Thesis Project

A Management System for Objects in Indoor Spaces

Andreas Naziris

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

May 2023

University of Cyprus

Department of Computer Science

Andreas Naziris

Supervising Professor
Demetris Zeinalipour

The Thesis Project was submitted for the partial completion of the requirements in order to earn the degree of Computer Science from the Department of Computer Science of the University of Cyprus

May 2023

Abstract

People are spending a major amount of their time in indoor spaces. When these spaces are big and foreign to a person navigation and localization can prove to be very difficult. As such indoor localization solutions have been poping up to solve this particular problem. With the rise of these new solutions however a new different problem appeared. The ability to visualize, manage and analyze data generated by those systems. This project focuses on object data inside indoor spaces. Many indoor localization systems collect data about objects either in order to localize a user using these very objects or maybe only for the purpose of tracking objects in order to achieve automation inside smart factories and to improve efficiency.

In the present thesis project an android application was developed in order to combat the aforementioned problem and offer a solution. This Object Management System that was implemented offers a few key functionalities for a nice user experience in the data visualization aspect of the problem, with future hopes of object management as well. The developed system has the ability to collect object data from sqlite database files of a certain schema generated by third party systems, or APIs and present the collected objects on an interactive map. A few filters were added such as the ability to filter objects by what floor they are on or what type of object they represent. A search bar allows users to quickly and easily find the objects they want to find. Sharing utilities may provide users with the ability to communicate to third persons about some information on the application easily. Approximate location via GPS helps people orient themselves in the map.

Code performance was measured for the application and any delays for a moderately sized datasets showed that operation was possible without any hiccups and without even noticing the delay in many cases.

Contents

Chapter 1

Introduction

1.1 Motivation

More and more people are spending a very big chunk of their time in indoor spaces [26]. As a result navigating in indoor spaces can become a challenge when a person arrives in a space that is extremely large and unknown to him. In addition major proportion of the population is fluent in the use of portable devices, especially smart phones with an ever increasing screen time metric. For this reason indoor localization techniques, systems and solutions in general have a lot of potential and their use is rising.

A large amount of localization data is generated and this includes Points of Interest and object location information. An example can be the Smart Alert System or SMAS [11] which is an indoor localization solution using no infrastructure, and only built-in device cameras with the combination of computer vision to identify objects in a space and calculate the location of the user in a space. For this reason SMAS initially collects information about objects in an internal space recording their location and some of their attributes.

This generated object data is in need of analysis and management. This is the objective and purpose of our Object Management System android application or OMS. OMS was

planned to be developed into a user friendly application for intuitively visualizing and managing objects in indoor spaces.

1.2 Project Outline

In the first chapter of this thesis project the motivation behind the project is shown, then we present the outline of the whole paper analyzing all chapters with their segments. There is a segment of a brief overview of the OMS android application and lastly we mention what contributions were made through this project.

In the second chapter an overview of the background and related work is shown. This chapter is segmented into two parts, the academic background and the industrial background. In the academic portion four papers are presented and their summary is given. These papers deal heavily with the anyplace system as an Indoor Navigation System [5], mention its use as a zero infrastructure geolocation service [11], show the 4.0 IoT version that enhances its capabilities with IoT devices [12] and finally present a privacy preserving indoor localization approach [1]. Next up in the industrial part, 5 indoor localization and asset tracker systems are visited and their features are discussed. These systems are MazeMap [23] from which OMS takes heavy inspiration, Google Maps: Indoor Maps [19], Inpixon's Indoor positioning [21] and by extension the Intranav [22] smart factory system, proximi.io [24] and finally Mapsted [20].

Following in chapter three the requirements and the design of OMS are presented. The chapter starts with the basic architecture of the system. It then continues with a list of twenty proposed features and a detailed description of each one, these features are thematically split in five relevant categories. Next in line a description of the database schema is given with ER diagrams. Following a mockup and design of the system is shown in addition to the description of the interface. Afterwards non functional requirements like responsiveness, speed and user experience are noted. Ultimately the features that were actually implemented during the length of the project are described.

Continuing with chapter four the implementation is described. For this chapter, five features were chosen, for each one of these features the task is given in detail and then alongside short code snippets of said features an explanation is given for the code and how it works. Those five features are not all the features that were implemented for

OMS, there are a few more whose implementation is not discussed. For the five features that are presented the full code of the mentioned files is presented in the Appendix. At the end of each feature explanation relevant screenshots of the system in use are shown and described highlighting the implemented features.

In chapter five an evaluation of the implementation is given. For each of the described features in chapter four, the time complexity is calculated for their functionalities and then metrics of time taken to run the functions of said features are shown. At the end of the chapter some conclusions are drawn about the user experience and efficiency of the implemented features.

In the sixth and final chapter of the project we analyse a few conclusions we came to and discuss what can be done in the future to improve the current work while also presenting a few problems in the present version of the implementation of OMS giving a few possible suggestions for improvement.

1.3 OMS Overview

OMS or Object Management System is an android application that was developed during the length of this thesis project. This application provides a user with the ability to visualize objects and Points of Interest or POIs that were either inputted into Anyplace through the anyplace API or collected with SMAS the Smart Alert System and its computer vision approach for localization. These objects are shown in OMS on an interactive map.

A handful of features were built into OMS in order to make the experience for the user better. These include a search bar that can search through objects giving an emphasis to objects that are closer to the user. A share button for sharing locations via links with friends and other people, alongside this functionality there is the ability to open the mentioned generated links with the OMS application. In addition a user can select a category of objects to display on the map in order to gain a better understanding, by combining this functionality with heatmaps, of where different types of objects are situated and often found on the map. Floor selection is another functionality that has been added to OMS, it supports selecting a floor with objects in it and while the heatmap and search bar will only handle objects in the selected floor, the user can still

see where markers appear on different floors since the makers for these objects still appear on the map, however the markers for these markers appear in a lower opacity in order to differentiate them from the other objects.

The last feature of the OMS application is the ability to view the physical location of the user on the map, though the technology used is GPS, it can still be helpful, even though in the future it would be best to use the existent anyplace technology for accurate indoor geolocation. The location of the user is updated every 20 seconds and by pressing a button the user of OMS can be taken to his location.

1.4 Contributions

OMS was developed as an android application. This application at this stage allows the visualization of object data that can be gathered from sqlite database files generated by SMAS [11] and by Point of Interest data from the Anyplace API [12], Anyplace being the predecessor to SMAS and an indoor localization system that uses wifi fingerprints and IoT devices such as BLE beacons, RFID tags etc. The stage of OMS is still a very early one with limited functionality implemented but some baseline features have been recorded that can extend the abilities of OMS in the future.

Chapter 2

Related Work and Background

2.1 Academic background

2.1.1 Internet based Indoor Navigation Services [5]

In the paper Internet based Indoor Navigation Services (IIN) the authors give a few categories with which indoor navigation systems can be segmented and then give a very brief introduction of an IIN called Anyplace.

The categories examined by the paper were the following: localization meaning how much equipment is needed to deploy an IIN system, crowdsourcing which looks at how localization data is collected for a service, privacy where we see if a service can track its users or if user devices can locate themselves without giving localization data to the central localization system and modeling which means whether a user can model a building, POIs and routes.

Localization can be split in two categories, infrastructure based and infrastructure free. Infra based refers to systems that need the installation of additional hardware in order to work. Things like bluetooth beacons [16], RFID tags [13], UWB tags [6]. This solution offers some very clear disadvantages like increased cost for new equipment and its installation, and of course maintenance. On the other hand we have infrastructure free solutions that use existing WiFi access points and cellular towers with the combination of the multitude of sensors that modern smart devices have. Infra free often means the use only of the sensors on the device. Studies have showed that infra free solutions can be as good and in some cases even better than infra based approaches in terms of accuracy [7].

Next is crowdsourcing that can be separated in non participatory and participatory systems. In non participatory the people that collect the geolocation data are either paid professionals or a small team of trained personnel. This of course can get quite expensive pretty fast especially for large spaces and multi floored buildings, additionally data can get outdated pretty fast since new access points can be added or old ones can get removed or moved. On the contrary with the participatory approach the task can be split between a bigger group of volunteers, reducing the load and having up to date data being collected constantly [15].

Privacy is the following aspect that was mentioned. The location can be either calculated by the service with the consequence of it constantly knowing the location of its users, or calculated by the terminal device preserving the user's privacy. With

terminal based approaches an algorithm on the device calculates its location. This offers privacy, offline functionality and no network overhead, the drawback however is that this is taxing on the user's device and more battery draw is the result. With network based methods the positions of users are computed by the central system and this has to be the case for devices that can't collect location dependent data.

Lastly there are systems that support modelling, meaning adding and managing floor plans of buildings, handling points of interest and drawing routes (graphs) in between POIs since the regular approach of Euclidean Geometric wayfinding wouldn't work because of walls, hallways, rooms etc [9].

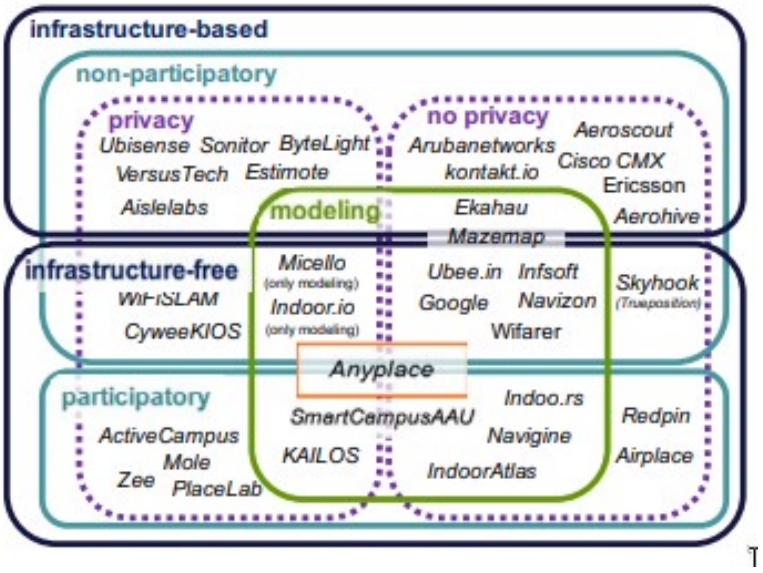


Fig 2.1: Taxonomy of IIN services with a Venn diagram [5]

Then the authors introduce Anyplace which is an IIN service comprised of the Server, the Architect that is the modelling software, the Viewer which is the web client of the service, a datastore, and an android client called Logger & Navigator. It is characterized as infra free, participatory, with modelling and a flexible privacy model. The Logger can record Radio Signal Strength fingerprints and send them to the server, the Navigator and Viewer do pretty much the same thing but the Navigator has better accuracy. As for privacy a user can either cache whole indoor models for maximum protection or download specific buildings intelligently hiding location data.

2.1.2 Zero Infrastructure Geolocation Of nearby First Responders On ro-ro Vessels [11]

In this paper the authors talk about a zero infrastructure indoor navigation system that was developed in order to help first responders in ro-ro vessels, in case of sudden fires, to locate themselves.

Many kinds of geolocation systems are available, however most require some kind of infrastructure either new or existing which can get costly really fast. In this case a solution was developed that uses computer vision and sensors on a device to locate itself meaning it needs 0 infrastructure. There are a few steps to the whole procedure. First training is needed on a dedicated deep learning computing server. Then logging where personnel walk around in the vessel and collect objects associating them to certain locations, this produces the fingerprint database. After these 2 steps are completed once per vessel type, users can locate themselves by using the application than uses the produced fingerprint database and computer vision to calculate the location of the user.

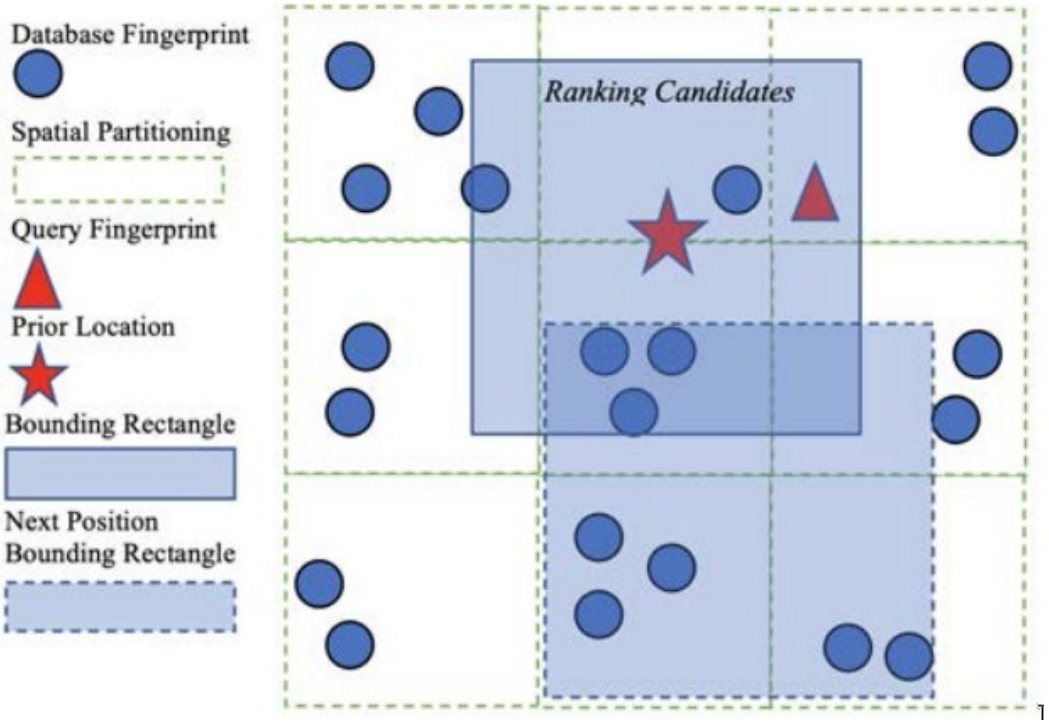


Fig 2.2: Example of the Surface Local algorithm in action [11]

These algorithms use object ranking functions. There is multiset subtraction where we find (x, y, deck) records that have the most similar objects with the set of objects from the query. Global Partitioned Frequency takes object frequencies that where computed when the database was created and uses important objects to rank results. Spatial Partitioning of fingerprints is used to cluster fingerprints that are close by, the default is a range of 10 meters. Lastly for surface local there is Bounding Box Filtering that only returns results within a difference of 100 meters and 1 deck compared to a previous location.

```
SELECT F.flid, F.X, F.Y, F.deck,
        ABS(x - "$prevX") as xDiff, ABS(y - "$prevY") as yDiff, ABS(deck - "$prevDeck") as deckDiff,
        (SELECT IFNULL(COUNT(*),0) FROM
  ( -- Ranking Criterion A: Multi-set Subtraction as
   SELECT ROW_NUMBER() OVER (PARTITION BY FLT.oid) AS RowNum, FLT.oid
   FROM FINGERPRINT_LOCALIZE_TEMP FLT WHERE FLT.uid="$uid"
   EXCEPT
   SELECT ROW_NUMBER() OVER (PARTITION BY FO.oid) AS RowNum, FO.oid
   FROM FINGERPRINT_OBJECT FO WHERE FO.flid=F.flid
   )
) AS dissimilarity, (SELECT IFNULL(AVG(weight),1) FROM
   (-- Ranking Criterion B: Multi-set Subtraction on Global Partitioned Frequency Counting with Spatial Partitioning of Fingerprints
   SELECT ROW_NUMBER() OVER (PARTITION BY FLT.oid) AS RowNum, FLT.oid, OF.weight as weight
   FROM FINGERPRINT_LOCALIZE_TEMP FLT, OBJECT_FREQUENCY OF
   WHERE FLT.oid = OF.oid and FLT.uid="$uid"
   EXCEPT
   SELECT ROW_NUMBER() OVER (PARTITION BY FO.oid) AS RowNum, FO.oid, OF.weight as weight
   FROM FINGERPRINT_OBJECT FO, OBJECT_FREQUENCY OF
   WHERE FO.oid = OF.oid and FO.flid=F.flid
   )
) AS weight
FROM FINGERPRINT F
-- Bounding Rectangle Filtering of Fingerprints
WHERE F.modelid="$modelid" and F.buid="$buid" and
      (x between "$prevX" - "$smas_db_location_bound_meters" and "$prevX" + "$smas_db_location_bound_meters") and
      (y between "$prevY" - "$smas_db_location_bound_meters" and "$prevY" + "$smas_db_location_bound_meters")
      and (deck between "$prevDeck" - 1 and "$prevDeck" + 1)
-- keep only x,y,deck rounded by 10m bounding box
GROUP BY ROUND(F.X,"$smas_db_location_bound_rounding"), ROUND(F.Y,"$smas_db_location_bound_rounding"), F.deck
-- include results that have at least 1 common object
HAVING dissimilarity < (SELECT COUNT(*) from FINGERPRINT_LOCALIZE_TEMP FLT WHERE FLT.uid="$uid")
-- rank by dissimilarity, then OBJECT_FREQUENCY weight than by distance from prior location
ORDER BY dissimilarity, weight, ABS(x - "$prevX") + ABS(y - "$prevY") + ABS(deck - "$prevDeck") ASC
-- return highest ranked result only
LIMIT 1;
```

Fig 2.3: Implementation of Surface Local in SQLite [11]

A few experiments were carried out where CV logs were collected from footage of the interior of a ro-ro vessel. Data was annotated and a neural network was trained on it using quantization to reduce the size of the model. The resulting model was extracted to tensorflow lite. The result was a database with 460 objects in 5 decks in 81 distinct locations. 150 localization attempts were attempted with an average success rate of 80%.

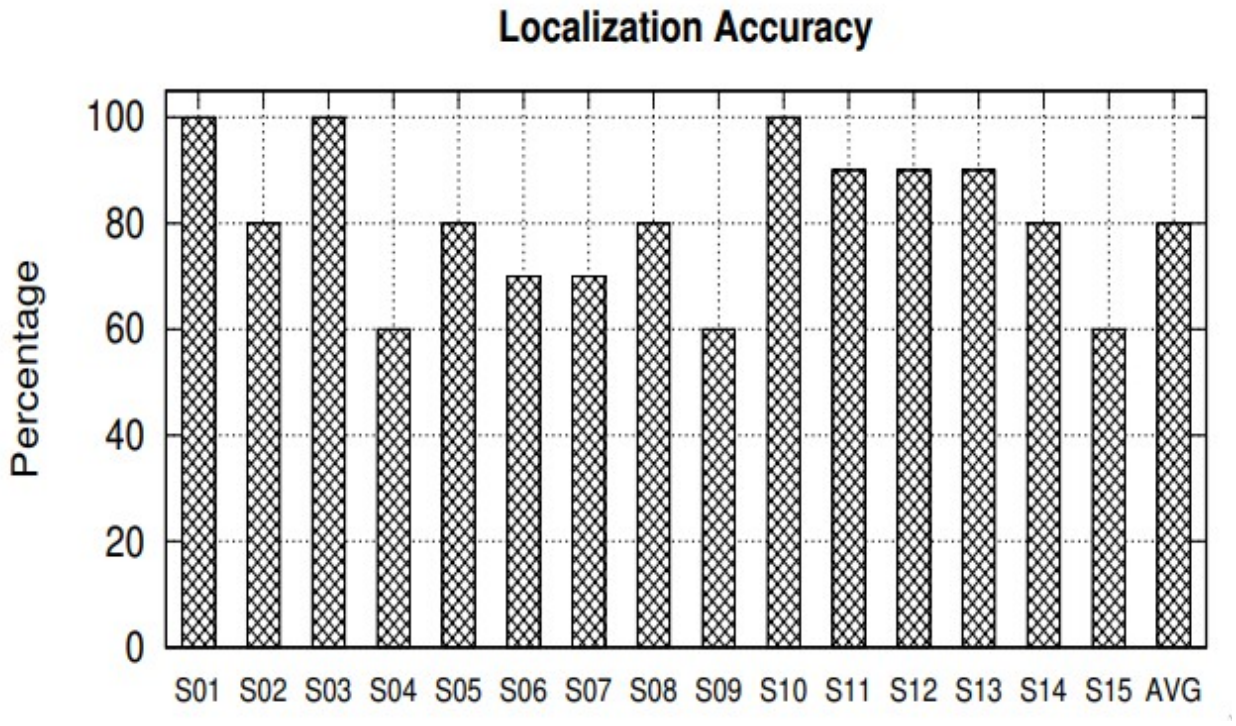


Fig 2.4: Localization Results for 15 scenes and the average of 80%. We can see that the highest percentage was 100% and the lowest 60% while two thirds of the scenes yielded results between 80% and 100% [11]

2.1.3 The Anyplace 4.0 IoT Localization Architecture [12]

Internet of Things or IoT devices are becoming extremely prevalent, they are devices that connect to the internet that can feel their environment and execute tasks. Anyplace 4.0 IoT is a new architecture for Anyplace that uses IoT devices for indoor localization, using many technologies like WiFi, Bluetooth Low Energy [8], Ultra-Wideband [6], Computer Vision [11] and more. This is the system the authors present in this paper based on the requirements of the Alstom SA smart factory, a leader in the transportation sector.

A4IoT works in the following manner, firstly fingerprints are collected which are Received Signal Strength Indicators of sensors in different locations (x, y) in a building. Then fingerprints are joint into NxM matrices called Radiomaps where N is the number of distinct fingerprints and M is the number of beacons. Lastly a terminal device can

query its observed fingerprint with the respective sensor type matrix and find the best match using KNN and WKNN algorithms [4].

A4IoT offers an architecture with a containerized backend which can run on both low-end devices and powerful servers with multiple nodes, a datastore that handles large quantities of spatiotemporal data from different kinds of sensors and real-time data visualization, and a library that helps with the development of clients on multiple platforms. This solution is capable of solving the needs of quite a few localization models, an example is Alstom's where there is the need for tracking assets and analytics using existing WiFi and maybe UWB [6] and LoRa. Another is Lash-Fire's RO-RO vessel fire hazard detection and localization using Computer Vision [11] which was presented in the previous paper above. And lastly Endorse's fleet of robots for healthcare purposes in the pursuit of help for the elderly.

The authors then presented the requirements of Alstom, why localization would help and how they would actually localize them. Alstom has 3 types of assets, workers, tools and parts. Each is associated with a building. Workers would get improved safety like alerts for entering a prohibited zone during dangerous tasks. Tools can get automatic check-out which would reduce losses and increase usage and availability maximizing efficiency and reducing costs. As for parts they can be tracked during their 3 phases of stationary storage, transit in containers and progress in the assembly line, quantities can be measured in order to always have the right quantity of parts. Another segmentation of assets can be done by their power supply. Assets with a persistent power supply like vehicles that have a long lifecycle and are large and expensive can have a Raspberry PI running the cli of anyplace or an Android device running the anyplace android client on the board of the asset, they will use the power supply of the vehicle and send RSSI fingerprints and GPS locations to the server. Assets with a battery like tools will have external modules with WiFi, cellular and GPS capabilities, while running custom programs using the Anyplace java library to report to the server. On the other hand workers will utilize smart android devices with a patched version of the Anyplace client. Lastly assets without power, big tools or parts will have active tracking where a low powered device will send data to the network a few times a day, for small parts

passive tags like UWB tags [6] will be used. As for consumables no tracking is required.

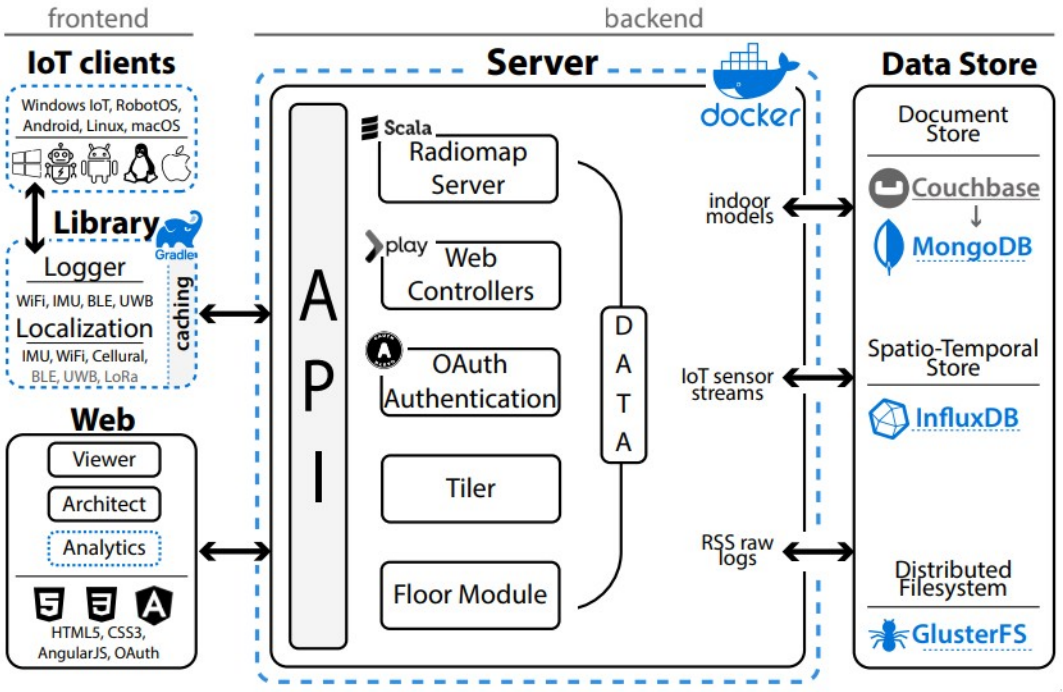


Fig 2.5: Architecture of Anyplace 4.0 IoT [12]

As for the architecture of the whole system there is a backend written in the MVC framework Play that exposes a REST API. There is a data store on a distributed filesystem, IoT data is managed by InfluxDB a time-series store, and JSON objects are saved in MongoDB. Architect, Viewer and Analytics are built with HTML, CSS3, AngularJS and feature a modelling interface, a map viewer and localization interface, and lastly a data visualization tool for fingerprints and wifi signals respectively. Logger & Navigator is an android application that records RSSI fingerprints and uses them for better localization than the Viewer. The server uses Docker for containers, HaProxy for load balancing, GlusterFS for DFS or Hadoop for massive loads in datacenters.

Anyplace also offers a few libraries and tools in order to easily automate tasks or develop new clients for custom solutions to problems that existing clients might not be able to solve. The anyplace-core Java library contains the standard functionality for IoT clients, it can switch between anyplace backends and request all sorts of data from the server, from buildings and their POIs, to connection data between POIs, estimated user

location or even a bounding box of generated Radiomaps to locally calculate current position. Next there is anyplace-android which contains anyplace-core and is a library for creating android anyplace clients with features like reading current RSSI values, secure offline data caching, background execution, appropriate permissions etc. Lastly anyplace-core has a cli and can be used to quickly build script based solutions.

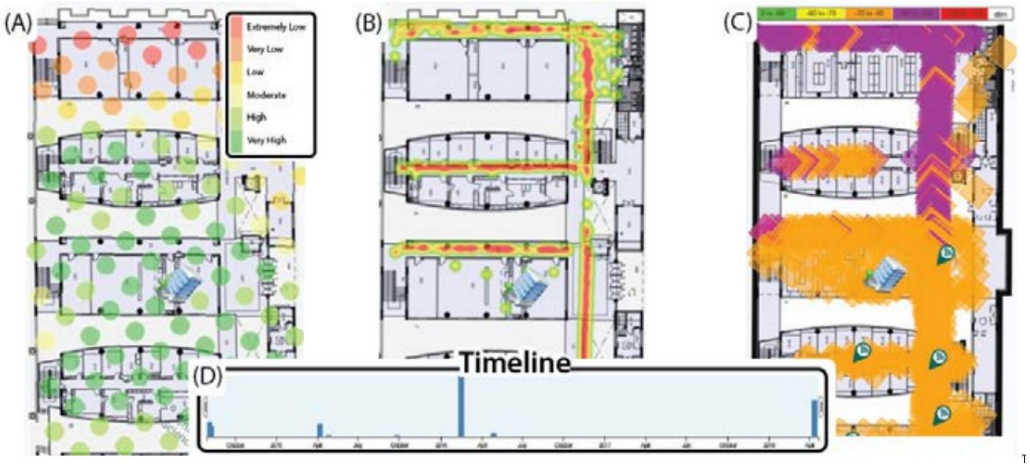


Fig 2.6: Analytics modules for (A) Accuracy estimation, (B) Fingerprint Maps, (C) WiFi coverage, (D) Timeline [12]

Analytics is another important feature that Anyplace 4.0 offers. There are 3 main analytics that can be visualized. Accuracy estimation is a heatmap that using a black box technique for fingerprint interpolation, predicts sensor values given a initial radiomap and derives a lower bound for the uncertainty for the estimate of location, this is the final value that appears on the heatmap. Fingerprint maps displays a heatmap of previously collected WiFi fingerprints and helps to aid volunteers by guiding them to collect values at areas with fewer collected fingerprints. Lastly WiFi coverage shows expected data rates and signal coverage in a building, this tool can help identify where new access points should be placed.

2.1.4 Privacy-Preserving Indoor Localization on Smartphones [1]

The authors of this paper present an algorithm for localization that preserves the privacy of the user. Many Indoor Localization Systems use approaches that can be exploited

either by hackers or by the very enterprises that offer those services in order to sell localization data to advertisers and profit. This algorithm was developed in order to solve this serious issue while also not using as much energy, memory and general resources as complete Client Side localization where the device downloads the whole Radiomap which can get huge very quickly.

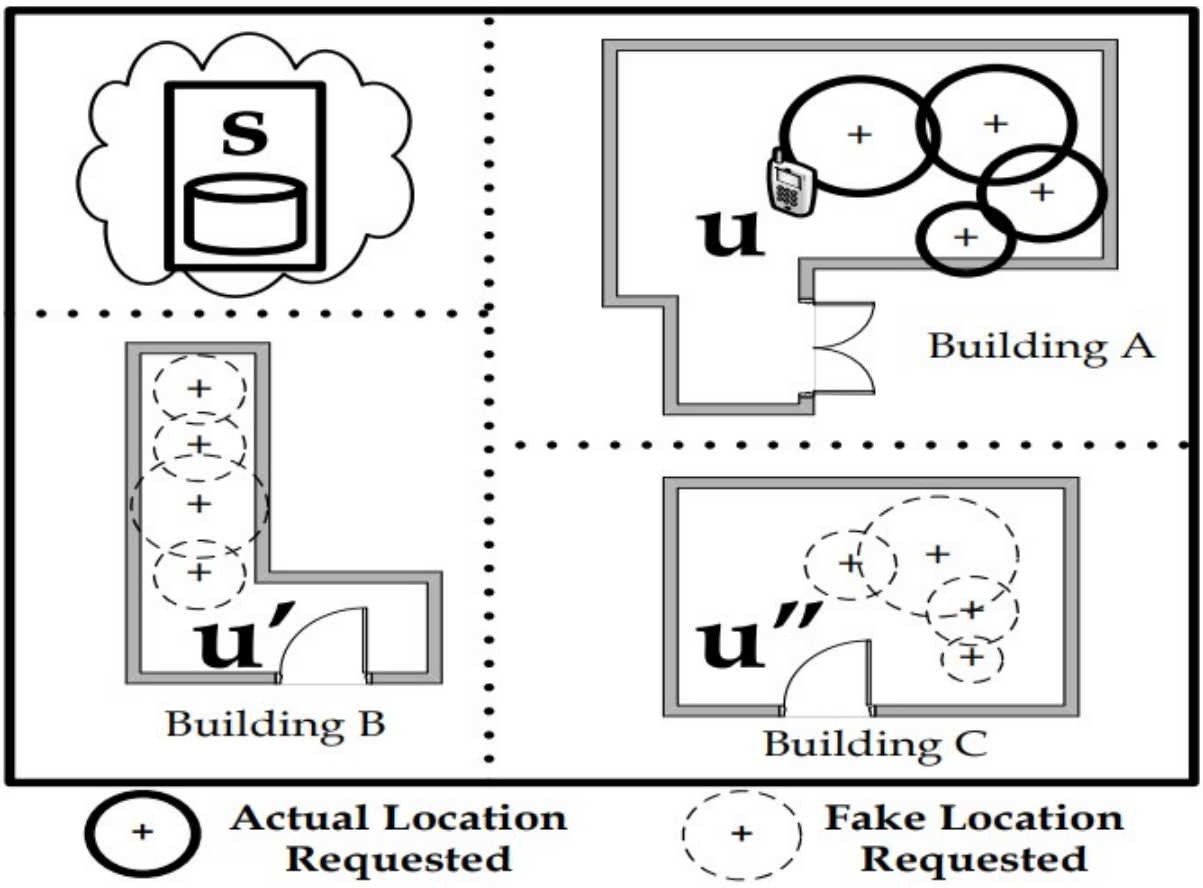


Fig 2.7: User u localizes using the IPS s and fetches u' and u'' as camouflaged positions [1]

Temporal Vector Map is the name of the new proposed algorithm. It takes in two arguments the current fingerprint of the user, the user's privacy preference which means that the service can't locate the user with a probability of more than the user's privacy preference, and lastly the Radiomap, it then returns the location of the user. The algorithm has two phases, the first phase is used for the initial localization (snapshot localization) and the second phase is used for continuous localization. The first phase

uses a kAB filter [2] to get k random APs to mask the user's location, after the kAB is created it is sent to the server, the server finds possible matching APs from the kAB. Then Radiomaps filtered by the matching APs are sent to the client, the client uses the received RMs and finds its location through WKNN, RBF or SNAP [3]. For the second phase instead of a kAB filter the algorithm uses a bestNeighbours approach, where the vector of the user's movement is calculated based on a previous and current AP and then that vector is applied to all APs in the RM received in the previous iteration. This creates a similar trajectory for all fake positions.

2.2 Industrial Background

2.2.1 Mazemap [23]

A system that is very similar to the one that was developed for this thesis is MazeMap. MazeMap is an Indoor Maps and Wayfinding system. It allows its clients to add building plans, POIs and other relevant information to make wayfinding and navigation easy and intuitive for its users. A user can see the POIs of a particular building, they can search through POIs and through a dropdown navigate to one. There is the ability to choose a category of POIs (toilets, elevators etc). A certain location or POI can be shared through a link. The floor of a building that is being viewed can be changed and a user can see himself on the map and can use wayfinding to guide himself to a desired location. Additionally it seamlessly integrates both outdoor and indoor Navigation.

Because GPS signals were to weak indoors mazemap uses other technologies to find the position of a user. It supports a few different technologies including WIFI triangulation, geomagnetic fields, beacons and sensor fusion. For assets and equipment, they can be tagged with RFID, BLE or GPS then they can be traced live on the map.

Many of the clients of MazeMap are universities, where finding a location for a student is pretty difficult because of how vast and complicated most campuses are, hospitals, where tracking equipment is vital and getting to certain places in case of emergency is an absolute priority. Another use case would be the reduction of missed appointments

that were caused by poor wayfinding, which was actually the case with St. Olavs hospital and thanks to mazemap a 31% reduction of missed appointments was achieved saving 1.5 million euros per year.

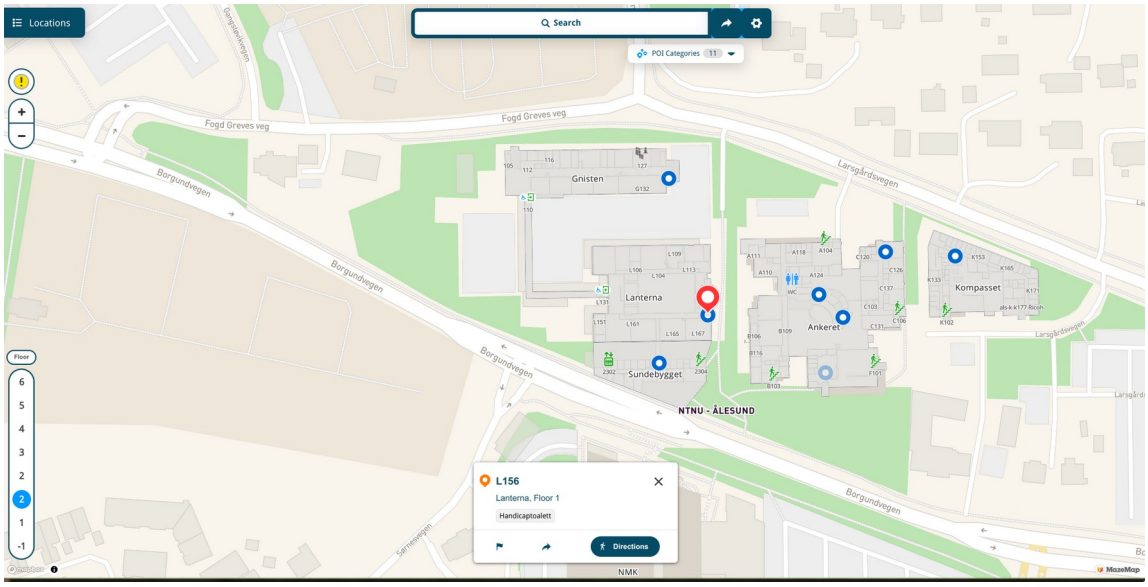
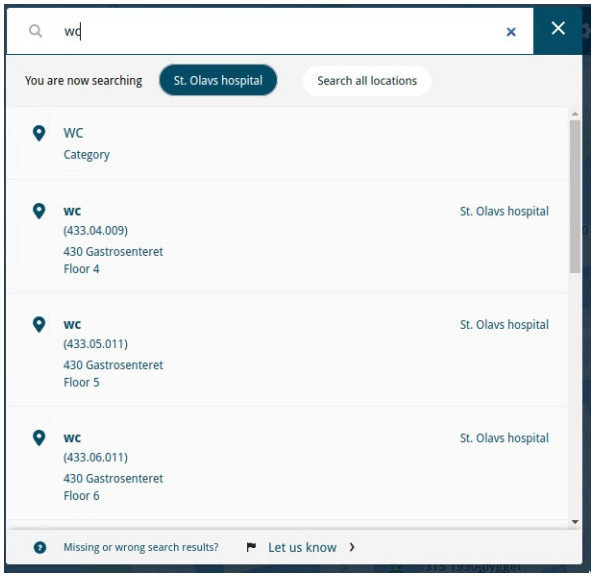


Fig 2.8: MazeMap system in use [23]

MazeMap can be used through 3rd party software like Facility Management Systems which would also offer automatic updates. In this way the manager of a Facility, in case that there is a reason to update the plans of a building, will only need to input the changes in one software. This saves time and provides users with the latest routes and POIs.



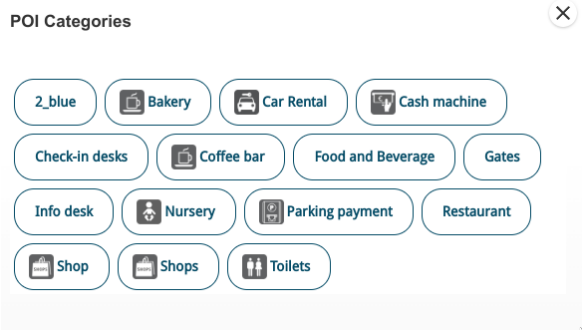


Fig 2.9: Searching POIs and selecting a POI category [23]

MazeMap has many more features that can make locations smarter using IoT devices. There is technology for data visualisation, bookable spaces that can have occupancy checked by sensors. Remote climate control, space planning etc. Even a patient tracking system can be achieved which is especially useful for patients with Alzheimer and Dementia. The data visualisation aspect is not to be overlooked as it can be used in many different ways to cut down costs.

2.2.2 Google Maps: Indoor Maps [19]

Indoor maps is an extension of google maps. It allows users to zoom in on a building on the regular google maps and see the plans of the building along with a floor switcher for navigating between floors. Points of interest are clearly shown on the map as well. Of course wayfinding is also present in indoor maps. All features that are supported by regular google maps are still present since this is just an extension of google maps. Indoor maps are also available through the google maps API, so anybody can integrate them in an application or website.

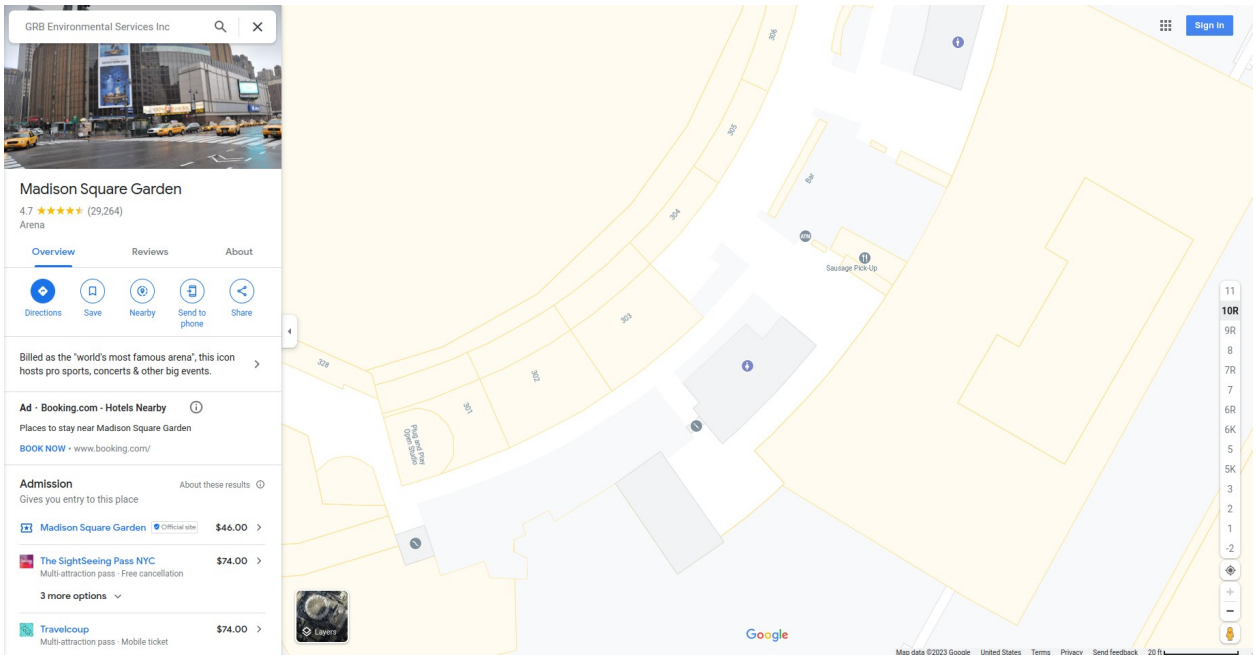


Fig 2.10: Indoor Maps in use [19]

Indoor Maps use GPS so the accuracy of location might not be as good as other solutions since GPS has difficulties penetrating walls and other surfaces. Even if GPS signal wasn't obstructed there, there is still a second challenge, GPS signals are only accurate up to a radius of 10 meters which for regular maps isn't a big error margin, but for indoor spaces this is not enough.

When indoor maps originally launched google had partnered with retailers, airports museums and transit stations to add an approximate 10k indoor Maps to their system. Later in 2016 for the Rio Olympics a few updates added indoor maps for the venues that would be used for the Olympics. Businesses can submit plans for their business to google and they will convert the plans to indoor maps and feature them on their maps. However they have made it clear that they are more interested in bigger venues where an indoor pathfinding system would actually make sense.

2.2.3 Inpixon Indoor Positioning [21]

An indoor positioning system that allows for accurate pinpointing of persons and assets in indoor spaces with the use of tracking tags, cellphones and other devices. Specifically this can be achieved with the internal sensors of portable devices or using anchors and radio frequency sensors placed in a building.

Inpixon's system has achieved centimeter accuracy, with very little delays and due to sensor fusion a very smooth location pointer. They use machine learning algorithms to speed up the positioning setup. Their service can be used regardless of internet coverage as they offer options between cloud, hybrid and local infrastructure. The switching between outdoor and indoor maps is smooth, and changing floors or exiting/entering a building can be automatically detected. Inpixon made sure to have minimal energy drain as to not inconvenience users, and even has the option to integrate their maps with other systems through REST APIs and MQTT [14] protocol speaking IoT devices.

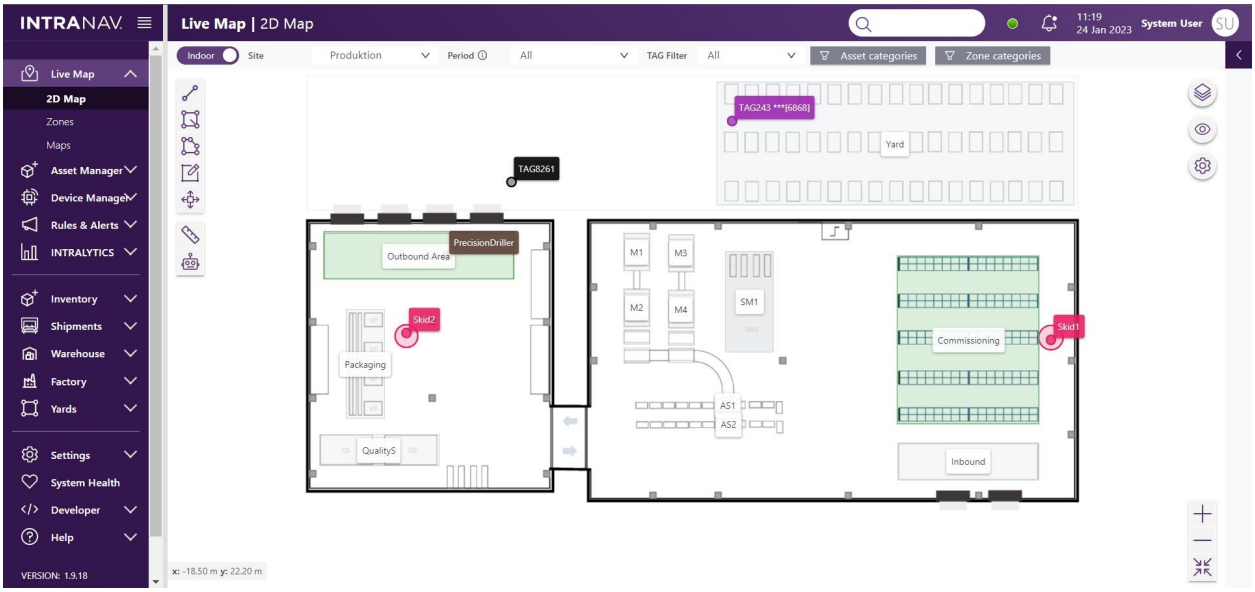


Fig 2.11: Inpixon's Intranav Industrial RTLS platform [22]

Inpixon in addition offers Indoor Analytics, it uses WiFi access points and specialized sensors that can better detect WiFi enabled devices or even detect cellular signals, Bluetooth, UWB and BLE tags. Once it collects relevant information, it analyzes and visualizes relevant data points like visitor counts and types, high traffic zones. It then can make decisions using AI. It supports heatmaps.

WiFi positioning is a method that uses signals from access points and mostly using received signal strength indicators where signals get categorized as strong or weak. Either WiFi access points are used or WiFi sensors. These devices collect location data corresponding to different devices. Using multiple sensor devices multilateration algorithms can calculate the correct positions of devices. RSSI fingerprinting is another solution where signal strengths of surrounding access points are recorder in a database. It however requires for calibration that can be tedious and time consuming. A more accurate approach would be Time of Flight where the distance between devices can be measured by counting the time a signal was in the air. For this approach to work correctly however clocks need to be synchronised and APs would need to be a numerous and concentrated which wouldn't be most cost effective in many cases. The

last method is Angle of Arrival where a single antenna sends signals using a multi antenna array, the phase shift then can be used to calculate the Angle giving a better accuracy than RSSI based approaches.

BLE or Bluetooth Low Energy [8] which is being used by some of the Inpixon's sensors is a technology which consumes very little power and is the advancement of the Classic Bluetooth. It is usually used for beacons and location tracking. BLE beacons are small devices that continually produce beats of BLE signals that can be detected by other BLE enabled gadgets. BLE usually is accurate up to 5 meters and has an optimal range up to 25m and if lucky up to 100m. BLE uses less power than WiFi and is more accurate however most organizations already have WiFi infrastructure but not BLE, its range and data transfer speed is also slower than that of WiFi.

UWB or Ultra Wideband [6] is a short range technology that sends nanosecond long pulses over an "ultra wide" frequency range. It's advantages include extremely high data rates and very accurate location detection in real time. Like BLE it consumes very little energy. It uses Time of Flight metrics to calculate location based on how long a pulse was traveling from one device to another. There are two primary ways UWB calculates location. The first is Time Difference of Arrival where we have some stationary anchors which will detect a device, timestamp received signals and send them to the indoor positioning system for multilateration to be used in order to get the location of the device. The second approach is Two Way Ranging where two devices communicate with each other and try to calculate their relative positioning. UWB has an accuracy of about half a meter and an optimal range of 50m or if lucky up to 200m. The only major downside is that its ecosystem is way smaller compared to something like Bluetooth.

Intranav is an inpixon company for automating smart factories and warehouses by using Inpixon technologies. Objects can be paired with tags that will then show up in a system with real time updates about location and status, making automation of factories an easy feat. Items can be quickly located, searched, filtered etc. Intranav offers geofences for applying different rules and alerts like emails, SMS or just API events. This allows clients to improve the efficiency of their processes and to cut down costs. It even supports features like collision avoidance for forklifts and other motor vehicles, or even employee tracking in order to quickly and effectively deal with accidents, sound alarms

if somebody unintentionally stepped in unauthorized zones and to improve workplace safety.

2.2.4 proximi.io [24]

proximi.io is a service for both indoor and outdoor maps that offers both two and three dimensional maps. The wayfinding can span both outdoor and indoor spaces even connecting multiple buildings together.

There are options to integrate proximi maps in a website or mobile application using the provided libraries. Some of the features proximi supports are: qr codes for locations, search with autocomplete, switching floors, analytics, filtering categories, smooth outdoor-indoor transition, managing POIs, privacy zones where users are not visible and accurate current location. To get an accurate indoor location proximi tries to determine the most accurate signal out of bluetooth, WiFi, GPS and Cellular. Currently it has an indoor accuracy of about 1 to 2 meters. Proximi provides its clients with about 50 beacons on startup for a better experience, however it can still work with other beacons as long as they support the iBeacon [16] or Eddystone [10] protocol.

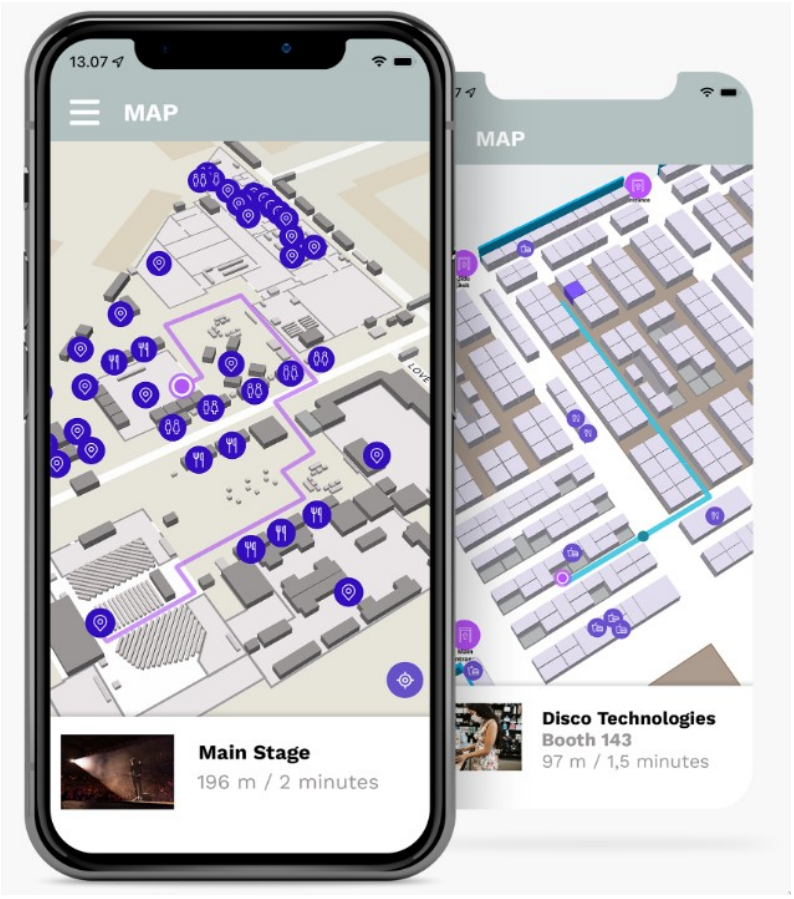


Fig 2.12: proxim.io in action [24]

Placing beacons in a space is something that should be done carefully warns the proximi team. If the job is done poorly, accuracy might be negatively affected. Some of the recommendations are the following: place the beacons at a height of around 2 meters, doing so will make it easier for devices to receive the signal. Secondly put the beacons in visible places as metal objects can hinder the signal. Beacons should be spaced out evenly where their reach is taken into account, they should be separated by a bit less than their reach in order to maximize precision, and at least three beacons' signal should be able to be received in a given location as three beacons is the minimum for trilateration, the process of calculating a device's location using RSSI (received signal strength indicators). Another recommendation is adding beacons right by the entrance/exit to make the indoor-outdoor transition as smooth as possible, as well as next to staircases and elevators so the switch between floors is also seamless. Lastly the beacons' settings should be checked, the optimal transmission power should be -4db and the transmission interval at around 350 milliseconds maximum.

Accessible audio navigation is also available adding the ability for blind users to also use the platform. Another feature is geofencing for both outdoors and indoors where a user can draw any shape on the map and collect data like how many users are inside the shape or trigger some content in the mobile etc. Lastly there are analytics, here you can see heatmaps with the most popular locations and dates, however information is anonymized to comply with GDPR rules so MAC addresses and advertising IDs are not collected.

For clients that want to add their location to proximi, the team has some excellent tools to make the process as easy as possible. They provide tools for creating new venues, then setting up floors and uploading architectural plans for each floor. For the next step, managers can set up navigation for the indoor places. For this they first need to add elevators escalators and stairs, then points of interest should be appended to the map, lastly paths may be added by drawing on the map, first indoor paths are added, then outdoor ones. In addition privacy functions can be used like private zones where the location data of users stops being updated displayed or saved, or the opposite can be

enabled with Geofence-only positioning where users will have their location updated only in certain areas denoted by geofences.

2.2.5 Mapsted [20]

Mapsted is another indoor navigation solution that is very similar to all the previously aforementioned systems. It offers one to three meter accuracy, needing only milliseconds to process through data.

The solutions of Mapsted are cross platform, featuring well integrated indoor-outdoor transitions. Like all previous products it offers wayfinding. One of its core differences with aforementioned tools, however is that it does not need BLE beacons, ultra-wideband tags, WiFi or external hardware, making it extremely fast and trivial to deploy while being also scalable. Mapsted additionally helps on the business side by providing features for marketing and analysis of data collected by the system.

Fig 2.13: The Mapsted interface [20]

Mapsted allows for adding points of interest with additional features like maintenance mode where particular POIs can be marked for maintenance, closing particular areas, creating emergency routes and tracking assets. It has a search feature where one can search by name or category and with real time autosuggestions. It also allows for customisation so it matches a venue.

The analytics module offers many features like heatmaps, usage metrics at entrances, conversion rates, visit times and path statistics. There is additionally traffic flow analysis where bottlenecks can be identified allowing for optimisation to improve user experience. Mapsted offers reports and real time data visualisations as well as AI driven insights based on customer behaviour and positional data.

Another way clients could cut costs and maximize profits is by using the marketing features, like location based targeting, geofencing for certain advertisements, sending personalized discounts to people near certain brands. The marketing intelligence that Mapsted built is capable of segmenting persons into relevant consumer categories. It can learn based of a person's habits and deliver effective marketing strategies.

The technology Mapsted uses for indoor location calculation is based on data-fusion of many sensors from a device like an off-the shelf smartphone and self learning algorithms. It detects magnetic and wireless disturbances in the aether to calculate the position of a user. Some of the data sources that Mapsted uses are Bluetooth, WiFi, Magnetometer, Gyroscope, Proximity Sensor, Accelerometer IOT data, Barometer and many more.

2.2.6 Comparison

	Mazemap	Google Maps	Inpixon	Proximi.io	Mapsted
Infrastructure	Based	Free	Based	Based	Free
Crowdsourcing	No	Yes	No	No	Yes
Privacy	Toggle Positioning	No	No	Privacy Zones, Toggle Positioning	No
Modeling	Yes	No (contact	Yes	Yes	Yes

		google to update floor plans)			
POIs	Yes	Yes	Yes	Yes	Yes
Asset tracking	Yes	No	Yes	No	No
Machine Learning	Modeling	No	Setup, Analytics	No	Analytics, Navigation
Analytics	Yes	Yes	Yes	Yes	Yes
Accuracy	2 - 10m	10 - 20m	From 0.3m to < 10m	1 – 2m	1 - 3m
Geolocation Technology	WiFi, BLE, Sensor fusion, Geomagnetic fields	GPS, WiFi, Cellular	UWB, BLE, chirp (CSS), Cellular, WiFi, TdoA, TWR, Blink	Bluetooth, Wifi, GPS, Cellular	Data fusion, sensor fusion, self learning

Table 2.1: Comparison of some features of the mentioned systems

UWB – Ultra Wide-band

CSS – Chirp Spread Spectrum

TdoA – Time difference of Arrival

TWR – Two Way ranging

Blink – Inpixon's open format for communication for UWB and chirp

We can see in table 2.1 a comparison of different features of the aforementioned systems. Almost all systems are infrastructure based using some kind of beacons, tags, or access points. Mapsted however uses sensor fusions and does not need any preexisting hardware, Google Maps usually uses GPS so it's infrastructure free but it can use WiFi and Cellular if high accuracy mode is used. Crowdsourcing is only done by Google for mapping the planet while for Mapsted it is used for localization. Mazemap and proximi.io allow for a user to disable positioning, and proximi also allows for privacy zones where a user is not tracked but otherwise the systems know the location of their users. All services allow for modeling except for Google Maps where

you need to contact Google to update the floor maps of your facility and that is only if it got accepted for Indoor Maps. POIs are supported by all services.

Mazemap and Inpixon offer asset tracking with the use of tags and beacons. Some solutions use machine learning in order to speed up and improve certain aspects of their workflow. Mazemap uses it for constructing maps, Inpixon for setup and Predictive analytics and Mapsted for Analytics and it uses self learning for positioning. Analytics are supported by all solutions. Accuracy varies for each service, for Mapsted and Inpixon WiFi gives a 10m accuracy while solutions like BLE might give 2m and UWB 0.4m. Google maps with GPS is less accurate but with high accuracy mode it can use WiFi to get more accurate. As for technology used the most common one is WiFi followed by BLE, GPS, Cellular and sensor fusion.

Chapter 3

Requirements and Design

3.1 Basic architecture and summary

This section examines the requirements and the design of the system that was developed for this paper. Let us call this system OMS which stands for Object Management System.

The general architecture of OMS is the following: a native android application written in then Kotlin programming language, it interfaces with an SQLite database that was generated by the SMAS [11] lashfire project. The database is used to get all the objects that where observed by its users in a space and display markers of those objects in a map providing some basic information about each marker/object. Using an API for fetching objects/Points of interests and their positions may also be achievable.

3.2 Proposed features

Following this paragraph is a list of requirements and features that were recorded in order to develop OMS. We recorded 20 features and categorized them in 5 groups, some features might belong to more than one group but we decide to put them in the

one that seemed to make the most sense. The five groups we decided on were the following: Map UI which focuses on features that are mainly features reliant on the Map and implement something or draw something on the map. Next up is Marker CRUD with features that create, read, update or delete markers. Filtering is the next category and this category encompasses functionality that narrows down the display of markers and only shows a select few. Following this we have Additional Interfaces where the features presented implement some interface not yet mentioned. Lastly we have "Advanced – Further down the Road" with features that should be implemented at a later stage as they might be a bit more advanced or just extra/extend previous functionality. Now for the full list of proposed features of OMS:

Map UI:

1. Display an interactive map, a user should be able to move the map, zoom in and zoom out. Google maps provides an SDK for this use case and it can be used.

2. Display a heatmap over the map, of all the markers satisfying current filters.

3. Be able to see the user's current location on the map, the location should be getting updates every few seconds. Additionally there should be a button which when pressed shall move the map so it has the location of the user in the center.

4. Ability to display floor plans from the anyplace API or other registered API endpoints. The floor plans should be rendered after zooming in to a specific zoom level and when the selected floor is changed they should obviously change.

5. Track changes of position for objects in between API fetch requests or in between updates of an existing database. Visualize these changes in positions with lines between the old and new position and show older lines of movements in increasingly transparent lines, this should help users study the movement patterns of different objects. Newly added objects or deleted objects can have a different type of marker icon to differentiate between stationary objects.

Marker CRUD:

6. Get objects and their recorded locations from the lashfire SMAS database [11] and add a marker for each recorded object on the map. On click the marker should display

some basic information about the object it represents. Markers should also disappear after zooming out to some threshold so the map does not look cluttered.

7. Have the ability to fetch information about POIs from the anyplace API [12] and display them in almost the same way as objects from the SMAS database, a slightly different marker would be good to give the user an easy way to differentiate between the two types of objects.

8. Ability to update markers. Be able to create new objects and position them somewhere on a map inside a building on a particular floor. Be able to delete markers. Be able to drag and drop markers in order to change their location, this action should also draw a line from the old to the new location showing the movement of the marker. These actions should update the associated database or API. If no such capability is available then warn the user that the current action is temporary. There should also be an undo button for the last few actions in case that the user makes a mistake.

Filtering:

9. Display a searchbar in which a user can type a few letters and get back a list of objects' names that have the search query string as a substring. When a search result gets, clicked the map should move in such a manner that would bring the clicked object's marker in the center of the view. Additionally the popup that displays information about the marker should also appear. The results of the search should be ordered based on which object is closer to the current center of the viewport on the map.

10. Be able to choose between floors in a building and only display the markers on that floor.

11. Be able to filter POIs and objects by category. This will aid users by reducing clutter on both the map and in the search results.

12. Be able to select a building or group of buildings and have the map display only objects in the selected building. This could also mean that when OMS is launched, only markers for all the available buildings are displayed until a marker of a building is clicked or a building is selected from a list of buildings possibly hidden in a drawer, at which point the markers for objects and POIs of that building get displayed.

Additional Interfaces:

13. Be able to customize the experience and change settings like language and accessibility via a modal under a settings button.

14. Be able to share a link for a particular coordinate or POI and be able to open links that where generated by OMS with OMS, opening such a link should position the map above the coordinate or POI encoded in the URL.

15. Statistics and analytics about the objects. Some examples would be most common objects per floor, most common objects per building, most frequently moved objects, objects that are added or deleted the most, 80/20 type charts where we can see what 20% of object types represent 80% of objects recorded.

Advanced – Further Down the Road:

16. Cache response from Anyplace [12] API so that points of interest can still be displayed even if the API is not available.

17. Have import functionality where a user can import database files in order to either add new objects or replace the previous database file. This could also expand to a list of database files where a user will be able to select to only show object from some selected files. Additionally management of said files must also be possible, so a user should be able to delete files, and add new ones.

18. Something similar can be done for API endpoints, where a user will be able to add, delete or update API endpoints that will be used for fetching POIs and buildings, floors etc. Again a filter would be very handy. Users should be able to choose which registered API endpoints to use and which to ignore.

19. Ability to use the application while offline. Instead of using the APIs, cached data of their responses should be used alongside the lashfire database files. The biggest challenge for this would be Google Maps, maybe we could use some other Map provider that also allows for offline use but we will need to research more on this subject before implementing it.

20. Use the anyplace-android library to get functionality like accurate indoor localization.

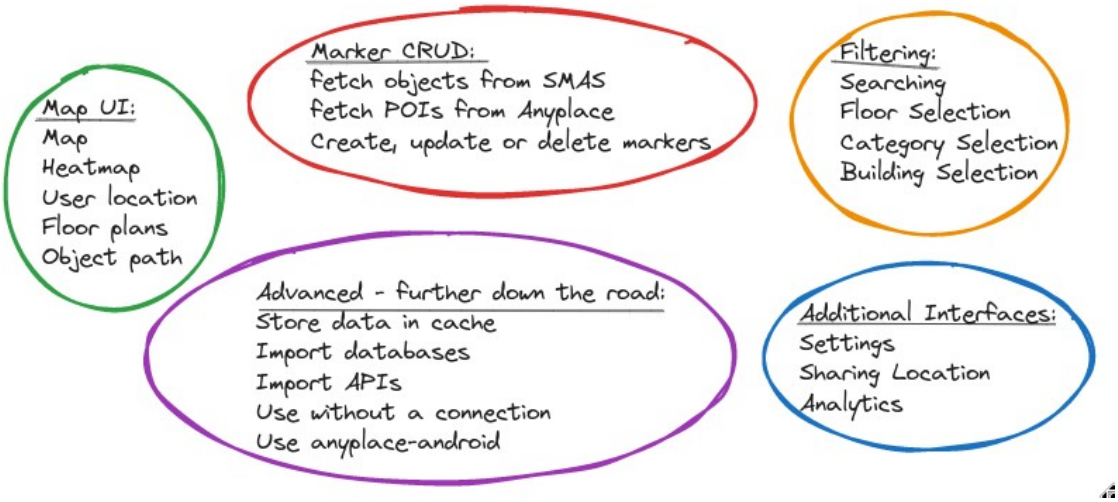


Fig 3.1: The 20 aforementioned features split in the 5 groups: Map UI, Marker CRUD, Filtering, Advanced – further down the road and Additional Interfaces. The figure was drawn with [18]

3.3 DB Schema

It has been mentioned a few times already that the database file being used for the application is the one from the SMAS [11] lashfire project. The relational schema of the database is provided in the following figure.

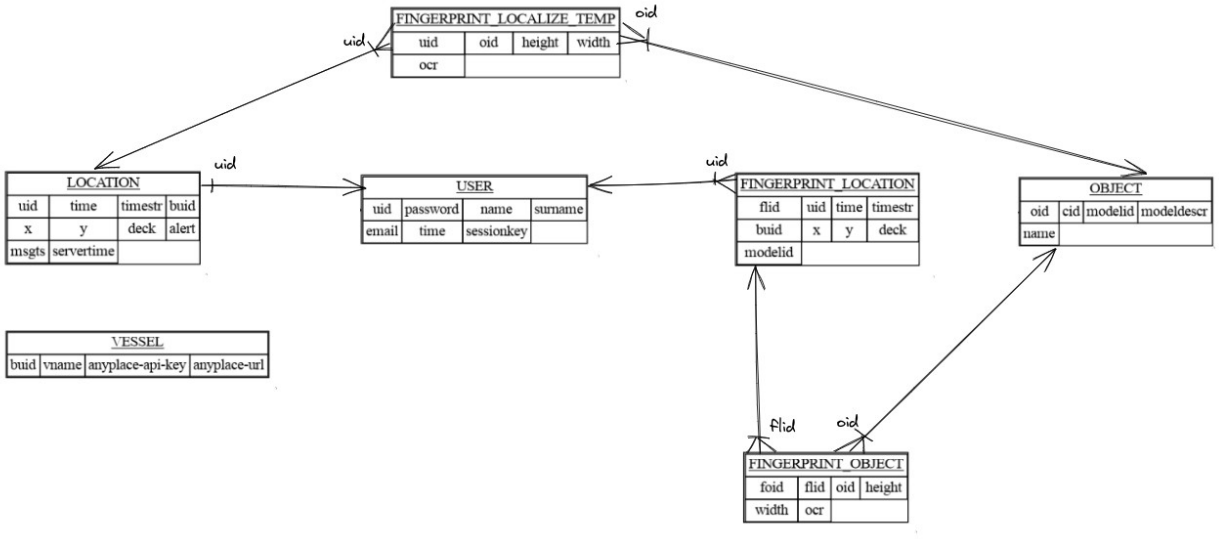


Fig 3.1: Database Relational Schema showing the base tables. The figure was drawn with [18] and [25].

In the schema we can see the tables of the SMAS database. There is the User with basic information about the SMAS user with his latest sessionKey. The Location table holds the latest location of each user along with some other useful information. The Object table contains information about objects we want to track. Vessel is a table that contains information about vessels on which the SMAS system is used. A Fingerprint Location stores the user and their location (vessel, x, y, deck) at different points of time, the Fingerprint Object table depends on Fingerprint Location and store a reference to one of its rows alongside an object, and the object's recorded dimensions and ocr. In OMS we do not need all the tables that are provided by SMAS. In fact we create a new view that only depends on Fingerprint Object, Fingerprint Location and Object. This view named as Object Location provides information about where each object was spotted alongside the object's name and ocr. This view is used for creating markers on the map. If we also implemented building selection we would also need the VESSEL table.

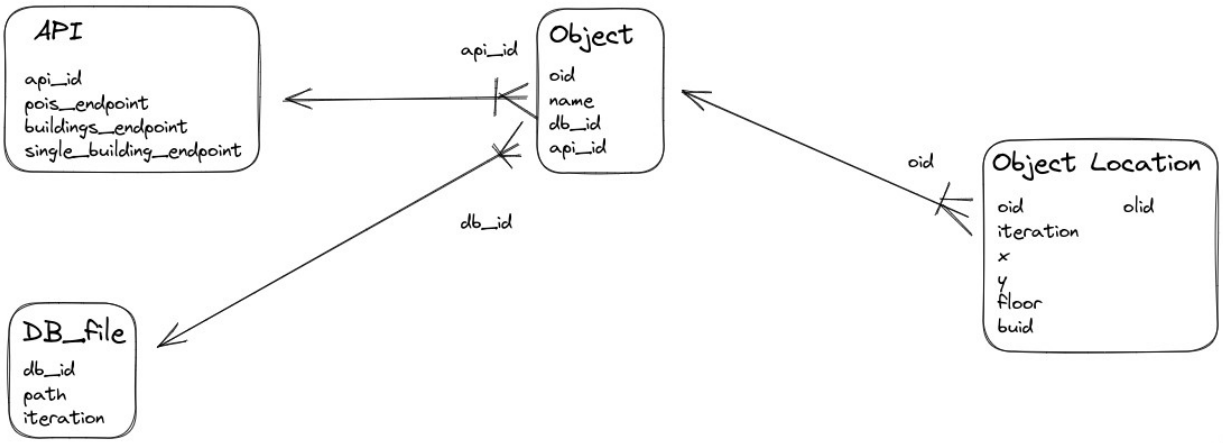


Fig 3.2: Database schema, for multiple APIs and DB files and for tracking changes between location of objects between request operations. The figure was drawn with [18].

In case that we decide to use a single database to manage SMAS database files and their updates and/or differences between positions from the API endpoints then we can design another schema (see Fig 3.2). We would have an APIs table and for each record

we would store in it urls for endpoints for getting buildings, or getting info about a building, and getting POIs in a building or floor. Another table could be created for each database SMAS database file. Then we could have a table that would contain all the types of objects and lastly a table with the locations of those objects as well as an iteration count which would increase each time we would make a new API request or a new version of a SMAS database gets imported.

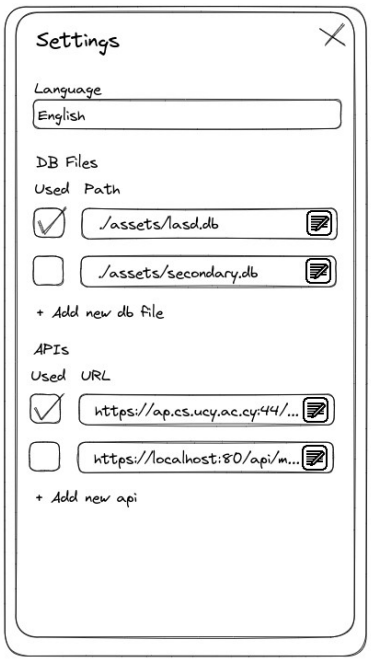
3.4 Mockup and Design

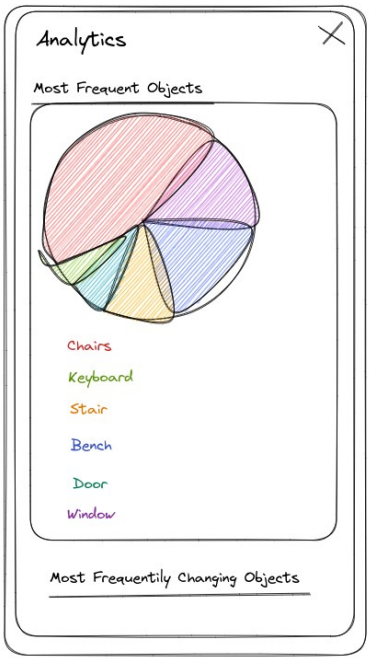
Fig 3.3: Sketch of the interface of the application. (Left) the application in its usual state, (Center) the application when some characters are typed in the search bar, (Right) the application when the Categories button is pressed. Figure was drawn with [18]

A mock-up was designed in order to visualize the final look of the application and to aid development. On the main usual screen immediately after a user opens up OMS he will be presented by a map occupying the whole screen, on the map there will be markers displayed either of the available buildings or of the actual objects. At the top of a screen resides a searchbar that allows a user to search for the name of an object and once a user clicks on a search result he will be taken to that object and the marker's popup will be displayed showing some basic information and a delete and update action. Some

markers may have a trail following them, this indicates previous positions of the marker and how it moved with the passing of time. A blue dot will indicate the current position of the user.

For the rest of the main interface, a share button under the searchbar will provide the ability to share one's location in the map or the viewing POI. A settings button near the share button shall open a modal with language settings and setup for DB files and API endpoints to use for fetching and displaying markers on the interactive map. A categories button under the settings button should show a modal with a list of object types that will give the user the ability to only display particular objects. At the bottom of the screen 3 buttons should be shown, up, down and a dropdown with all the available floors, these three buttons will be the interface for changing the floor that is being viewed. Above these three buttons there will be a "go to my location" button which when clicked will move the map so that the blue dot that represents the user's location moves to the center of the map. Near that button will be displayed a button for creating new POIs. At last we should have a page with all the analytics of our objects, this page would consist of different kinds of charts and statistics that would help a user better analyse and get the whole picture of the objects that occupy a building.





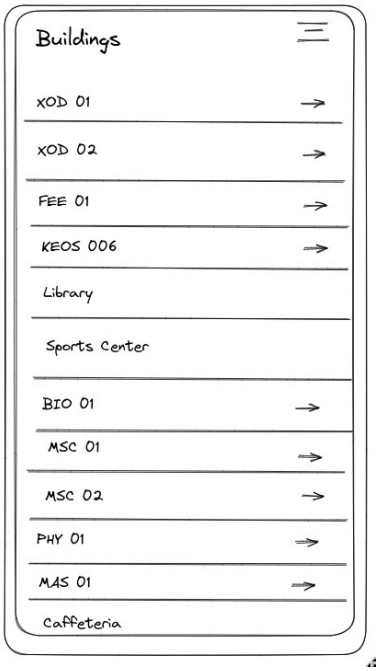


Fig 3.4: Design of the interface of the application. (Left) Settings modal with language selector, and management of DB Files and API endpoints, (Center) OMS analytics like most frequent objects, most frequently changing object etc, (Right) Buildings drawer showing all the available buildings. The figure was drawn with [18]

For APIs the user might need to authenticate themselves or maybe create an account in order to update POI data, we can have another screen that allows users to login or register to a service when they add a service's endpoints to the APIs section in settings. In case that something goes wrong or a user is not allowed to update data about an api a simple error popup can appear telling the user the exact error. DBs do not need authentication since the application can access them directly usually so none of this is an issue. On the other hand we will need an export function to easily extract a DB after a user made whatever updates he saw fit.

3.5 Non functional Requirements

The application should have some attributes that would make a user want to use it and not get frustrated with it resulting in the abandonment of the application. OMS should offer a pleasant user experience so adoption is not hindered in any way.

OMS should be fast and responsive, slow applications and systems that need a lot of time to load have been proven time and time again to negatively affect user experience. Moving the map shouldn't procure any noticeable delays which is a pretty valid concern when taking in mind the fact that a pretty big amount of markers might be added to the map. This could be mitigated by only rendering markers that are currently visible on the map.

The application has to be easy to use with a very smooth learning curve. Users must be able to navigate the user interface without trouble and find whatever settings or functionality they are searching for in mere seconds. This can be achieved by using buttons and icons that can be very easily identified immediately (icons that are usually used like the share icon and the "my current location" icon) and in the case that text is

used, it has to be concise and accurate. No functionality may be hidden in weird places and unnatural spots. A user should be able to navigate to any functionality or place in the application in at most three clicks or taps, this rule shall prevent the overcomplication of any interfaces. In case that the project's scope expands greatly this rule can become more lax but in the duration of the initial round of this project this rule should be enough.

OMS should not use an exorbitant amount of resources. The application has to easily function properly on modern smartphones and even slightly older devices. Users shouldn't have a need to update their hardware to use a mere android application. The main concern is space since we have mentioned a few things about storing older iterations of markers, storing responses from multiple apis and importing multiple database files. A possible solution for this problem would be to store up to a certain number of iterations, and only store objects that have actually moved in between iterations. As for issues regarding performance the previously mentioned solution of only displaying currently visible markers on the map might be the way to go.

3.6 Implemented features

This section can be closely tied to the next chapter, here we give a brief overview of the features that were implemented in the length of the project. If we reference the list of proposed features in section 3.2 of this paper we can say that the features that were implemented were number 1, 2, 3, 6, 7, 9, 10, 11 and 14.

Giving a brief summary of the aforementioned features they are the following: Display an interactive map as the main interface of OMS. Get marker data from a SMAS DB file and the anyplace API and display those markers on the map with the ability to click on one and reveal a little popup with some information about the marker. Markers disappear after zooming out to a particular zoom level and only the heatmap that was generated by them remains. A searchbar was added and it provides the functionality to search for markers by their title, when clicking on a search result the user is moved to that marker on the map, search results are ordered by which marker is close to the current center of the map. There is the ability to choose a category of

markers and only display markers belonging to it. We can also choose a floor and show which markers belong to it, other markers can still be seen on the map but they have a lower opacity to differentiate them easily. The previous two features update the heatmap and search results from the search bar each time the selected category or selected floor changes in order to show only the filtered markers. The ability to share links to certain positions on a map was added alongside the ability to open such generated links with the OMS application. Lastly the user can view their current physical location on the map and move the map to that location in a single tap of a button.

Chapter 4

Implementation

In this section we will introduce 5 of the features that were implemented in the span of the project. For each of the five features we will give a short description of the task at hand. Then relevant code snippets will be shown with a brief explanation for each one. Some code may be left behind as it is not very important for a feature or it is just boilerplate code. At then end of each feature explanation we provide a few screenshots relevant to the particular feature. In addition these are not all the features that were implemented, we just picked five of the ones that were developed and showcased them.

4.1 Getting markers and displaying them on the map

The task at this point of the project was to display a map and then fetch POI data from the anyplace API as well as the SMAS database file and display the returned markers on the map.

We are using the Google Maps SDK to display the map on the screen. We wait for the map to load and when it loads we run a few actions. First we define an event listener for

when the map stops moving, when the listener is triggered we iterate over the list with all the markers and only display them if the zoom level is under a certain threshold. After the event listener we collect the objects from the database and from the api and add them to the map and to a list for later use. The heatmap and search results are also initialized in this step. In case that the app was launched by clicking a link that was generated by the sharing functionality at this point the map moves to the point that was shared.

MapsActivity.kt

Creates and updates the heatmap and the search results, currently however it only creates the heatmap and only runs at the beginning. Later it will also run each time the selected floor or selected category changes and it will update the search results as well. It copies all existing markers to a new list if they match certain filters currently we are only searching for non null markers. If the new list is zero we delete the heatmap. If there is no existing heatmap we create one otherwise we update it.

```
// Creates the heatmap
private fun updateHeatmapAndSearch() {
        val newList: ArrayList<LatLng> = ArrayList()
        val filteredList: ArrayList<CustomMarker> = ArrayList()
        for (m in markerList) {
                if (m.marker != null) {
                        newList.add(m.marker.position)
                        filteredList.add(m)
                }
        }

        if (newList.size == 0) {
                heatmapOverlay?.remove()
                heatmapProvider = null
                return
        }

        if (heatmapProvider == null) {
                // Create a heat map tile provider, passing it the latlngs of the markers.
                heatmapProvider = HeatmapTileProvider.Builder()
                        .data(newList)
                        .build()

                // Add a tile overlay to the map, using the heat map tile provider.
```

```
                HeatmapOverlay = mMap.addTileOverlay(
                        TileOverlayOptions().tileProvider(
                                heatmapProvider as HeatmapTileProvider
                        )
                )
        } else {
                heatmapProvider!!.setData(newList)
                heatmapOverlay?.clearTileCache()
        }
}
```

This function takes all the objects from the SMAS db file. It then adds them to the map and the markerList and creates the heatmap.

```
private fun addMarkers(search: String?) {
        val latlongs: MutableList<LatLng?> = ArrayList()
        val dao = database.fingerprint_locationDao()
        val markers = dao.getLocations()
        Log.i("markers", markers.toString())

        var lastMarker: LatLng? = null
        for (a in markers) {
                val ll = LatLng(a.x.toDouble(), a.y.toDouble())
                latlongs.add(ll)
                lastMarker = ll
                var title = a.name + if (a.ocr !== null && a.ocr.isNotEmpty()) " | " + a.ocr
                        else ""
                val marker = mMap.addMarker(MarkerOptions().position(ll).title(title))
                markerList.add(CustomMarker(marker, title))
        }

        updateHeatmapAndSearch()

        if (lastMarker != null && initialCenter == null)
        mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(lastMarker, 18F))
}
```

Lastly onMapReady is the function that runs when the map is ready and it sets up the event listener for when the map is idle to check if the markers need to be rendered based on the zoom level. Additionally it calls the functions for getting the markers and moves the map to the location of the clicked url if the user clicked a url that was shared by OMS.

```
override fun onMapReady(googleMap: GoogleMap) {
        mMap = googleMap

        val action: String? = intent?.action
        val data: Uri? = intent?.data
        var coords = data?.pathSegments?.get(0)?.split("&")

        mMap.setOnCameraIdleListener {
                Log.i("zoom level", mMap.cameraPosition.zoom.toString())
                filterMarkers()
        }

        addMarkers("")
        fetchAnyplaceMarkers()

        if (coords == null) {
                coords = mutableListOf()
                coords.add(markerList.last().marker?.position?.latitude.toString())
                coords.add(markerList.last().marker?.position?.longitude.toString())
        }

        initialCenter = LatLng(coords[0].toDouble(), coords[1].toDouble())
        mMap.moveCamera(CameraUpdateFactory
                .newLatLngZoom(initialCenter!!, 20F))
        Log.i("coords", LatLng(coords[0].toDouble(), coords[1].toDouble()).toString())
}
```

ObjectLocation.kt

In this file we construct the ObjectLocation view we talked about in the previous section. For this view to be possible we also had to add kotlin files for each of the related tables.

```
@DatabaseView("
        SELECT FL.*, ocr, name
        FROM FINGERPRINT_LOCATION FL, FINGERPRINT_OBJECT FO, OBJECT O
        WHERE FL.flid = FO.flid and O.oid = FO.oid
        GROUP BY x, y"
)
data class ObjectLocation (
        val flid: Int,
        val uid: String,
        val time: Int,
        val timestr: String,
        val buid: String,
        val y: Float,
        val x: Float,
```

```
        val deck: Int,
        val modelid: Int,
        val ocr: String?,
        val name: String
)
```

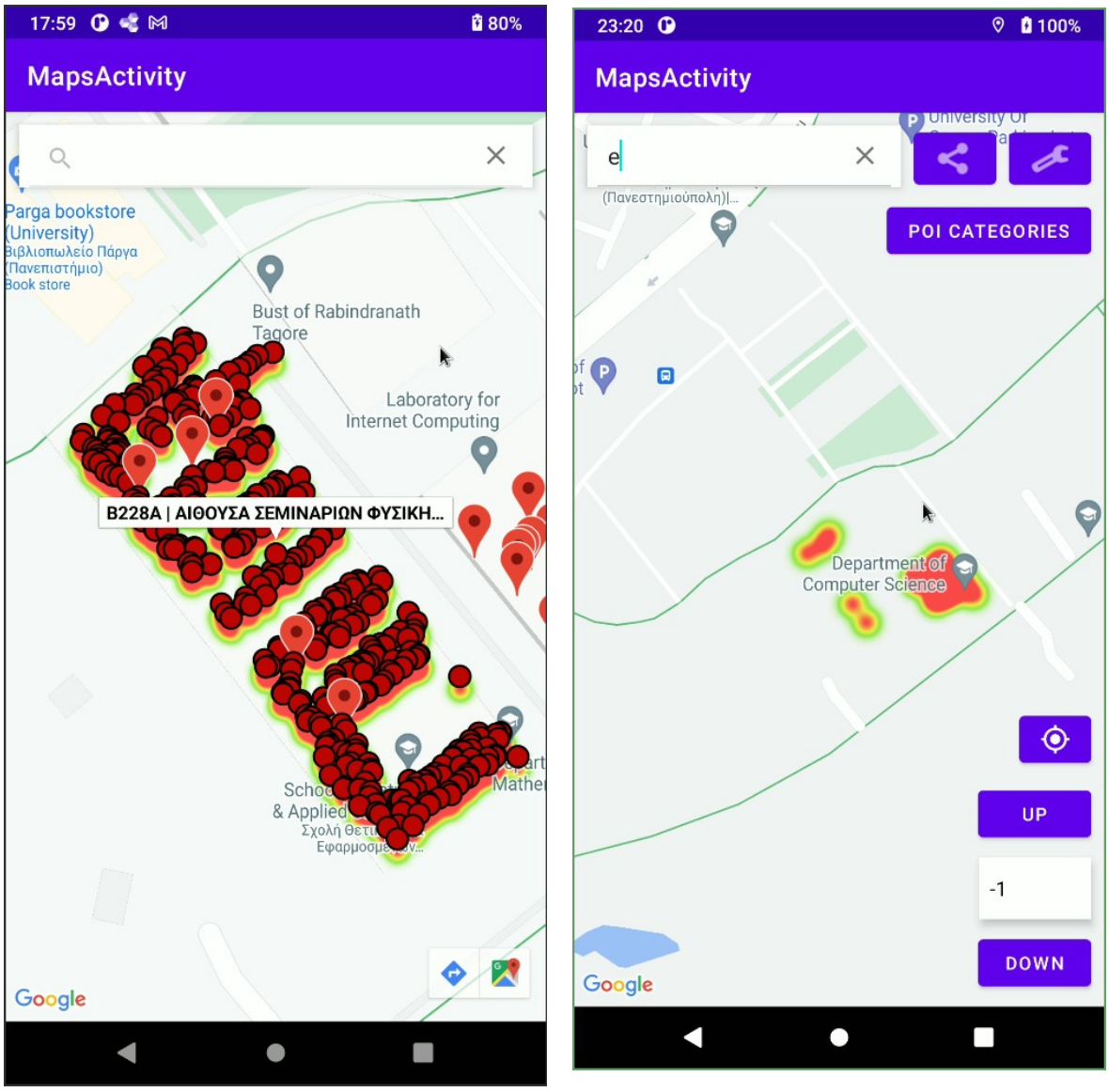


Fig 4.1: (Left) The map with all the collected POIs from anyplace as small red dots and all the collected objects from SMAS as bigger markers. You can see the underneath the markers there is a heatmap. The earlier version from which the screenshot was taken only had a heatmap for POIs from anyplace and not for objects from SMAS. (Right) We can see how the map looks when we zoom out past a certain level. The markers disappear and only the heatmap remains.

4.2 POI searchbar

For this feature the task was to add a searchbar to the interface with which a user would be able to search and locate certain markers on the map. The results the search returns should be sorted by which one is closer to the location the user is viewing in the map.

When the main activity gets created we bind our new views, a searchview and a listview, to their respective properties and we define some event listeners. When an item from the search results is clicked we hide the search results, navigate the map to that marker and display its popup window with the information about the particular marker. For the searchview, whenever its text query changes we display the search results then rerun the filtering with the new text and whenever the close button is pressed we hide the search results. In the previous section in the listener for when the map stops moving we added a call to the ordering function of the search results based on the maps current location.

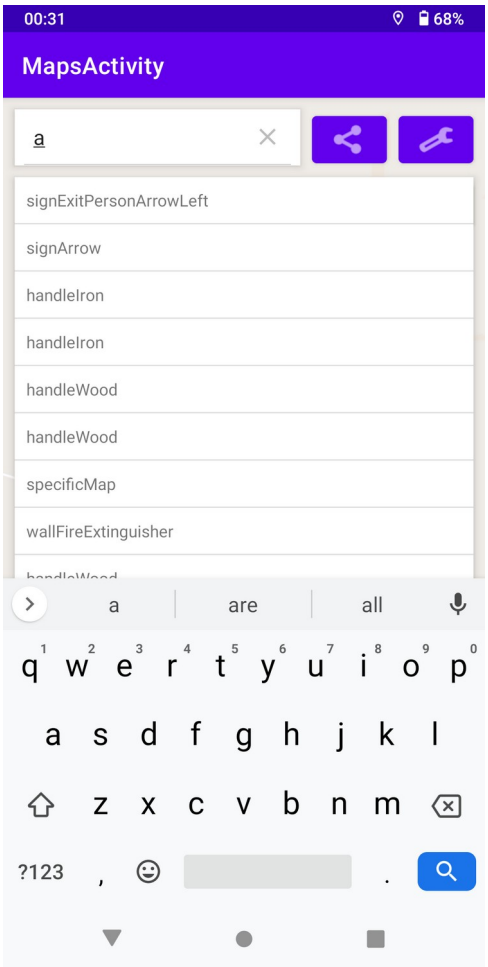


Fig 4.2: The interface of OMS when a user starts typing text in the search bar. The results at the top are items that are closer to the area that the user is currently viewing on the map and the ones at the bottom are further away.

MarkerAdapter.kt

This is the function that performs the filtering each time the text in the search bar changes. It just iterates over the whole dataset of our list adapter, which are currently all the markers but in the next features will only be the markers that where filtered through all the filtering, and only returns the ones whose title in lowercase contains the text query in lowercase.

```
override fun performFiltering(charSequence: CharSequence?): Filter.FilterResults {
        val queryString = charSequence?.toString()?.lowercase()

        val filterResults = Filter.FilterResults()
        filterResults.values = if (queryString == null || queryString.isEmpty())
                dataset
        else
                dataset.filter {
                        it.title.lowercase().contains(queryString)
                }
        return filterResults
}
```

Here we have the function for calculating the distance between two points on a map. Note this is not the correct way to calculate distance for curved shapes like the earth but as a quick prototype it worked pretty well for its purpose.

```
private fun latLngDistance(l1: LatLng, l2: LatLng): Double {
        return sqrt(
                (l1.latitude - l2.latitude).pow(2.0) + (l1.longitude – l2.longitude).pow(2.0)
        )
}
```

Here we have the code snippet for ordering the search results and it runs every time the map stops moving. We define a comparator based on our distance function and compare

the distance of the latlongs from the center of the map. We then make a copy of our dataset, we sort it and then set the dataset to its sorted version.

```
fun orderData(center: LatLng) {
        val comparator = Comparator { m1: CustomMarker, m2: CustomMarker ->
                var l1 = m1.marker!!.position
                var l2 = m2.marker!!.position

                var res = (latLngDistance(center, l1) - latLngDistance(center, l2))
                if (res == 0.0) 0 else if (res < 0.0) -1 else 1
        }
        val copy = arrayListOf<CustomMarker>().apply { addAll(dataset) }
        copy.sortWith(comparator)
        dataset = copy
        notifyDataSetChanged()
}
```

4.3 Floor selection

At the next step the task at hand was the ability to select a floor and be able to only see the markers belonging to that floor. However we want to still be able to see the markers of other floors so we should display them with a lesser opacity to differentiate them.

We added three views for this feature, an up and down button, and a dropdown where a user can select any floor that is available directly. Similarly to the previous feature, when the main activity gets created we bind our new views to their respective properties and then set event listeners for them. When an item from the dropdown gets selected we set the selected floor to the floor the user clicked. When the up or down button is clicked we select the next available floor up or down from the current one. When fetching POIs or objects from anyplace and SMAS we now also add the floor of each marker to the marker list in addition to adding the floor of each marker to a set that has the purpose of storing all the floors with markers in them. This means that some floors might be missing. For example we might have the ability to only jump to floors -1, 0, 1, 3 and 7 if the collected markers only appear at those floors. Then after all markers are fetched we initialize the dropdown with all the available floors and now each time we change the selected floor only the markers in the floor appear in search results and the heatmap while in the map we can see all of them with the makers on other floors being displayed in a lighter opacity.

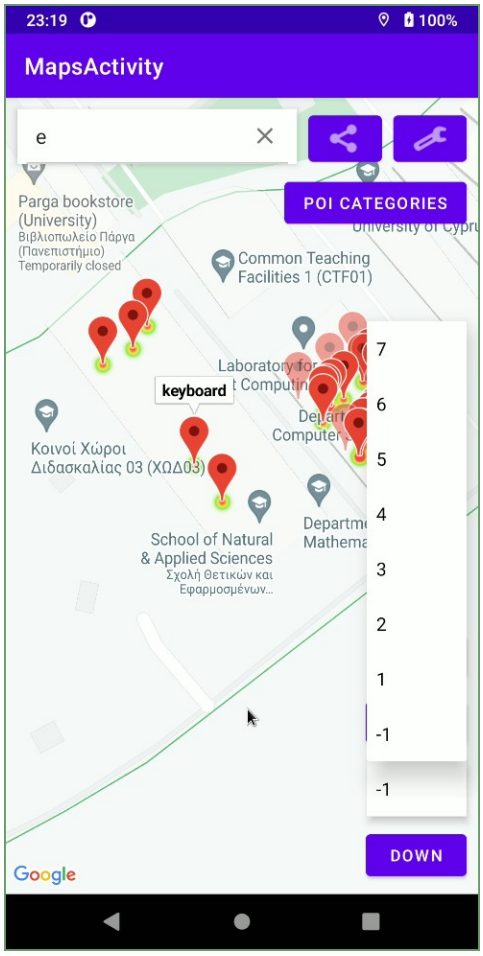


Fig 4.3: The interface of OMS when the dropdown is clicked to reveal all the available floors a user can now select. Under the dropdown we can see the down button and above the dropdown is the up down but it is hidden by the popup of the dropdown. Notice Floor 0 is absent. That means there were no markers on floor 0 and so there is no way to select that floor. We can also see that some of the markers are more opaque than others and the ones that have less opacity do not have a heatmap at their base like the opaque ones. This is because the ones with less opacity are in other floors.

MapsActivity.kt

In filterMarkers we add a line for changing opacity based on whether a marker is on the current floor or not.

```
private fun filterMarkers() {
        for (m in markerList) {
                m.marker?.isVisible = mMap.cameraPosition.zoom > 18
                m.marker?.alpha = if (m.floor == floor) 1f else .5f
        }
```

```
}
```

In updateHeatmapAndSearch we check if a marker is on the selected floor and only then do we include it in the heatmap and search results.

```
private fun updateHeatmapAndSearch() {
        val newList: ArrayList<LatLng> = ArrayList()
        val filteredList: ArrayList<CustomMarker> = ArrayList()
        for (m in markerList) {
                if (m.marker != null && m.floor == floor) {
                        newList.add(m.marker.position)
                        filteredList.add(m)
                }
        }
        ...
}
```

Inside the onCreate function we handle the new buttons and their actions. Firstly we bind all new views to their respective properties and then define the event listeners. The up button goes up one floor, the down goes one down, and the floorView's onItemSelected goes to the selected floor each one of them calls the function that updates the selected floor and then updates the heatmap, search and opacity of the markers. Going up and down is not guaranteed to be a difference of 1 floor level, if we are at floor level 1 and there were no markers on floor 2 but there were markers on floor 3, going up would take us to floor 3. In general there would be no way to select floor 2 if it had to markers there.

```
floorUpView = findViewById(R.id.floor_up)
floorDownView = findViewById(R.id.floor_down)
floorView = findViewById(R.id.floor)
floorView.onItemSelectedListener = object: AdapterView.OnItemSelectedListener {
        override fun onNothingSelected(parent: AdapterView<*>?) {}
        override fun onItemSelected(
                parent: AdapterView<*>?, view: View?, position: Int, id: Long
        ) {
                floor = floor_list[position]
                setFloorViewText()
        }
}
floorUpView.setOnClickListener {
        val cur_index = floor_list.indexOf(floor)
```

```
        if (cur_index > 0) {
                floor = floor_list[cur_index - 1]
                setFloorViewText()
        }
}
floorDownView.setOnClickListener {
        val cur_index = floor_list.indexOf(floor)
        if (cur_index < floor_list.size - 1) {
                floor = floor_list[cur_index + 1]
                setFloorViewText()
        }
}
```

4.4 Filter by category

This task is very similar to the previous one. For a user to better find an object he is searching for, filtering by category can be extremely useful.

In this feature we added a button to the layout for opening the category selection dialog. At the beginning we bind the button view to a property and setup an on click listener that opens up the category dialog when the button is clicked. When fetching object and POI data alongside the floor we also now add the marker's category to the marker list in addition to a category set. When fetching from anyplace finishes the category dialog is built. The dialog includes all the categories from the set alongside an "All" option at the top that is used for viewing all the objects. When an item from the dialog is clicked a category is set and the heatmap, search and the displayed markers on the map get updated showing only the filtered results.

MapsActivity.kt

A new function called getFilters was defined which when given a marker checks if it should be displayed based on the selected category.
filterMarkers was updated so it uses the getFilters function.

```
private fun getFilters(marker: CustomMarker): Boolean {
        return (cur_category == null || cur_category == marker.category)
}

private fun filterMarkers() {
```

```
        for (m in markerList) {
            m.marker?.isVisible = mMap.cameraPosition.zoom > 18 &&
                getFilters(m)
            m.marker?.alpha = if (m.floor == floor) 1f else .5f
        }
}
```

updateHeatmapAndSearch also now uses getFilters to include marker data in the search results and the heatmap based on the selected category.

```
private fun updateHeatmapAndSearch() {
        val newList: ArrayList<LatLng> = ArrayList()
        val filteredList: ArrayList<CustomMarker> = ArrayList()
        for (m in markerList) {
            if (m.marker != null && m.floor == floor && getFilters(m)) {
                ...
            }
        }
        ...
}
```

The function createCategoryDialog was added in order to initialize the dialog that displays the list of available categories as well as an extra item called “All” which shows all markers. This also has the onClick event listener which sets the cur_category property to the selected category and updates the heatmap, search results and displayed markers.

```
private fun createCategoryDialog() {
        val builder = AlertDialog.Builder(this)
        val list = categories.toMutableList()
        list.sort()
        list.add(0, "All")
        builder.setTitle("POI categories")
            .setItems(
                list.toTypedArray(),
                DialogInterface.OnClickListener { dialog, which ->
                    cur_category = if (which == 0) null else list[which]
                    updateFilters()
                    Log.i("cur_category", cur_category.toString())
                }
            )
        categoryDialog = builder.create()
}
```

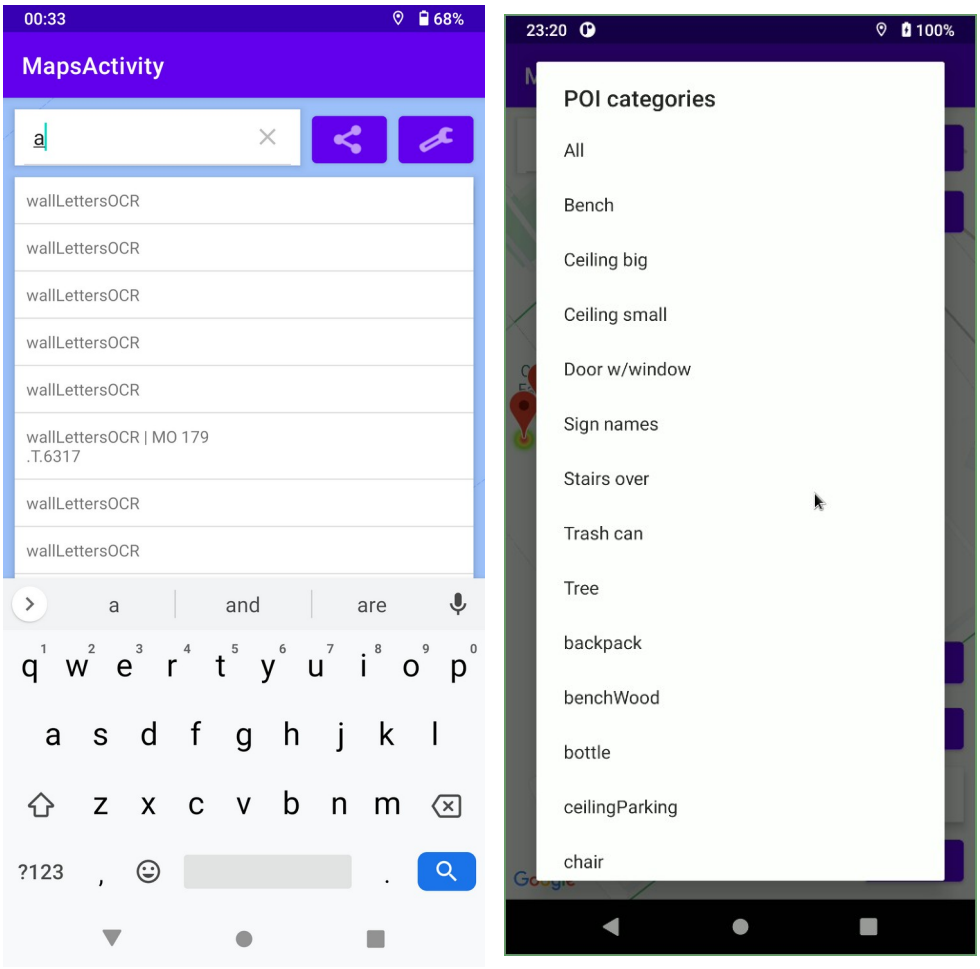


Fig 4.4: (Right) The category selection dialog when it gets opened. (Left) how the search results look after wallLettersOCR gets selected from the category selection dialog.

4.5 Display the user's location

We need to display the user's location on the map. It should update every few seconds so that it is kept up to date. Since we are not using anyplace-android we will have to settle with GPS.

We added google play services' location module to the dependencies, added the Access Fine Location permission to our manifest and then added code that checks to see if the user granted the required permissions. If he did we start fetching location data every 20 seconds and set a marker on the map to that location. Additionally we added a button that when it gets pressed it moves the map to the user's location and if it doesn't find

the location of the user like when it is still fetching or when the user didn't grant permissions it shows a toast that says that location information is not available.

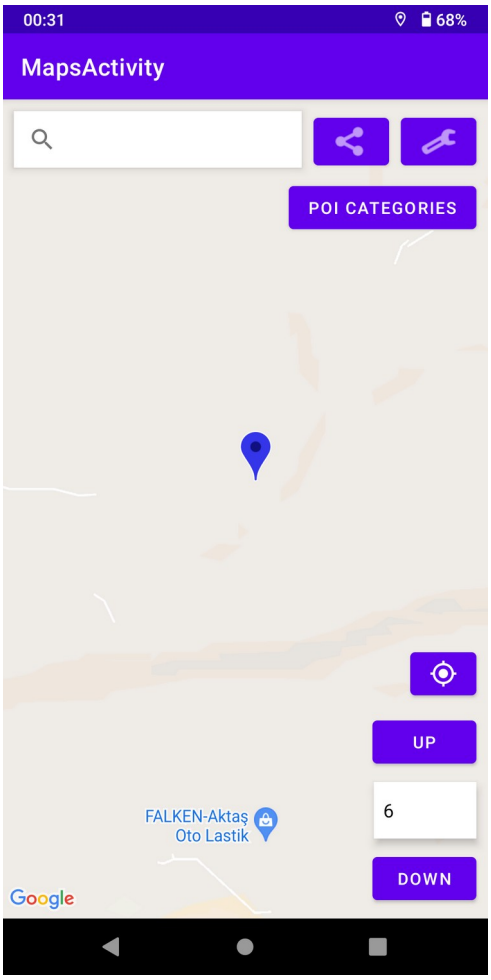


Fig 4.5: Marked as a blue marker is the location of the user. In the bottom right corner above the "UP" button we can see the navigation button which moves the map to the user's location.

MapsActivity.kt

onRequestPermissionResult triggers when the user grants or rejects the ACCESS FINE LOCATION permission. If the permission gets granted we call another new function called fetchLocation.

```
override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<out String>,
        grantResults: IntArray
) {
```

```
        super.onRequestPermissionsResult(requestCode, permissions, grantResults)
        if (requestCode == PERMISSION_REQUEST_ACCESS_FINE_LOCATION) {
                Log.d("permission", grantResults.toString())
                when (grantResults[0]) {
                        PackageManager.PERMISSION_GRANTED -> fetchLocation()
                        PackageManager.PERMISSION_DENIED →
                                Log.e("Permission denied", "User didn't like it")
                }
        }
}
companion object {
        private const val PERMISSION_REQUEST_ACCESS_FINE_LOCATION = 100
}
```

fetchLocation first checks if the permissions were correctly granted. If they were then it sets up a callback that gets called every 20 seconds. This callback whenever is called has a new location as a parameter which is then used to position the marker on the map.

```
private fun fetchLocation() {
        var fusedLocationProviderClient = LocationServices
                .getFusedLocationProviderClient(this@MapsActivity)
        if (ActivityCompat.checkSelfPermission(this,
                Manifest.permission.ACCESS_FINE_LOCATION) !=
                PackageManager.PERMISSION_GRANTED &&
                ActivityCompat.checkSelfPermission( this,
                        Manifest.permission.ACCESS_COARSE_LOCATION) !=
                        PackageManager.PERMISSION_GRANTED
        ) {
                ActivityCompat.requestPermissions(
                        this,
                        arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
                        PERMISSION_REQUEST_ACCESS_FINE_LOCATION
                )
                return
        }
        Log.d("permissions", "are correct")
        var locationRequest = LocationRequest.create();
        locationRequest!!.priority = LocationRequest.PRIORITY_HIGH_ACCURACY;
        locationRequest!!.interval = 20 * 1000;
        Log.d("location request", "create")
        var locationCallback = object: LocationCallback() {
                override fun onLocationResult(locationResult: LocationResult) {
                        Log.d("location request", "got results")
                        if (locationResult == null) {
                                return;
                        }
```

```
                        Log.d("location request", "is not null")
                        for (location in locationResult.locations) {
                                if (location != null) {
                                        Log.d(
                                                "location request",
                                                 location.latitude.toString() + " " +
                                                        location.longitude.toString()
                                        )
                                        var ll = LatLng(
                                                location.latitude, location.longitude
                                        )
                                        if (curLocation != null) {
                                                curLocation!!.position = ll
                                        } else {
                                                curLocation = mMap.addMarker(
                                                MarkerOptions()
                                                        .position(ll)
                                                        .icon(
                                                                BitmapDescriptorFactory
                                                                        .defaultMarker(
                                                                BitmapDescriptorFactory
                                                                        .HUE_BLUE
                                                                        )
                                                        )
                                                )
                                        }
                                }
                        }
                }
        };
        fusedLocationProviderClient.requestLocationUpdates(locationRequest,
        locationCallback, Looper.getMainLooper())
}
```

In addition we call fetchLocation at the end of the onCreate function.

In the onCreate function we also bind the location button with the correct property and then like always setup the relevant event listener. In this case it is an onClickListener that moves the map to the users location if it is there, else it shows a toast notifying the user that the Location is still not available. This could be triggered for a few reasons. Mainly a user didn't grant the required permissions, secondly the location wasn't fetched yet, the user clicked on the button too quickly, and lastly maybe there is no gps enabled on the device or it doesn't have gps signal.

```
locationView = findViewById(R.id.my_location)
```

```
locationView.setOnClickListener {
        if (curLocation != null) {
                mMap.moveCamera(
                        CameraUpdateFactory.newLatLngZoom(
                                curLocation!!.position, 20F
                        )
                )
        } else {
                val toast = Toast.makeText(
                        applicationContext,
                        "Location not yet available",
                        Toast.LENGTH_SHORT
                )
                toast.show()
        }
}
```

Chapter 5

Evaluation

This chapter deals with a few metrics of the previously introduced five features. For each one of those features we measured the time the feature took to run in addition to calculating approximate time complexity. The device that was used for the tests was an S62 Pro Caterpillar smart phone running Android 11, here are a few more of its specs: Qualcomm Snapdragon 660 Octo Core 2.0GHz, 6GB RAM, 128GB of storage, 4000mAh non-removable lithium battery with a battery life of up to two days [17].

5.1 Displaying markers on the map

Here is the code of the feature with only code relevant to the calculation of the time complexity left in.

We can see that filterMarkers has a time complexity of O(N) where N is the number of markers.

updateHeatmapAndSearch also is categorized as O(N)
updateFilters uses the previous two functions so we have O(N) + O(N) = O(N)

fetchAnyplaceMarkers has a for loop going through all the markers (M) that were returned from the anyplace api, so it has a time complexity of O(M), additionally it calls updateFilters so we get, O(M) + O(N) = O(N)

Same goes for addMarkers, we have a for loop going through all the objects returned from the lashfire db file (K), and then it calls updateHeatmapAndSearch so O(K) + O(N) = O(N)

onMapReady initially calls addMarkers and fetchAnyplaceMarkers so O(N) + O(N) = O(N) if we did the math normally however we would get about (K + N) + (M + N + N) = 3N + K + M. Lastly each time the map stops moving filterMarkers runs so that is another O(N)

```
private fun filterMarkers() {
        for (m in markerList) {
                ...
        }
}

// Creates the heatmap
private fun updateHeatmapAndSearch() {
        ...
        for (m in markerList) {
                ...
        }
        ...
}

private fun updateFilters() {
        filterMarkers()
        updateHeatmapAndSearch()
}

private fun fetchAnyplaceMarkers() {
        scope.launch {
                try {
                        ...
                        for (a in result.pois) {
                                ...
                        }

                        ...
                } catch (e: Exception) {
                        Log.e("fetch_response", e.toString())
```

```
                } finally {
                        updateFilters()
                }
        }
}

private fun addMarkers(search: String?) {
        ...
        for (a in markers) {
                ...
        }
        updateHeatmapAndSearch()
        ....
}

override fun onMapReady(googleMap: GoogleMap) {
        ...
        mMap.setOnCameraIdleListener {
                Log.i("zoom level", mMap.cameraPosition.zoom.toString())
                filterMarkers()
        }

        addMarkers("")
        fetchAnyplaceMarkers()
        ...
}
```

A few measurements were taken of the time it took to run the previous functions and here are the results for a K = 465, M = 0 and N = 465

The filterMarkers an average (26 runs) took 63ms to run.
For updateHeatmapAndSearch on the first run when the heatmap is first constructed runs usually on avg (3 runs) for 69ms, while for each run after the first one the average (8 runs) is 18ms.

The updateFilters function on average (8 runs) needed 81ms to complete.

fetchAnyplaceMarkers was tested however the anyplace API was not functioning so we could only test it in its down state the result was on average (3 runs) 11ms. However that does not count the asynchronous code that actually is supposed to fetch the data for that part of the code we got an average of 7413ms.

On the similar addMarkers function which did have a database to query and was functioning properly on average (3 runs) gave us a metric of 411ms. This seems like a pretty slow function even though db queries are usually fast, the reason is that we are creating half a thousand markers and that takes the most amount of time. The query itself is really fast. For example in the section 5.5 when comparing a run that created a marker vs one that moved it the difference was 28ms.

The map idle listener yielded over (18 runs) an average of 69ms.

Finally the onMapReady function took on average (3 runs) 519ms.

5.2 POI searchbar

MarkerAdapter.kt

In getFilter's performFiltering, the filter can take up to O(N) where N the number of markers in the dataset, while the contains inside the filter can take up to O(l*m) if l is the length of one string and m the length of the other, this gives us a total of about O(Nlm)

For orderData we make a copy of the dataset and then sort it, the copy takes about O(N) time while the sort takes O(NlogN) so the total time complexity of the function is O(NlogN)

```
class MarkerAdapter(...) : BaseAdapter(), Filterable {
        override fun getFilter(): Filter {
                return object : Filter() {
                        ...
                        override fun performFiltering(...): Filter.FilterResults {
                                ...
                                dataset.filter {
                                        it.title.lowercase().contains(queryString)
                                }
                                ...
                        }
                }
        }

        fun orderData(center: LatLng) {
                val copy = arrayListOf<CustomMarker>().apply { addAll(dataset) }
                copy.sortWith(comparator)
```

```
                ...
        }
}
```

MapsActivity.kt

In the onQueryTextChange listener we call the filter we defined in the previous file so each time a character is pressed or deleted in the search bar we have an O(Nlm) function running

```
searchView.setOnQueryTextListener(object : SearchView.OnQueryTextListener {
        ...
        override fun onQueryTextChange(query: String?): Boolean {
                listView.visibility = View.VISIBLE
                listAdapter.filter.filter(query)
                return false;
        }
})
```

In the onMapRead function in the event listener for when the map stops moving we call orderData which has a time complexity of O(NlogN) adding the previous O(N) that this listener had, we still get O(NlogN)

```
mMap.setOnCameraIdleListener {
        ...
        listAdapter.orderData(mMap.cameraPosition.target)
}
```

Now for the time metrics that were observed for this feature:

perform filtering over (10 runs) run for an average of 2ms.

orderData over (9 runs) run for an average of 90ms.

onQueryTextChange over (10 runs) run for 2ms.

The idleListener's duration (9 runs) on average increased to 175ms.

5.3 Floor selection

MapsActivity.kt

setFloorViewText calls updateFilters so it's an O(N) time complexity

```
private fun setFloorViewText() {
        ...
        updateFilters()
}
```

For the onItemSelected listener, and the setOnClickListener of the floor views, each one of them calls setFloorViewText so their time complexity is of O(N).

```
floorView.onItemSelectedListener = object: AdapterView.OnItemSelectedListener {
        override fun onItemSelected(...) {
                ...
                setFloorViewText()
        }
}

floorUpView.setOnClickListener {
        ...
        setFloorViewText()
        ...
}

floorDownView.setOnClickListener {
        ...
        setFloorViewText()
        ...
}
```

Inside the onMapReady we have toMutableList which we can say is a maximum of O(F), for sortDescending the most probable time complexity is O(FlogF) for the new adapter its most probably again O(F), last() can be O(1) and setFloorViewText is for sure O(N) where N the total number of markers while F is the total number of Floors so we have a total time complexity of O(N) + O(FlogF) but F is usually much smaller than N (think 7 maximum floors vs hundreds or thousands of markers) so we can say onMapReady is still of O(N).

```
floor_list = floors.toMutableList()
floor_list.sortDescending()
floorView.adapter = ArrayAdapter<Int>(
        this,
```

```
        android.R.layout.simple_spinner_dropdown_item,
        floor_list
)
floor = floor_list.last()
setFloorViewText()
```

For the floor selection feature here are the times recorded:

Over (11 runs) setFloorViewText run for 99ms on average.

For onItemSelected an average (6 runs) we observed a duration of 98ms.

For the onClick listeners of the floorUp and floorDown view for (4 runs) the average duration was of 95ms.

5.4 Filter by category

MapsActivity.kt

In the createCategoryDialog we have a sort function on all categories (C) so O(ClogC), additionally when an item is clicked because of updateFilters we have a time complexity of O(N).

```
private fun createCategoryDialog() {
        ...
        list.sort()
        ...
        builder.setTitle("POI categories")
                .setItems(
                        list.toTypedArray(),
                        DialogInterface.OnClickListener { dialog, which ->
                                cur_category = if (which == 0) null else list[which]
                                updateFilters()
                                Log.i("cur_category", cur_category.toString())
                        }
                )
        ....
}
```

In the fetchAnyplaceMarkers we call createCategoryDialog so we get a time complexity of O(N) + O(C) = O(N)

```
private fun fetchAnyplaceMarkers() {
        ...
        createCategoryDialog()
        ...
}
```

The functionCreateCategoryDialog was measured and we found that over 3 runs on average it would run for 22ms for a total number of 83 categories.

5.5 Display the user's location

MapsActivity.kt

We can see here that for L the number of locations received from the location request, every 20s we have a function of time complexity O(L) running. L should be very small or even 1 usually.

```
private fun fetchLocation() {
        ...
        var locationCallback = object: LocationCallback() {
                override fun onLocationResult(locationResult: LocationResult) {
                        ...
                        for (location in locationResult.locations) {
                                ...
                        }
                }
        };
        ...
}
```

In this case when onLocationResult was measured two cases where observed, firstly on the first onLocationResult invocation when a marker needs to be added to the map OMS needs a bit more time at 29ms over (3 runs) while on all subsequent invocations over (11 runs) an average run of 1ms was observed.

5.6 Conclusions

Overall the time it took for all the functionality was not noticeable to the end user. The only barely noticeable delays were on startup of the application were the application has to initially render the map and fetch data from the lashfire database file. Things will surely get worse on much bigger datasets, and a few problems do exist. Optimization potential does exist. One such example is filterMarkers currently this function always loops over the whole marker list whenever it is called, a more reasonable approach would be to check at the beginning of the function if the zoom level has crossed the determined threshold for showing or hiding markers, or if the selected category or selected floor have been changed and only then loop over the array. This would reduce lag in the most common case where the user just moves the map without zooming.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In the present project we developed an android application for managing objects in indoor spaces. The project is still in its infancy and can be greatly improved and extended in the future. OMS, the application we developed has the purpose of fulfilling a rising problem that appeared from the creation of indoor localization systems and their use of object identification for localization or general collection of points of interest for a greater user experience.

Due to the prominent use of indoor spaces alongside the fluency of the general population in the use of modern mobile smart phone devices, the need for indoor localization systems became great. As solutions for this problem arose so did the problem of managing objects that these systems collected over the span of their use. Such a system is the Smart Alert System or SMAS [11] that was developed by the University of Cyprus, it utilizes computer vision and no existent infrastructure in order to localize the user. This system in its first stage collects objects in a space through computer vision. OMS can use a database file that was generated by SMAS and visualize collected objects on an interactive map. OMS can do the same for objects fetched from the Anyplace API [12], Anyplace being the predecessor and basis of SMAS, and an indoor localization system using wifi fingerprints and later IoT devices like BLE beacons, Ultra wide-band tags and other such devices.

The OMS implementation was evaluated with metrics taken of its performance. At this stage the user experience is pretty satisfying, it can handle half a thousand objects without the user noticing any delays. Optimizations can be done however they should be done at a later stage when OMS becomes more feature complete.

6.2 Future Work

OMS can be considered to be at a decent place as a first time implementation from scratch. However there is a lot of functionality currently missing. In this section we make a few suggestions for what should be done to continue this work and to complete the OMS android-application.

A building selection should be implemented, displaying all the markers in the whole database can become slow with an extraordinarily big amount of objects. A way to limit that would be to add the ability for the user to see the location of buildings on the map and to select which building to view. This will improve efficiency as less markers will need to be displayed and the user will be able to navigate and visualise all the buildings that have objects in them. Another feature could be the display of floor plans of buildings for the better perception of objects in an indoor space. Additionally updating objects either by moving them, changing other details about the objects, deleting them or even creating new ones can be very useful in the future giving the user an intuitive interface for improving, correcting or just updating possible mistakes about collected objects. A great addition to the list of improvements would be the use of anyplace's android library in order to incorporate already implemented functionality like indoor localization and navigation, this would greatly increase the accuracy of the location of the user as the current implementation uses GPS which can be extremely inaccurate for indoor spaces. After all sorts of important functionality is implemented possible optimizations can be detected and incorporated to the existing system in so the user experience is boosted further.

Chapter 7

Bibliography

1 A. Konstantinidis, G. Chatzimilioudis, D. Zeinalipour-Yazti, P. Mpeis, N. Pelekis and Y. Theodoridis, "Privacy-Preserving Indoor Localization on Smartphones," in IEEE Transactions on Knowledge and Data Engineering, vol. 27, no. 11, pp. 3042-3055, 1 Nov. 2015, doi: 10.1109/TKDE.2015.2441724.

2 B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. Commun. ACM, 13(7):422–426, July 1970.

3 C. Laoudias, G. Constantinou, M. Constantinides, S. Nicolaou, D. Zeinalipour-Yazti, and C. G. Panayiotou. The airplace indoor positioning platform for android smartphones. In Proceedings of the 13th IEEE Intl. Conference on Mobile Data Management, MDM ’12, pages 312–315, 2012.

4 C. Rizos, A. G. Dempster, B. Li, and J. Salter, “Indoor positioning techniques based on wireless LAN,” in 1st IEEE Intl. Conf. on Wireless Broadband and Ultra Wideband Communications. IEEE, 2007, pp. 13–16.

5 D. Zeinalipour-Yazti, C. Laoudias, K. Georgiou and G. Chatzimilioudis, "Internet-Based Indoor Navigation Services," in IEEE Internet Computing, vol. 21, no. 4, pp. 54-63, 2017, doi: 10.1109/MIC.2017.2911420.

6 D. Coppens, A. Shahid, S. Lemey, B. Van Herbruggen, C. Marshall and E. De Poorter, "An Overview of UWB Standards and Organizations (IEEE 802.15.4, FiRa, Apple): Interoperability Aspects and Future Research Directions," in IEEE Access, vol. 10, pp. 70219-70241, 2022, doi: 10.1109/ACCESS.2022.3187410.

7 D. Lymberopoulos, et. al., “A realistic evaluation and comparison of indoor location technologies: Experiences and lessons learned”, In ACM/IEEE IPSN 2015.

8 E. Mackensen, M. Lai and T. M. Wendt, "Bluetooth Low Energy (BLE) based wireless sensors," SENSORS, 2012 IEEE, Taipei, Taiwan, 2012, pp. 1-4, doi: 10.1109/ICSENS.2012.6411303.

9 H. Lu, X. Cao, and C.S. Jensen, “A foundation for efficient indoor distance-aware query processing”, In IEEE ICDE 2012.

10 L. David, A. Hassidim, Y. Matias, M. Yung and A. Ziv, "Eddystone-EID: Secure and Private Infrastructural Protocol for BLE Beacons," in IEEE Transactions on

Information Forensics and Security, vol. 17, pp. 3877-3889, 2022, doi: 10.1109/TIFS.2022.3214074.

11 Mpeis, Paschalis & Zeinalipour-Yazti, D & Vicario, J. (2022). ZERO INFRASTRUCTURE GEOLOCATION OF NEARBY FIRST RESPONDERS ON RO-RO VESSELS. ICCAS 2022. 10.3940/rina.iccas.2022.25.

12 P. Mpeis et al., "The Anyplace 4.0 IoT Localization Architecture," 2020 21st IEEE International Conference on Mobile Data Management (MDM), Versailles, France, 2020, pp. 218-225, doi: 10.1109/MDM48529.2020.00045.

13 R. Want, "An introduction to RFID technology," in IEEE Pervasive Computing, vol. 5, no. 1, pp. 25-33, Jan.-March 2006, doi: 10.1109/MPRV.2006.2.

14 S. Quincozes, T. Emilio and J. Kazienko, "MQTT Protocol: Fundamentals, Tools and Future Directions," in IEEE Latin America Transactions, vol. 17, no. 09, pp. 1439-1448, September 2019, doi: 10.1109/TLA.2019.8931137.

15 S. He and S. . -H. G. Chan, "Wi-Fi Fingerprint-Based Indoor Positioning: Recent Advances and Comparisons," in IEEE Communications Surveys & Tutorials, vol. 18, no. 1, pp. 466-490, Firstquarter 2016, doi: 10.1109/COMST.2015.2464084.

16 Aislelabs, “The Hitchhikers Guide to iBeacon Hardware: A Comprehensive Report by Aislelabs” Available at: http://goo.gl/eCsUp9, May 4, 2015.

17 *Cat S62 Pro Tech Specs | Cat phones USA*. (2022, August 17). Cat Phones USA. https://www.catphones.com/en-us/cat-s62-pro-smartphone/cat-s62-pro-tech-specs/

18 *Excalidraw — Collaborative whiteboarding made easy*. (n.d.). Excalidraw. https://excalidraw.com/

19 *Indoor Maps - About - Google Maps*. (n.d.). https://www.google.com/maps/about/partners/indoormaps/

20 *Indoor Positioning System Company with Advanced Navigation Solutions | Mapsted*. (n.d.). https://mapsted.com/

21 Inpixon. (n.d.). *Inpixon: Real-Time Location Systems for Industrial IoT*. Inpixon. https://inpixon.com/

22 INTRANAV. (2023, January 24). *INTRANAV.IO The Enterprise IoT RTLS Platform for the Digital-Twin - INTRANAV*. https://intranav.com/en/iot-rtls-suite/intranav-io-the-enterprise-iot-rtls-platform-for-the-digital-twin/

23 *MazeMap Indoor Maps and Wayfinding*. (n.d.). https://www.mazemap.com/

24 Proximi.io. (2023, April 27). *API-First Indoor Navigation solution for Mobile, Web and Kiosks - Proximi.io*. https://proximi.io/

25 Sqlite3todot. (n.d.). *GitHub - chunky/sqlite3todot: Convert sqlite3 databases to directed graph .dot files*. GitHub. https://github.com/chunky/sqlite3todot

26 U.S. Environmental Protection Agency, “Report to Congress on indoor air quality”, Volume 2. EPA/400/1-89/001C. Washington, DC. 1989.

Appendix

MapsActivity.kt

```
package cy.ac.ucy.cs.anyplace.bms

import android.Manifest
import android.content.DialogInterface
import android.content.Intent
import android.content.pm.PackageManager
import android.net.Uri
import android.os.Bundle
import android.os.Looper
import android.util.Log
import android.view.View
import android.widget.*
import androidx.appcompat.app.AlertDialog
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import com.google.android.gms.location.*
import com.google.android.gms.maps.*
import com.google.android.gms.maps.model.*
import com.google.maps.android.heatmaps.HeatmapTileProvider
import cy.ac.ucy.cs.anyplace.bms.databinding.ActivityMapsBinding
import cy.ac.ucy.cs.anyplace.bms.db.AppDatabase
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.Job
import kotlinx.coroutines.launch
import kotlin.system.measureTimeMillis


class MapsActivity : AppCompatActivity(), OnMapReadyCallback {

        val database: AppDatabase by lazy { AppDatabase.getDatabase(this) }

        private lateinit var mMap: GoogleMap
        private var heatmapProvider: HeatmapTileProvider? = null
        private var heatmapOverlay: TileOverlay? = null
        private var initialCenter: LatLng? = null
        private lateinit var binding: ActivityMapsBinding
        private lateinit var searchView: SearchView
        private lateinit var listView: ListView
        private lateinit var floorView: Spinner
        private lateinit var floorUpView: Button
        private lateinit var floorDownView: Button
```

```
private lateinit var locationView: Button
private lateinit var shareView: Button
private lateinit var categoriesButtonView: Button
private var categoryDialog: AlertDialog? = null
private var floor = 0
private var floors = mutableSetOf<Int>()
private var floor_list = mutableListOf<Int>()
private var categories = mutableSetOf<String>()
private var cur_category: String? = null
lateinit var listAdapter: MarkerAdapter
var markerList: ArrayList<CustomMarker> = ArrayList();
private var curLocation: Marker? = null
var txt: String = ""

protected val scope = CoroutineScope(
        Job() + Dispatchers.Main
)

private fun getFilters(marker: CustomMarker): Boolean {
        return (cur_category == null || cur_category == marker.category)
}

private fun filterMarkers() {
        val t = measureTimeMillis {
                for (m in markerList) {
                        m.marker?.isVisible = mMap.cameraPosition.zoom > 18
                                && getFilters(m)
                        m.marker?.alpha = if (m.floor == floor) 1f else .5f
                }
        }
        txt += "filterMarkers: " + markerList.size + " | " + t + "ms\n"
}

private fun updateHeatmapAndSearch() {
        val t = measureTimeMillis {
                val newList: ArrayList<LatLng> = ArrayList()
                val filteredList: ArrayList<CustomMarker> = ArrayList()
                for (m in markerList) {
                        if (m.marker != null && m.floor == floor &&
                                getFilters(m)) {
                                newList.add(m.marker.position)
                                filteredList.add(m)
                        }
                }
                listAdapter.newData(filteredList)
                listAdapter.notifyDataSetChanged()
```

```
                    if (newList.size == 0) {
                        heatmapOverlay?.remove()
                        heatmapProvider = null
                        return
                    }

                    if (heatmapProvider == null) {
// Create a heat map tile provider, passing it the latlngs of the police stations.
                        heatmapProvider = HeatmapTileProvider.Builder()
                            .data(newList)
                            .build()

// Add a tile overlay to the map, using the heat map tile provider.
                        HeatmapOverlay = mMap.addTileOverlay(
                            TileOverlayOptions().tileProvider(
                                heatmapProvider as HeatmapTileProvider
                            )
                        )
                    } else {
                        heatmapProvider!!.setData(newList)
                        heatmapOverlay?.clearTileCache()
                    }
                }
                txt += "heatmap: " + markerList.size + " | " + t + "ms\n"
            }

            private fun updateFilters() {
                val t = measureTimeMillis {
                    filterMarkers()
                    updateHeatmapAndSearch()
                }
                txt += "filters: " + markerList.size + " | " + t + "ms\n"
            }

            private fun setFloorViewText() {
                val t = measureTimeMillis {
                    floorView.setSelection(floor_list.indexOf(floor))
                    updateFilters()
                }
                Log.d("steFloorViewText", "" + t + "ms")
            }

            private fun createCategoryDialog() {
                val t = measureTimeMillis {
                    val builder = AlertDialog.Builder(this)
                    val list = categories.toMutableList()
                    list.sort()
```

```
                list.add(0, "All")
                builder.setTitle("POI categories")
                        .setItems(
                                list.toTypedArray(),
                                DialogInterface.OnClickListener { dialog, which ->
                                        cur_category = if (which == 0) null else
                                                list[which]
                                        updateFilters()
                                        Log.i(
                                                "cur_category",
                                                cur_category.toString()
                                        )
                                }
                        )
                categoryDialog = builder.create()
        }
        Log.d("createCategory", "" + t + "ms |" + categories.size)
}

override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = ActivityMapsBinding.inflate(layoutInflater)
        setContentView(binding.root)

// Obtain the SupportMapFragment and get notified when the map is ready to be
used.
        val mapFragment = supportFragmentManager
                .findFragmentById(R.id.map) as SupportMapFragment
        mapFragment.getMapAsync(this)

        searchView = findViewById(R.id.idSv)
        listView = findViewById(R.id.list)

        listAdapter = MarkerAdapter(
                this,
                ArrayList<CustomMarker>()
        )
        listView.adapter = listAdapter

        listView.setOnItemClickListener { parent, view, position, id ->
                val element = parent.getItemAtPosition(position) as
                        CustomMarker
                Log.i("clicked element", element.toString())
                listView.visibility = View.INVISIBLE
                if (element.marker != null) {
                        mMap.animateCamera(CameraUpdateFactory
```

```
                        .newLatLngZoom(element.marker.position, 22F))
                element.marker.showInfoWindow()
        }
}

locationView = findViewById(R.id.my_location)
floorUpView = findViewById(R.id.floor_up)
floorDownView = findViewById(R.id.floor_down)
floorView = findViewById(R.id.floor)
floorView.onItemSelectedListener = object:AdapterView
        .OnItemSelectedListener {
        override fun onNothingSelected(parent: AdapterView<*>?) {}

        override fun onItemSelected(
                parent: AdapterView<*>?,
                view: View?,
                position: Int,
                id: Long
        ) {
                val t = measureTimeMillis {
                        floor = floor_list[position]
                        setFloorViewText()
                }
                Log.d("item selected", "" + t + "ms")
        }
}

floorUpView.setOnClickListener {
        val t = measureTimeMillis {
                val cur_index = floor_list.indexOf(floor)
                if (cur_index > 0) {
                        floor = floor_list[cur_index - 1]
                        setFloorViewText()
                }
        }
        Log.d("floor up", "" + t + "ms")
}

floorDownView.setOnClickListener {
        val t = measureTimeMillis {
                val cur_index = floor_list.indexOf(floor)
                if (cur_index < floor_list.size - 1) {
                        floor = floor_list[cur_index + 1]
                        setFloorViewText()
                }
        }
        Log.d("floor down", "" + t + "ms")
```

```
        }

        locationView.setOnClickListener {
            if (curLocation != null) {
                mMap.moveCamera(CameraUpdateFactory
                    .newLatLngZoom(curLocation!!.position, 20F))
            } else {
                val toast = Toast.makeText(
                    applicationContext,
                    "Location not yet available",
                    Toast.LENGTH_SHORT
                )
                toast.show()
            }
        }

        shareView = findViewById(R.id.share)
        shareView.setOnClickListener {
            val center = mMap.cameraPosition.target
            val link = "https://bms.ucy.ac.cy/" + center.latitude.toString() +
                "&" + center.longitude.toString()
            val intent= Intent()
            intent.action=Intent.ACTION_SEND
            intent.putExtra(Intent.EXTRA_TEXT, link)
            intent.type="text/plain"
            startActivity(Intent.createChooser(intent,"Share To:"))
        }

        categoriesButtonView = findViewById(R.id.categories_button)
        categoriesButtonView.setOnClickListener {
            categoryDialog?.show()
        }

// on below line we are adding on query
// listener for our search view.
        searchView.setOnQueryTextListener(
            object : SearchView.OnQueryTextListener {
                override fun onQueryTextSubmit(query: String?):
                    Boolean {
                        return false
                    }

                override fun onQueryTextChange(query: String?):
                    Boolean {
                    val t = measureTimeMillis {
                        listView.visibility = View.VISIBLE
                        listAdapter.filter.filter(query)
```

```
                                }
                                Log.d("txt change: ", "" + t + "ms")
                                return false;
                        }

                }
        )
        searchView.setOnCloseListener {
                listView.visibility = View.INVISIBLE
                false
        }

        fetchLocation()
}

private fun fetchLocation() {
        var fusedLocationProviderClient = LocationServices
                .getFusedLocationProviderClient(this@MapsActivity)
        if (ActivityCompat.checkSelfPermission(
                this, Manifest.permission.ACCESS_FINE_LOCATION) !=
                PackageManager.PERMISSION_GRANTED &&
                ActivityCompat.checkSelfPermission(
                        this, Manifest.permission.ACCESS_COARSE_LOCATION) !=
                        PackageManager.PERMISSION_GRANTED)
        {
                ActivityCompat.requestPermissions(this,
                        arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
                        PERMISSION_REQUEST_ACCESS_FINE_LOCATION)
                return
        }
        Log.d("permissions", "are correct")

        var locationRequest = LocationRequest.create();
        locationRequest!!.priority = LocationRequest
                .PRIORITY_HIGH_ACCURACY;
        locationRequest!!.interval = 20 * 1000;
        Log.d("location request", "create")
        var locationCallback = object: LocationCallback() {
                override fun onLocationResult(locationResult: LocationResult) {
                        val t = measureTimeMillis {
                                Log.d("location request", "got results")
                                if (locationResult == null) {
                                        return;
                                }
                                Log.d("location request", "is not null")
                                for (location in locationResult.locations) {
                                        if (location != null) {
```

```
                                        Log.d(
                                                "location request",
                                                location.latitude.toString()
                                                + " " +
                                                location.longitude
                                                        .toString())
                                        var ll = LatLng(
                                                location.latitude,
                                                location.longitude
                                        )
                                        if (curLocation != null) {
                                                curLocation!!.position = ll
                                        } else {
                                                curLocation = mMap
                                                        .addMarker(
                                                MarkerOptions()
                                                        .position(ll)
                                                        .icon(
                                                BitmapDescriptorFactory
                                                        .defaultMarker(
                                                BitmapDescriptorFactory
                                                        .HUE_BLUE
                                                )
                                                        )
                                                        )
                                        }
                                }
                        }
                        Log.d("fetchLocation", "" + t + "ms |" +
                                locationResult.locations.size)
                }
        };
        fusedLocationProviderClient.requestLocationUpdates(locationRequest,
        locationCallback, Looper.getMainLooper())
}

private fun fetchAnyplaceMarkers() {
        scope.launch {
                val t = measureTimeMillis {
                        try {
                                val result = AnyplaceApi
                                        .retrofitService.getPOIs(DataPOI(
                                                buid = "building_3ae47293-69d1-
45ec-96a3-f59f95a70705_1423000957534"
                                        ))
                                Log.i("fetch_response", result.toString())
```

```
                                        var lastMarker: LatLng? = null
                                        val latlongs: MutableList<LatLng?> = ArrayList()

                                        for (a in result.pois) {
                                                if (a.name == "Connector") continue
                                                Log.i("pios: ", a.coordinates_lat + "
                                                        " + a.coordinates_lon)
                                                val ll =  LatLng(
                                                        a.coordinates_lat.toDouble(),
                                                        a.coordinates_lon.toDouble()
                                                )
                                                latlongs.add(ll)
                                                lastMarker = ll
                                                val title = a.name + if (a.name !== null &&
a.description !== null && a.description.isNotEmpty()) " | " + a.description else ""
                                                val marker = mMap
                                                        .addMarker(
                                                        MarkerOptions()
                                                        .position(ll).title(title)
                                .icon(BitmapDescriptorFactory.fromAsset("red_dot.png")))

                                                val marker_category = a.pois_type
                                                markerList.add(CustomMarker(
                                                        marker, title,
                                                        a.floor_number,
                                                        marker_category
                                                ))
                                                categories.add(marker_category)
                                                floors.add(a.floor_number)
                                        }

                                        if (lastMarker != null && initialCenter == null)
mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(lastMarker, 18F))

                                } catch (e: Exception) {
                                        Log.e("fetch_response", e.toString())
                                } finally {
                                        Log.i("categories", categories.toString())
                                        createCategoryDialog()
                                        updateFilters()
                                }
                        }
                        txt += "fetch anyplace: " + markerList.size + " | " + t + "ms\n"
                }
        }
```

```
private fun addMarkers(search: String?) {
        val t = measureTimeMillis {
                val latlongs: MutableList<LatLng?> = ArrayList()

                val dao = database.fingerprint_locationDao()
                val markers = dao.getLocations()
                Log.i("markers", markers.toString())

                var lastMarker: LatLng? = null
                for (a in markers) {
                        val ll = LatLng(a.x.toDouble(), a.y.toDouble())
                        latlongs.add(ll)
                        lastMarker = ll
                        var title = a.name + if (a.ocr !== null &&
                                a.ocr.isNotEmpty()) " | " + a.ocr else ""
                        val marker = mMap
                                .addMarker(MarkerOptions()
                                        .position(ll).title(title))
                        val marker_category = a.name
                        markerList.add(CustomMarker(marker, title, a.deck,
                                marker_category))
                        categories.add(marker_category)
                        floors.add(a.deck)
                }

                updateHeatmapAndSearch()

                if (lastMarker != null && initialCenter == null)
                        mMap.moveCamera(CameraUpdateFactory
                                .newLatLngZoom(lastMarker, 18F))
        }
        txt += "addMarkers: " + markerList.size + " | " + t + "ms\n"
}

override fun onRequestPermissionsResult(
        requestCode: Int, permissions: Array<out String>,
        grantResults: IntArray
) {
        super.onRequestPermissionsResult(requestCode, permissions,
                grantResults)
        if (requestCode == PERMISSION_REQUEST_ACCESS_FINE_LOCATION) {
                Log.d("permission", grantResults.toString())
                when (grantResults[0]) {
                        PackageManager.PERMISSION_GRANTED ->
                                fetchLocation()
                        PackageManager.PERMISSION_DENIED ->
                        Log.e("Permission denied", "User didn't like it")
```

```
                }
        }
}

companion object {
        private const val PERMISSION_REQUEST_ACCESS_FINE_LOCATION = 100
}

override fun onMapReady(googleMap: GoogleMap) {
        val t = measureTimeMillis {
                mMap = googleMap

                val action: String? = intent?.action
                val data: Uri? = intent?.data
                var coords = data?.pathSegments?.get(0)?.split("&")

                mMap.setOnCameraIdleListener {
                        Log.i("zoom level",
                                mMap.cameraPosition.zoom
                                        .toString())
                        val t2 = measureTimeMillis {
                                filterMarkers()
                                listAdapter.orderData(mMap
                                        .cameraPosition.target)
                        }
                        txt += "idle camera: " + markerList.size + " | " + t2 +
                                "ms\n"
                        Log.d("txt", txt)
                        txt = ""
                }

                addMarkers("")
                fetchAnyplaceMarkers()

                floor_list = floors.toMutableList()
                floor_list.sortDescending()
                floorView.adapter = ArrayAdapter<Int>(this,
                        android.R.layout.simple_spinner_dropdown_item,
                        floor_list)
                floor = floor_list.last()

                if (coords == null) {
                        coords = mutableListOf()
                        coords.add(markerList.last().marker
                                ?.position?.latitude.toString())
                        coords.add(markerList.last().marker
                                ?.position?.longitude.toString())
```

```
                        floor = markerList.last().floor
                    }

                    setFloorViewText()
                    initialCenter = LatLng(coords[0].toDouble(),
                        coords[1].toDouble())

                    mMap.moveCamera(CameraUpdateFactory
                        .newLatLngZoom(initialCenter!!, 20F))
                    Log.i("coords", LatLng(coords[0].toDouble(),
                        coords[1].toDouble()).toString())
                }
                txt += "onMapReady: " + markerList.size + " | " + t + "ms\n"
                Log.d("txt", txt)
                txt = ""
            }
        }
```

MarkerAdapter.kt

```
package cy.ac.ucy.cs.anyplace.bms


import android.content.Context
import android.util.Log
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.view.animation.Animation
import android.widget.*
import androidx.recyclerview.widget.RecyclerView
import com.google.android.gms.maps.model.LatLng
import com.google.android.material.snackbar.Snackbar
import cy.ac.ucy.cs.anyplace.bms.db.User
import kotlin.math.pow
import kotlin.math.sqrt
import kotlin.system.measureTimeMillis

class MarkerAdapter(
        private val context: Context,
        private var dataset: List<CustomMarker>
) : BaseAdapter(), Filterable {
        private lateinit var item_title: TextView
        private var markers: List<CustomMarker> = dataset

        override fun getCount(): Int {
                return markers.size
        }
```

```
override fun getItem(position: Int): Any {
        return markers[position]
}

override fun getItemId(position: Int): Long {
        return position.toLong()
}

override fun getView(position: Int, convertView: View?, parent: ViewGroup):
        View? {
        var convertView = convertView
        convertView = LayoutInflater.from(context)
                .inflate(R.layout.list_item, parent, false)
        item_title = convertView.findViewById<TextView>(R.id.item_title)
        item_title.text = markers[position].title
        return convertView
}

override fun getFilter(): Filter {
        return object : Filter() {
                override fun publishResults(charSequence: CharSequence?,
                        filterResults: Filter.FilterResults) {
                        markers = filterResults.values as List<CustomMarker>
                        notifyDataSetChanged()
                }

                override fun performFiltering(charSequence: CharSequence?):
                        Filter.FilterResults {
                        var filterResults: FilterResults
                        val t = measureTimeMillis {
                                val queryString = charSequence
                                        ?.toString()?.lowercase()

                                filterResults = FilterResults()
                                filterResults.values = if (queryString == null ||
                                        queryString.isEmpty())
                                                dataset
                                        else
                                                dataset.filter {
                                                        it.title.lowercase()
                                                                .contains(
                                                                        queryString
                                                                )
                                                }
                        }
                        Log.d("performFiltering: ", "" + t + "ms")
```

```
                        return filterResults
                    }
                }
            }

            fun newData(data: List<CustomMarker>) {
                dataset = data
            }

            private fun latLngDistance(l1: LatLng, l2: LatLng): Double {
                return sqrt((l1.latitude - l2.latitude).pow(2.0) + (l1.longitude -
                        l2.longitude).pow(2.0))
            }

            fun orderData(center: LatLng) {
                val t = measureTimeMillis {
                    val comparator = Comparator { m1: CustomMarker, m2:
                            CustomMarker ->
                        var l1 = m1.marker!!.position
                        var l2 = m2.marker!!.position

                        var res = (latLngDistance(center, l1) -
                                latLngDistance(center, l2))
                        if (res == 0.0) 0 else if (res < 0.0) -1 else 1
                    }
                    val copy = arrayListOf<CustomMarker>().apply
                            { addAll(dataset) }
                    copy.sortWith(comparator)
                    dataset = copy
                    notifyDataSetChanged()
                }
                Log.d("orderData: ", "" + t + "ms")
            }
        }
```

ObjectLocation.kt

```
package cy.ac.ucy.cs.anyplace.bms.db

import androidx.room.ColumnInfo
import androidx.room.DatabaseView
import androidx.room.Entity
import androidx.room.ForeignKey
import androidx.room.PrimaryKey

@DatabaseView("
        SELECT FL.*, ocr, name
```

```
        FROM FINGERPRINT_LOCATION FL, FINGERPRINT_OBJECT FO, OBJECT O
        WHERE FL.flid = FO.flid and O.oid = FO.oid
        GROUP BY x, y
")
data class ObjectLocation (
        val flid: Int,
        val uid: String,
        val time: Int,
        val timestr: String,
        val buid: String,
        val y: Float,
        val x: Float,
        val deck: Int,
        val modelid: Int,
        val ocr: String?,
        val name: String
)
```