Thesis Dissertation

A MODULAR LIBRARY FOR CLIENT-SIDE APPLICATIONS IN THE A4IoT INDOOR NAVIGATION PLATFORM

Christakis Achilleos

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

May 2023

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

A MODULAR LIBRARY FOR CLIENT-SIDE APPLICATIONS IN THE A4IoT INDOOR NAVIGATION PLATFORM

Christakis Achilleos

Advisor:

Associate Professor Demetris Zeinalipour

Thesis submitted in partial fulfilment of the requirements for the award of degree of Bachelor's in Computer Science at University of Cyprus.

May 2023

Acknowledgments

Firstly, I would like to thank my Professor Dr. Demetris Zeinalipour for all the help that he has given me during the writing of this paper. He has provided me with everything I needed to work on my thesis and has given me the great opportunity to work on such an incredible project that is the A4IoT. He was always there to help when I needed him, and I am very thankful for that.

Secondly, I would like to thank Paschalis Mpeis for answering any questions that I had during my work. He helped guide me through the difficult aspects of understanding certain parts of the Anyplace system that I had found trouble with.

Lastly, I would like to thank my parents for also being my emotional support whenever I needed it and for all the understanding they had shown.

Abstract

Owing to the prevalence of enclosed environments like shopping malls, airports, medical facilities, cultural institutions and educational institutions, the significance of indoor navigation systems has surged significantly. This surge stems from the inherent limitations of conventional GPS systems, rendering them inadequate for seamless indoor navigation. A4IoT, an open-source indoor navigation system, addresses this need by harnessing Wi-Fi signals to provide cutting-edge indoor positioning and routing services.

For coders who are unfamiliar with the A4IoT design, utilising the A4IoT API can be difficult. To tackle this challenge head-on, we have created a Java library, empowering developers to effortlessly tap into the expansive capabilities of the A4IoT API across a diverse range of devices. Our comprehensive toolkit significantly streamlines the development process of innovative indoor navigation applications, abstracting the complexities of the A4IoT client-side design.

By offering modularity to development of client-based applications for the A4IoT platform that programmers can use to build new indoor navigation apps and progress the field of GPS-less indoor navigation, this thesis makes a contribution to the field of indoor navigation systems.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

The global population is witnessing a relentless surge in smartphone usage, with each passing day. Outdoor navigation systems are already very popular since they are able to provide great assistance to people. Smartphone GPS navigation has replaced the conventional map navigation and has effectively made it obsolete. Given the great popularity with outdoor navigation it can be argued that there would be a need for indoor navigation systems as well. Some businesses like warehouse logistics companies, large cruise-line ships or even factories could potentially find this navigation system useful. Anyplace was created to provide users an indoor navigation system capable of navigating people within closed areas like schools, large buildings or even warehouses for example. Since the previous version of anyplace was not very modular and easy to implement into new projects we have decided to make it more modular so that other developers are able to make use it. The old anyplace was reworked into the now known A4IoT. The motivation behind this thesis is to create a readily available library for the A4IoT interface that is capable of being easily integrated into any mobile application that runs on an Android OS or just the use of Java.

1.2 Thesis Overview

For the purpose of expanding the A4IoT front-end support, we have developed a set of libraries that encapsulate the core functionality for IoT clients. Implementing a modular structure, we give the ability to choose the appropriate library for a given environment or operating system.

1.3 Thesis Contribution

This thesis contributes with the creation of 2 libraries, the anyplace-core library that contains the basic functions and runs on JAVA and the anyplace-android library that contains methods used by the android system and is the basis for creating android projects. Two client-side application are also given to demonstrate the use of the anyplace-android library. The source code is also updated to meet the new android requirements for AndroidX. The new code base also allowed for the further improvement of the anyplace platform towards the creation of A4IoT.

1.4 Thesis Outline

This particular Thesis Dissertation begins with a small introduction in Chapter 1, which explains to the reader the motivation to carry out this work, its impact to future works and its general structure. In addition, it briefly explains the concept and the contents for each one of the chapters that is going to follow.

Moving on to Chapter 2, the “Related Work”, the reader will have the chance to get familiarized with some terms and technologies associated with AndroidX, Gradle, Signal Fingerprinting and the classifications of Internet-based Indoor Navigation Services.

In Chapter 3 we have the Architecture upon which the project is based on. The A4IoT and how each part plays its role in creating a complete IIN system.

Chapter 4 goes on to talk about the Anyplace-core library used for client-side development. Some potential applications and considerations are presented along with the outline of its structure and how it can be implemented into other projects.

Chapter 5 follows with the introduction of the android Java Module that builds more features on top of the Anyplace-core library. The chapter goes on to give an explanation of the library's capabilities. The three-module design for application development is discussed along with two app modules that create a Navigator and Logger applications for the A4IoT platform.

Finally, in Chapter 6 a final conclusion is given along with potential future work considerations.

Chapter 2

Related Work & Background

2.1 Related Work

In modern times people continually find themselves in large indoor spaces such as museums, hospitals, ships, malls, etc. Since the introduction of such spaces people had to rely in some form of indoor navigation which typically consists of handheld maps or signs and indicators for directions. However, technology has presented the possibility for a new form of indoor navigation.

Google and Apple for example have introduced the ability to add maps for indoor spaces and let people use the already available map software on their phones to navigate through indoor spaces. However, this has simply moved the handheld physical map to a digital format that is more interactable.

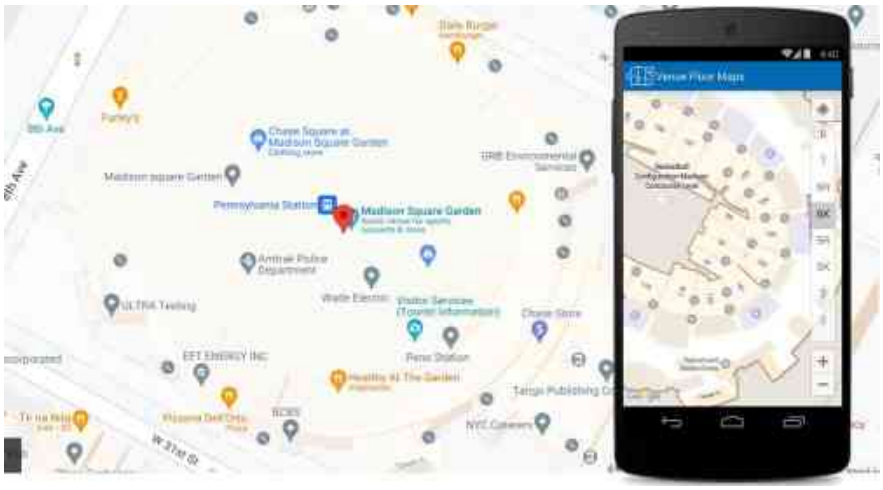

Figure 2.1 Google Indoor Maps [8]. Display of a buildings floor plan through a smartphone.

Going a step further we have the ability to position ourselves within the indoor space, similar to how the GPS system would give positioning, but instead of using satellites we can use a different method of localization that uses RSS fingerprinting [5]. One such indoor navigation system, upon which this project expands on, is the Anyplace A4IoT [4].

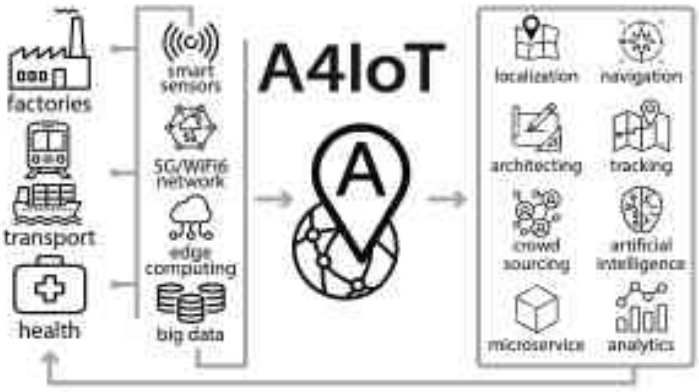

Figure 2.2 A4IoT: A localization service designed for smart spaces [4].

2.1.1 Classification of Internet-based Indoor Navigation Systems

With the rise of smartphones and highly computationally capable devices present in modern everyday life, a new form of indoor navigation has started to slowly replace the more conventional satellite-based localization technologies, more broadly identified as Internet-based Indoor Navigation Services. When combined with the fact people spend more than three quarters of their time in enclosed spaces, it creates opportunities for a

vast range of applications that utilize indoor positioning systems, such as indoor navigation, inventory tracking, and elderly support through Assisted Living [7]. Internet-based Indoor Navigation Systems can offer a large range of information to the user [3].

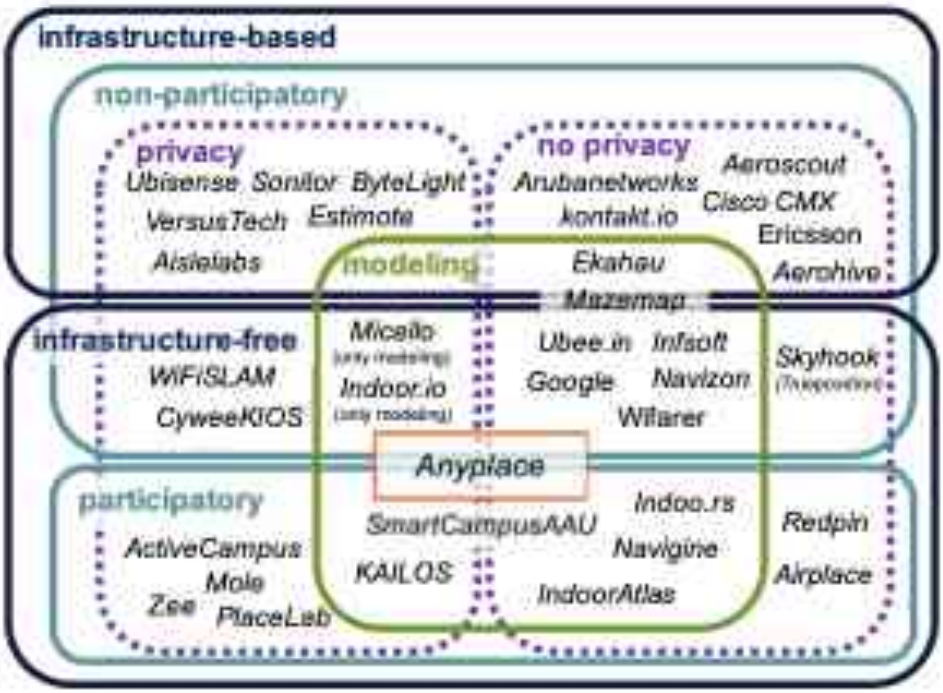

Figure 2.3 Taxonomy of Internet-based Indoor Navigation (IIN) services [3].

Infrastructure-based:

Systems that require dedicated equipment for the specific purpose of locating items or people are known as Real-Time Locating Systems (RTLS). The equipment used typically comes in the form of tags or badges placed on objects, beacons that serve as anchors in the environment and a central processing system for the location data. These systems utilise various technologies such as active or passive Radio Frequency Identification (RFID), Bluetooth Low Energy (BLE Smart) beacons, Wi-Fi beacons or Ultra-Wide-Band (UWB) chips.

Such systems offer superb accuracy but usually come with a high cost and in the form of very specialised solutions. Some infrastructure-based system designs such as passive RFID [20] have shown promise in reducing the overall cost of such systems but they still come at the cost of some accuracy. These Indoor Navigation Systems have proven their worth in numerous applications where precise real-time information is crucial. Overall, they have greatly contributed to the efficiency, productivity and safety of industries such as healthcare, manufacturing, logistics and retail.

Infrastructure-free:

IIN solutions that don't require dedicated equipment or infrastructure for the purpose of localization. They do so by exploiting already present infrastructure that serves a different purpose. The typical infrastructure found in such systems are wireless communication and cellular base infrastructure. These systems are capable of serving as IIN solutions by having dedicated software at the device side that uses Received Signal Strength (RSS) measurements of the available Wi-Fi or cellular beacons in combination with other sensory data from onboard Inertial Measurement Units (IMU) to provide localization estimates. Another solution implements Computer Vision which has shown to be effective in specific applications such first responder localization for fire suppression on Ro-Ro vessels [21].

Such systems are significantly cheaper to implement compared to their infrastructure-based counterparts. They are also very scalable, easy to implement and have shown to give comparable accuracy to RTLS solutions.

Privacy:

Since an indoor navigation service uses personal information such as the user's location, there are privacy concerns in regard to such technologies. Some Indoor Navigation Services choose to address such concerns whilst others don't. The way privacy is typically protected is decided by where the algorithmic localisation is executed. If this is done at the server-side, then the service provider has access to the end-user's location.

The distinction that can be made for IIN systems based on privacy stems from where the localization algorithm and the indoor signal measurement data are located. The two main scenarios are:

- If the localisation is done at the terminal device, then this prevents the Indoor Navigation service provider from tracking your device. By simply withholding the critical information from the server the device can preserve its location privacy.
- If the location estimation is done over the Network infrastructure. Devices such as iPhones or Windows phones do not have the ability to natively offer a way of gathering signal measurements, something that android phones can do, which gives them limited choices in IIN services when they want to keep their privacy. In this scenario the provider of the service can monitor the user's MAC address which fundamentally violates his privacy.

Crowdsourcing:

Another aspect of IIN systems is how the database's data was populated. The main distinction here being whether it was by experts or non-experts.

Participatory:

When it comes to data collections it has become quite common to outsource the task to common users of the system. This has the effect of significantly decreasing maintenance costs and also improving the scalability of the underlying system. Given however the unpredictable nature of common users, the data collected must be filtered to avoid data contamination by false or inaccurate measurements which would consequently lower the efficiency and accuracy of the system. An example of such system is the Anyplace [12] indoor navigation system, which by using the help of students was able to crowdsource the data collection of several buildings.

Non-participatory:

Some IIN systems are designed in such a way that trained personnel are needed to gather the data for the system. The systems that employ such methods are usually Real-Time Locating systems that find application in the Manufacturing industry or in large Enterprises. Due to the data becoming outdated over time, the system can become increasingly inaccurate and this creates a maintenance overhead which needs to be accounted for by the system operators. This reliance on trained personnel for data collection introduces challenges in terms of cost, time and complexity.

Modelling:

As a final consideration when classifying a IIN system we have the way pathing is calculated for the indoor spaces. The nature of indoor spaces makes navigating them a complex problem that typically relies on graph algorithms to solve. By annotating points of interest within an indoor space (e.g., walkways, rooms) the navigation problem becomes graph based. By employing graph algorithms an appropriate navigation path can be generated. These services typically use client-side libraries to provide the necessary tile maps and POIs for navigation.

2.2 Background

2.2.1 AndroidX

AndroidX [10], the new namespace, emerges as a substantial advancement supplanting the now outdated Android Support libraries. Comprising the Android Jetpack libraries, AndroidX boasts backward compatibility with its predecessor. Each library within AndroidX undergoes individual updates and maintenance, ensuring discrete versions. Commencing from Version 28 and beyond, AndroidX completely supersedes the old system. This revolutionary introduction by Google aims to furnish developers with enhanced functionalities, improved performance, and a more stable foundation for backward compatibility in the realm of Android app development.

AndroidX distinguishes itself with a modular and streamlined approach, distinguishing it from its predecessor, the Android Support Library. It presents an all-encompassing suite of libraries designed to assist developers in diverse tasks, including UI design, data management, testing, and beyond. These libraries harmoniously operate across various Android versions, empowering developers to craft code that seamlessly adapts to different devices and operating system iterations.

Key features and benefits of AndroidX encompass an array of notable aspects. Foremost among them are the Jetpack Components—libraries intricately crafted to simplify common Android development tasks. These components encompass diverse realms, spanning navigation, data persistence, app compatibility, lifecycle management, and more.

AndroidX places significant emphasis on backward compatibility, ensuring developers can leverage the latest features and APIs while still retaining compatibility with older Android versions. This facet empowers developers to harness the full potential of cutting-edge advancements without leaving behind users on earlier platform iterations.

Optimized performance stands as another highlight of AndroidX, with libraries finely tuned to execute code faster and more efficiently. This optimization results in heightened overall app performance and improved user experiences.

Moreover, AndroidX introduces a simplified packaging approach, enabling developers to include only the specific components necessary for their app. This approach translates into reduced app package sizes, facilitating smoother distribution and enhanced storage efficiency.

Finally, AndroidX provides comprehensive support for the Kotlin programming language—a rising star among Android developers. This seamless integration of Kotlin into the AndroidX ecosystem bolsters the language's popularity and offers developers increased flexibility and productivity when building Android apps.

2.2.2 Fingerprint based localization

Fingerprint-based localisation is a core concept that revolves around the meticulous collection of signal measurements within a designated area of interest. These measurements, encompassing signal strength and more, are collected to construct a comprehensive reference database. This database forms the foundation for establishing the important associations between locations and their distinctive signal fingerprints, effectively manifesting as a radiomap, the file that encapsulates crucial insights into the signal sources, such as WiFi routers.

When a device needs to know its location, it retrieves signal measurements from its current position and compares them to the stored fingerprints in the database. By identifying patterns that closely match the observed signal readings, the device utilises sophisticated algorithms like KNN (K Nearest Neighbour) to accurately estimate its precise location.

The accuracy of fingerprint-based localisation hinges upon an array of critical factors, with paramount importance placed on the density of reference points, the quality of signal

measurements, and the influence of wireless interference prevalent in the environment. To attain a desired level of accuracy, a larger number of reference points is often required, while the database necessitates periodic updates to adapt to changes in source positioning or notable transformations within the broader wireless landscape.

2.2.3 Inertial navigation

We can use the inertial sensors on the phone to create an inertial navigation system, however this poses the problem of having a significant amount of inaccuracy [6]. The accelerometers and gyroscopes on a smartphone are designed with size constraints. This leads to there being several reasons for inaccuracies when used for inertial navigation on smartphones.

1. Drift. This is a phenomenon where the error in the accelerometer and gyroscope accumulates over time. At some point it reaches an extent that even when the device is stationary there is movement in the estimated position compared to the actual position.
2. Noise. Given the design of these sensors when compared to more professional ones, they produce a certain level of random data that is considered noise since it does not reflect any actual change in movement of the device. This noise can accumulate in the measurements over time and cause significant inaccuracies.
3. Vibration and shock. Given that real world movements can be quite radical for the sensor to properly measure. Movements such as high frequency vibrations or shocks from rapid acceleration and deceleration, will result in inaccuracies in the measurements.
4. Calibration Error. The smartphones that come with these sensors need to be calibrated when produce. This calibration is not possible to reach a perfect level in the mass production of smartphones. This results in the devices not being properly calibrated from the start. Over time other factors can also influence the calibration of the sensors such as temperature changes.

Although using purely inertial navigation on a smartphone is not enough to create a reliable indoor navigation system, the integration of this type of navigation with a different one such as signal fingerprinting can result in significantly more accurate navigation systems.

2.2.4 Gradle

Developing software applications is a multifaceted endeavour, demanding a sophisticated solution to streamline the process. Gradle [9] is an acclaimed open-source build automation tool purposefully designed to simplify the intricate tasks of building, testing, and deploying software. Renowned for its adaptability and rich feature set, Gradle has emerged as a favoured choice among developers worldwide.

At its core, Gradle boasts an exceptional Build Automation functionality, captivating users with its remarkable capabilities. Leveraging a domain-specific language rooted in Groovy or Kotlin, developers can craft scripts that automate diverse tasks—compiling source code, running tests, packaging applications, and generating artifacts. These scripts can be tailored to execute precise tasks, effortlessly accommodating essential dependencies crucial to a seamless development workflow.

Effortless dependency management is a key strength of Gradle. Seamlessly accessing repositories such as Maven, JCenter, Google, or custom repositories based on project specifications, Gradle ensures the availability of vital dependencies throughout the build process, effectively mitigating potential obstacles.

An expansive ecosystem of plugins distinguishes Gradle as a standout tool. These plugins furnish supplementary capabilities and seamless integration with various frameworks, catering to an extensive array of requirements and project types encompassing Java, Android, Kotlin, Groovy, and beyond. From comprehensive code analysis to automated documentation generation, these plugins empower developers across diverse domains.

Notably, Gradle's intelligent build process delivers unparalleled efficiency gains. By intelligently rebuilding solely the components that have undergone changes since the previous build, Gradle significantly accelerates build times. Moreover, incorporating sophisticated caching mechanisms facilitates the reuse of compiled resources, augmenting productivity and performance.

When embarking on multi-project builds, Gradle shines in its adept resource management, meticulous oversight of the overall build structure, and robust mechanisms for seamless dependency and resource sharing across projects.

2.2.5 JitPack

JitPack [11] is an online project and package repository for Gradle and Maven projects. It simplifies the process of compiling a GitHub project to extract the artefacts that will be used by developers to integrate into their projects. It is free to use for public repositories. It works in conjunction with the Gradle and Maven build automation tools. The popularity of these two tools increases the appeal of JitPack over other alternatives

Chapter 3

Architecture

3.1 Architecture overview

The A4IoT architecture represents a cutting-edge evolution of the esteemed Anyplace software stack, purpose-built to address the unique demands of IoT (Internet of Things) applications. Within its framework, a range of back-end and front-end components collaboratively operate, seamlessly integrating to fulfil the diverse requirements of this dynamic domain.

At its core, the Web component stands tall, encompassing vital modules such as Architect, Viewer, and Analytics. Architect serves as a robust web application, empowering users to effortlessly introduce buildings by uploading architectural floorplans and actively engage in the interactive design of routes and Points of Interest (POIs). Viewer, on the other hand, establishes itself as a formidable search and navigation engine, harnessing the power of crowdsourced data to unlock comprehensive location-based insights. Finally, Analytics steps forward as an illuminating data visualization dashboard, forging a

seamless connection with the Data Store and furnishing invaluable perspectives into the stored data.

Meanwhile, the Library component emerges as a beacon of versatility and efficiency, catering to the intrinsic needs of IoT applications. Armed with an extensive repertoire of tools, APIs, and libraries, the Library component empowers developers to swiftly construct IoT-driven applications with utmost dexterity and effectiveness, fostering modular design and reusable functionality.

Within the IoT Clients realm, an array of application possibilities flourishes, each intimately tied to the Library component's remarkable capabilities. Noteworthy among these is the Logger—an indigenous Android application that empowers users to seamlessly crowdsource Wi-Fi RSSI (Received Signal Strength Indicator) values. Enabling users to record Wi-Fi readings from nearby access points and conveniently batch-feed them to the Server for further analysis, the Logger reigns as a pivotal tool for data collection and processing. Further augmenting the IoT Clients domain is the Navigator—a native Android application that elevates the indoor navigation experience to unprecedented heights. By harnessing the prowess of Wi-Fi RSSI values in tandem with additional sensor inputs such as the accelerometer, compass, and gyroscope, the Navigator ensures unrivaled accuracy and user satisfaction in the realm of indoor navigation.

Stepping back to examine the larger picture, the A4IoT Localisation Architecture stands as a comprehensive solution tailored explicitly to the unique requirements of IoT applications. Its unwavering focus on delivering impeccable indoor localization, seamless navigation, and the ability to thrive even in remote areas or infrastructures resistant to conventional GPS localization systems marks it as a transformative force within the IoT landscape.

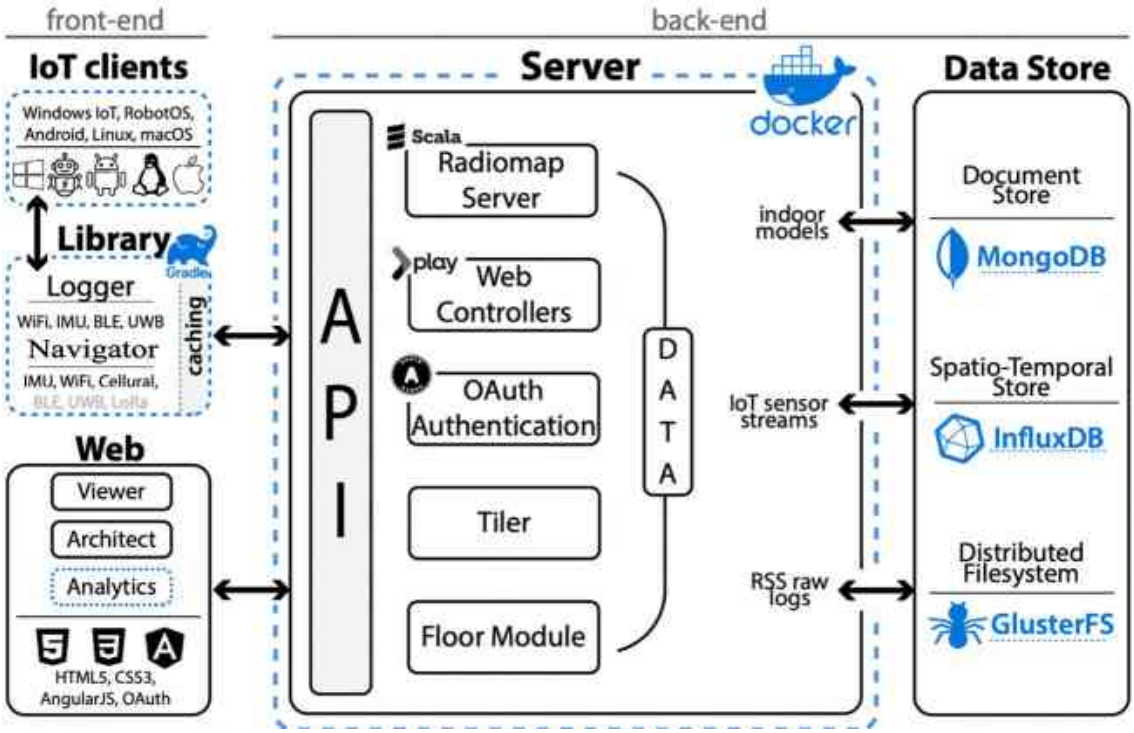

Figure 3.2 Overview of the Anyplace A4IoT Localization Architecture

3.2 Front-end

3.2.1 Anyplace Viewer

The Anyplace Viewer [1] is a web application that allows users to quickly visualize buildings modelled in Anyplace without the need for installing a separate app. It is ideal for a first-time user that doesn't want to invest considerable time before launching the service through an app downloaded from a popular mobile market. By simply using a web browser, users can access the Anyplace Viewer without the need for installations or special hardware.

The Viewer offers off-the-shelf usage, eliminating the need for app installation. Users can access it directly through a web browser, making it easily accessible across various devices without additional downloads.

The Web App was designed for mobile users with a good user experience in mind and so it incorporates things such as large interactive buttons, a clutter-free design and straightforward UI. The application is an efficient way to explore and navigate through buildings without much effort.

The Viewer is however lacking in more advanced functionalities such as better accuracy or caching which are provided by the Navigator app. Since it is a web application the Viewer also requires an internet connection to access the server which provides the building floor plans.

Figure 3.3 Web based Anyplace Viewer [1]

3.2.2 Anyplace Architect

The Anyplace Architect [1] is a web application designed for managing indoor models in Anyplace. The application enables users to overlay building blueprints onto Google Maps, with support for multiple floors. The floor editor feature allows users to upload, scale, and rotate blueprints to accurately fit them within the model.

Users can add, annotate, and geo-tag points of interest (POIs) inside the building and establish connections between them to facilitate navigation instructions. The user can do this through cross-browser compatible drag-n-drop functionality, which can even be used on tablets and smartphones during field deployments for on-the-go adjustments.

In addition to its foundational capabilities, the Anyplace Architect encompasses an array of features that further enhance its utility. Among these notable functionalities is the ability to diligently monitor the progress of crowdsourcing endeavors by proficiently

collecting Wi-Fi radiomaps and presenting them in the form of visually informative color heat-maps. This powerful feature enables assigners to meticulously evaluate the quality of the crowdsourced data, drawing upon predefined acceptance criteria to discern the adequacy and fidelity of the collected information. By empowering users to gain comprehensive insights into the effectiveness of crowdsourcing initiatives, the Anyplace Architect furnishes an indispensable tool for meticulous assessment and data-driven decision-making.

The Architect also provides options for setting a building as public or private. Public buildings are shared on the Anyplace Viewer interface, while private buildings can be shared with specific users via a URL. The application also supports exporting and importing indoor models and radiomaps, enabling backup/restore functionality, expediting POI input, and facilitating the creation of new models for different purposes.

Figure 3.4 Anyplace Architect [1]

3.2.3 Library

The Library component serves as a powerful and versatile resource for developers working on IoT applications. Its modular Java library specifically caters to the development of client-side indoor navigation applications. When combined with a server-side API platform for indoor navigation, this library becomes exceptionally valuable. It serves as the intermediary between the client-side application and the server API, seamlessly facilitating communication and interaction.

By providing a compact and concise Library, it empowers developers to streamline their development process. Its inclusion simplifies the implementation of indoor navigation

features and enhances the overall functionality of the A4IoT platform. Consequently, this library significantly elevates the appeal and desirability of the A4IoT platform among developers, as it provides a seamless and efficient solution for incorporating indoor navigation capabilities into their projects.

Language	Type	Published
Java	Library (anyplace-core)	Gradle
OS	**Type**	**Published**
Linux-based	Tool	-
macOS	Tool	-
Windows	Tool	-
Android	Library (anyplace-android)	Gradle
Android	Application	Google Play
ROS	Application	Robot App Store

TABLE I: List of libraries, applications, and tools that are developed in the scope of the A4IoT architecture.

Figure 3.4 List of Libraries in A4IoT [1]

3.2.4 IoT Clients

The IoT clients utilise the available libraries in a layered Java-based code structure to deploy a desired application that interacts with the A4IoT platform to deliver its intended functionality. One such available app is the Logger app, capable of creating signal fingerprint logs to populate the Data Store of a specific building. Another such app is the Navigator. The Navigator plots courses inside buildings and structures for the users to navigate the Points of Interest in them.

3.3 Back-end

3.3.1 Data Store

The Data Store [1] component is responsible for storing and retrieving data. It utilizes a Distributed Filesystem (DFS) for storing raw data, a time-series store for IoT data, and a document store for JSON objects. This allows efficient and scalable storage and retrieval of different types of data.

In terms of the time-series store, the incorporation of a time-series database, specifically InfluxDB, enables real-time analytics and visualizations. InfluxDB is chosen for its suitability in write-intensive workloads and its ability to handle spatio-temporal queries efficiently. This enhancement enables the consumption of high-volume input streams, providing real-time IoT tracking capabilities. The document-store aspect of the Data Store utilises MongoDB, an efficient document storing option in terms of resource consumption. The Distributed Filesystem part of the Data Store is handled by the GlusterFS which handles the raw RSS logs of the system.

3.3.2 Server

The Server [1] component handles the application logic of the A4IoT service. It is implemented using the Play [13] framework, a web application software design framework based on the MVC (Model-View-Controller) architectural pattern. The Server exposes a RESTful API that enables functionalities such as crowdsourcing, spatio-temporal queries, and interfaces with various data stores.

Chapter 4

Anyplace-core Library

A simple bare-bones modular Java library for developing client-side indoor navigation applications based on the A4IoT platform. This Java library provides other java projects a concise way to interact with the API of the server. By leveraging Gradle's build automation capabilities, this library can easily be imported into a plethora of projects by the developer. As such it makes the use of the A4IoT platform more desirable by developers.

4.1 Important Considerations

The library was created with a lot of considerations in mind. Programming an indoor navigation service can be a very daunting task for any developer and so the choice to

simply adopt an already existing service is very appealing to many. With the potential users of this library in mind, we attempted to implement these considerations to the best of our ability.

Main considerations:

- Simplified Development
- Seamless Integration
- Consistent Development Experience
- Customisation and Extensibility
- Compatibility Issues

4.1.1 Simplified Development

Companies and organisations have limited time and resources to develop new software. We decided that a simple interface was a good choice for that. It's simple design, minimises integration time into projects. We also provide ways for testing and debugging the system so that the developer can avoid any delays in their workflow.

4.1.2 Seamless Integration

In the current rapidly evolving technological age, companies find themselves adopting a best-of-breed approach in their search of software solutions to any problem they want to solve [18]. This creates the need for software developers to design software that can easily integrate into other systems. With this consideration in mind, the library uses data formatting, like JSON, to make exported information more approachable.

4.1.3 Consistent Development Experience

By utilising the library, developers can ensure a consistent debugging experience across different client applications. The library can provide standardised functions, interaction patterns and testing methods to create a more consistent development workflow.

4.1.4 Customisation and Extensibility

Each user is unique, and so are his requirements. Since it’s practically impossible to have a library that can do everything that the user might require, it’s best to focus on only what can be considered essential. The user/developer can fill in the rest by extending the functionality that is offered.

4.1.5 Compatibility Issues

The library would need to offer executability to users that run on a multitude of Operating Systems like Windows, Linux or macOS. Since the devices that will implement the use of the A4IoT platform are in the sphere of Internet of Things then it’s important to consider that the device using the library can fall under a very niche section of operating systems. We have taken this into consideration in order to offer Command-Line Interfaces in some versions of the library.

4.2 Technical Implementation

4.2.1 Code Design Pattern

The architecture of the Anyplace-core library bears a striking resemblance to the Façade Pattern, showcasing its design elegance and practicality. The library adopts a centralized approach, where a single prominent class assumes the role of the facade, serving as the gateway to access all of the library's functionality. This thoughtful design choice greatly simplifies the usage and interaction with the library, fostering a user-friendly and intuitive experience for developers.

Drawing inspiration from the widely adopted Façade Pattern in object-oriented programming, the Anyplace-core library embodies the essence of this design paradigm. In the Façade Pattern, a key class, referred to as the "facade," emerges as the central hub that encapsulates the complexities of a subsystem or a collection of interrelated classes.

Acting as a streamlined interface, the facade shields the clients or other classes from the intricate details of the subsystem, providing a cohesive and easily navigable entry point.

Within the context of the anyplace-core library, the role of the facade is assumed by the Anyplace class. This pivotal class orchestrates the interactions between the clients and the underlying subsystem, effectively simplifying the overall usage and integration of the library. By abstracting away the complexities of the subsystem, the Anyplace class offers developers a unified and coherent interface, empowering them to effortlessly leverage the rich functionality of the library in their Java-based codebases.

4.2.2 Dependencies

The anyplace-core library, an integral component of Java, is underpinned by four dependencies that are central to its operation. This library was crafted using OpenJDK version 11.0.2 in combination with the OpenJDK Runtime Environment 18.9 (build 11.0.2+9). These dependencies denote external libraries that the anyplace-core library leverages to fulfil precise tasks.

- OkHttp (version 4.3.1) [16]
- Okio (version 2.4.1) [15]
- Json Library (version 20180813) [17]
- JUnit (version 4.13) [14]

The OkHttp library, a preeminent HTTP client library within the realm of Java applications, is employed by the anyplace-core library to establish a link with the Server API through the generation of POST and GET requests.

The Okio library, a compact yet potent I/O library for Java applications, facilitates the decompression of response data retrieved from the Server API within the anyplace-core library. It is instrumental in the extraction of data that is delivered in the gzip format.

The json library, a renowned library for manipulating JSON (JavaScript Object Notation) data in Java, is relied upon by the anyplace-core library to parse and process the extracted response data received from the Server API.

Lastly, JUnit, an extensively utilized testing framework within Java applications, is deployed in the anyplace-core library to formulate test cases. These ensure the accurate and reliable operation of its diverse components and attributes.

4.2.3 Codebase Structure

The anyplace-core Java library has a codebase consisting of 10 classes, the Anyplace, Algorithms, AnyplaceException, LocDistance, JsonHelper, LogRecord, Preferences, RadioMap, RestClient, and UnzippingInterceptor.classes that work together to provide functionality for accessing the A4IoT localisation platform. The main class is the Anyplace class, which acts as the interface to the library's capabilities. It communicates with the Anyplace Server API using the RestClient class, and it can cache floor plans and radiomaps locally for offline usage.

The Algorithms class supports offline localisation and is called by the Anyplace class. It utilizes the LocDistance class to store distance information from signal points recorded during the localisation process. The AnyplaceException class handles exceptions related to file or directory creation during caching.

The RestClient class uses the OkHttp library to generate POST and GET requests. The server API uses mostly POST requests. The responses are compressed in the gzip format. The UnizippingInterceptior class decompresses the response and then using methods from the JSON library it is parsed. See Figure 4.1.

Other supporting classes include JsonHelper for parsing server responses, LogRecord for storing BSSID and RSS information for device localisation, Preferences for reading local files containing server request information, and RadioMap for holding Radiomap data for offline localisation.

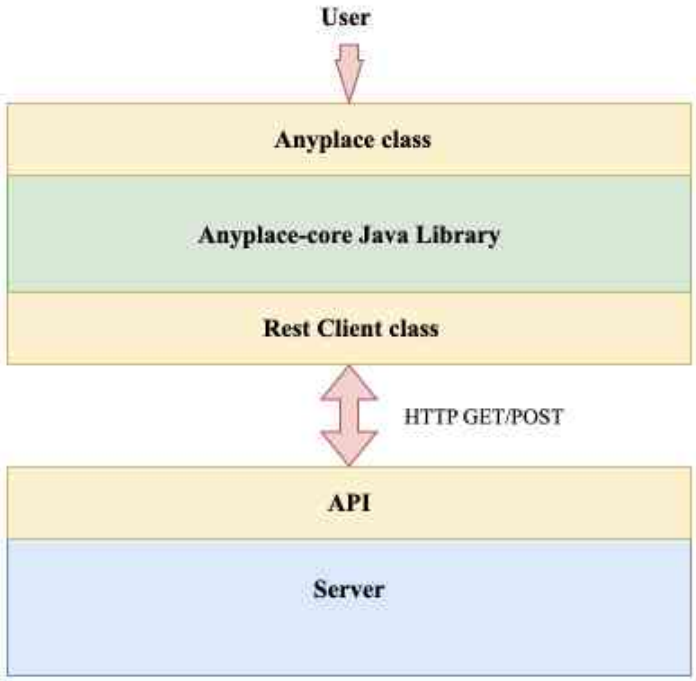

Figure 4.1 Anyplace-core server communication diagram.

4.2.4 Exception Handling

A major consideration when writing any piece of code is error and exception handling. The way exceptions are handled can vastly affect development times. The library implements error handling in the class that encounters the exceptions.

When dealing with a lot of I/O operations it is expected to run into errors regarding reading and writing. Socket exceptions and timeout exceptions are also a common example of those likely to be encountered by this type of library. As such we implemented specialised handling for this kind of exceptions in order to have a clear understanding of the origin and cause.

We have ensured the reliability and robustness of the library by testing it against a variety of scenarios in order to comprehensively shield it from any potential bugs in the future.

By implementing a comprehensive exception handling framework throughout the library's codebase, we ensure its resilience and reliability when faced with errors and exceptions. This holistic approach strengthens the library's ability to navigate different scenarios and promptly recover from exceptional conditions that may occur during I/O operations and server communication. Ultimately, the conscientious implementation of error and exception handling enhances the library's robustness, enabling it to adapt seamlessly to various circumstances.

4.2.5 Unit Testing

macOS:

The TesterMac class verifies the compatibility of the library with macOS platforms. It manly tests the library’s ability to take the signal measurements of the Wi-Fi network and parse them to execute the API endpoints.

Endpoints:

The Tester class’s purpose is to check if the library can access all Server endpoints and can properly execute the code to parse the data retrieved. The class takes as parameters the access token, a building id, a floor, coordinates, two points of interest, algorithm choice and signal fingerprint measurements to test all the endpoints.

Local file functionality:

The TesterPreferences runs the same endpoint tests as the Tester class, only this one instead of having the parameters be included in the class it reads them from a file.
More test scenarios can be written depending on the features that need to be implemented.

4.3 Integration Steps

In order to use the JitPack dependency through the gradle files of the Android project you have to include the following lines into the top level build.gradle file of the Android project.

In the build.gradle file of the module that you wish to have the library dependency you must include the following line.

```
implementation 'com.github.dmsl:anyplace-lib-core:4.0.2'
```

More documentation available on GitHub [19].

4.4 Command-Line Interface

The library offers a CLI for integration with other projects, Java-based or not. We envisioned the possibility of integration in different IoT systems. The command-line nature of the interaction makes it language and platform independent. Shell scripts are also capable of using the library and this can be used with piping the output to a different library. This in turn can make the use of the library independent of a project in development. It can simply be used externally through a shell.

The library's CLI was tried and tested on Linux and macOS, making sure that the signal measurements generated from the operating systems can be properly parsed. This cross-platform compatibility allows developers to integrate the anyplace-core library into their projects running on such operating systems, expanding the range of applications where the library can be used.

The CLI provides a convenient way to develop indoor localisation clients with minimal effort. By interacting with the anyplace-core library through the CLI, developers can quickly utilize the library's indoor navigation capabilities without extensive integration or modification of their existing project's codebase.

A local Preferences file can be created that contains the user's id, access key and algorithm preference. When provided alongside the building, floor and RSSI readings the user can localise themselves.

Figure 4.5 Localisation on Linux using the A4IoT CLI [4]

Chapter 5

Android Java Module

In this chapter we will be discussing the Anyplace Android library and the android applications that utilize it. The basis for the functionality offered by the library is the refactored code that was developed for the existing anyplace android application. That code was tuned to comply with AndroidX libraries and can be built in newer sdk versions.

Our library has turned to the new AndroidX library framework to make sure it can stay compatible with newer versions of android. The library is a Java Module and works as an expansion on the anyplace-core library discussed in the previous chapter. It includes the basic code needed to run the A4IoT platform on an android device.

5.1 Android-specific Functionality

The anyplace-android library, an extension of the anyplace-core library, introduces a range of supplementary features that significantly augment the indoor navigation experience on Android devices. These features leverage the inherent capabilities of the device to deliver a range of functionalities while adhering to the A4IoT design principles tailored for mobile environments.

Logging BSSID and RSS Measurements:

The library includes classes for logging Basic Service Set Identifier (BSSID) and Received Signal Strength (RSS) measurements. This is the code associated with the Logger logic. Developers trying to build a crowdsourcing application for data collection will find this very useful.

Base Navigation Logic:

The Java Module provides the base navigation logic required for building navigation and tracking applications. The functions provided allow for the ease of use of building and floor models along with the POI data associated with them.

Downloading and caching building data:

Necessary code for downloading and caching building models is provided. By allowing the user of your application to used locally stored data along with its signal measurements for the location estimation, you avoid violating his location privacy.

Motion Detection and Orientation:

Utilising the IMU of the android device, the application can programmatically estimate if the user is walking or standing still. This estimation in conjunction with the location estimate from Signal Fingerprinting allows for a real-time navigation feature.

Asynchronous Server API Calls:

Android applications require the use of asynchronous calls for any function that would take an indefinite amount of time to complete. Such as an HTTP request. Implemented from refactored legacy code, they provide a prime example of the modular nature of the anyplace-core library.

WiFi Signal Measurements:

Android offers the ability to read Receive Signal Strength measurements of Wi-Fi routers. The library provides the way to collect and use that data for the purpose on localisation, navigation or logging.

5.2 Handling Permissions

For the Android module, the network connectivity and Wi-Fi access are needed for its features. As such we must request the following permissions.

ACCESS_NETWORK_STATE:

Gives access to the ConnectivityManager that handles information about networks.

ACCESS_WIFI_STATE:

Allows the application to access the WifiManager. From the WifiManager the application can get Signal Strength measurements that are used for location estimation.

CHANGE_WIFI_STATE:

This permission enables the app to change or modify Wi-Fi connectivity settings. In the Navigator and Logger apps if the network connectivity check determines that there is no internet connection, the app will prompt the user to enable Wi-Fi.
The APP Module (Figure 5.1) needs more permissions beyond these to function correctly. These permissions are required for only the Features Module.

5.3 AndroidX Refactoring

With the introduction of AndroidX, a transformation unfolded within the android development community. The structure and use of the Android Support Library were revamped with the aim to shift to a more modular, up-to-date and backward-compatible Support library. Consequently, the existing Android applications for Anyplace required a refactoring in order to adopt the new suite of libraries and thus build to a higher SDK version.

New SDK versions use the AndroidX libraries which means that in order to build to such version developers must resolve the namespace conflicts that arise from the existence of similar naming in the old and new support libraries. Due to the deprecation of the old libraries, there is no longer support offered thus, to keep using such libraries we must change to the AndroidX backward-compatible library.

Google provides more frequent updates and bug fixes for AndroidX libraries. We get better modularity, selective inclusion of specific libraries, better build times, reduced

APK size and better compatibility with different Android versions. Google has deprecated the previous support library so to take advantage of new improvements to the android support libraries we must migrate to AndroidX.

5.4 Integration Steps

In order to import the Android Java module into an android project you have to use the JitPack package repository through Gradle in order to download the dependency. In the root build.gradle of the Android project you add the code shown in the Figure 5.1.

Figure 5.1 JitPack package repository for Git

Following that you go to the desired module within the project that will have the anyplace-android library as a dependency, and you add the following line.

```
implementation 'com.github.dmsl:anyplace-lib-android:v4.0.1'
```

After adding the code make sure to sync the project with the Gradle files.
It is also possible to use the tag main-SNAPSHOT to get the latest version on GitHub.

5.5 IoT Client-side Applications

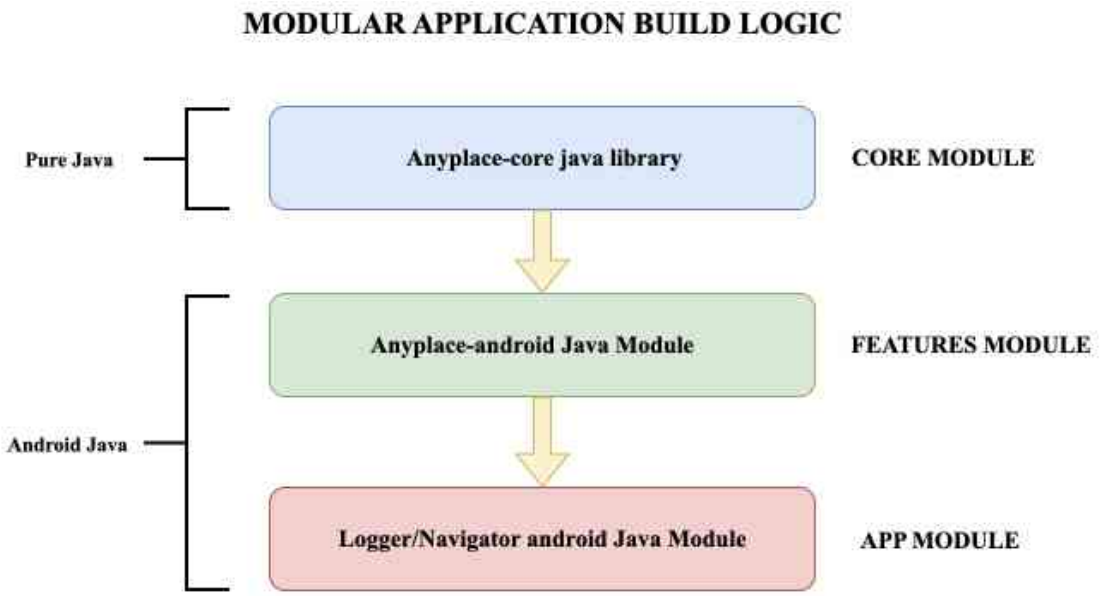

Figure 5.1 A4IoT client build layers

To complete the IoT client applications a modular approach was used to build them. Three Java modules are required in our case to build a functioning A4IoT client application. These three modules each serve their own specific purpose. One is the Core Module, the second is the Features Module and the third is the App Module. See Figure 5.1.

- Core Module: This is our anyplace-core java library. It serves as the core providing the basic functions that belong to the client-side of the A4IoT architecture.
- Features Module: This is the anyplace-library Java Module discussed in this chapter. It builds on the core functionality of the Core Module by providing features that stem from the capabilities of either the device or the android platform.
- App Module. This is either the Logger, Navigator or a Module of your development. It provides the layout for the app along with the Activity codebase that put the Features Module and Core Module to use, bringing forth a client-side application native to the A4IoT ecosystem.

5.3.1 IoT Clients

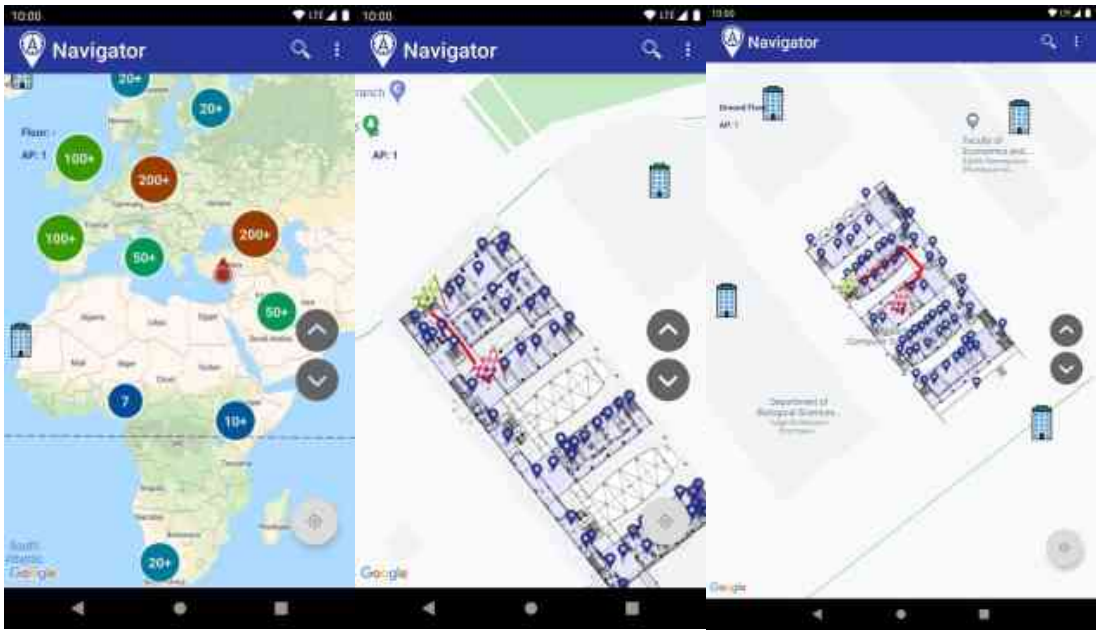

Figure 5.2 A4IoT client Navigator

The Navigator App demonstrates how to utilise the Anyplace-android Java Module to make an application for indoor navigation. Using the Java Modules provided you have access to a ready-made navigation algorithm and the sensor logic to pair it with real time movement.

Activity	Uses Anyplace-android
About	No
Preferences	Yes
Search POI	Yes
Search Building	Yes
Settings	No
Start	No
Unified Navigation	Yes

Table 5.1 Activities that utilise the anyplace-android Java Module.

The Navigator Demo allows the user to navigate through the buildings that have been registered on the Anyplace A4IoT server. These images show snapshots of the Demo App in use. On the left image we have a screenshot of the app showing how the registered buildings around the world are distributed. The bubbles have numbers in them that indicate the number of buildings in that area. In the middle image we have an example of how you would navigate from one point of interest to another. On the right image we have also got an example of navigation, but it is on a 10-inch device instead.

The Android application uses Gradle to import and manage the dependencies on the anyplace-android library.

The Java module for the application is a bundle of Activity classes and several xml files for the display format along with their respective resources. The essential navigation and localisation logic is in the other two Java Modules that are the anyplace-core Java library and Anyplace-android module respectively.

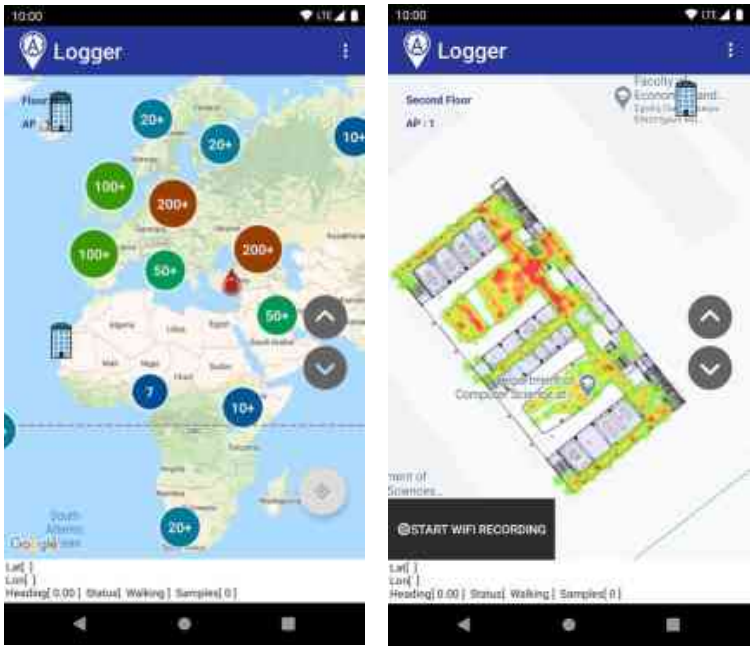

Figure 5.3 A4IoT client Logger

The creation of the Logger App demonstrates how the anyplace-android Java Module can be used to create an application for crowdsourcing signal fingerprints your A4IoT server.

5.3.2 Permissions

In the event of the A4IoT client application detecting the absence of internet connection on the device, a pop-up window is presented during the application's start-up phase. This prompt urges the user to activate his Wi-Fi connection by navigating to the settings tab on their device, ensuring access to the necessary network resources for an uninterrupted navigation experience. As it is not a far-fetched request, the user should comply.

1. Upon detecting the lack of internet access, the application triggers a pop-up window giving the user the choice to go to his device's settings to enable Wi-Fi or he can decline.

2. The pop-up window informs the user clearly of the lack of internet connection with the text "No internet connection. Would you like to change settings?". See third image in figure 5.2.

3. The user can select the button that directs them to the device settings or decline. When the user taps on the "SETTINGS" button, the application opens the device settings page where Wi-Fi settings can be modified.

4. When the user is back, the application then verifies the internet connection again and proceed accordingly based on the availability of a network connection. If the application finds that there is no internet connection, then the user is met with the pop-up message in figure 5.1.

Figure 5.4 Application has no internet connection.

If the application detects that location permissions are disabled, it displays a pop-up window, prompting the user to give access to the device's location as shown in Figure 5.2.

It is good practice to have graceful permission handling for any android application. One might also argue that an indoor navigation app would need to be even more mindful of its permission handling. Bad permission handling ruins application user experience. As a result, if the user feels that his device permissions are not handled appropriately his opinion of the app will deteriorate.

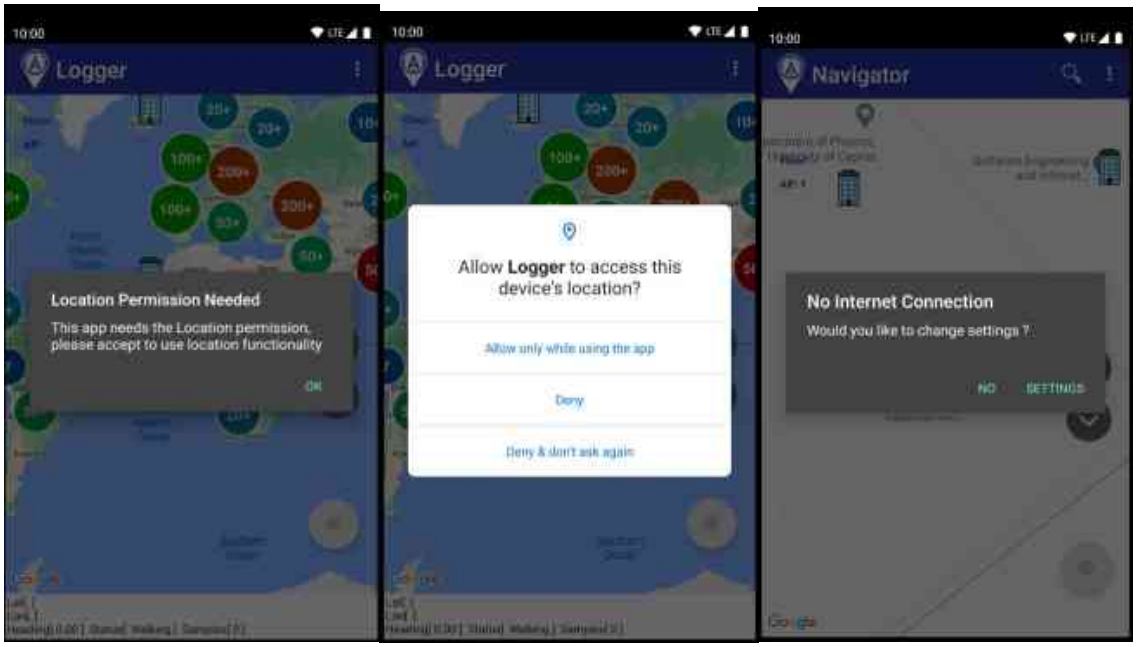

Figure 5.5 Enabling permissions.

5.3.3 Preferences settings

An Anyplace Preferences settings menu (Figure 5.6, left image) was created that implements a file saving feature for the Server IP, port and access token used to make server requests and for the caching or deleting of building data. Username and password are needed to upload RSS logs to the server.

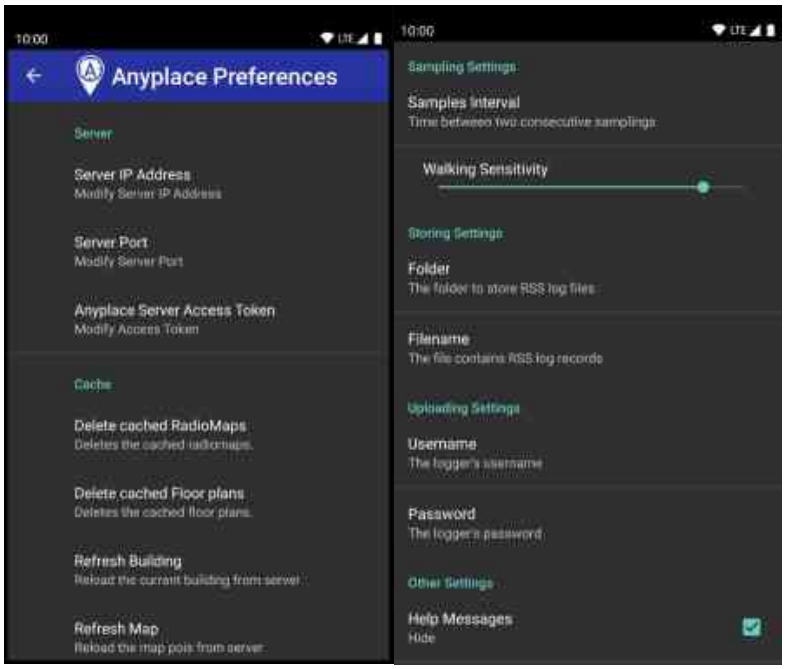

Figure 5.6 *App preferences menu. Right: Logger Preferences. Left: Server info and caching preferences.*

Chapter 6

Conclusion

6.1 Conclusions

In conclusion, this Thesis Dissertation provided a comprehensive exploration of the topic of Internet-based Indoor Navigation Services, with a focus on the development and implementation of the Anyplace library and associated Android modules, showing how a modular library is a valuable asset for client-side development.

6.2 Future Work

The A4IoT ecosystem is now at a point that it can expand its capability into more niche areas of indoor navigation. By pushing forward in the path of infrastructure-free systems, A4IoT can stand out as a leader of open-source navigation projects.

References

[1] "The Anatomy of the Anyplace Indoor Navigation Service", Demetrios Zeinalipour-Yazti and Christos Laoudias "ACM SIGSPATIAL Special" (SIGSPATIAL '17), ACM Press, Volume 9, Pages: 3-10, 2017

[2] Laoudias, C.; Constantinou, G.; Constantinides, M.; Nicolaou, S.; Zeinalipour-Yazti, D.; Panayiotou, C. G. (2012). "The Airplace Indoor Positioning Platform for Android Smartphones". 2012 IEEE 13th International Conference on Mobile Data Management. pp. 312–315. DOI:10.1109/MDM.2012.68

[3] "Internet-based Indoor Navigation Services", Demetrios Zeinalipour-Yazti, Christos Laoudias, Kyriakos Georgiou and Georgios Chatzimiloudis, IEEE Internet Computing (IC '17), IEEE Computer Society, Vol. 21, Iss. 4, pp. 54-63, Los Alamitos, CA, USA, 2017. DOI: 10.1109/MIC.2016.21

[4] "The Anyplace 4.0 IoT Localization Architecture", Paschalis Mpeis, Thierry Roussel, Manish Kumar, Constantinos Costa, Christos Laoudias, Denis Capot-Ray, Demetrios Zeinalipour-Yazti, Proceedings of the 21st IEEE International Conference on Mobile Data Management (MDM'20), IEEE Computer Society, ISBN:, pp. 218-225, June 30 - July 3, 2020, Versailles, France, DOI: 10.1109/MDM48529.2020.00045, **2020**.

[5] Sagar V. Ramani , Yagnik N. Tank. "Indoor Navigation on Google Maps and Indoor Localization Using RSS Fingerprinting", International Journal of Engineering Trends and Technology (IJETT), V11(4),171-173 May 2014. ISSN:2231-5381.

[6] Paolo Dabove, Giorgio Ghinamo, Andrea LINGUA "Inertial Sensors for smartphones navigation",

[7] Y. Gu, A. Lo and I. Niemegeers, “A survey of indoor positioning systems for wireless personal networks”, In IEEE Comm. Surv. Tutor., vol. 11, no. 1, pp. 13-32, 2009

[8] Google Indoor Maps, “https://www.google.com/maps/about/partners/indoormaps”

[9] Gradle main website: “https://www.gradle.org/”

[10] AndroidX website: “https://developer.android.com/jetpack/androidx”

[11] JitPack website: “https://www.jitpack.io”

[12] "Anyplace: A Crowdsourced Indoor Information Service", Kyriakos Georgiou, Timotheos Constambeys, Christos Laoudias, Lambros Petrou, Georgios Chatzimilioudis and Demetrios Zeinalipour-Yazti. IEEE Mobile Data Management (MDM '15), IEEE Press, Volume 2, Pages: 291-294, 2015

[13] Play Framework, "The High Velocity Web Framework For Java and Scala". "https://www.playframework.com"

[14] JUnit. https://junit.org/junit4

[15] Okio, "A modern I/O library for Android, Java, and Kotlin Multiplatform.". https://github.com/square/okio

[16] OkHttp. https://square.github.io/okhttp

[17] Json library. https://stleary.github.io/JSON-java/index.html

[18] Saidi, A. (2002). "Seamless integration". Paper presented at Project Management Institute Annual Seminars & Symposium, San Antonio, TX. Newtown Square, PA: Project Management Institute.

[19] Anyplace v4.0, "https://github.com/dmsl/anyplace/tree/v4.0"

[20] Ting SL, Kwok SK, Tsang AHC, Ho GTS. The Study on Using Passive RFID Tags for Indoor Positioning. *International Journal of Engineering Business Management*. 2011;3. doi:10.5772/45678

[21] P Mpeis, J Bleye Vicario, D Zeinalipour-Yazti. "ZERO INFRASTRUCTURE GEOLOCATION OF NEARBY FIRST RESPONDERS ON RO-RO VESSELS". International Conference on Computer Applications in Shipbuilding 2022, 13-15 September, 2022, Yokohama, Japan.

Appendix A