ User Datagram Protocol (UDP) serve distinct yet complementary roles. The protocols are explained technologically extensive enough, such that one does not need to read the connected theses, but it will only delve as deep into the technical aspects as is necessary for a basic understanding. A similar comparison of different protocols and their metrics has been done by [14], which has been consulted as a guideline for this section. The main parts of the protocol subsections after the technical introduction will be dedicated to the security aspects and how the protocols position themselves in the IoT bubble. [15] has done a comparison as well and was used as inspiration for gathering security related knowledge.

### 2.1 Background

The *ZigBee* protocol will not be elaborated upon extensively, as it is the primary element of [10] and is described thoroughly there. In summary, Zigbee is a wireless communication standard created for low-power, low-data-rate, and short-range uses, especially in the IoT environment. Based on the Institute of Electrical and Electronics Engineers (IEEE) 802.15.4 standard, Zigbee enables the development of Wireless Personal Area Networks (WPANs) that allow for mesh networking structures [16]. This mesh feature enables devices to transmit data via intermediate nodes, improving network reliability and coverage without relying on centralized infrastructure. Functioning mainly in the 2.4 GHz frequency, Zigbee provides data speeds of up to 250 kbps, making it ideal for uses such as home automation and smart lighting. Zigbee's design incorporates specific device roles such as coordinator, router, and end device, to effectively handle network formation and data routing. Its uniform application profiles guarantee seamless interaction between devices from various manufacturers, promoting a unified and scalable IoT ecosystem.

### 2.1.1 Bluetooth Low Energy

BLE is a WPAN protocol operating in the 2.4 GHz band, designed for low power consumption and short-range communication. It was introduced as part of Bluetooth 4.0, offering a lightweight alternative to 'classic' Bluetooth for connecting devices like sensors, wearables, and beacons. It supports data rates up to 1 Mbps and typically ranges between 10 - 100 meters. BLE's efficiency and support in virtually all modern smartphones have made it a de facto standard for IoT devices requiring intermittent data transfer and long battery life. Notably, Bluetooth 4.2 added the Internet Protocol Support Profile (IPSP), allowing BLE devices to directly support IPv6 for IoT connectivity, further solidifying BLE's role in the Internet of Things [17][18].

In IoT ecosystems, BLE often serves as the local communication link between resource-constrained devices and gateway devices like phones or hubs. The devices typically operate in a star topology, with a central device like a phone and multiple peripherals like sensors. This suits IoT scenarios where a smartphone app configures or collects data from several sensors. Another factor is Generic Attribute Profile (GATT), that provides a flexible data model for IoT where devices can define services and characteristics that can be discovered and accessed by others. BLE's ubiquity and support for broadcast advertising, a connectionless mode for beacons, allow IoT devices to periodically transmit small data or presence announcements efficiently.

#### Security

BLE implements link-layer security through its Security Manager protocol. Devices pair to establish trust and then use CCM encryption<sup>1</sup> for link-layer confidentiality and integrity. BLE pairing can operate in several modes, the strongest of which provide mutual authentication and protection against man-in-the-middle (MITM) interception. However, 'Just Works' pairing, that requires no user verification, provides no MITM defense. It is effectively an unauthenticated key exchange. In BLE 4.0/4.1 (legacy pairing), the pairing process was vulnerable to eavesdropping if an attacker could sniff the initial key exchange, since the temporary key in 'Just Works' or a fixed PIN was guessable. BLE 4.2 introduced LE Secure Connections, which uses Elliptic Curve Diffie-Hellman (ECDH) to derive encryption keys, significantly improving security. This update, along with features like Resolvable Private Addresses (RPAs) for device anonymity, made BLE more robust against snooping and tracking [2]. Once paired, BLE devices encrypt all attribute read/write operations. At the GATT layer, permissions can require encryption or authentication for specific characteristics like making a lock control characteristic only be writable if paired with an authenticated key.

Despite these security mechanisms, vulnerabilities persist in practice. One issue is that many IoT developers opt for 'Just Works' pairing for usability, leaving them susceptible to MITM attacks if an attacker is present during pairing. Interestingly, researchers have found that some BLE implementations could be forced into the weaker legacy pairing, even when secure connections were supported by exploiting flaws in the BLE protocol negotiation [19]. This kind of downgrade attack could allow an attacker to intercept or modify communications if they can act as a fake intermediary during device pairing.

---

<sup>1</sup>[https://en.wikipedia.org/wiki/CCM\\_mode](https://en.wikipedia.org/wiki/CCM_mode)

BLE's GATT also does not inherently enforce access control beyond the pairing bond. It's up to devices to mark attributes as requiring encryption or authentication. Many IoT devices have been shown to expose sensitive data or commands over BLE without proper restrictions. In [19] found that of 17'000+ BLE-enabled Android apps for IoT devices, over 70% had at least one known BLE security issue in how they interacted with devices. Common problems included characteristics that could be read or written by any BLE scanner in range if the device didn't enforce pairing, the use of default or well-known Universally Unique Identifier (UUID) that made sensitive data easy to locate, and lack of encryption on transmitted data. Furthermore, many low-cost IoT gadgets omit firmware update mechanisms, meaning known security flaws cannot be patched, exacerbating long-term risk.

Although BLE introduced address randomization to prevent device tracking, a protocol-level flaw enables de-anonymization of devices' BLE addresses under certain conditions. For instance, if a device used a whitelist of trusted peers, an attacker listening to the protocol could infer the real address from side-channel timing or repetition patterns [19]. This defeats the privacy goal and could allow tracking of devices across advertising address changes. Another concern is that many BLE IoT devices continuously advertise identifiable information, like fitness trackers broadcasting a fixed device name or service UUIDs. Without precautions, this can be abused for fingerprinting or tracking users' physical movements, which was proven to be possible with [20]. Modern BLE specs encourage using resolvable private addresses, which change periodically and can only be resolved by bonded devices holding the private Identity Resolving Key (IRK) [2]. When used properly, this makes it difficult for outsiders to correlate advertising packets over time to the same device.

### 2.1.2 UWB

UWB is a wireless protocol characterized by signal bandwidth in the hundreds of MHz to enable high temporal resolution communication. It typically operates in frequencies from around 3.1 GHz to 10.6 GHz with at least 500 MHz of bandwidth per channel [21]. Unlike narrowband radios, UWB transmits short pulses or chirps spread over a broad spectrum. This yields advantages in high data rates of up to around 1 Gbps and extremely precise time-of-flight measurements. Modern UWB devices support data communications and, most notably, a distance measurement method called ranging between devices with centimeter-level accuracy [22]. UWB's ability to measure distance by timing radio signal propagation is a key feature driving its adoption in IoT applications like indoor localization, asset tracking and secure access control.

UWB has recently gained traction in consumer and IoT devices. Examples include UWB transceivers in modern smartphones and smart tags to enable precision finding of lost items [23]. Another interesting example is secure car keys: automotive manufacturers are adopting UWB in key fobs and cars to defeat relay attacks, because UWB can verify that the key is physically near the car by measuring the round-trip time. This adds a layer of security for passive keyless entry that RSSI-based methods like BLE cannot reliably provide. UWB thus complements other IoT radios, although it is not as ubiquitous or

low power as BLE, but for applications needing location precision or enhanced security, UWB fills a critical niche.

### Security

UWB's security model is centered around its ranging capability. The IEEE 802.15.4z standard introduced Secure Ranging protocols for UWB, recognizing that if UWB is to be used for access control or payments, it must resist sophisticated attackers who attempt to spoof distances [24]. The core idea is to use cryptographic techniques to make the exchange of ranging messages unpredictable to outsiders. This is achieved by techniques such as Scrambled Timestamp Sequences (STS) where the initiator and the responder share a secret or exchange one securely, which they use to randomize the timing or modulation of UWB pulses. If an attacker does not know the secret, they cannot easily manipulate the signals to fake a shorter distance. One approach is that the responder's reply is cryptographically dependent on the challenger's message in such a way that only the legitimate responder with the key can generate the correct reply quickly. An attacker trying to relay or replay it will inevitably introduce a delay or an incorrect sequence. By then measuring the time-of-flight (ToF) with nanosecond precision, UWB can detect if there is any abnormal delay indicating an attack. Secure ranging is designed to thwart both relay attacks, where an attacker simply forwards messages over a longer distance, trying to make two far-apart devices appear nearby and distance fraud, where an attacker device tries to pretend it is closer or farther than it really is by cheating the physics.

While UWB's very design gives it some inherent resilience since it is hard to intercept and modify signals in real-time without specialized hardware, [25] has demonstrated that UWB is not foolproof. Their security analysis showed the first practical over-the-air distance reduction attack against secure UWB ranging. In their experiment, attackers used an inexpensive device to subtly manipulate the UWB signals between an Apple U1 and an AirTag, successfully spoofing the distance from an actual 12m separation down to effectively 0m, tricking the devices into thinking they were at the same location. Notably, this was done without knowing any cryptographic keys. The attack exploited imperfections in the radio implementation, essentially finding a way to send an early reply that partially collides with the legitimate signal, shaving off time. The success rate was only a few percent, but even a 4% chance of a false 'device is nearby' reading could be unacceptable for security-critical uses like door locks. This highlights that even with secure ranging protocols, physical-layer attacks like signal injection or delaying can still pose a threat.

Other challenges for UWB include jamming and interference. Because UWB spreads energy over a wide band, it is fairly resistant to narrowband jamming, however a malicious attacker with a UWB transmitter could emit noise across the UWB band to degrade ranging accuracy or block communication. Regulatory limits on the UWB transmission power to avoid interfering with other devices mean that the UWB range is limited to often less than 50m. Attackers might use this to their advantage by getting closer than normal usage would require. An interesting privacy aspect of UWB in IoT is that, as devices exchange ranging signals, they potentially announce their presence, unlike BLE which can use anonymous advertising. UWB ranging usually happens between devices that have discovered each other via another channel, i.e. often BLE is used to find and then UWB to range.

### 2.1.3 TCP / UDP

TCP and UDP are transport-layer protocols that underpin most internet communications, including IoT systems that use Internet Protocol (IP) networking. Unlike BLE or UWB, TCP/UDP are not wireless link protocols by themselves, they operate over any IP-based network which could be Ethernet, Wireless Fidelity (Wi-Fi) or cellular. Many IoT deployments rely on IP for end-to-end connectivity, especially for devices that communicate with cloud servers or web services.

*TCP* is a connection-oriented, reliable protocol. It establishes a session between two endpoints with a three-way handshake to synchronize sequence numbers, then guarantees in-order, error-checked delivery of a byte stream. It achieves reliability through mechanisms like sequence numbers and acknowledgments, retransmission of lost packets, and flow control/congestion control to prevent network overload [26]. This makes TCP heavier weight but suitable for applications where all data must arrive correctly. In IoT, protocols such as Hypertext Transfer Protocol (HTTP) and Message Queueing Telemetry Transport (MQTT) ride on top of TCP.

*UDP*, in contrast, is connectionless and does not guarantee delivery. It simply sends individual packets, called datagrams, from one host to another with no built-in retry or ordering. UDP has much lower overhead, using no handshake, no sequence numbers in the protocol apart from a length and checksum. Therefore, UDP is useful for applications that value timeliness over reliability or where the application layer will handle any needed reliability [26]. In IoT, UDP is very common due to its lightweight nature like the Constrained Application Protocol (CoAP), a RESTful protocol for IoT devices that runs over UDP and adds its own simple acknowledgment mechanism instead of using a full TCP stack [27]. Low-power wireless networks prefer UDP because maintaining long-lived TCP connections can be problematic in lossy networks.

The choice depends on the use case, UDP is favored in constrained environments since it is simpler and has smaller headers (8 bytes vs 20 bytes for TCP), while TCP is used when IoT devices need the full reliability or stream-oriented communication. It has been shown that TCP outperforms UDP, given the same circumstances [28]. For example, a security camera streaming video might use UDP for the video itself due to latency concerns, but an IoT gateway reporting logs to a server might use HTTP over Transport Layer Security (TLS) /TCP to ensure everything arrives. IoT deployments often involve gateways that translate between local protocols like BLE or Zigbee and TCP/IP to send data over the Internet. Thus TCP/UDP often come into play at the gateway or cloud communication layer.

#### Security

By themselves, TCP and UDP provide no encryption or authentication. They were designed decades ago when the internet was trusted, so any security must be provided by layers above such as TLS/ Secure Sockets Layer (SSL) for TCP or Datagram Transport Layer Security (DTLS) for UDP. TCP does incorporate certain controls that indirectly affect security like sequence numbers and checksums [29]. This offers some resistance to blind injection attacks, though an attacker who can sniff the traffic or guess them within a window can inject malicious data or send forged control flags. UDP, having no handshake

or sequence, is even more vulnerable to packet injection or spoofing. Any entity can send a UDP packet claiming to be from an arbitrary source. There is no concept of a session in UDP that the network layer enforces.

The standard approach is to utilize application-layer security. An IoT device connecting to a cloud Application Programming Interface (API) will use TLS over TCP, ensuring encryption and server authentication. Similarly, when using CoAP/UDP, the device would use DTLS to encrypt and authenticate the datagrams. These protocols (TLS/DTLS) handle key exchange, often via certificates or pre-shared keys in constrained IoT, and provide confidentiality and integrity. Without such measures, any data an IoT device sends over TCP/UDP, especially over wireless networks, could be intercepted or altered by an attacker with relative ease.

IoT deployments have to contend with numerous potential threats if higher-layer protections are not in place. [30] gives a good overview, some notable issues are:

- Denial-of-Service (DoS): TCP is susceptible to flood attacks, where an attacker initiates many TCP handshakes but never completes them, causing the server to allocate resources for half-open connections until it is overwhelmed. Mitigations include cookies or rate-limiting inbound connections. [31] showed that a Universal Plug and Play (UPnP) discovery request, which is HTTP over UDP in LAN environments, could be spoofed to amplify traffic 30x towards a target. IoT devices running services that respond to UDP requests must implement anti-spoofing or not be openly accessible to the internet to prevent unwilling participation in Distributed Denial-of-Service (DDoS) attacks.
- Packet Spoofing/Hijacking: Because UDP lacks connection state, an attacker can easily spoof UDP packets. This is especially problematic for services that trust certain IP addresses. In IoT, an attacker could spoof commands to a device if the device accepts UDP commands from an authorized IP without additional authentication. With TCP, spoofing is harder, since one cannot easily spoof and complete the 3-way handshake unless the attacker is on-path. However, if an attacker can predict TCP sequence numbers, they can hijack or reset a connection [32]. IoT devices often communicate over local Wi-Fi and if that Wi-Fi is open or compromised, TCP sessions from IoT devices to the router could be intercepted and tampered with. Session hijacking at the transport layer could allow an attacker to inject false sensor readings or commands by taking over an unencrypted stream.
- Resource Constraints and Protocol Handling: Some IoT devices have very limited processing power and may run minimal TCP/IP stacks. This can lead to vulnerabilities like the mishandling of malformed packets. A malformed TCP packet or sequence of out-of-order packets might crash a poorly implemented TCP/IP stack on a tiny IoT microcontroller, causing a DoS or potentially a buffer overflow that could be exploited. Notable past vulnerabilities, like the Ripple20 set [33], included flaws in embedded TCP/IP libraries widely used in IoT, where an attacker sending crafted packets could take control of the device. These are not flaws in the TCP/UDP protocols per se, but in their implementations under constrained conditions.

Secure by design for TCP/UDP means using them in conjunction with appropriate safeguards. Best practices include using TLS/DTLS for encryption, implementing secure bootstrapping such that the initial key exchange is protected, and closing unnecessary ports on devices. IoT-focused networks also use gateway/firewall devices that restrict inbound connections from the internet. Since most IoT devices act as clients, there's usually no need to allow unsolicited inbound TCP connections or UDP datagrams from arbitrary sources. By blocking those, one can mitigate spoofing and scanning threats. Another trend is the use of alternative transports like Quick UDP Internet Connections (QUIC), which operates over UDP but includes built-in encryption and connection semantics similar to TCP. QUIC might eventually be relevant for IoT if its complexity can be managed, as it provides confidentiality by default and better performance on unstable networks [34].

### 2.1.4 Privacy Requirements

As billions of IoT devices collect and exchange data, privacy has become a paramount concern. IoT sensors gather personal or sensitive information, from health metrics on wearables to video footage on security cameras, raising the question of how this data is protected and used. Privacy in IoT is not only a technical issue but also a regulatory one. In recent years, laws around the world have been enacted to ensure organizations respect users' data privacy.

The General Data Protection Regulation (GDPR), implemented in the EU in 2018, is one of the strictest and most influential privacy laws. It defines personal data broadly and imposes various obligations on entities processing such data [35][36]. The key principles of GDPR include *data minimization* (collect only what is necessary), *purpose limitation* (use data only for the stated purpose), and *storage limitation* (do not keep data longer than needed). For IoT this means that device manufacturers and service providers should avoid collecting extraneous data from sensors and should transparently inform users about what data is collected and for what purpose. GDPR also requires obtaining explicit user consent for personal data collection in many cases, which is why IoT devices often need to have users agree to terms or settings that control data sharing [37]. Users, data subjects, have rights such as access to their data, the ability to request deletion (called the 'right to be forgotten'), and to correct inaccuracies. These rights can be challenging to implement in IoT scenarios like an environment that has hundreds of sensors continuously logging data. Honoring a deletion request might require scrubbing that person's data from all logs.

One of GDPR's requirements is 'privacy by design and by default', meaning IoT systems should be designed from the ground up with privacy in mind. This leads to design choices like anonymizing data at the edge, aggregating information to avoid personal identifiers, and providing strong access control [38]. Encryption is implicitly encouraged, since GDPR mandates 'appropriate technical and organizational measures' to secure personal data. Protecting data in transit and at rest is critical since IoT devices often transmit data over wireless links that could be intercepted. A breach of personal data can lead to severe GDPR fines, so there is a strong incentive for IoT companies to use state-of-the-art

cryptography to prevent unauthorized access. GDPR also requires breach notification within 72 hours to regulators and possibly to users, which has driven organizations to implement better monitoring of their IoT backends for any intrusion.

Other regions have implemented similar laws. In the United States, there is no GDPR-equivalent at the federal level, but several laws address data privacy. The California Consumer Privacy Act (CCPA)<sup>2</sup>, in effect since 2020, gives California residents rights over their personal data somewhat akin to GDPR, including knowing what is collected and the right to deletion, and it applies to businesses over certain size thresholds. CCPA has pushed IoT companies operating in California to provide privacy notices and opt-outs for data sharing. Federally, sector-specific laws can dictate privacy requirements for certain IoT devices like a smart health device collecting medical info. There is also the IoT Cybersecurity Improvement Act<sup>3</sup> in the US that, while focused on security of IoT used by government, indirectly helps privacy by requiring things like unique device identities and no hardcoded passwords.

In Asia, several countries have introduced comprehensive data protection laws inspired by GDPR. For example, Japan's Act on Protection of Personal Information<sup>4</sup> and South Korea's Personal Information Protection Act<sup>5</sup> have stringent rules on handling personal data with many similarities to GDPR like consent and breach notification. One of the most significant is China's Personal Information Protection Law (PIPL), enacted in 2021<sup>6</sup>. PIPL is often called 'China's GDPR', it gives Chinese users rights over their data and places restrictions on data handling, with an emphasis on data localization and government access. IoT manufacturers selling in China must comply with PIPL's requirements, which include clearly delineating the purpose of data collection and obtaining consent, especially for sensitive personal information. Violation of PIPL can result in heavy fines and even criminal liability. These global regulations show a clear trend: IoT systems worldwide are expected to protect user data and privacy by default, not as an optional add-on.

### Impact on IoT and Protocol Design

Privacy regulations have both direct and indirect effects on how IoT communication protocols are used and designed. Directly, laws might dictate certain features, like the EU's ePrivacy directive<sup>7</sup>, an adjunct to GDPR focusing on communications, requires user consent before storing or accessing information on a user's device. Indirectly, to comply with privacy principles, IoT designers incorporate measures such as data anonymization, encryption and access control, device and user consent, as well as logging and audit. Some protocols now explicitly incorporate privacy considerations:

- The BLE spec enhanced its privacy mode after recognizing that BLE beacons could be used to track phones. Devices can now use a resolvable private address that changes frequently. This was not directly due to GDPR, seeing as how it predates

---

<sup>2</sup><https://oag.ca.gov/privacy/ccpa>

<sup>3</sup><https://www.congress.gov/bill/116th-congress/house-bill/1668>

<sup>4</sup><https://www.dlapiperdataprotection.com/?t=law&c=JP>

<sup>5</sup><https://www.dlapiperdataprotection.com/index.html?t=law&c=KR>

<sup>6</sup><https://www.dlapiperdataprotection.com/?t=law&c=CN>

<sup>7</sup>[https://www.edps.europa.eu/data-protection/our-work/subjects/eprivacy-directive\\_en](https://www.edps.europa.eu/data-protection/our-work/subjects/eprivacy-directive_en)



GDPR, but it is a privacy-by-design improvement that aligns with regulatory expectations.

- Modern Wi-Fi standards have introduced Media Access Control (MAC) address randomization for devices when scanning for networks, to prevent tracking of devices by MAC between networks. Apple and Android also randomize Wi-Fi MAC addresses when connecting to networks, a clear privacy feature influenced by the desire to limit long-term identifiers. Enterprises adapting to this had to change how they do network access control, since they cannot rely on fixed MAC addresses as an identity.
- Mobile IoT technologies like Narrowband-IoT and LTE-M<sup>8</sup> inherit the robust security of cellular networks, but privacy laws push operators to ensure that any user-identifying data, like Subscriber Identity Module (SIM) IDs or phone numbers associated with IoT modules, are protected and not improperly shared. eSIM/soft SIM technologies also incorporate more security to avoid cloning, which could lead to impersonation and privacy breaches.

Privacy regulations affect not just the data transfer, but the entire lifecycle of IoT data. This means wireless protocols must often integrate with broader privacy-preserving systems. An IoT device might use end-to-end encryption such that even the service provider cannot read the data, which is going beyond what regulations explicitly require, but to assure users of privacy. An example is Apple's approach with AirTag in their 'Find My' network: location data, that is collected and transmitted, is encrypted on an Apple server such that they cannot view it, only the user can upon request by using their private key [39]. This is a selling point for privacy and ensures compliance since even a breach at their servers would not expose usable information.

IoT architects must choose protocols not just on technical merit, but on privacy considerations. If data is extremely sensitive like medical IoT device data, designers might avoid broad-range wireless protocols like BLE that could be sniffed from outside a home and instead use more contained communication. They will also ensure all traffic is encrypted at a higher layer, using TLS over BLE's GATT via BLE IPv6 mode or similar. Some IoT systems implement additional layers of privacy. Onion routing or mix networks for IoT data have been proposed in research to hide metadata [40]. While not common yet, if privacy laws become more stringent about metadata, IoT protocols might incorporate such concepts. IoT devices often require an explicit user action to start interfacing, like scanning a QR code or pressing a button to pair. This can be seen as aligning with privacy requirements since it ensures the user is in control of who can connect to the device.

In conclusion, privacy requirements have elevated the importance of data protection, user consent, and transparency in IoT communication. Regulations like GDPR have global reach, since any IoT service handling EU residents' data must comply, and they have set a high bar that other jurisdictions are following. IoT manufacturers now compete on privacy features, it's common to see products marketed as respecting privacy. From a technical perspective, this has driven widespread adoption of encryption in transit,

---

<sup>8</sup><https://en.wikipedia.org/wiki/LTE-M>

minimization of data and features in protocols to avoid persistent identifiers. Wireless protocols are being designed or amended with privacy by design in mind, ensuring that the convenience of ubiquitous sensing does not come at the cost of individual privacy.

## 2.2 Related Work

Users are highly concerned about privacy and security. Surveys rank privacy among the biggest concerns for IoT adoption, with consumers desiring control over the personal data these devices collect [41]. However, evaluating the privacy of smart home devices is challenging due to the scale and heterogeneity of device ecosystems, which may contain dozens of diverse devices from different vendors. Academic research has begun developing holistic frameworks to systematically assess IoT privacy and security across devices, apps, and platforms.

### 2.2.1 Privacy Evaluation Frameworks

Researchers have proposed frameworks to evaluate the privacy and security of individual IoT devices by analyzing both the device and its companion smartphone app. One approach is to leverage the companion app as a proxy for the device’s functionality. [42] used a ‘mirror’ strategy, analyzing the mobile apps associated with smart-home devices to infer device behavior and detect vulnerabilities. This platform performed program analysis on thousands of IoT apps and identified hundreds of devices likely affected by known security flaws without needing physical access to each device. Such techniques, while aimed at security, can indirectly flag privacy issues like insecure data transmissions at scale. Similarly, [43] evaluated smart security devices, including smart locks, cameras and alarms, by scanning their Android companion app code for vulnerabilities. They found that every tested app contained at least one common security or privacy weakness, regardless of the brand’s reputation. Many apps exhibited ‘subtle threats’ introduced by developers, like tracking device usage or illegitimately collecting data in ways that violate user privacy. These studies underscore that companion apps can be rich sources of insight into device privacy posture.

[44] focuses on automated compliance auditing for individual IoT products. They introduced the Compliance-Oriented IoT Security and Privacy Evaluation (COPSEC) framework. COPSEC extracts measurable privacy requirements from regulations like GDPR and best-practice security guidelines, then runs a battery of automated tests on IoT devices to check conformance. For example, it will scan a device’s network traffic, open ports, and data transmissions while also parsing the device’s privacy policy, then compare the device’s actual behavior against its privacy claims. The output is a ‘certification’ report on whether the device meets the specified privacy and security criteria. From initial results testing popular consumer IoT products like speakers, cameras and appliances, the authors report that devices largely fail to comply with even basic security guidelines, and their data collection practices often introduce privacy risks for users. Prior to COPSEC, there was effectively no general framework for auditing IoT products’ adherence to privacy

	CIA Triad	Task	Google Home	Apple HomeKit
Deploy	Confidentiality	Discover new devices	<b>Difficult since devices may use different commissioning mechanisms</b>	Apple MFi certification simplifies discovering and commissioning new devices
		Manage credentials	<b>Users need to manage their own credentials</b>	Simplified with Apple Keychain
		Manage different protocols	<b>Difficult to manage due to varied ecosystem</b>	MFi certification makes interoperability easier
	Integrity	Establish access control	Some controls are in place	Highly controlled via HomeKit framework
	Availability	Manage scale	<b>Difficult as number of devices increases</b>	<b>Difficult as number of devices increases</b>
		Accommodate device heterogeneity	<b>Difficult to manage due to diverse ecosystem</b>	<b>Only certified devices permitted</b>
Operate	Confidentiality	Prevent privacy interference	<b>Some integrations may not encrypt data<sup>9</sup>; collects usage statistics<sup>13</sup></b>	Data might be encrypted but still susceptible to inference attacks
	Integrity	Update access control	<b>Controlled through app updates</b>	Managed via system updates
		Patch/upgrade	<b>Not supported, might have to use companion application to administer updates</b>	<b>Not supported, might have to use companion application to administer updates</b>
	Availability	Repair	<b>Support varies by device</b>	Standardized support through AppleCare
Decommission	Confidentiality	Remove sensitive information	<b>Manual process required</b>	Automated removal through settings

Figure 2.1: Comparison of how Apple HomeKit vs. Google Home handle various security and privacy tasks across device lifecycle stages (Deploy, Operate, Decommission), source: [45]

and security requirements. While still in the early stages, such frameworks could help regulators, consumers, and third parties to benchmark IoT devices' privacy compliance on an ongoing basis, rather than relying on vendor assurances alone.

Beyond individual devices, [45] have looked at entire smart home platforms and ecosystems to identify systemic privacy and security issues. They consider the lifecycle of devices, from deployment to operation and decommissioning, and use the classic Confidentiality-Integrity-Availability (CIA) triad to evaluate challenges at each stage. They applied their framework to two major ecosystems, Apple's HomeKit<sup>9</sup> and Google Home<sup>10</sup>, and uncovered a range of shortcomings in how each platform supports secure and private device management. Notably, both ecosystems were found to have issues across nearly every lifecycle phase, starting at onboarding devices to updating and ultimately removing them from service. Apple's ecosystem did fare better in several respects due to its stricter controls. Google requires manual credential management and may not encrypt some data, while Apple restricts device variety via certification. See Figure 2.1 for an informed comparison.

<sup>9</sup>[https://en.wikipedia.org/wiki/Apple\\_Home](https://en.wikipedia.org/wiki/Apple_Home)

<sup>10</sup><https://home.google.com/welcome/>

### 2.2.2 Specialized Tools and NLP

Not all IoT privacy risks come from obvious smart home gadgets, some arise from abuse of IoT technologies in the environment. A salient example is unwanted tracking via Bluetooth tags. AirGuard, a tool by [46], exemplifies a focused privacy-defense solution in this space. Apple built stalking-detection into their OS, yet billions of Android users were left unprotected aside from a very rudimentary scanner app Apple released. In response, they reverse-engineered Apple’s tracking protections and developed AirGuard as a free Android app to safeguard users from covert trackers. It continuously scans for Apple Find My devices and notifies the user if it suspects an unknown tracker is following them. AirGuard’s detection performance was on par with or better than Apple’s solution, and a field study in [46] with tens of thousands of users demonstrated its effectiveness in the wild.

AirGuard’s success highlights the importance of independent privacy tools complementing IoT ecosystems. It addresses a privacy issue that falls outside of smart home device evaluations. Instead of evaluating a device’s compliance, it monitors the environment for malicious devices. While such tools are not holistic frameworks, they play a practical role in the IoT privacy ecosystem by giving users actionable protection in specific domains.

#### NLP

Given the vast amounts of textual and log data associated with IoT devices (privacy policies, app descriptions, user interfaces, permission lists, device logs, etc.), researchers have explored Natural Language Processing (NLP) techniques to automate privacy analyses. One prominent area is parsing and analyzing IoT privacy policies. [47] conducted a large-scale study of smart home device privacy policies, using NLP to demystify their content and coverage. They augmented an existing policy analysis tool with IoT-specific named entity recognition, and analyzed 284 smart home privacy policies, extracting thousands of data practice statements. They revealed major challenges like finding the policies in the first place, since IoT privacy disclosures are fragmented across websites, apps, manuals. And even when obtained, the policies often lacked standardization. The authors had to incorporate ‘out-of-band’ context, mapping device model names to companies, to correctly identify which policy text applied to which device. The outcome was a curated dataset of labeled policy texts and an improved model for extracting data collection and sharing statements. Simply analyzing policy text ‘in a vacuum’ can be ineffective thus future frameworks should combine policy NLP with other information sources like linking privacy clauses with the device’s actual sensor capabilities or network behavior, to catch inconsistencies.

Other possible scenarios include analyzing smart speaker voice assistant transcripts or skill descriptions with NLP to identify potential eavesdropping skills or scanning IoT device user interfaces like permission prompts or settings menus to evaluate how transparently privacy options are presented. So far, research specifically using NLP on device logs or UIs in the IoT context is sparse, but it is a logical extension. However, while NLP can significantly reduce manual effort, achieving a high degree of accuracy and completeness in automated privacy audits will likely require a hybrid approach, leveraging text analysis, program analysis, and perhaps even computer vision for UIs in tandem.

### 2.2.3 Research Gap

Academic research on smart home privacy evaluation is gaining momentum, moving from isolated analyses toward more comprehensive frameworks. There are now tools to scan IoT apps for flaws, monitor device traffic, parse privacy promises, and even detect rogue devices in our midst. A truly holistic privacy evaluation framework would allow stakeholders, be it users, auditors, or regulators, to assess an IoT device or an entire smart home setup against a unified set of privacy requirements. Current studies illuminate important pieces of this puzzle. They show that many devices fall short of basic privacy protections [44], that mainstream platforms still have systemic issues [45], and that automation is possible but needs refinement [47]. By addressing the gaps in coverage, context, and integration, future research can work toward a holistic solution.



# Chapter 3

## Design

As the main part of this thesis, a desktop application is being developed<sup>1</sup>. This chapter will cover the decision-making processes for all design choices, guided by the restrictions that are given through the existing projects and the thesis project description. Further, architectural choices are explained and outline the structure of the application.

### 3.1 Requirements Engineering

Modern requirements engineering is a process to reduce the risk of delivering a system that does not meet the desires and needs of stakeholders [48]. This means that one only specifies as many requirements as is feasible within a given product boundary. With the structure of a bachelor's thesis, we are limited by a set time frame and the description of work. For a pure software product, this would mean a full-fledged description of the software system according to [49], which would be great, but not feasible to create.

The 2 reasonable work products that create value are the following:

#### **Risk assessment**

Here, we determine which features will have the highest impact on development when implemented poorly. A matrix from [50] is used as a reference and modified, since we do not determine the stakeholders and especially the distinction into importance is not particularly fitting. The reason for this is because the requirements elicitor is also the developer, and the pool of possible other stakeholders is not big. Therefore, we cannot expect to have a distribution of stakeholders with different levels of importance, which in turn leads to the loss of meaningfulness of this factor.

The 3 points in Figure 3.1 refer to the 3 requirements of the thesis' description of work:

1. Feed the collected data stream into the requirements mapper
2. The projects need to be able to run as standalone on the application

---

<sup>1</sup><https://github.com/iKusii/BA-ModularIntegrationPlatform>

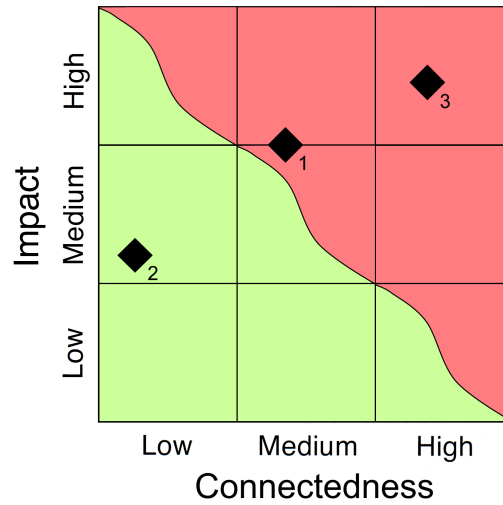


Figure 3.1: Risk assessment

3. The projects need to be able to run concurrently without significant loss in performance

As we can see, we must pay additional attention to the selection of a framework that will allow concurrent processes and data handling.

### Context boundary

Without a clear distinction from the system and its context, there might arise confusion as to what is still part of the system and what is not. One important separation within the system that has not been highlighted before is the affiliation of [12] to the requirements mapper part, which is conceptually separated from the other projects that contribute data.

The system can then be set into context with two different kinds of environments, live and offline. Figure 3.2 shows an abridged version of what this looks like when put together. This thesis is built on the premise that the application will process offline data, but there is an emphasis on building it sufficiently well such that it can be expanded to live data later.

## 3.2 Relevant Data Formats and Technologies

What follows is an overview of the technical stacks of the five projects that build the foundation of the application and what the resulting requirements are.



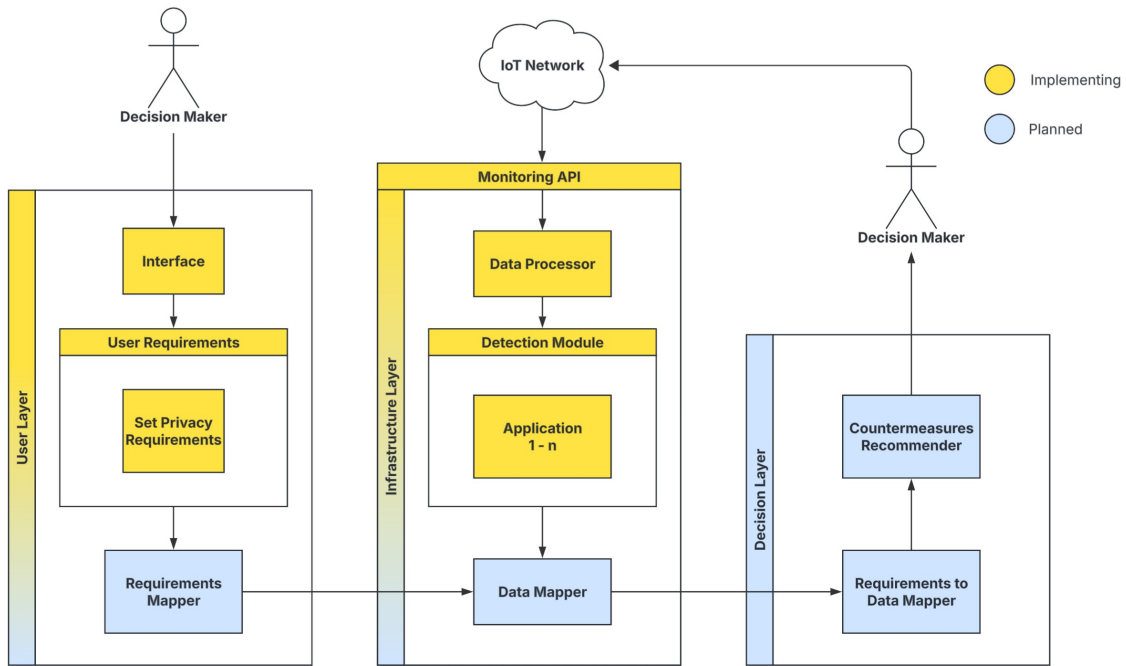


Figure 3.2: Abridged overview, original from Katharina O.E. Mueller

### 3.2.1 Privacy Requirements

The culmination of [12] is a Python<sup>2</sup> command-line interface (CLI) script that runs a user through a privacy use case and informs them about the possible vulnerabilities of their situation. The user can keep a history of their requests in the console, but no files are generated.

This program cannot be used as-is and needs to be changed to output something tangible, since this project is supposed to be the privacy requirements mapper / starting point creator.

### 3.2.2 HomeScout

The name-giving app of [9] was developed for Android-based mobile phones and written in Kotlin<sup>3</sup>. The app keeps a database of encountered devices and flagged devices. Since the whole reason for that work is to study the behavior of tracking devices, it would not make sense to emulate this application on a desktop device which is stationary. However, there exist solutions to mirror a phone screen onto a desktop PC. These solutions will be discussed in 3.3.

<sup>2</sup><https://www.python.org/>

<sup>3</sup><https://kotlinlang.org/docs/home.html>

Baseline				
Work	Language	Has Live Mode?	Input	Output
Privacy Requirements	Python	No	Manual Selection	Overview
HomeScout	Kotlin	Yes	Events	Database
ZigBee Packet Sniffer	Python	Yes	Packets	Event Logs
IoT Network Topology	Python	Yes	Packets	Graph
High-traffic Dataset	Python	No	Dataset	Statistics

Table 3.1: Hard Requirements

### 3.2.3 ZigBee Packet Sniffer

The work of [10] has 2 environments to consider for an implementation. One is bound to the live collection of packets with an nRF board, the other one analyzes existing .pcap files using Python scripts and Wireshark<sup>4</sup>, which can be installed on any OS.

### 3.2.4 IoT Network Topology

In the same manner as the previous entry, [11] has 2 environment setups. The active sniffing of a network requires a hardware component called Ubertooth One<sup>5</sup>, which builds on libraries that can be installed on any OS (even though Linux is recommended), all in combination with the already seen Wireshark software. The evaluation of the data can be done with already generated data and uses a Python script.

### 3.2.5 High-traffic Dataset Generation

Finally, [13] uses Python scripts to process data from pre-generated datasets to identify BLE devices in high-traffic environments. The dataset is extensive with a size of 3.7 GB, which needs to be kept in mind when choosing the application environment.

## 3.3 Platform Specifics Choice

Consolidating the knowledge from the previous section into a conclusive table with Table 3.1, we can see that there is a common language that is used for all of them, which is *Python*. This is not surprising since it is one of the friendliest languages to create custom programs, with popular libraries in numerous areas of application [51].

The only outlier here is HomeScout, which is built in Kotlin<sup>6</sup>, another popular cross-platform language that is based on Java. As mentioned before in 3.2.2, a database is

<sup>4</sup><https://www.wireshark.org>

<sup>5</sup>[https://ubertooth.readthedocs.io/en/latest/ubertooth\\_one.html](https://ubertooth.readthedocs.io/en/latest/ubertooth_one.html)

<sup>6</sup><https://kotlinlang.org/docs/home.html>

cultivated during the execution of the app, which can be simulated for the offline version by creating a dataset for an evaluation. The live version though will need to connect to a mobile device to get periodic updates to the database. One method to do this is through virtual network computing (VNC), this allows for a direct connection from a computer to a mobile device through Wi-Fi [52]. There exist 3rd party solutions for this but it is possible to set this up without them, even though security is a concern due to the nature of the connection [53].

### 3.3.1 Web-based vs Desktop Application

Starting the comparison with a web-based application, the architecture would consist of 4 dedicated servers with a standardized interface for the respective programs and a load balancer to distribute incoming requests efficiently. This allows for parallel processing and good resource isolation and management, while being easily horizontally scalable through the addition of new servers. The challenges here are the initial cost of acquiring and setting up the servers, potential latency issues due to the number of requests that will occur and the existence of a single point of failure in the load balancer. The complexity of the system is another concern when considering authentication and server monitoring.

On the other hand, a desktop application would execute the programs asynchronously. This is very CPU-intensive when all programs should run concurrently at the same time but needs less coordination since each program runs with their own input. The disadvantage here is the lacking ability to scale when the number of programs increases, save for the option to upgrade the hardware, and the single point of failure in the host machine's lifecycle.

The application needs to run on any OS, thus implementing it as a web application is a simple but effective solution. However, since 3 of the projects have live modes that deal with high volumes of data and consistency is a core aspect of any respectable application, the risk of a bottleneck through a possible bad connection is too high. Also, a constant required internet connection is another minor downside, when considering the application's specified ability to process offline data. This, combined with the cost of setting up multiple servers for a project of this scope, leads to a clear decision towards the desktop application.

### 3.3.2 Language Selection

The next step involves deciding on a language that is suitable for the task at hand. Python seems to be the obvious choice but there are limitations with this language that concern multiprocessing. Python has a so-called Global Interpreter Lock (GIL)<sup>7</sup> that prevents multiple threads to execute bytecode at the same time. This prevents race conditions and ensures thread safety but it is diagonally opposed to the goal of the application to run multiple programs concurrently. At the time of this thesis, Python 3.13 has introduced the ability to turn off the GIL optionally in CPython [54], the reference implementation of

---

<sup>7</sup><https://wiki.python.org/moin/GlobalInterpreterLock>

the Python programming language<sup>8</sup>. However, this comes with additional overhead and difficulty implementing this correctly. Finally, the most popular Graphical User Interface (GUI) creator for Python is QtPy<sup>9</sup>, which has a steep learning curve but has good cross-platform support.

Another contender is Electron<sup>10</sup>, an option to run web-based code in a dedicated Chromium browser instance, while Node.JS<sup>11</sup> handles system-level operations. The main advantage is the low entry barrier, since it uses web technologies that are fast to pick up, and the ability to run seamlessly on all OS. It is important to note though that it is not very efficient, and it has no mobile platform support. It also has a CPU usage that is usually higher than that of native applications due to running multiple instances of JavaScript and a larger file size [55]. Despite the cross-platform support, due to the performance reason it will not be considered as an option.

Kotlin has already been mentioned before and is a suitable candidate for the role. One of the GUI frameworks is TornadoFX, a wrapper for JavaFX, a prevalent GUI framework for Java, but it is no longer being developed as of September 2024<sup>12</sup>. Another popular GUI framework is Compose<sup>13</sup> which is developed by JetBrains and prides itself for its ability be used to create cross-platform applications. Given that the application needs to be flexible enough to accommodate future additions of programs that might be written in other languages than Python, it makes for a compelling argument to decide upon this language.

### 3.3.3 Similar existing Solutions

It is important to look at similar solutions implemented by existing companies that have a presence in the Smart Home market to gain insight into suitable architecture approaches. The first one is Samsung's SmartThings [56], it collects data from end nodes in a hub device that is connected to a so-called swarm service in the cloud to evaluate the collected data and return fitting commands for the end nodes. There is a clear separation of concerns that allows for a smooth operation of the whole system while preserving reliability.

The second interesting solution is Home Assistant, an open-source project managed by the Open Home Foundation [57]. This flexible application can be installed at any level of convenience, and it focuses on local data collection with optional cloud connectivity. It achieves this by running the end nodes in containers, which allows users to add any node to their network, given that there exists an integration library for that specific node type. The project has a large community that cultivates those libraries.

The main take-aways for this project are the containerization approach, which seems to be a good fit to run programs independently and collect their outputs for further processing down the line in a hub-like component.

---

<sup>8</sup><https://en.wikipedia.org/wiki/CPython>

<sup>9</sup><https://wiki.python.org/moin/PyQt>

<sup>10</sup><https://www.electronjs.org/>

<sup>11</sup><https://nodejs.org/en>

<sup>12</sup><https://github.com/edvin/tornadofx>

<sup>13</sup><https://www.jetbrains.com/compose-multiplatform/>

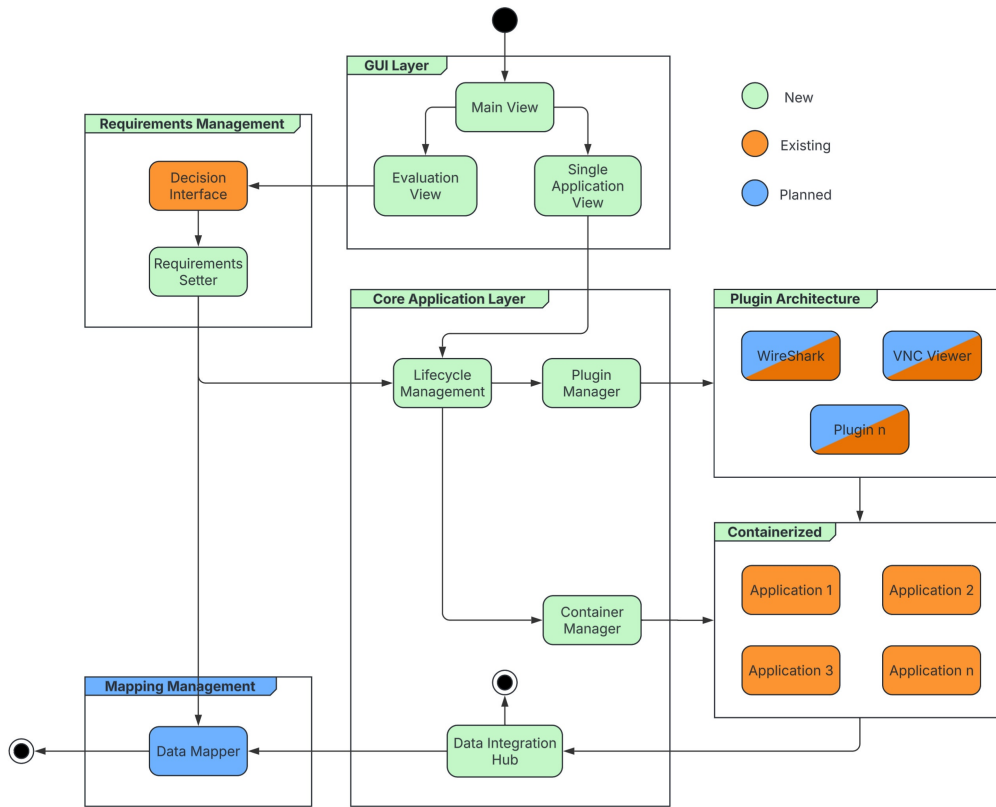


Figure 3.3: High-level system view

### 3.4 Projected Class Diagram

With all the important decisions finalized, a refined class diagram has been created, as illustrated in Figure 3.3. This diagram presents a slightly more detailed and expanded version of Figure 3.2, offering a clearer perspective on the system's architecture. It also highlights which parts are new additions from this thesis and which ones are from previous works. The application supports 2 distinct execution paths: one by following the 'Single Application View' route that yields a specific program in isolation, and another by choosing the 'Evaluation View' route, choosing a privacy requirements setting and having the application take care of managing the appropriate next steps.

In the 'Core Application Layer' there exist 2 managers. The 'Plugin Manager' is a collection of configurations that are necessary to start any additional software for the live modes of the programs. The 'Container Manager' is responsible for running and terminating the containers with their respective programs. Any data that is output by a program is streamlined into the 'Data Integration Hub' and relayed to the next part.

The final step varies depending on the execution path chosen. In the 'Evaluation View', the Data Mapper acts as a placeholder for the last stage of the evaluation process, whereas in the 'Single Application View', the process simply concludes with the program's standard output.

## 3.5 Test Suite

The knowledge and considerations for approaching and applying testing is derived from [58]. Apart from the core management logic that will be implemented with object-oriented programming, this system relies heavily on configurations related to the containerization of individual programs and plugins. Since these configurations are not easily testable, adopting a test-driven development (TDD) approach would be impractical. As a result, the feedback loop for identifying design flaws and managing complexity may be longer than in a typical TDD workflow.

To balance this, tests will be written after the integration of each individual program, marking the completion of incremental development milestones. This approach ensures that testing remains manageable and prevents an overwhelming backlog of untested functionality as the application nears completion, when major changes become more challenging. While this method introduces the risk of delayed issue detection, it allows for greater flexibility in adapting to unforeseen design considerations during development.

To mitigate these risks, a structured testing strategy will be followed, ensuring that unit tests verify the correctness of individual components while integration tests confirm smooth interactions between containerized elements. Ultimately, broader testing phases, including system tests and evaluation structures, will be incorporated to validate the overall integrity and performance of the application. By structuring testing in this way, the application maintains a strong balance between development agility and long-term reliability.

# Chapter 4

## Implementation

This chapter is dedicated to the actual implementation of the design plans and any necessary changes that were made along the way. After introducing the environment for the application, all core components will be explained sufficiently well enough, such that one can start working on this project.

### 4.1 Compose

Kotlin Compose Multiplatform, henceforth only called 'Compose' for brevity, is a modern declarative User Interface (UI) framework that enables a single Kotlin codebase for user interfaces across multiple platforms like Android, iOS, web, and desktop for all OS. It is based on Android's UI toolkit Jetpack Compose and developed by JetBrains, leveraging Kotlin Multiplatform (KMP) technology. On desktop, Compose targets the Java Virtual Machine (JVM) and provides high-performance rendering on all major desktop OS via the Skia graphics library [59].

Compose provides desktop-specific UI components and extensions to integrate with the windowing system and OS features. This means desktop apps can behave like native apps. One can structure an application with common code for data handling, domain logic, and even UI structure, and have minimal platform-specific code, like app entry points or specific integrations. JetBrains provides tooling support for Compose in **IntelliJ IDEA**<sup>1</sup> and it is highly recommended to use it for development. For desktop specifically, the development cycle is quite straightforward. The desktop app runs directly on the development machine, since it is a JVM app, and one can get fast build times and hot reload-style experiences.

Compose apps leverage Graphics Processing Unit (GPU) acceleration for UI, making them performant for rich graphics. Start-up times may be a bit higher than a pure native app due to the JVM startup and Skia library loading, but the runtime performance is generally smooth. Since everything is compiled to JVM bytecode, and possibly native

---

<sup>1</sup><https://www.jetbrains.com/idea/>

code on other targets, heavy computations can be as fast as any JVM program. Using truly native UI components within Compose beyond what the framework offers is not straightforward. For example, there is no built-in support to embed a native macOS control directly in Compose, short of using a third-party bridge or writing a custom interface. Another consideration is application size, a Compose desktop app must bundle the Kotlin runtime and the Skia native libraries, and often a stripped-down Java Runtime Environment (JRE). This means even a simple 'Hello World' app can be tens of megabytes in size. However, this is usually acceptable given modern distribution methods and disk sizes.

## 4.2 Packaging

Since this application is developed with future additions in mind, packaging and distribution may not be an immediate concern. Developers can, as mentioned above, run the application from the IDE and benefit from this intermediate use. However, it is important to look at how a finished product can be shipped in the end.

Packaging a Compose app for desktop involves creating a self-contained, installable binary for each target OS. The goal is to provide end-users with a familiar installation process, such as an .exe or .msi installer on Windows, a .dmg or .pkg on macOS, or a .deb/.rpm on Linux, that includes everything needed to run the app. This can be a bit more involved than packaging a traditional native app because a Compose Desktop app runs on the JVM and thus needs a Java runtime. Fortunately, the process is streamlined by official tools.

The Gradle plugin, which greatly simplifies packaging, wraps the functionality of the Java Development Kit (JDK)'s `jpackage` tool and automates many steps of creating a distributable package<sup>2</sup>. One can create self-contained installations that bundle the application's code, resources, and a tailored runtime, such that the user does not need to have Java installed separately. To make the app truly self-contained, the packaging process will bundle a JRE that the application uses. To avoid shipping a full JVM, which could be 300+ MB, the plugin leverages `jlink`<sup>3</sup>, to include only the necessary Java modules needed to run the app. By default, a minimal set of modules is included.

The plugin defines tasks such as `package<Format>` to create an installer, and a convenient `createDistributable` to assemble the raw distributable application image with the embedded JDK. There are also corresponding `runDistributable` tasks that let you test-run the packaged app on your machine, which is handy to verify it works with the bundled JRE. In the Gradle build file, inside the `compose.desktop.application` block, you can configure various settings:

- Main class is the entry point of your app (-> `mainClass = 'MainKt'`)
- App metadata is the name of the package, version and description

---

<sup>2</sup><https://www.jetbrains.com/help/kotlin-multiplatform-dev/compose-native-distribution.html>

<sup>3</sup><https://dev.java/learn/jlink/>



```
43
44     compose.desktop {
45         application {
46             mainClass = "evostar.mainKt"
47
48             nativeDistributions {
49                 targetFormats(TargetFormat.Dmg, TargetFormat.Msi, TargetFormat.Deb)
50                 packageName = "evostar"
51                 packageVersion = "1.0.0"
52
53                 macOS {
54                     iconFile.set(project.file("resources/icon.icns"))
55                 }
56                 windows {
57                     iconFile.set(project.file("resources/icon.ico"))
58                 }
59                 linux {
60                     iconFile.set(project.file("resources/icon.png"))
61                 }
62             }
63         }
64     }
```

Figure 4.1: Basic settings with icon packaging options

- Icons provide icons for each platform (.ICO for Windows, .ICNS for Mac, .PNG for Linux) so that your packaged app has a proper icon
- In extra files bundle additional files or resources if needed, via the resources folder or by adding to the distribution
- JVM options, if your app needs specific JVM arguments or system properties, which can be set in the launcher
- Signing/Notarization for macOS, to configure signing identities and notarization steps if distributing to users online, see 4.2.2

Building installers for a given OS typically requires that OS. The jpackage tool is part of the JDK and technically can target other OS if their packaging tools are available, but in practice it is easiest to build Windows installers on Windows, Mac installers on macOS, etc. The Compose Gradle plugin currently follows this pattern. The generated installers cover installation of the app and placement of shortcuts, but beyond that, some 'expected' features might need custom implementation. For example, auto-update is not provided out of the box. to update itself, one would have to implement a solution, perhaps the app checks a server for updates and then downloads a new installer or uses a third-party updater library.

### 4.2.1 Windows

Code signing an installer, and possibly the .exe, is important to avoid SmartScreen warnings like 'Unknown publisher'. The Compose plugin currently does not have built-in code signing steps for Windows. This means after generating the .exe or .msi, one would need

to sign it using `signtool`<sup>4</sup> or a similar tool for a code signing certificate. The `upgradeUuid` is particularly important, it should remain consistent between versions of the application.

```
1 windows {
2     menuGroup = "Application Group"
3     shortcut = true
4     upgradeUuid = "uuid-for-updates"
5     iconFile.set(project.file("path/to/icon.ico"))
6 }
```

Listing 4.1: Windows bundling specifics

## 4.2.2 MacOS

When distributing an unsigned app, users will be blocked by Gatekeeper<sup>5</sup> with a message that the app 'cannot be opened because it is from an unidentified developer'. To avoid this, signing the application with an Apple Developer certificate and notarizing it with Apple's notary service is crucial. The Compose plugin can offer some support, since it can invoke `codesign` and notarization for `.pkg` or `.dmg` if the Developer ID credentials in the Gradle properties are configured. For Mac App Store distribution, one would also need to set the app category and other metadata, which is supported by the plugin. Another small thing, is the familiar prompt 'Drag the app to Applications folder' when opening a DMG. By default, `jpackage`'s DMG doesn't show that fancy UI, since it is a plain volume. The customization of the DMG background image and script is needed for that behavior, which again, is possible with the plugin.

```
1 macOS {
2     bundleID = "com.domain.app"
3     signing {
4         sign.set(true)
5         identity.set("Developer ID Application: Name (TEAM_ID)")
6     }
7     notarization {
8         appleID.set("email@example.com")
9         password.set("@keychain:NOTARIZATION_PASSWORD")
10        teamID.set("TEAM_ID")
11    }
12    iconFile.set(project.file("path/to/icon.icns"))
13 }
```

Listing 4.2: macOS bundling specifics

## 4.2.3 Linux

Linux typically does not enforce signing of binaries, except perhaps some package repositories, so `.deb` and `.rpm` from `jpackage` are usually fine as-is. However, there may be a need to specify maintainer emails or other fields for the package. The plugin has set

---

<sup>4</sup><https://learn.microsoft.com/en-us/dotnet/framework/tools/signtool-exe>

<sup>5</sup>[https://en.wikipedia.org/wiki/Gatekeeper\\_\(macOS\)](https://en.wikipedia.org/wiki/Gatekeeper_(macOS))

Linux-specific fields like maintainer name/email, package license, etc., to satisfy distro package conventions.

```

1 linux {
2     packageName = "app-name"
3     debMaintainer = "name@example.com"
4     menuGroup = "Development"
5     iconFile.set(project.file("path/to/icon.png"))
6 }

```

Listing 4.3: Linux bundling specifics

## 4.3 Containerization

Containerizing desktop GUI applications is an unusual but useful approach. One might consider running an app in a container to encapsulate its environment and avoiding installing dependencies on the host or to deploy the app in a controlled sandbox. Two popular containerization tools are Docker<sup>6</sup> and Podman<sup>7</sup>. Both are Open Container Initiative (OCI)-compliant<sup>8</sup> container engines that can run an app in an isolated environment, but they have different architectures and trade-offs.

Docker is the classic container platform that introduced widespread container use. It uses a client-server model with a background daemon, called the Docker daemon, running as root on the host, which manages images and containers. The user interacts via the Docker CLI, which talks to the daemon to start or stop containers. Podman, on the other hand, is a newer container engine by Red Hat that is daemonless, meaning it runs containers as child processes directly, without a persistent root-privileged service. Podman's CLI is largely compatible with Docker's, but it emphasizes running containers 'rootless', as a normal user, for better security.

Table 4.1 compares key aspects of Docker and Podman as they relate to running a Compose Desktop app. [60] and [61] for comparisons.

Aspect	Docker	Podman
Architecture	Daemon-based, a central dockerd service manages containers. Containers run as sub-processes of the daemon, giving namespace isolation	Daemonless, no always-running service. Containers are launched as separate processes directly via fork/exec. Each container is supervised by a helper process (common), not a monolithic daemon

<sup>6</sup><https://www.docker.com>

<sup>7</sup><https://podman.io/>

<sup>8</sup><https://opencontainers.org/>

Aspect	Docker	Podman
Rootless Support	By default, Docker daemon requires root privileges. A rootless mode is available since Docker 20.x, but not the default and requires a setup	Designed for rootless operation from the ground up. A regular user can run podman without root, and Podman will use user namespaces to emulate the root inside the container. This improves security by containing the app with no root daemon on the host
Compatibility	Universal support, Docker is available on Linux, and via Docker Desktop on macOS/Windows, which provides a VM. Docker CLI and Docker Compose are standard tools widely supported by development workflows	Linux-native, runs on Linux without extra layers. On macOS/Windows, Podman can be used via a VM or Podman Desktop, which is similar to Docker Desktop. Podman supports the Docker OCI image format, so it can run images from Docker Hub directly. Most Docker CLI commands work the same, though some Docker-specific features are not present
Ecosystem & Tools	Very rich ecosystem with Docker Hub registry, extensive documentation, and tools like Docker Compose for multi-container orchestration. Many third-party tools and libraries assume Docker's presence. Docker has features like volumes, networks, and plugins	Podman can use Docker images and has its own Compose equivalent. Podman integrates with systemd, managing containers as systemd services easily. Generally, Podman aims for parity with Docker's functionality in most areas
Performance	Near-native performance for most workloads. Docker's overhead is very low, it uses Linux kernel features. CPU and memory usage under Docker are also very close to bare metal. The Docker daemon itself uses some memory, but for a single container running an app, this is negligible. Docker's network stack tends to be fast, though in rootless mode it may use slirp <sup>9</sup> , which is slower	Also near-native performance. Podman's daemonless approach removes the idle overhead of a daemon, potentially using a bit less memory when no containers are running. In practice, for running one or a few containers, performance differences with Docker are tiny. Podman's rootless mode uses user-space networking which can be a bit slower for high throughput networking compared to Docker's rootful networking

---

<sup>9</sup><https://en.wikipedia.org/wiki/Slirp>

Aspect	Docker	Podman
Concurrency & Scale	Docker's daemon can handle multiple containers simultaneously and is well-tested under heavy loads. The centralized management might introduce a bottleneck in theory, but it also optimizes resource sharing. Docker has long been used in large-scale environments, so its stability with concurrent containers is proven	Podman can run many containers as well, but since each invocation spawns processes, very high concurrency could hit process limits or contention. Podman's advantage is that each container is independent, there is no single point of failure daemon. One stuck container does not impact control of others, whereas if Docker's daemon hangs, all containers become inaccessible via CLI

Table 4.1: Docker vs Podman Comparison

As shown above, Docker and Podman offer very similar capabilities in terms of running a containerized application, with differences primarily in the ecosystem. Meaning, it comes down to environment and preference rather than technical necessity. Both can achieve the goal of containerizing the app with almost equal performance, with differences on the order of a few percent as per [61]. Docker shines in convenience if it is already used and benefits from the vast documentation available. Podman shines in security and not needing a root daemon, making it possible to give a container image to a less privileged environment and running it without worrying about installing Docker. Podman works on all major OS through Windows-Subsystem for Linux (WSL) on Windows, has native support on Linux and macOS support through virtual machines. This more consistent behavior across platforms is the deciding factor in favor of Podman. The instantiation details are in 4.4.7.

## 4.4 Core Components

Figure 4.2 shows a well-structured system with clear separation of concerns between container management, data integration, and lifecycle management. Notably, there is a change to the planned role of the PluginManager compared to 3.3.

Core Components:

- LifecycleManager: The main orchestrator that manages app level access to containers and data integration
- ContainerManager: Handles container creation and management
- Container: Represents a single container instance with its own lifecycle
- DataIntegrationHub: Manages data communication between containers and the application

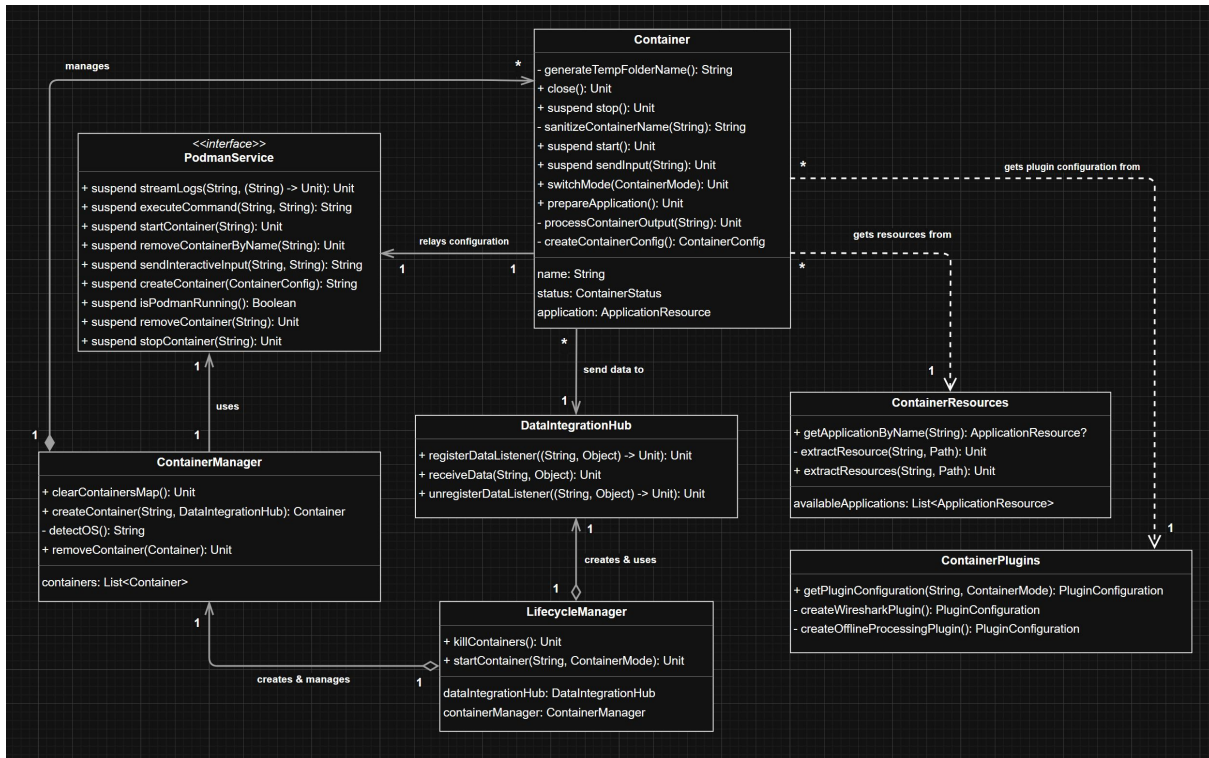


Figure 4.2: Implementation Class Diagram

Supporting Components:

- **ContainerResources**: Project-specific information data
- **ContainerPlugins**: Project-specific plugin data

Key Relationships:

- **LifecycleManager** manages both **ContainerManager** and **DataIntegrationHub**
- **ContainerManager** creates and manages **Container** instances
- **Container** uses **DataIntegrationHub** for data communication
- **Container** creates a **ContainerConfig** for the image configuration, tailored to a use case

Data Flow:

- Data flows through the **DataIntegrationHub** using the observer pattern
- Containers communicate through the **DataIntegrationHub** interface
- Container lifecycle is managed through the **LifecycleManager**

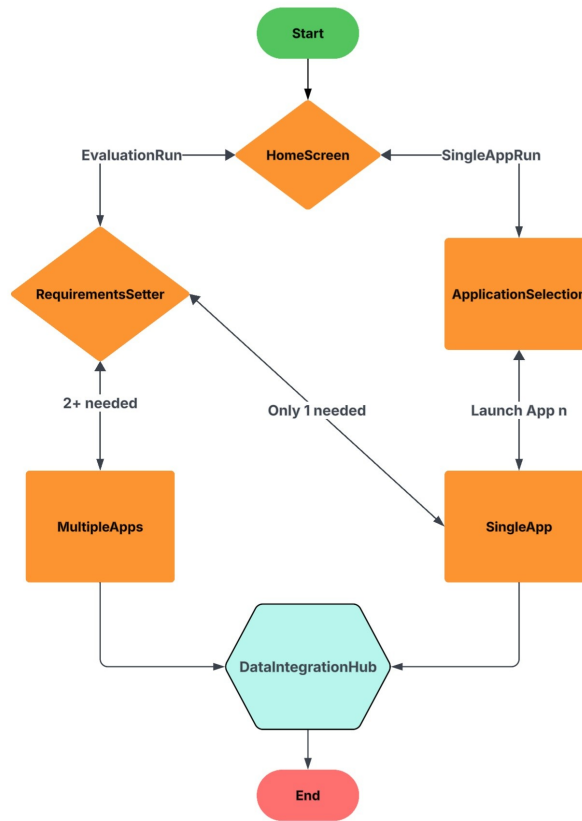


Figure 4.3: User Interaction Flow Diagram

#### 4.4.1 Views

Compose’s UIs are constructed using composable functions, which are annotated with *@Composable*. These functions serve as the fundamental units for declaring UI elements in a declarative programming model. Rather than manually updating the UI, developers define what the interface should look like for a given application state, and the Compose runtime automatically manages updates when that state changes. Common composables include Text, Button, and layout containers such as Column, Row, and Box, which facilitate structured UI composition. State management is typically handled through constructs such as *remember* and *mutableStateOf*, enabling reactive behavior within the UI. A Compose application is launched using a ‘Window’ container inside the application function, which hosts the root composable, thus establishing the application’s visual interface. This model emphasizes simplicity, readability, and responsiveness, aligning with modern functional UI paradigms.

The term ‘View’ is used in Android development for the equivalent of Composables in Compose for Desktop. The execution flow of the UI components allow for 2 distinct paths, shown in Figure 4.3. The 2 paths are called ‘SingleApplicationRun’ and ‘EvaluationRun’, corresponding to the 2 required modes of execution stated in 1.2. Both paths can be traversed back at any time, removing the necessity to restart the program once a mode is selected.

Back to Home

1.: What issues inside the AirTag system do you want to analyze?

☒ 1.1 iOS-related AirTag issues

1.1: These are all Item Safety Alert related-issues. Choose one:

☐ 1.1.1 Inadvertent Disabling of the ISA

☐ 1.1.2 No Manual Scanning Possibility

☐ 1.1.3 Low Reliability

☒ 1.1.4 ISA Trigger Blocking

1.1.4: These are the methods used for blocking the triggering of ISA's. Choose one

☐ 1.1.4.1 Changing of Status Bytes in the BLE advertisement

☒ 1.1.4.2 Periodic Key changing

☒ 1.2 Android-related issues

1.2: These are all issues related to Apple's Tracker Detect App on the Google Play Store. Choose one:

☐ 1.2.1 No Background Scanning Possibility

☐ 1.2.2 Low Awareness and Detectability

☐ 1.3 Sound-related issues

☒ 1.4 BLE-related issues

1.4: The following issues are BLE-related issues. Choose one:

☐ 1.4.1 RSSI-based Fingerprinting

☐ 1.4.2 Byte changing of BLE advertisements to block ISA from triggering

☐ 1.5 macOS-related issues

☒ 1.6 FindMy Authentication-related issues

Submit selection

Figure 4.4: Privacy Requirements Mapping UI

In `EvaluationRun`, the project of [12] has been adapted into a user-friendly UI that maps the privacy requirements decisions to be evaluated onto a set of required programs to do the actual evaluation. As mentioned before, the actual mapping from the requirements onto the relevant data to be evaluated will be a subject for future work, but both the UI and the ability to run multiple containers at once, as well as the ability to navigate to the correct proceeding View, is already implemented.

In `SingleApplicationRun`, the user may choose one of the 4 other projects to be executed in either live or offline mode. During both modes, the currently processed data is displayed in a logging feature. This also notifies the user when the program inside the container has completed. Afterwards, the user can find the processed data inside a folder for personal use.

#### 4.4.2 Lifecycle Manager

The `LifecycleManager` serves as the central orchestrator for the application's lifecycle, managing the creation and operation of containers through the `ContainerManager` and facilitating data flow via the `DataIntegrationHub`. It provides essential methods to start



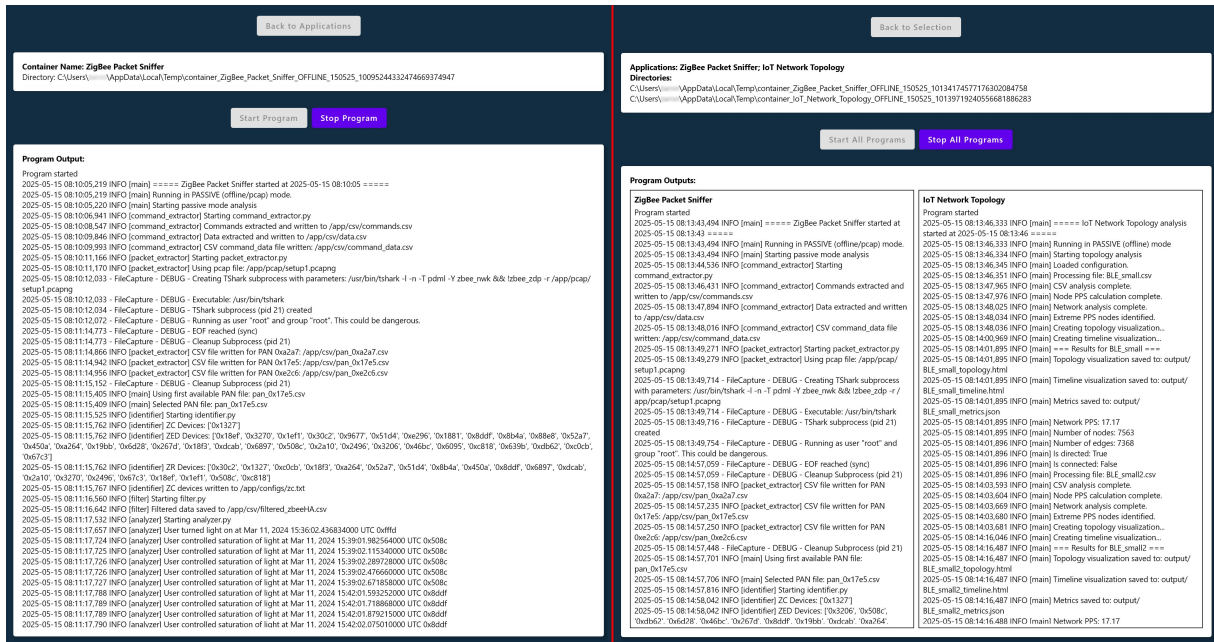


Figure 4.5: Left: single application container, right: multiple applications container

and stop containers, ensuring seamless interaction with container functionalities. As the entry point for application components, it is responsible for initializing containers, utilizing the ContainerManager to create and manage these containers. The LifecycleManager also ensures that all containers are properly terminated, handling any exceptions that may arise during the shutdown process to maintain system integrity.

### 4.4.3 Data Integration Hub

In the initial design phase, this component was thought to collect any processed data in itself for further analysis. However, the actual application building process has shown that it is unnecessary to do this, since the programs mount themselves into temporary directories where the data is accessible for use. This shifted the intended usage of this component to what it is now, a feedback piece. It implements the Observer pattern<sup>10</sup> for data communication, meaning that components can register and unregister as data listeners. This way it can distribute container output to registered listeners which provides a clean separation between containers and UI components. This also work for multiple containers at the same time.

```

1 DisposableEffect(containerName) {
2   container = lifecycleManager.containerManager.getContainers().find {
3     it.name == containerName }
4   val listener: (String, Any) -> Unit = { receivedContainerName, data
5     ->
6     if (receivedContainerName == container?.name) {
7       processContainerOutput(data.toString())
8     }
9   }
10  }
```

<sup>10</sup><https://refactoring.guru/design-patterns/observer>

```

6      }
7  }
8  lifecycleManager.dataIntegrationHub.registerDataListener(listener)
9  onDispose {
10      lifecycleManager.dataIntegrationHub.unregisterDataListener(
11          listener)
12  }

```

Listing 4.4: Registering a data listener (+ automatic clean-up on dispose)

#### 4.4.4 Container Manager

This component manages container instances in a thread-safe `ConcurrentHashMap`. It initializes the container creation and deletion by acting as a middleware between the lifecycle manager and the containers themselves. This is also where the `PodmanService` implementation resides, since there is an important distinction between creating a container and starting one. 4.4.7 will explore this in-depth.

#### 4.4.5 Container

The Container component is responsible for **preparing** and interfacing a container, not running it! The actual instance of a Podman container will be with the `PodmanService`. This component handles the configuration of the container image according to predefined specifications that can be found inside the `ContainerResources` and `ContainerPlugins` Singleton objects. The configuration starts with an `ApplicationResource` object, that defines what parts of the directory that holds the project code, will be mounted into the temporary directory in which the container will be run. It does this recursively for subfolders and files, so bundling scripts or data necessary for the execution is not unnecessarily extensive. See Listing 4.4.5

```

1 private fun getAvailableApplications(): List<ApplicationResource> {
2     return listOf(
3         ApplicationResource(
4             name = "ZigBee Packet Sniffer",
5             configPath = "$RESOURCE_BASE_PATH/applications/ZigBee Packet
6                           Sniffer/config.json",
7             scriptPath = "$RESOURCE_BASE_PATH/applications/ZigBee Packet
8                           Sniffer/main.py",
9             extraResources = listOf("data", "scripts", "csv", "configs",
10                                    "pcap")
11         ),
12         ApplicationResource(
13             name = "IoT Network Topology",
14             configPath = "$RESOURCE_BASE_PATH/applications/IoT Network
15                           Topology/config.json",
16             scriptPath = "$RESOURCE_BASE_PATH/applications/IoT Network
17                           Topology/main.py",
18             extraResources = listOf("data", "scripts")
19         )
20     )
21 }

```

```

15         ...
16     )
17 }

```

Listing 4.5: ApplicationResource data

It continues with selecting an appropriate image for the container base. This will hold any dependencies like Python versions, external libraries or programs that should be included in the installation of the container. These images are built on the first ever run of the application and are then cached in the Podman instance for further runs. A single *Containerfile* and *requirements.txt* are all that is necessary for this.

```

1 FROM python:3.12-slim
2
3 WORKDIR /app
4
5 # Install required Python packages
6 COPY requirements.txt .
7 RUN pip install --no-cache-dir -r requirements.txt
8
9 # Create necessary directories
10 RUN mkdir -p /app/data
11
12 # Set environment variables
13 ENV PYTHONUNBUFFERED=1
14 ENV DATA_PATH=/app/data
15 ENV CONFIG_PATH=/app/config.json
16
17 # Default command (will be overridden)
18 CMD ["python", "main.py"]

```

Listing 4.6: Containerfile for a basic image that uses Python

This is also where it will look for any additional dependencies that may be introduced through plugins, but the offline modes all do not need those. Afterward, it sets the environment variables like the entry point script name, the selected mode of operation (live or offline) and returns this to the PodmanService for instantiation. The remaining functionality is related to handling container lifecycle operations and sending data to the DataIntegrationHub for logging. Once a container has been stopped and removed, there is no functionality in place to regain access to it inside the application.

#### 4.4.6 Plugin Manager

Initially this was supposed to be its own component that handles the augmentation of container images with additional software and hardware requirement based on a use case. During the implementation it became clear that this will not be more than a Singleton object that stores the configuration for **live mode** that might become necessary in the future. There is a great deal of detail that can be prepared beforehand like network access configurations, hardware requirements and permissions. See Listing 4.4.6 for a detailed depiction of a nRF52840 development kit configuration.

```

1  return PluginConfiguration(
2      name = "nrf-sniffer",
3      version = "1.0.0",
4      requirements = PluginRequirements(
5          software = listOf(
6              SoftwareRequirement("tshark", "3.0.0", verifyCommand = "
              tshark --version", type = SoftwareType.SYSTEM_PACKAGE),
7              SoftwareRequirement(
8                  name = "nrf802154_sniffer_extcap",
9                  minVersion = "1.0.0",
10                 verifyCommand = "tshark -G extcaps | grep -q
                  $tsharkCaptureInterface",
11                 type = SoftwareType.PYTHON_PACKAGE
12             )
13         ),
14         hardware = listOf(
15             HardwareRequirement(
16                 type = HardwareType.SERIAL_PORT,
17                 identifier = hostDevicePath,
18                 permissions = listOf("rwm")
19             )
20         ),
21         permissions = emptyList(),
22         networkAccess = null,
23         extcapOptions = emptyList()
24     ),
25     containerConfig = ContainerPluginConfig(
26         volumes = listOf(
27             VolumeMount(
28                 // Ensure user.home is appropriate for all OS, or make
29                 // configurable
30                 hostPath = System.getProperty("user.home") + "/"
31                 Evostar_Captures/ZigBee",
32                 containerPath = "/app/captures"
33             )
34         ),
35         devices = listOf(
36             DeviceMount(
37                 hostDevice = hostDevicePath,
38                 containerDevice = containerSideSerialPort,
39                 permissions = "rwm"
40             )
41         ),
42         capabilities = emptyList(),
43         environmentVariables = mapOf(
44             "TSHARK_CAPTURE_INTERFACE" to tsharkCaptureInterface,
45             "NRF_SERIAL_PORT_INTERNAL" to containerSideSerialPort
46         ),
47         inheritHostGroups = true
48     )
49 )

```

Listing 4.7: Setup data for an external nRF board configuration

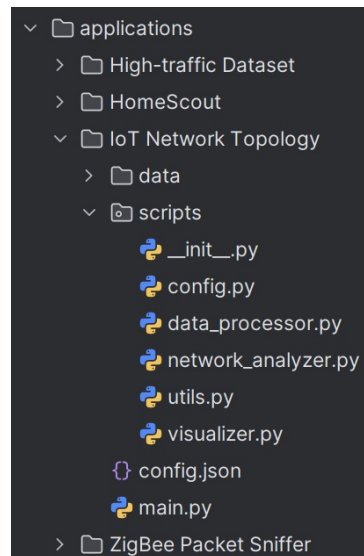


Figure 4.6: Folder structure for adapted projects

#### 4.4.7 PodmanService

The PodmanService abstracts away all the low-level details of container operations, providing a clean interface for the rest of the application. It handles platform-specific concerns, error conditions, and provides the building blocks for container lifecycle management. There are a lot of Podman-specific error cases that need handling, since the Podman virtual machine instance will usually continue to exist even after the application has ended. This means that potential errors can span multiple sessions or even days. The implementation uses coroutines with the Input/Output (IO) dispatcher for non-blocking execution of potentially long-running operations inside the application. This service is a crucial part of the architecture, serving as the bridge between the high-level container management logic and the low-level Podman container runtime.

#### 4.4.8 Adjusted Projects

The privacy requirements project adaption was already explained in 4.4.1. The other projects exist in different formats, ranging from Python scripts, over Jupyter notebooks<sup>11</sup> to Kotlin mobile apps. The goal of the adaptation of these projects was to streamline the workflow with little to no user interaction, since this is connected to overhead for possible multiple container execution. This meant rewriting the projects to simple scripts and changing parts where needed while keeping the original outputs. A sample project folder structure for this application can be seen in Figure 4.6. The entrance point is called 'main.py', and any project related specifications have been abstracted away into a 'config.json' file. Alternatively, one could include parameters into environment variables during the container configuration phase, but this approach makes it easier to change variables on the go in one place.

---

<sup>11</sup><https://jupyter.org/>

Additionally, functionality to process user input has been created, since not everything could be properly solved with the *config.json* settings. However, it is advised to not rely on this if possible. Especially when using multiple containers at once this can lead to race conditions from the input pop-ups. As summarized in Table 4.2, the projects have different levels of support features that relate to their general workflow.

Project	Interactivity?	Images	Input	Output
HomeScout	No	Base / VNC	Database files in /data	Same as input
ZigBee Packet Sniffer	Supported	Base / nRF-capture	pcap files in /p-cap	Event log report
IoT Network Topology	No	Base / nRF-capture	.csv files in /data	summaries in /output
High-traffic Dataset	Currently not supported. Would need rewrite, but does make sense	Base / nRF-capture	.csv files in /data/raw	tables, plots and reports in /outputs

Table 4.2: Project Usages

The image column relates to a predefined set of system level dependencies that will be available in the Podman container environment. The 'Base' image contains a Python version 3.12 environment with all needed external libraries and the Wireshark CLI tool *tshark*, since this is needed for most of the projects to process .pcap files. There was no need to make multiple variants of the base image that are tailored to each project, since these images are only built once when first launching the application and then used from Podman's cache. The second image name in that column relates to the image used for live mode. The 'Wireshark' image is proposed to be equipped with more functionality than what is needed for pcap processing like port discovery and external hardware harmonization relative to the container environment. The 'VNC' image will be used to connect and emulate a smartphone for the HomeScout application.

[9] currently only exists in offline mode, where the program dumps a database in a dedicated folder for further processing.

[10] is the only project so far to support interactivity. The .pcap files are processed and there is a step in the workflow where the user can choose a .PAN file that was extracted earlier. The interactivity is toggleable and if not active it will simply choose the first .PAN file in the folder.

[11] is pretty well streamlined. It processes .csv files through its own pipeline and puts the graphs, timelines and metrics into a folder called /output.

[13] is currently not supporting interactivity, even though the project it is based on relies heavily on it. Basically, after each substep in the initial project, the user was able to choose to continue with the intermediate result or to run another iteration with new data.

There are 4 main 'pipelines' that have been identified from this and used as processing steps. After each step, there is a checkpoint created, such that the program could resume from that checkpoint with the corresponding generated data. Currently, the *main.py* orchestrator executes all 4 pipelines in succession and stores the results in */outputs/logs*. This project might present itself to add another exclusive mode that mirrors the initial fully interactive suite.

## 4.5 Testing

Compose provides sufficient tools to test both the UI layer and the underlying logic of an application. The testing frameworks used for the test suite are JUnit4 and Mockito.

### 4.5.1 UI

The Compose testing API allows to write tests that interact with a Composable via semantics such as finding nodes by tags or text and performing clicks. Under the hood, the Compose test framework takes care of launching a headless version of the UI, such that there is no need for a visible window for tests, meaning it can render the UI in memory and interact with it this way.

For the UI part there was no emphasis on getting the branch coverage up to 100% since this would mean testing UI settings that are not reachable. Changing the code to allow for more branch testing would mean opening up the code to modes that it currently does not support and may actually introduce more problems than it would solve.

### 4.5.2 Logic

The logic part testing has been put off for the most part, since the application is not a finished product as of the writing of this thesis. There will be additions to the structure and interoperability that result from the proposed live modes. Thus any tests that try to encapsulate the behavior of the logic components will most definitely need to be changed again. This is further amplified by the workflow of the core components that necessitate integration tests and they do not present themselves for simple unit testing.





# Chapter 5

## Evaluation

With the development and integration of the application complete, it is essential to evaluate its effectiveness in relation to the original standalone projects. This comparative assessment provides insight into the performance implications and potential trade-offs introduced by the integration process. To facilitate a fair and consistent evaluation, a set of measuring standards has been defined. These metrics serve as the foundation for analyzing the application's behavior under controlled conditions and are critical for drawing meaningful conclusions about its efficiency, scalability, and overall impact. The following sections introduce these measurement criteria in detail and subsequently apply them across a range of test scenarios to assess the application's relative performance.

### 5.1 Benchmarks

Benchmarking is essential to objectively evaluate the performance and behavior of different systems under comparable conditions. The aim is to assess and compare the performance characteristics of CLI projects and a graphical application, when executed independently and for the application combined in three specific pairings. The objective is to assess system efficiency, overhead behavior under multitasking conditions, and container-induced performance penalties.

The most useful performance metric for this use case is runtime, measured from initiation to completion. The projects will be used in their adapted form, since they have been outfitted with a logging feature that includes starting, ending and total time spent executing. Only one of the initial projects could have been used as is, with a time measuring wrapper utility, while the others would have used too many user interactions that would have made measurements inaccurate and difficult.

To ensure that results are robust and comparable, the following testing scenarios have been chosen:

- **Standalone CLI Run:** The projects are executed independently in the CLI, with a predefined input set and runtime configuration.

- **Standalone App Run:** The App is running a single container with no interactivity, using the same input set.
- **Pairing 1:** ZigBee Packet Sniffer and IoT Network Topology are run in 2 containers at the same time.
- **Pairing 2:** ZigBee Packet Sniffer and High-traffic Dataset are run in 2 containers at the same time.
- **Pairing 3:** IoT Network Topology and High-traffic Dataset are run in 2 containers at the same time.
- **Pairing 4:** All 3 projects are run at the same time.

The HomeScout project was excluded since it is currently a simple data dump and thus would not contribute meaningfully. Each scenario is executed multiple times under controlled conditions to minimize noise and capture consistent performance data. The tests are run on 3 different machines to give a good perspective on the system usage, see Table 5.1 for the specifications. Executing multiple projects at the same time in the CLI was not feasible to do.

OS	Brand & Name	CPU	RAM
Windows 11	ROG Strix GL10	AMD Ryzen 5 3400G	16GB
macOS	MacBook Pro 2021	Apple M1 Pro	16GB
Windows 11	Legion T5 28IMB05	Intel i7-10700	32GB

Table 5.1: Evaluation system specifications

## 5.2 Comparison

The input data for each project used in the comparison corresponds to the datasets and configurations made available in the project’s repository<sup>1</sup> at the time of submission. To ensure consistency and reproducibility, all input files were sourced directly from this repository without modification. Runtime measurements were recorded in seconds, leveraging Python’s built-in *time* library, which offers sufficient granularity for performance analysis at this level. To minimize variability and isolate system performance, no additional programs or background processes were allowed to run during the evaluation phase, thereby harmonizing the system environment across all tests. Each scenario, defined as a specific project executed under a particular configuration, was executed ten times consecutively. This repetition enabled the computation of average runtimes and standard deviations, contributing to a more stable and statistically relevant performance baseline. For transparency and further analysis, the complete set of raw captured values is presented in Figure A.1 and as a file in repository.

<sup>1</sup><https://github.com/iKusii/BA-ModularIntegrationPlatform>

## 5.3 Analysis

What follows are explanations of statistical measures that are computed onto the raw data. The actual values can be checked in A.2.

To establish a foundational understanding of the system behavior, descriptive statistics on the raw runtime data have been calculated. For each workload and combination:

- **Mean runtime** offers an overall indicator of expected performance
- **Standard deviation** measures variability and indicates consistency
- **Minimum and maximum runtimes** reveal the bounds of performance
- **Median runtime** helps identify skewed data and is more robust to outliers

This first analysis helps characterize the baseline performance of each configuration in isolation and under parallelization load.

**Performance Overhead** is one of the key evaluation objectives, which aims to quantify the performance overhead introduced when the projects are executed in the application vs the CLI. This overhead is computed as:

$$Overhead (\%) = \left( \frac{App \text{ Mean Runtime} - CLI \text{ Mean Runtime}}{CLI \text{ Mean Runtime}} \right) \times 100$$

System	ZigBee	IoT	Dataset	Zig + IoT	Zig + Data	IoT + Data	All 3
PC 1	-21.5	-1.67	7.77	-16.92 / 14.98	-15.06 / 10.65	19.42 / 10.93	-1.32 / 58.88 / 29.62
macOS	9.59	20.16	7.3	12.63 / 18.13	13.92 / 11.48	22.55 / 8.69	17.72 / 25.52 / 17.03
PC 2	-2.61	-8.7	-0.96	-1.18 / - 4.45	0.93 / 1.94	-3.87 / 0.96	3.69 / 2.04 / 5.23

Table 5.2: Overhead % compared to CLI baseline

By applying this metric to each combination, the identification of how significantly application tasks stress the system, compared to their CLI execution, is possible.

When only considering this chart, it seems as if macOS handles containerization worse than Windows-based systems, but it is also important to notice that the actual runtimes on macOS are consistently faster in both CLI and single run scenarios.

To understand how runtime behavior evolves under concurrent load, the examination of overhead patterns using slope charts and direct overhead calculations is used. See Figure 5.1.

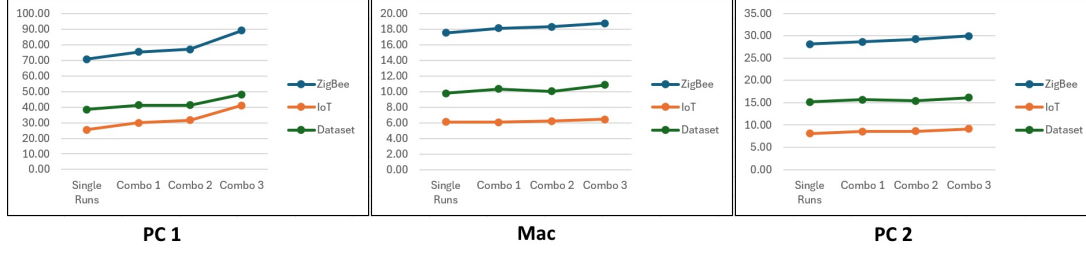


Figure 5.1: Slope graphs

**System stability** under load is another critical factor. The evaluation for this uses the Coefficient of Variation (CV), calculated as:

$$CV = \frac{\text{Standard Deviation}}{\text{Mean}}$$

System	ZigBee	IoT	Dataset	Zig + IoT	Zig + Data	IoT + Data	All 3
PC 1	0.023	0.060	0.086	0.044	0.005	0.057	0.102
macOS	0.021	0.050	0.036	0.028	0.030	0.050	0.067
PC 2	0.006	0.030	0.010	0.055	0.012	0.041	0.020

Table 5.3: CV values per system and scenario

The values are rounded to 3 digits and for combinations the higher value was picked. A high CV indicates erratic performance, while a low CV suggests predictable and stable execution. This is particularly important for real-time or latency-sensitive operations, which might be an issue in future additions to the application.

### Cross-Platform Performance Comparison

As expected, all projects generally take longer to run when they are part of a combo compared to when they run alone. This is due to resource contention for CPU, memory and I/O. The slowdown percentage for each project is highest when it is part of the 3-project combo. This is logical as there is more competition for resources.

To holistically assess each system's ability to handle containerized workloads, a **composite performance score** is introduced. The score penalizes high runtime, large container overhead, and unstable performance, measured via CV:

$$Score_{system} = \sum_{project} [\alpha * NR + \beta * Overhead\% + \gamma * CV]$$

NR is the *normalized runtime*, calculated as the median container runtime over the median CLI runtime. The weights were empirically set to  $\alpha=0.5$ ,  $\beta=0.3$ ,  $\gamma=0.2$  to prioritize relative speed and overheads. Lower scores indicate better overall performance in the context of container usage.

<b>System</b>	<b>Score</b>
PC 2	3.88
PC 1	35.27
Mac	62.9

Table 5.4: Performance scores

This ranking aligns with earlier observations and underscores PC 2’s efficiency in handling multi-container workloads. This also reaffirms that macOS has a good base performance with CLI runs and that it handles containers worse than its Windows-based counterparts. The runtimes are still faster on macOS but the scaling is in favor of Windows.

These findings highlight the importance of considering not just raw runtime performance, but also efficiency under concurrent workloads and runtime stability when evaluating systems for deployment in containerized environments. The weighted scoring model, which incorporates normalized runtime, overhead during concurrent execution, and runtime variability, reveals that systems with modest single-run performance may underperform under load due to higher overhead or instability. This suggests that future development and system selection should prioritize balanced performance characteristics rather than peak speed alone. Hardware definitely has an impact on not only the startup time, but also the runtime in total. These finding also need to be appended and correlated with network data, as soon as the application expands into this territory.



# Chapter 6

## Final Considerations

### 6.1 Summary

The thesis goals outlined in 1.2 have been fully met. Starting with a thorough theoretical background knowledge presentation in chapter 2, a design for a prototype had been developed according to the needs of the underlying projects. In general, this outline had the right idea when it came to the execution workflow of the desired application. However, major components of the logic implementation were not placed where they have ended up now. Mainly, the plugin manager and the data integration hub have undergone changes that shifted the intended contribution to a more appropriate place.

The implementation phase was shaped by iterative cycles of attempts of getting the containers to run and rewriting the projects in a streamlined manner. Not all projects were the same amount of effort when it came to rewriting them, since some had already been in an advantageous format while others needed to be built up from scratch. The harmonizing of data into a pipeline was achieved rather quickly and the simultaneous execution of containers happened faster than anticipated.

The evaluation phase proved to be quite enlightening, since the results showed metrics that are going to be useful for further development. Especially the OS-specific scaling abilities are good to know at this point of the application life cycle.

### 6.2 Challenges and Accomplishments

The main goal of implementing the offline modes was successfully achieved. The rewriting of the programs was accompanied by a logging feature that was later used for the evaluation phase. The privacy requirements project has been nicely adapted into a user-friendly selection page.

The container environment itself was a major concern due to the differences that may be impeding the execution on different OS. The research showed that Podman would be the

better fit for this use case, but with no prior knowledge of containers, this was effectively a gamble. However, apart from one minor bug that occurred once due to insufficient memory allocation, the OS-specific functionality was sufficiently well abstracted and the implementation process has shown that it was indeed the correct choice.

One step that has not been taken is the packaging of the application into an executable file. This is not necessary yet, since the application will still evolve and a direct usage from an IDE is currently the superior way of using it. All the steps involved to undergo this endeavor have been documented though.

An additional goal was to implement a live mode that uses a nRF developer kit for packet capturing. [10] has this functionality in the base project already. This has been attempted but was abandoned due to time restrictions. The source code contains an artifact that tries to establish a connection with the external hardware. It is located in the 'PluginConfigurations' file.

The main challenge here is connected to the discovery of the external device port from inside the container environment. This differs for each OS and needs a proper setup to guarantee an error-free execution across platforms. For Windows this might include mounting ports into WSL through an admin shell. More technical details about this can be found in A.4.

This functionality can also be used for [11] with close to no additional overhead. [13] would be able to process the .pcap files but it currently does not fit into the pipeline structure, thus more work would be necessary to integrate this fully.

Yet another live mode that was touched upon is the connection to a smartphone for the 'HomeScout' project. Research was done, as to how this could be achieved using a VNC, but nothing specific has been attempted yet.

One last unofficial goal was to implement a mode that supports multiple containers at once, which was also achieved. This was beneficial for the evaluation, as well as processing different data sources at the same time.

## 6.3 Conclusions

This project marked a significant personal and academic milestone, not only due to the technical complexity involved, but also because it encompassed and applied a wide range of skills that have been developed throughout the studies. From setting up isolated container environments and analyzing system performance to designing a comprehensive and fair evaluation framework, the work demanded a deep engagement with both theory and practice. Despite beginning with no prior knowledge of containerization, benchmarking, or the underlying systems, a gradually strong understanding, and even a genuine interest, in the topic was developed. The challenges were many, but each one contributed to a clearer picture of how real-world system evaluation is done, making this project a fitting culmination for an academic journey.



## 6.4 Future Work

The application is functional in its current state and can host the chosen projects in their simple form. The implementation is sufficiently well done such that other developers can easily add to its scope. This is a strong base to extend the functionality with either more projects, live mode for the existing ones or new modes altogether. The following list of possible additions has been cultivated over the development of this application:

- **Most importantly:** Implementing the privacy requirements mapping and subsequent data analysis.
- Finishing live mode capture using the nRF PluginConfiguration.
- Improving the testability of the core logic components.
- Abstracting away OS-specific functionality into a utility package.
- Possibly rewriting or adding another mode for [13] with the initial user interactivity from the notebooks.



# Bibliography

- [1] Bluetooth, “2024 bluetooth market update,” 2024, last accessed 01 April 2025. [Online]: <https://www.bluetooth.com/2024-market-update/>
- [2] G. Koulouras, S. Katsoulis, and F. Zantalis, “Evolution of bluetooth technology: Ble in the iot ecosystem,” *Sensors*, Vol. 25, No. 4, 2025. [Online]: <https://www.mdpi.com/1424-8220/25/4/996>
- [3] K. E. Jeon, J. She, P. Soonsawad, and P. C. Ng, “Ble beacons for internet of things applications: Survey, challenges, and opportunities,” *IEEE Internet of Things Journal*, Vol. 5, No. 2, pp. 811–828, 2018.
- [4] Statistica, “Smart home - united states | statista market forecast,” 2024, last accessed 12 April 2025. [Online]: <https://www.statista.com/outlook/cmo/smart-home/united-states#product-types>
- [5] Y. Zhang, J. Weng, R. Dey, and X. Fu, *Bluetooth Low Energy (BLE) Security and Privacy*. Cham, Springer International Publishing, 2020, pp. 123–134. [Online]: [https://doi.org/10.1007/978-3-319-78262-1\\_298](https://doi.org/10.1007/978-3-319-78262-1_298)
- [6] M. Caesar, T. Pawelke, J. Steffan, and G. Terhorst, “A survey on bluetooth low energy security and privacy,” *Computer Networks*, Vol. 205, p. 108712, 2022. [Online]: <https://www.sciencedirect.com/science/article/pii/S1389128621005697>
- [7] J. Yang, C. Poellabauer, P. Mitra, and C. Neubecker, “Beyond beaconing: Emerging applications and challenges of ble,” *Ad Hoc Networks*, Vol. 97, p. 102015, 2020. [Online]: <https://www.sciencedirect.com/science/article/pii/S1570870518307170>
- [8] S. Singh, P. K. Sharma, S. Y. Moon, and J. H. Park, “Advanced lightweight encryption algorithms for iot devices: survey, challenges and solutions,” *Journal of Ambient Intelligence and Humanized Computing*, Vol. 15, pp. 1625–1642, 2024.
- [9] L. Bienz, “Homescout: a modular bluetooth low energy sensing android app,” Master’s thesis, University of Zurich, 2023.
- [10] D. Datsomor, “Homescout extension: Investigation of lightbulb-based user-profiling and privacy-preservation,” 2024, bachelor’s Thesis.
- [11] A. Vincenz, “Design and evaluation of a generic iot network topology detection and visualization approach,” Master’s thesis, University of Zurich, 2023.

- [12] D. Vogel, "Mapping boundaries: An analytical dive into airtags and respective privacy concerns," 2024, bachelor's Thesis.
- [13] S. R. Saxer, "Dataset generation and feature extraction for high-traffic environment personal tracker identification," 2024, bachelor's Thesis.
- [14] S. Al-Sarawi, M. Anbar, K. Alieyan, and M. Alzubaidi, "Internet of things (iot) communication protocols: Review," *2017 8th International Conference on Information Technology (ICIT)*, 2017, pp. 685–690.
- [15] I. Coston, E. Plotnizky, and M. Nojournian, "Comprehensive study of iot vulnerabilities and countermeasures," *Applied Sciences*, Vol. 15, No. 6, 2025. [Online]: <https://www.mdpi.com/2076-3417/15/6/3036>
- [16] C. M. Ramya, M. Shanmugaraj, and R. Prabakaran, "Study on zigbee technology," *2011 3rd International Conference on Electronics Computer Technology*, Vol. 6, 2011, pp. 297–301.
- [17] C. Gomez, J. Oller, and J. Paradells, "Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology," *Sensors*, Vol. 12, No. 9, pp. 11 734–11 753, 2012.
- [18] M. Al-Shareeda, M. Ali, S. Manickam, and S. Karuppayah, "Bluetooth low energy for internet of things: review, challenges, and open issues," *Indonesian Journal of Electrical Engineering and Computer Science*, Vol. 31, pp. 1182–1189, 08 2023.
- [19] Q. Zhao, C. Zuo, J. Blasco, and Z. Lin, "Periscope: Comprehensive vulnerability analysis of mobile app-defined bluetooth peripherals," *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '22. New York, NY, USA, Association for Computing Machinery, 2022, p. 521â533. [Online]: <https://doi.org/10.1145/3488932.3517410>
- [20] P. Locatelli, M. Perri, D. M. Jimenez Gutierrez, A. Lacava, and F. Cuomo, "Device discovery and tracing in the bluetooth low energy domain," *Computer Communications*, Vol. 202, pp. 42–56, 2023. [Online]: <https://www.sciencedirect.com/science/article/pii/S0140366423000452>
- [21] G. Aiello and G. Rogerson, "Ultra-wideband wireless systems," *IEEE Microwave Magazine*, Vol. 4, No. 2, pp. 36–47, 2003.
- [22] W. Hirt, "Ultra-wideband radio technology: overview and future research," *Computer Communications*, Vol. 26, No. 1, pp. 46–52, 2003. [Online]: <https://www.sciencedirect.com/science/article/pii/S0140366402001196>
- [23] M. M. Khan, "Precision finding and ultra-wideband technology," July 2021.
- [24] X. Luo, C. Kalkanli, H. Zhou, P. Zhan, and M. Cohen, "Secure ranging with ieee 802.15.4z hrp uwb," *2024 IEEE Symposium on Security and Privacy (SP)*, 2024, pp. 2794–2811.

- [25] P. Leu, G. Camurati, A. Heinrich, M. Roeschlin, C. Anliker, M. Hollick, S. Capkun, and J. Classen, "Ghost peak: Practical distance reduction attacks against HRP UWB ranging," *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA, USENIX Association, 2022, pp. 1343–1359. [Online]: <https://www.usenix.org/conference/usenixsecurity22/presentation/leu>
- [26] K. R. Fall and W. R. Stevens, *Tcp/ip illustrated*. Addison-Wesley Professional, 2012, Vol. 1.
- [27] D. Singh, M. K. Mishra, A. Lamba, and S. Swagatika, "Security issues in different layers of iot and their possible mitigation," *International Journal of Scientific & Technology Research*, Vol. 9, No. 04, pp. 2762–2771, 2020.
- [28] F. T. AL-Dhief, N. Sabri, N. A. Latiff, N. Malik, M. Abbas, A. Albader, M. A. Mohammed, R. N. AL-Haddad, Y. D. Salman, M. Khanapi *et al.*, "Performance comparison between tcp and udp protocols in different simulation scenarios," *International Journal of Engineering & Technology*, Vol. 7, No. 4.36, pp. 172–176, 2018.
- [29] O. O. Felix, "Tcp/ip stack transport layer performance, privacy, and security issues," *World Journal of Advanced Engineering Technology and Sciences*, Vol. 11, No. 2, pp. 175–200, 2024.
- [30] A. M. Alotaibi, B. F. Alrashidi, S. Naz, and Z. Parveen, "Security issues in protocols of tcp/ip model at layers level," *International Journal of Computer Networks and Communications Security*, Vol. 5, No. 5, p. 96, 2017.
- [31] G. Kayas, M. Hossain, J. Payton, and S. R. Islam, "An overview of upnp-based iot security: threats, vulnerabilities, and prospective solutions," *2020 11th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, 2020, pp. 0452–0460.
- [32] K. Hemanth, T. Ravikiran, M. V. Naveen, and T. Ravi, "Security problems and their defenses in tcp/ip protocol suite," *International Journal of Scientific and Research Publications*, Vol. 2, No. 12, 2012.
- [33] D. dos Santos, "Identifying and protecting devices vulnerable to ripple20," 2020, last accessed 16. April 2025. [Online]: <https://www.forescout.com/blog/identifying-and-protecting-devices-vulnerable-to-ripple20/>
- [34] K. Nepomuceno, I. N. d. Oliveira, R. R. Aschoff, D. Bezerra, M. S. Ito, W. Melo, D. Sadok, and G. Szab<sup>3</sup>, "*Quicandtcp : Aperformanceevaluation*," *2018 IEEE Symposium on Computers and Communications (ISCC)*, pp. 00051.
- [35] P. Voigt and A. Von dem Bussche, *The eu general data protection regulation (gdpr)*. Springer, 2017, Vol. 10, No. 3152676.
- [36] H. Mildebrath, "Understanding eu data protection policy," 2025, last accessed 19. April 2025. [Online]: [https://www.europarl.europa.eu/RegData/etudes/BRIE/2022/698898/EPRS\\_BRI\(2022\)698898\\_EN.pdf](https://www.europarl.europa.eu/RegData/etudes/BRIE/2022/698898/EPRS_BRI(2022)698898_EN.pdf)

- [37] V. Morel, M. Cunche, and D. Le Maître, “A generic information and consent framework for the iot,” *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 2019, pp. 366–373.
- [38] S. Wachter, “The gdpr and the internet of things: a three-step transparency model,” *Law, Innovation and Technology*, Vol. 10, No. 2, pp. 266–294, 2018.
- [39] Apple, “Find my,” last accessed 19. April 2025. [Online]: <https://www.apple.com/icloud/find-my/>
- [40] N. P. Hoang and D. PISHVA, “A tor-based anonymous communication approach to secure smart home appliances,” *ICACT Transactions on Advanced Communications Technology*, Vol. 3, pp. 517–525, 09 2014.
- [41] P. Emami-Naeini, H. Dixon, Y. Agarwal, and L. F. Cranor, “Exploring how privacy and security factor into iot device purchase behavior,” *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI ’19. New York, NY, USA, Association for Computing Machinery, 2019, p. 1â12. [Online]: <https://doi.org/10.1145/3290605.3300764>
- [42] X. Wang, Y. Sun, S. Nanda, and X. Wang, “Looking from the mirror: Evaluating IoT device security through mobile companion apps,” *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA, USENIX Association, 2019, pp. 1151–1167. [Online]: <https://www.usenix.org/conference/usenixsecurity19/presentation/wang-xueqiang>
- [43] A. Allen, A. Mylonas, S. Vidalis, and D. Gritzalis, “Security evaluation of companion android applications in iot: The case of smart security devices,” *Sensors*, Vol. 24, No. 17, 2024. [Online]: <https://www.mdpi.com/1424-8220/24/17/5465>
- [44] G. Anselmi, A. M. Mandalari, S. Lazzaro, and V. De Angelis, “Copsec: Compliance-oriented iot security and privacy evaluation framework,” *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, ser. ACM MobiCom ’23. New York, NY, USA, Association for Computing Machinery, 2023. [Online]: <https://doi.org/10.1145/3570361.3615747>
- [45] R. Mangar, T. J. Pierson, and D. Kotz, “A framework for evaluating the security and privacy of smart-home devices, and its application to common platforms,” *IEEE Pervasive Computing*, Vol. 23, No. 3, pp. 7–19, 2024.
- [46] A. Heinrich, N. Bittner, and M. Hollick, “Airguard - protecting android users from stalking attacks by apple find my devices,” *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec ’22. New York, NY, USA, Association for Computing Machinery, 2022, p. 26â38. [Online]: <https://doi.org/10.1145/3507657.3528546>
- [47] S. Manandhar, K. Kifle, B. Andow, K. Singh, and A. Nadkarni, “Smart home privacy policies demystified: A study of availability, content, and coverage,” *31st*

- USENIX Security Symposium (USENIX Security 22)*. Boston, MA, USENIX Association, 2022, pp. 3521–3538. [Online]: <https://www.usenix.org/conference/usenixsecurity22/presentation/manandhar>
- [48] M. Glinz, “A glossary of requirements engineering terminology,” *Standard Glossary of the Certified Professional for Requirements Engineering (CPRE) Studies and Exam, Version*, Vol. 1, p. 18, 2011.
- [49] “Iso/iec/ieee international standard - systems and software engineering – life cycle processes – requirements engineering,” *ISO/IEC/IEEE 29148:2018(E)*, pp. 1–104, 2018.
- [50] M. Glinz, “A risk-based, value-oriented approach to quality requirements,” *IEEE Software*, Vol. 25, No. 2, pp. 34–41, 2008.
- [51] K. Srinath, “Python—the fastest growing programming language,” *International Research Journal of Engineering and Technology*, Vol. 4, No. 12, pp. 354–357, 2017.
- [52] A. Jadhav, V. Oswal, S. Madane, H. Zope, and V. Hatmode, “Vnc architecture based remote desktop access through android mobile phones,” *International Journal of Advanced Research in Computer and Communication Engineering*, Vol. 1, No. 2, pp. 98–103, 2012.
- [53] T. Reidt, “Android screen mirroring to pc: Options with and without root,” 2024, last accessed 28 February 2025. [Online]: <https://emteria.com/blog/android-screen-mirroring>
- [54] A. Turner and T. Wouters, “What’s new in python 3.13,” 2024, last accessed 05 March 2025. [Online]: <https://docs.python.org/3/whatsnew/3.13.html>
- [55] P. Mugunthan, “Electron software framework: The best way to build desktop apps?” 2024, last accessed 05 March 2025. [Online]: <https://pangea.ai/resources/electron-software-framework-the-best-way-to-build-desktop-apps#disadvantages-of-electron>
- [56] SmartThings, “The architecture of smartthings,” 2025, last accessed 05 March 2025. [Online]: <https://developer.smartthings.com/docs/getting-started/architecture-of-smartthings>
- [57] H. Assistant, “Architecture overview,” 2024, last accessed 05 March 2025. [Online]: [https://developers.home-assistant.io/docs/architecture\\_index/](https://developers.home-assistant.io/docs/architecture_index/)
- [58] M. Aniche, *Effective Software Testing: A developer’s guide*. Manning, 2022. [Online]: <https://books.google.ch/books?id=U4BlEAAAQBAJ>
- [59] K. B. Berko, “Exploring the viability of cross-platform ui development with compose multiplatform,” last accessed 26. April 2025. [Online]: <https://www.droidcon.com/2024/07/05/exploring-the-viability-of-cross-platform-ui-development-with-compose-multiplatform>

- [60] Uptrace, “The complete podman vs docker analysis: Features, performance security,” last accessed 28. April 2025. [Online]: <https://uptrace.medium.com/the-complete-podman-vs-docker-analysis-features-performance-security-eb40fa6046c3>
- [61] Solid-Future, “Podman vs docker: Which is better?” last accessed 28. April 2025. [Online]: <https://solid-future.com/podman-vs-docker-vs-kubernetes>



# Abbreviations

API	Application Programming Interface
BLE	Bluetooth Low Energy
CIA	Confidentiality-Integrity-Availability
CLI	Command-line interface
CV	Coefficient of Variation
CoAP	Constrained Application Protocol
COPSEC	Compliance-Oriented IoT Security and Privacy Evaluation
CCPA	California Consumer Privacy Act
CSG	Communications Systems Group
DDos	Distributed Denial-of-Service
DoS	Denial-of-Service
DTLS	Datagram Transport Layer Security
ECDH	Elliptic Curve Diffie-Hellman
GATT	Generic Attribute Profile
GDPR	General Data Protection Regulation
GIL	Global Interpreter Lock
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IO	Input/Output
IoT	Internet of Things
IP	Internet Protocol
IPSP	Internet Protocol Support Profile
IRK	Identity Resolving Key
JDK	Java Development Kit
JRE	Java Runtime Environment
JVM	Java Virtual Machine
KMP	Kotlin Multiplatform
MAC	Media Access Control
MITM	Man-in-the-middle
MQTT	Message Queueing Telemetry Transport
NLP	Natural Language Processing
OCI	Open Container Initiative
OS	Operating System
PIPL	Personal Information Protection Law

QUIC	Quick UDP Internet Connections
RPA	Resolvable Private Address
SIM	Subscriber Identity Module
SSL	Secure Sockets Layer
STS	Scrambled Timestamp Sequences
TCP	Transmission Control Protocol
TDD	Test-driven Development
TLS	Transport Layer Security
ToF	Time-of-flight
UDP	User Datagram Protocol
UI	User Interface
UPnP	Universal Plug and Play
UUID	Universally Unique Identifier
UWB	Ultra-Wideband
VNC	Virtual Network Computing
Wi-Fi	Wireless Fidelity
WPAN	Wireless Personal Area Network
WSL	Windows-Subsystem for Linux

# List of Figures

2.1	Comparison of how Apple HomeKit vs. Google Home handle various security and privacy tasks across device lifecycle stages (Deploy, Operate, Decommission), source: [45]	15
3.1	Risk assessment	20
3.2	Abridged overview, original from Katharina O.E. Mueller	21
3.3	High-level system view	25
4.1	Basic settings with icon packaging options	29
4.2	Implementation Class Diagram	34
4.3	User Interaction Flow Diagram	35
4.4	Privacy Requirements Mapping UI	36
4.5	Left: single application container, right: multiple applications container	37
4.6	Folder structure for adapted projects	41
5.1	Slope graphs	48
A.1	Evaluation results	70
A.2	Evaluation stats	71



# List of Tables

3.1	Hard Requirements . . . . .	22
4.1	Docker vs Podman Comparison . . . . .	33
4.2	Project Usages . . . . .	42
5.1	Evaluation system specifications . . . . .	46
5.2	Overhead % compared to CLI baseline . . . . .	47
5.3	CV values per system and scenario . . . . .	48
5.4	Performance scores . . . . .	49



# Listings

4.1	Windows bundling specifics . . . . .	30
4.2	macOS bundling specifics . . . . .	30
4.3	Linux bundling specifics . . . . .	31
4.4	Registering a data listener (+ automatic clean-up on dispose) . . . . .	37
4.5	ApplicationResource data . . . . .	38
4.6	Containerfile for a basic image that uses Python . . . . .	39
4.7	Setup data for an external nRF board configuration . . . . .	40





# Appendix A

## Contents of the Repository

The code repository contains the following content:

### A.1 README

This holds all the information that is needed for the installation and operation of the application. For further help, there are additional READMEs in the /Instructions folder.

### A.2 Source code

<https://github.com/iKusii/BA-ModularIntegrationPlatform/tree/main/composeApp>

### A.3 Evaluations

#### A.3.1 File

<https://github.com/iKusii/BA-ModularIntegrationPlatform/blob/main/evaluations.xlsx>

### A.3.2 Results raw

	Single Runs			Combos								
	ZigBee	IoT	Dataset	Zig + IoT		Zig + Data		IoT + Data		ZigBee + IoT + Data		
PC 1	73.03	24.47	37.71	74.67	28.66	76.54	41.23	34.50	41.87	89.16	40.76	47.18
	70.39	28.45	47.67	74.20	29.07	77.76	41.07	29.00	41.31	88.49	35.90	46.25
	72.70	27.23	38.11	75.47	29.03	76.47	41.26	32.00	41.15	92.10	48.98	50.73
	73.79	23.51	36.90	76.69	32.54	77.15	41.35	29.22	40.63	87.66	37.97	46.49
	68.20	24.88	37.68	75.58	31.82	76.72	41.22	32.58	41.74	89.18	41.39	47.89
	70.63	25.34	40.08	76.40	30.08	76.89	40.95	30.52	41.88	90.76	42.87	47.51
	71.02	26.84	37.68	75.19	31.57	77.20	41.08	29.81	40.30	89.60	46.82	48.28
	71.53	27.46	42.73	74.83	29.74	77.47	41.34	31.17	40.76	91.57	45.77	49.40
	69.86	24.87	43.17	75.70	30.53	76.82	41.14	32.93	41.54	87.59	39.63	48.78
	70.50	25.66	38.93	74.41	29.48	77.05	40.75	32.47	41.24	88.52	37.94	49.39
CLI	94.39	25.99	40.03									
	90.16	27.09	35.18									
	90.24	26.14	37.81									
	88.05	26.71	36.87									
	90.17	25.44	33.80									
	91.35	26.42	37.47									
	90.72	27.52	38.93									
	89.28	25.70	41.88									
	90.47	26.23	34.37									
	91.73	25.87	35.44									
Mac	18.09	5.84	10.33	18.24	6.30	18.10	10.15	6.42	10.50	19.20	7.13	11.29
	17.30	5.99	9.74	17.81	5.95	18.71	10.91	5.82	9.96	18.66	6.04	10.42
	17.56	6.23	9.60	17.60	6.14	17.72	9.94	6.20	10.02	18.68	6.10	10.61
	17.23	5.94	9.71	18.14	6.04	18.29	10.25	6.01	9.89	18.64	5.98	10.56
	17.15	6.79	9.65	18.30	5.77	18.39	10.50	6.11	10.01	19.39	6.75	11.14
	17.85	6.21	10.43	18.46	6.07	18.27	10.26	6.64	10.05	18.76	6.58	10.72
	18.21	6.34	10.55	18.28	5.84	18.39	10.37	6.79	10.16	18.69	6.39	10.59
	17.43	5.78	9.67	18.04	5.92	18.57	10.49	6.25	10.48	18.86	5.84	10.96
	17.22	5.96	9.99	17.93	6.24	18.32	9.89	5.95	9.82	19.13	6.90	11.08
	17.64	6.37	9.82	17.74	6.14	17.85	10.60	6.48	9.89	18.70	6.48	11.14
CLI	15.95	5.16	9.30									
	16.21	5.33	9.28									
	15.70	4.86	9.26									
	16.04	5.03	9.35									
	15.79	5.20	9.29									
	16.33	4.94	9.25									
	16.47	5.10	9.18									
	15.80	5.16	9.35									
	15.96	5.27	9.25									
	16.05	5.09	9.21									
PC 2	28.56	7.82	15.13	28.48	8.79	29.30	15.57	9.29	15.38	29.63	9.15	16.24
	28.07	8.40	15.39	28.75	9.32	28.94	15.46	8.76	15.43	30.13	9.32	16.15
	28.22	8.07	15.07	28.70	7.92	29.20	15.50	8.88	15.44	29.87	8.95	16.30
	28.09	8.08	15.39	28.61	8.20	29.22	15.69	8.57	15.25	29.79	9.22	15.98
	28.09	8.63	15.16	28.65	8.53	29.36	15.67	8.17	15.42	30.03	9.10	16.37
	28.17	8.40	15.22	28.59	8.39	29.16	15.84	8.34	15.46	30.16	9.43	15.94
	27.94	8.04	14.99	28.47	7.95	29.43	15.62	8.63	16.50	29.43	9.37	16.08
	28.07	8.17	15.40	28.05	9.16	28.79	16.07	8.40	15.67	29.67	9.04	16.03
	28.16	8.33	15.35	28.46	8.84	28.93	15.49	8.94	15.37	30.60	9.06	16.48
	28.20	7.94	15.09	28.94	8.59	29.46	15.73	8.23	15.21	30.47	8.87	16.12
CLI	29.08	8.52	15.76									
	28.78	9.09	14.98									
	28.90	9.31	15.21									
	29.13	9.43	15.36									
	28.96	8.47	15.83									
	28.78	9.10	15.48									
	28.85	8.68	15.60									
	29.04	9.07	15.34									
	28.67	8.77	15.24									
	28.92	9.24	14.86									

Figure A.1: Evaluation results

## A.3.3 Results evaluated

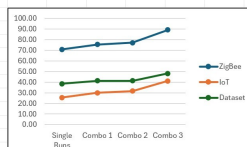
	Single Runs				Combos								Single Runs				Combos													
	ZigBee	IoT	Dataset		Zig + IoT	Zig + Data	IoT + Data	ZigBee + IoT + Data	ZigBee	IoT	Dataset		Zig + IoT	Zig + Data	IoT + Data	ZigBee + IoT + Data	ZigBee	IoT	Dataset		Zig + IoT	Zig + Data	IoT + Data	ZigBee + IoT + Data	ZigBee	IoT	Dataset			
PC 1	73.03	24.47	37.71		74.67	28.66	76.54	41.23	34.50	41.87	89.16	40.76	47.18	28.56	7.82	15.13		28.48	8.79	29.30	15.57	9.29	15.38	29.63	9.15	16.24				
	70.39	28.45	47.67		72.20	29.07	77.76	41.07	29.00	41.31	88.49	35.90	46.25	28.07	8.40	15.39		28.75	9.32	28.94	15.46	8.76	15.43	30.13	9.32	16.15				
	72.70	27.23	38.11		75.47	29.03	76.47	41.26	32.90	41.54	92.10	48.90	50.73	28.22	8.07	15.07		28.70	7.92	29.20	15.50	8.88	15.44	29.87	8.95	16.30				
	73.79	23.51	36.90		76.69	32.54	77.15	41.35	29.22	40.63	87.66	37.97	46.49	28.09	8.08	15.39		28.61	8.20	29.22	15.69	8.57	15.25	29.79	9.22	15.98				
	68.20	24.88	37.68		75.58	31.82	76.72	41.22	32.58	41.74	89.18	41.39	48.98	28.09	8.63	15.16		28.65	8.53	29.36	15.67	8.17	15.42	30.03	9.10	16.37				
	70.63	25.34	40.08		76.40	30.08	76.89	40.95	30.52	41.88	90.76	42.87	47.51	28.17	8.40	15.22		28.59	8.39	29.16	15.84	8.34	15.46	30.16	9.43	15.94				
	71.02	26.84	37.68		75.19	31.57	77.20	41.08	29.81	40.30	89.60	46.82	48.28	27.94	8.04	14.99		28.47	7.95	29.43	15.62	8.34	15.40	29.43	9.37	16.08				
	71.53	27.46	42.73		74.83	29.74	77.47	41.34	31.17	40.76	91.57	45.77	49.30	28.07	8.17	15.40		28.05	9.16	28.79	16.07	8.40	15.67	29.67	9.04	16.03				
	69.86	24.87	43.17		75.70	30.53	76.82	41.18	31.59	41.29	89.17	39.63	48.78	28.16	8.33	15.35		28.46	8.84	28.93	15.49	8.94	15.37	30.60	9.06	16.48				
	70.50	25.66	38.93		74.41	29.48	77.05	40.75	32.47	41.24	88.52	37.94	49.49	28.20	7.94	15.09		28.94	8.59	29.46	15.73	8.23	15.21	30.47	8.87	16.12				
Mean	71.17	25.87	40.07		75.31	30.25	77.01	41.14	31.42	41.24	89.46	41.80	48.19	28.16	8.19	15.22		28.57	8.57	29.18	15.66	8.62	15.51	29.98	9.15	16.17				
Std. Dev.	1.66	1.56	3.44		0.82	1.33	0.41	0.19	1.79	0.54	1.56	4.27	1.41	0.16	0.25	0.15		0.23	0.47	0.23	0.19	0.35	0.37	0.37	0.18	0.18				
Min	68.20	23.51	36.90		74.20	28.66	76.47	40.75	29.00	40.30	87.59	33.90	46.25	27.94	7.82	14.99		28.05	7.92	28.79	15.46	8.17	15.21	29.43	8.87	15.94				
Max	73.79	28.45	47.67		76.69	32.54	77.76	41.35	34.50	41.88	92.10	48.90	50.73	28.56	8.63	15.40		28.94	9.32	29.46	16.07	9.29	15.50	30.60	9.43	16.48				
Median	70.83	25.50	38.52		75.33	29.74	76.97	41.18	31.59	41.29	89.17	41.08	48.98	28.13	8.13	15.19		28.60	8.56	29.21	15.65	8.60	15.43	29.95	9.13	16.14				
Overhead %	-21.50	-1.67	7.77		-16.92	14.98	-15.06	10.65	19.42	10.93	-1.32	58.88	29.62	-2.61	-8.70	-0.96		-1.18	-4.45	0.93	1.94	-3.87	0.96	3.69	2.04	5.23				
Overhead Increase					4.58	16.65	6.44	2.89	21.09	3.16	20.18	60.55	21.85					1.43	4.25	3.53	2.90	4.83	1.91	6.30	10.74	6.18				
CV	0.023	0.060	0.086		0.011	0.044	0.005	0.057	0.013	0.017	0.102	0.029		0.006	0.030	0.010		0.008	0.055	0.008	0.012	0.041	0.024	0.012	0.020	0.011				
CLI	94.39	25.99	40.03											29.08	8.52	15.76														
	90.16	27.09	35.18											28.78	9.09	14.98														
	90.24	26.14	37.81											28.90	9.31	15.21														
	88.05	26.71	36.87											29.13	9.43	15.36														
	90.17	25.44	33.80											28.96	8.47	15.83														
	91.35	26.42	37.47											28.78	9.10	15.48														
	90.72	27.52	38.93											28.85	8.68	15.60														
	89.28	25.70	41.88											29.04	9.07	15.34														
	90.47	26.23	34.37											28.67	8.77	15.24														
	91.73	25.87	35.44											28.92	9.24	14.86														
Mean	90.66	26.31	37.18											28.91	8.97	15.37														
Std. Dev.	1.67	0.64	2.59											0.15	0.34	0.31														
Min	88.05	25.44	33.80											28.67	8.47	14.86														
Max	94.39	27.52	41.88											29.13	9.43	15.83														
Median	90.36	26.19	37.17											28.91	9.08	15.35														
				Score 35.27																										
																														
Mac	18.09	5.84	10.33		18.24	6.30	18.10	10.15	6.42	10.50	19.20	7.13	11.29																	
	17.30	5.99	9.74		17.81	5.95	18.71	10.91	5.82	9.96	18.66	6.04	10.42																	
	17.23	6.23	9.60		17.60	6.14	17.72	9.94	6.20	10.02	18.68	6.10	10.81																	
	17.23	5.94	9.71		18.14	6.04	18.29	10.25	6.01	9.89	18.64	5.98	10.56																	
	17.15	6.79	9.65		18.30	5.77	18.39	10.50	6.11	10.01	19.39	6.75	11.14																	
	17.85	6.21	10.43		18.46	6.07	18.27	10.26	6.64	10.05	18.76	6.58	10.72																	
	18.21	6.34	10.55		18.28	5.84	18.39	10.37	6.79	10.16	18.69	6.39	10.59																	
	17.43	5.78	9.67		18.04	5.92	18.57	10.49	6.25	10.48	18.86	5.84	10.06																	
	17.22	5.96	9.69		17.93	6.24	17.32	9.89	5.95	9.82	19.13	6.90	11.08																	
	17.64	6.37	9.82		17.74	6.14	18.35	10.60	6.48	9.89	18.70	6.48	11.14																	
Mean	17.57	6.15	9.95		18.05	6.04	18.26	10.34	6.27	10.08	18.87	6.42	10.81																	
Std. Dev.	0.38	0.31	0.36		0.28	0.17	0.30	0.31	0.31	0.24	0.27	0.43	0.35																	
Min	17.15	5.78	9.60		17.60	5.77	17.72	9.89	5.82	9.82	18.64	5.84	10.42																	
Max	18.21	6.79	10.55		18.46	6.30	18.71	10.91	6.79	10.50	19.39	7.13	11.29																	
Median	17.50	6.10	9.78		18.09	6.06	18.31	10.32	6.23	10.02	18.73	6.44	10.84																	
Overhead %	9.59	20.16	7.30		12.63	18.13	13.92	11.48	22.55	8.69	17.72	25.52	17.03																	
Overhead Increase					3.03	-2.03	4.32	4.17	2.39	1.39	8.13	5.36	9.73																	
CV	0.021	0.050	0.036		0.015	0.028	0.017	0.030	0.050	0.024	0.014	0.067	0.028																	
CLI	15.96	5.16	9.30																											
	16.21	5.33	9.28																											
	15.70	4.86	9.26																											
	16.04	5.03	9.35																											
	15.79	5.20	9.29																											
	16.33	4.94	9.25																											
	16.47	5.10	9.18																											
	15.80	5.16	9.35																											
	15.96	5.27	9.25																											
	16.05	5.09	9.21																											
Mean	16.03	5.11	9.27																											
Std. Dev.	0.25	0.14	0.05																											
Min	15.70	4.86	9.18																											
Max	16.47	5.33																												

Figure A.2: Evaluation stats

## A.4 Instructions

<https://github.com/iKusii/BA-ModularIntegrationPlatform/tree/main/Instructions>