

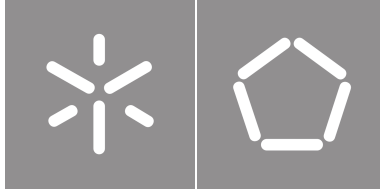


Universidade do Minho

Escola de Engenharia

Ulisses Tiago Simões Araújo

Where@UM – Where is the classroom for my next lecture? – The problem of the space's geometry



Universidade do Minho

Escola de Engenharia

Ulisses Tiago Simões Araújo

Where@UM – Where is the classroom for my next lecture? – The problem of the space's geometry

Master Thesis

Master in Informatics Engineering

Work developed under the supervision of:

Adriano Jorge Cardoso Moreira

Filipe Miguel Lopes Meneses

October, 2022

COPYRIGHT AND TERMS OF USE OF THIS WORK BY A THIRD PARTY

This is academic work that can be used by third parties as long as internationally accepted rules and good practices regarding copyright and related rights are respected.

Accordingly, this work may be used under the license provided below.

If the user needs permission to make use of the work under conditions not provided for in the indicated licensing, they should contact the author through the RepositoriUM of Universidade do Minho.

License granted to the users of this work



Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International CC BY-NC-SA 4.0

<https://creativecommons.org/licenses/by-nc-sa/4.0/deed.en>

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the Universidade do Minho.

Braga

26th November 2022

(Place)

(Date)

(Ulisses Tiago Simões Araújo)

Abstract

Where@UM – Where is the classroom for my next lecture? – The problem of the space's geometry

In recent years more and more complex structures have been built. Buildings and locations which users must navigate efficiently so they can reach their appointments in a timely fashion, such as hospitals, universities and airports. Unfortunately technologies such as GPS are not well adapted to indoor locations and therefore do not provide a solution to this problem. Indoor mapping has been subject to increased amounts of research in the past few years and a plethora of different solutions have started to arise although none completely fulfill every requirement this problem presents. This thesis is done in conjunction with others with the final objective of creating the prototype of a mobile application and system that will be able to precisely locate where a user is inside of an indoor location through showing them their location on a floorplan. It will more specifically focus on the modeling aspect of the space geometry in an efficient way that can be used by this application. The purpose of this dissertation is to document the research done to choose the most appropriate data format, the development of a conversion method of the available data to the chosen format, and the development of web services and mobile application components that will provide this information to the end user. Additionally the development of a web application that, with the results obtained throughout this investigation process, helps keep track of the progress of the radio mapping will also be documented.

Keywords: Indoor Positioning, Floorplans Service, GeoJSON, Mobile Application, Webservices

Resumo

Where@UM – Onde é a sala da minha próxima aula? – O problema da geometria do espaço

Nos últimos anos estruturas cada vez mais complexas tem sido construídas. Edifícios e localizações que devem ser eficientemente percorridos pelos utilizadores para que estes possam chegar aos seus destinos e marcações atempadamente. Estas localizações podem incluir instituições como hospitais, universidades e até mesmo aeroportos. Infelizmente tecnologias como o GPS não funcionam corretamente dentro destes edifícios devido ao seu sinal ser bloqueado pelas paredes dos edifícios. Por esta razão medidas alternativas de mapeamento indoor tem vindo a ser estudadas nos últimos anos e uma variedade de diferentes soluções tem começado a surgir, no entanto, até à data nenhuma destas soluções resolve completamente o problema em questão. A presente tese é feita em conjunto com outras com o objetivo final de produzir o protótipo de uma aplicação móvel e sistema que sejam capazes de localizar precisamente onde um utilizador se encontra dentro de uma localização interior, através do suporte de plantas dos edifícios em questão. No caso desta dissertação, o foco será, mais especificamente, o aspeto da modelação da geometria do espaço tendo como seu propósito a documentação da pesquisa do formato de dados mais apropriado para este use case, o desenvolvimento de um método de conversão de formatos que possibilite a conversão dos dados existentes para o formato escolhido, e todo o desenvolvimento e implementação dos web services e da aplicação móvel. Adicionalmente também é feita a documentação do desenvolvimento de uma aplicação web que, utilizando as funcionalidades do web service das plantas, facilita a visualização do progresso do processo de radio mapping.

Palavras-chave: Posicionamento Indoor, Serviço de Plantas, GeoJSON, Aplicação Mobile, Webservices

Contents

List of Figures	ix
List of Tables	xi
Acronyms	xiv
1 Introduction	1
1.1 Motivation	1
1.2 Objectives	2
1.3 Structure	2
2 State of the art	4
2.1 Requirements	4
2.2 Standard indoor map formats and models	5
2.2.1 Indoor OpenStreetMap	5
2.2.2 OGC IndoorGML	6
2.2.3 CityGML	8
2.2.4 IFC	9
2.2.5 IMDF	9
2.2.6 SVG	10
2.2.7 Discussion and comparison	10
2.3 Conclusion	11
3 Floorplan Service Development	13
3.1 Process to obtain the geographically referenced GeoJSON files	13
3.2 Server interface definition	23
3.2.1 “getByFloor“ endpoint	24
3.2.2 “getByIds“ endpoint	25

3.2.3	“getByLocation“ endpoint	26
3.3	Possible database systems	28
3.3.1	Systems already implemented	28
3.3.2	File system based floorplan storage solutions	31
3.3.3	Conclusion and chosen solution	32
3.4	Server database model definition and description	32
3.4.1	Version 1	32
3.4.2	Version 2	34
3.4.3	Version 3	35
3.5	Technologies used and file structure	35
3.6	Implemented Solution Validation	37
4	Mobile application	38
4.1	Objective of the integration of the floorplan service	39
4.2	Components used	40
4.3	Strategy	41
5	Web application for collected data	43
5.1	Objectives of the web application development	43
5.2	Requirements of the web application	44
5.3	Ways of representing the data	44
5.3.1	Marker Clustering	44
5.3.2	Heatmap	46
5.3.3	Conclusions	46
5.4	Technologies used	47
5.5	Interface definition and description	48
5.5.1	All samples endpoint	49
5.5.2	Temporal endpoints	50
5.6	Implemented Solution Validation	51
6	Conclusions and future work	53
6.1	Future work	54
6.1.1	Mobile application	54
6.1.2	Web application	54
6.1.3	Floorplans web service	54
	Bibliography	55
	Appendices	

Annexes

I Annex 1

57

List of Figures

1	An example of an adjacency graph as perceived by IndoorGML [18]	7
2	Example of a Multi-Layered Space Model [7]	7
3	Different LOD levels of CityGML [8]	8
4	Visualisation of CityGML LOD4 building model for Berlin main train station [9]	8
5	Settings to import SVG file into Inkscape	15
6	Ungrouping SVG file in Inkscape	15
7	Cleaned up and usable SVG file	16
8	settings to correctly export to PDF	16
9	Settings to import a PDF file into Inkscape	17
10	Settings to export a PDF as DXF to use in QGIS	17
11	Button to create a shapefile	18
12	Shapefile creation settings	18
13	Four example control points in an ungeoreferenced floorplan	18
14	Four equivalent control points in the desired georeferenced location	19
15	Shapefile exportation settings	19
16	Example of a CSV containing the Ungeoreferenced control point coordinates	19
17	AnotherDXFImport settings	20
18	Georeferenced GeoJSON result	21
19	Setting the appropriate Coordinate Reference System (CRS) on the GeoJSON layers	21
20	Saving the layers to GeoJSON	22
21	Settings to correctly save as GeoJSON	22
22	Removed properties from the GeoJSON files	23
23	Example response of the getByLocation endpoint	28
24	Version 1 of the floorplan's server database model	33
25	Version 2 of the floorplan's server database model	34
26	Version 3 of the floorplan's server database model	35

27	Floorplan service architecture	36
28	Snippet of a GeoJSON file	37
29	System architecture that supports the mobile application	38
30	Mobile application home page	39
31	Pick Location method of crowdsourcing	40
32	Representative Diagram of the Strategy to keep the floorplan being displayed updated . .	42
33	Example of a more clustered Marker Clustering representation of the crowdsourcing samples	45
34	Example of a less clustered Marker Clustering representation of the crowdsourcing samples	45
35	Example of an heatmap representation of the crowdsourcing samples	46
36	Web Application Architecture	47
37	Table containing the crowdsourced samples relevant to the Web Application	48
38	Example of an all samples endpoint response	49
39	Results of the test for 500 users 5000 requests	51
40	Results of the test for 750 users 7500 requests	52
41	Results of the test for 1000 users 10000 requests	52

List of Tables

1	Comparison between IndoorOSM, IndoorGML,IFC, CityGML,IMDF and SVG [19]	11
2	Request parameters for the getByFloor GET request	24
3	Possible errors for the getByFloor GET request	24
4	Request parameters for the getByIds GET request	25
5	Possible errors for the getByIds GET request	25
6	Request parameters for the getByLocation GET request	26
7	Possible errors for the getByLocation GET request	27
8	Comparison between MongoDB and PostGIS [6]	30
9	Request parameters for all Web Application GET requests	49
10	Possible errors for the all samples GET request	50
11	Possible errors for the temporal samples endpoints GET request	51

List of Listings

I.1	Script to remove extra properties	57
-----	---	----

Acronyms

CRS Coordinate Reference System [ix](#), [8](#), [10](#), [11](#), [21](#), [22](#)

LOD Level Of Detail [8](#), [11](#)

OGC Open Geospatial Consortium [6](#), [8–10](#), [29](#)

ORM Object Relational Mapper [35](#)

Introduction

1.1 Motivation

The continuous growth and extension on the size and complexity of public buildings, airports, shopping centers, hospitals and university campi has reinforced the need for an effective indoor navigation system. However, there are very few solutions implemented for this problem in the current days. Some of the industry giants such as Microsoft, Apple and Google have recently been investing in this market with some products already available on the market, such as Apple Indoor Mapping [3].

The majority of the research so far has been done on indoor navigation, with studies concentrating on the positioning technology and its viability, with several techniques such as BLE beacons and Wi-Fi fingerprinting having been tested in several technical implementations, since the GPS technology used for outdoor navigation is not feasible in an indoor setting. Furthermore, it is also a must to not only take into account the growing complexity and size of the buildings in question, but also the different characteristics that differentiate indoor from outdoor environments, such as the navigation, orientation and the constant changes in accessibility to spaces, as well as the different landmarks, which, as a consequence, turn the navigation task into an even bigger challenge.

A fundamental component of an indoor navigation system is a data model to accurately represent the entirety of the information needed for the good functioning of not only the navigation system but also the accurate presentation of data to the end-user.

Nonetheless, it is also a requirement to think about how to present this information to the user in the application's front-end. Commonly, the floor plans approach is used, but considering the high level of detail that more complex buildings can have, this is not a trivial decision to make. Three-dimensional models are also a possibility. Unfortunately, more complex interfaces are required to enable the user's good model manipulation. Furthermore, complications can arise with both approaches when the buildings are complex and the navigating tasks can present a real challenge. Currently, there is no real consensus on which approach is superior to its competitor.

1.2 Objectives

The main expectation of this project is the idealization and development of the prototype of a mobile application and its application services intended to allow a user to, without complications and in a friendly fashion, be able to navigate through complex indoor locations. It is essential to remember that there may be different types of users with different permissions or accessibility needs that the application should take into account. Therefore, the application should essentially be a Google Maps for indoor locations but still be integrated with the outdoors if needed.

This dissertation's main goal and theme, integrated into the *Lab4U&Spaces* project, are to investigate which is the best way to represent the space and routes, as in, what is the best data model to store the information needed and the most compatible with the current map visualization tools. This leads us to a few objectives of this dissertation:

- Figuring out which is the best data format to represent the floorplans of the indoor buildings and the respective meta-data.
- What is the best way to show the building's floor plans to the end-user through a mobile application.
- What is the best way to show the routes provided by the application services to the end-user along with the building's floor plans.

Ultimately, the expected result of this dissertation is the prototype of an indoor navigation system that will work inside of the University of Minho but hopefully be modular enough to be easily extrapolated to other use cases such as airports, shopping centers, among others.

1.3 Structure

This document is divided into seven main chapters, each having a few subsections:

- The first chapter is essentially a short introduction to the dissertation, its motivation, structure and the main objectives it seeks to achieve.
- The second chapter describes the state of the art of the space's geometry problem in indoor mapping, briefly explaining 5 data models and comparing their respective upsides and downsides.
- The third chapter describes the whole process from beginning to end of how the floorplan service was conceived and the measures that had to be taken to obtain the correct data for its healthy functioning.
- The fourth chapter gives insight into how and why the integration between the mobile application and the floorplan service described in the previous chapter was done.

- The fifth chapter explains the thought process and development process behind a web application, based on the floorplan service, used to monitor the process of (radio) mapping the university buildings.
- Finally, the sixth and last chapter introduces a final point to this document where conclusions are made, some setbacks during development are exposed and possible future work is discussed.

State of the art

Good space models are fundamental when seeking an appropriate solution for indoor navigation, but especially when the objective is to show their results to the end-user through an application. This is because one of their main requirements will be exchanging data between systems and services. Therefore, for this to be accomplished, there must be an excellent middle ground between expressive power and efficiency; that is, a standard data format should express the amount of information needed for the perfect functioning of the system and lessen the loss of information, while, on the other hand, any overheads that may happen during transmission should also be reduced. Nonetheless, it is essential to remember that this is not the only requirement that any standard format for indoor mapping should fulfill. Some others will be mentioned, therefore, in the following section.

2.1 Requirements

As is always the case, whenever we wish to find a solution to a problem, we must first define the requirements that the desired solution must fulfill. Therefore, this section will present the minimum requirements for an indoor map format to be a possible solution.

- *Coordinate Reference System in indoor spaces:* The use of an (x,y) in 2D or (x,y,z) in a 3D coordinate reference system is fundamental in an indoor space to track the state of the user inside of an indoor location, while latitude and longitude coordinates become relevant as a way to ease the transition between indoor and outdoor locations.
- *Structures of indoor space:* While not as common in outdoor locations, indoor spaces often have some structure or hierarchy to them, with areas often being subdivided into wings, levels, and zones. This information is also relevant and should be kept in the indoor maps.
- *Hierarchical representations of the space:* Some indoor position algorithms can exploit hierarchical representations of the space, such as with buildings, floors, corridors, rooms, etc., being then able to provide more accurate estimates of the location of a user. Therefore, these maps should provide

representations for inclusion (a space being inside another), adjacency (spaces being beside each other), proximity, accessibility (whether it is possible to access one space from another or not), and other properties of this nature.

- *Constraints of indoor spaces:* The accessibility constraints must also be considered since they vary with the user, time and location. This is not only due to the opening hours in commercial areas, universities, and other types of places, but also because some people possess physical limitations that sometimes do not allow them to use stairs or go through specific locations (for example, a wheelchair user).
- *Representation of indoor features:* In addition to doors, windows and stairs the model should represent other constraints such as temporary venues or events.
- *Seamless integration between outdoor and indoor locations:* It is required for there to be no issue transitioning between outdoor locations and indoor spaces in case the user wants to go from one indoor location to another at any point, from outdoors to indoors or vice versa.
- *Compatibility with maps used for visualization of routes:* For an easier way for the end-user to perceive the routes provided by the application services and understand where they must go, it is required for there to be a smooth integration of the floor plans with the visualization libraries for it to be possible to show the current location exactly on the floor plan provided by the indoor map data format.
- *Navigable areas:* While the accessibility has been previously mentioned, it is also important to keep in mind the trajectories that a user cannot follow due to the desired area being restricted or simply inaccessible so this should also be adequately represented.
- *Integration and compatibility with topological models:* It is imperative for the model to be compatible with topological models since these are fundamental for computing the routes required for the well-functioning of the application.

2.2 Standard indoor map formats and models

This section will present the indoor map formats and models subject to analysis and comparison among themselves in a later section.

2.2.1 Indoor OpenStreetMap

OpenStreetMap (OSM)[23] was idealized as a free editable map of the whole Earth created entirely by an Open Source Community of volunteers. The maps are released under the Open Database License.

Upon creating or modifying a map in OSM, a user tags a point, line, polygon or relation. Tags change the semantics of the place by adding properties like the name, type of building, type of amenity and opening hours, among other kinds of features. Although created for outdoors, it was possible to adapt it to indoors. This suggestion for indoor environments was made in 2011 by Marcus Goetz[13]. OSM-based indoor maps utilize plans from architects and any floor plans in a raster (JPG) or vector (SVG) based formats. Some important projects in indoor mapping in OSM are: OpenStreetMap[22] and OpenIndoor[21], both of which show 3D indoor building representation solutions.

The main drawback of OSM when it comes to indoor locations is the lack of a real standard and the decreasing interest in this initiative during recent years which led to many attempts at a solution being outdated and no longer all that relevant or extendable.

2.2.2 OGC IndoorGML

IndoorGML[18] was published as a standard for exchange format and data model of indoor map data in 2016 by the [Open Geospatial Consortium \(OGC\)](#). The main goals are to enrich the expressive power whilst reducing the loss of information during the data conversion process and establish the foundations of an indoor spatial data model.

It is called a cellular space model since it presumes that the space is represented by several non-overlapping units called *cells* that aim to represent spaces such as rooms, corridors, or toilets. The union of all cells is also a subset of the given indoor space which means there may be shadow areas not covered by any cells, and every position does not necessarily belong to a cell. Hence IndoorGML provides a model to describe details of these *cells* such as *cell geometry and semantics*, *topology between cells*, and *multi-layered space model*:

- *Cell Geometry*: To build a map using IndoorGML, every cell must have closed geometry such as a surface or solid and have its geometry specified as a point, 2D surface or 3D solid.
- *Cell Semantics*: Along with the geometry of the cell, semantics such as its properties and features are added.
- *Topology between Cells*: Figure 1 is an example of an adjacency graph (topology) derived from an indoor layout. In this case, the topology between cells is the adjacency and connectivity between two cells since in IndoorGML there is no overlapping between cells. Therefore, if two cells share a passable boundary (doors, for instance), it is called a connection. There are two options to represent the graph derived from the topographic space. The first option is to include the geometries of the node and edge as a point and curve, respectively, with this being called the geometric properties geometric graph. And the second option is not to represent any geometric properties, thus being called a logical graph. However, in most applications for indoor navigation, we need geometric data to calculate indoor distance; therefore, the first approach is often used.

- *Multi-layer Space Model*: An indoor space may be perceived differently depending on who views it. As seen in Figure 2, Room 3 is a single cell for walking pedestrians; however, for someone with different accessibility needs, such as a person with a wheelchair due to the step in the middle of the room, it is no longer a single room but now divided in Room 3a and 3b. Therefore, this leads to two different *space layers* consisting of two indoor layout configurations as perceived by two different types of users. With a *multi-layer space model* it is possible to integrate a wide array of space layers through an inter-layer connection. With this model, it becomes easily possible to represent the hierarchical structure of an indoor space, for instance.

For the sake of making extensibility easier, IndoorGML possesses a modular structure as well. The core of IndoorGML contains the data model for cell geometry, topology and the multi-layered space model, whilst the indoor navigation extension model, which provides the semantic extension model for indoor navigation, is so far the only extension module defined on top of the core model. However, many other extension modules may yet be defined depending on the required applications.

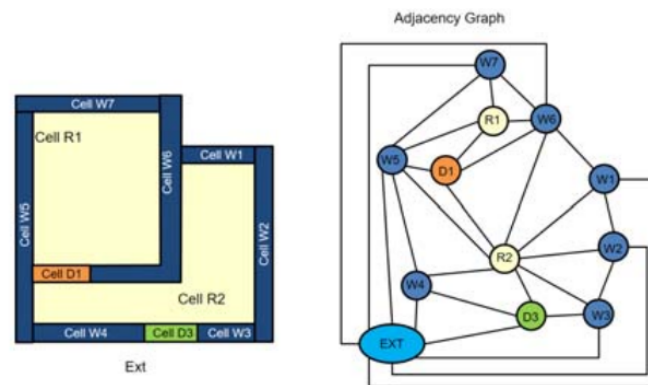


Figure 1: An example of an adjacency graph as perceived by IndoorGML [18]

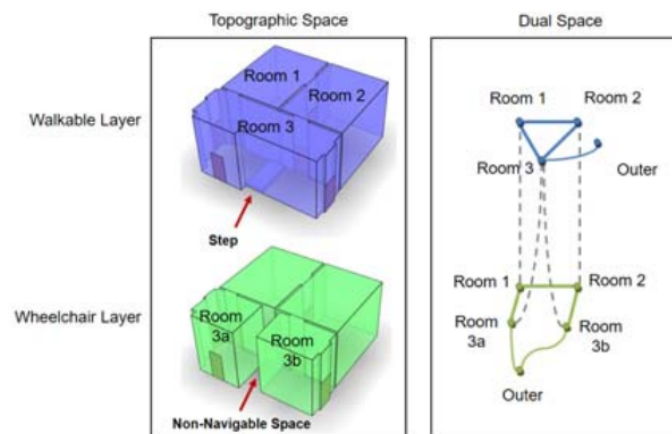


Figure 2: Example of a Multi-Layered Space Model [7]

2.2.3 CityGML

City Geography Markup Language (CityGML)[10] is an open data model and XML-based format for the storing and exchanging 3D city models, an official OGC Standard that can be used free of charge. It is a concept for the modeling and exchanging of 3D city and landscape models while being defined as an application XML schema of GML 3.1.1 that aims to provide an appearance model and a basic entity model with 3D geometry representations of city objects.

It possesses different Level Of Detail (LOD)s, as seen in Figure 3, with Level 4 representing a 3D indoor model. It provides a great level of support for the many requirements of indoor mapping, supporting rich semantics and accommodating both geodetic and engineering CRSs.



Figure 3: Different LOD levels of CityGML [8]

A shortcoming of this format is its lack of multiple LODs for indoor spaces. There is only the singular LOD 4 for indoor locations, which makes it an all-or-nothing approach, where LOD 4 shows everything indoors whilst LOD3 shows the hollow shell of the building solely. The ability to provide more LODs for indoor spaces would permit many levels of generalization, such as generalizations of rooms or entire floors. It also includes support for using the world coordinate system, allowing outdoor and indoor spatial information to be seamlessly integrated to route a user outdoors between buildings. It presents limited support for fixed and movable obstacles and absolutely no support for dynamic ones.

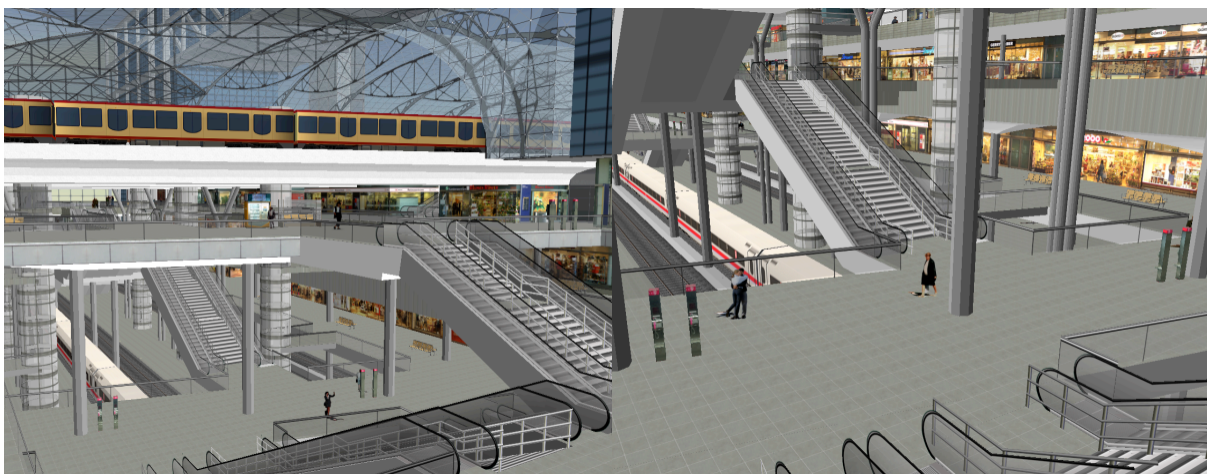


Figure 4: Visualisation of CityGML LOD4 building model for Berlin main train station [9]

While CityGML is a viable framework for generating and storing indoor maps, it does not exactly fulfill the map visualization requirement since it is not optimized for that function. For that capability, it can be exported to X3D and SVG formats, which work with most web browsers, or to 2D GIS web mapping formats, with X3D being a promising display for 3D indoor representations and SVG holding some promise for its 2D visualization.

2.2.4 IFC

IFC[15] is both a standard data exchange model for building information modeling (BIM) and a file format used by the architecture, engineering, construction and facilities management industries. It is semantically rich, object-oriented and possesses native 3D geometry. IFC has a large number of classes dedicated to buildings. It keeps on being extended with more and more classes to allow this format to comprise the complex construction management of large civil engineering projects.

It is a very suitable model to provide a precise 2D and 3D indoor map due to the high level of detail, native 3D geometry and rich semantics. Aside from the notations for walls, slabs, connections between different floors (such as stairs and elevators), doors and windows, these models can contain information about spaces in rooms and furniture. There are also a few particular classes, such as `ifcSpace` and `ifcVirtualElement`, among others, that are of particular interest to indoor positioning and location-based services. The information related to connectivity obtained indirectly through the information about doors, windows and stairs makes the automatic creation of a network relatively simple and straightforward. The notion of space in this format opens up new horizons for enhancing the localization and navigation of users and assets. Due to its high geometric resolution, IFC can also be a valuable source for map-based localization of assets or people when kept up to date. However, BIM models only exist for newer structures natively in BIM.

2.2.5 IMDF

Indoor Mapping Data Format (IMDF)[4] is, since 2021, an [OGC](#) Community Standard, originally developed by Apple for use in their project Indoor Maps Program. It provides a generalized and comprehensive model for any indoor location, giving a basis for orientation, navigation and discovery. This format is heavily based on GeoJSON (its output partially consists of various GeoJSON files with the different data types defined).

It is a mobile-friendly, compact, human-readable, temporally aware (can express information dependant on time) and highly extensible data model for any indoor space, specifically developed with this purpose in mind. Due to this fact, various features are available to help describe indoor spaces. A few of these that may be used as an example are the following:

- *Venue*: Models the presence, location and approximate extent of a place. It is a feature of the type Polygon and can additionally have a category, restriction, name, hours of operation, phone number, among other details.

- *Detail*: A detail represents the location and extent of a physical object.
- *Section*: A section describes the approximate extent of an area representing a theme (such as a duty-free store at an airport, entertainment area, etc.). It is particularly useful to establish a sort of hierarchy among a building's locations.
- *Amenity*: A feature used to describe pedestrian amenities such as elevators or stairs.

Aside from these, there are still dozens of other features provided by this format, but, obviously, analysing and mentioning all of this is not within the scope of the present document.

2.2.5.1 GeoJSON

GeoJSON[5] is an open standard geospatial data interchange format, that provides the capability to represent simple geographic features and their nonspatial attributes. It supports the geometric types *Point*, *LineString*, *Polygon*, *MultiPoint*, *MultiLineString* and *MultiPolygon*. It is important to note also that GeoJSON features are not only used to represent entities of the physical world, but can also be used by mobile routing and navigation apps to describe their service coverage.

While not an OGC standard, a small mention is made in this document to this standard since, as stated above, IMDF is heavily based on and built upon it.

2.2.6 SVG

SVG is an XML-based markup language for describing 2D vector graphics viewable with web browsers. It is a text-based open web standard for describing images that can be rendered without any size issues. Its files are well defined in XML, thus providing the capability to be searched, indexed, scripted and compressed. The latest version, 1.1, includes support for lines and areas (i.e., rectangle, circle, ellipse, and polygon) as well as rasters; however, SVG currently does not support point geometry. SVG also allows the definition of re-usable groups and links to other XML files for custom attributes, although not a native capability [1]. It natively supports engineering CRSs and has a flexible system for specifying geodetic CRSs using one of three methods: a web-based uniform resource identifier (URI), a well-known CRS identifier, or directly defining the CRS within the XML document.

2.2.7 Discussion and comparison

The table presented below compares the discussed standard formats, including Indoor OSM, OGC IndoorGML, IFC, OGC CityGML LoD 4, IMDF and SVG.

While in IndoorGML the geometry has to be closed such as solids in 3D or polygons in 2D, the geometry of the other standards consists of boundaries. Considering that except SVG and Indoor OSM every other standard is based on well-defined data models and schema they are capable of transmitting sizeable amounts of metadata. However, Indoor OSM uses simple-tag based representation, and its expressive

	Indoor OSM	IndoorGML	IFC	CityGML	IMDF	SVG
2D vs 3D	2D,3D	2D,3D	3D	3D	2D	2D
Modelling Scope	Feature	Space	Feature	Feature	Feature	-
Geometry	Boundary	Closed Geometry	Boundary	Boundary	Boundary	Boundary
Expressive Power	Low	High	High	High	High	Low
Efficiency	High	Low	Low	Low	High	High
Encoding	Tag and XML	XML (GML)	Express and XML	XML(GML)	GeoJSON	XML

Table 1: Comparison between IndoorOSM, IndoorGML,IFC,CityGML,IMDF and SVG [19]

power is relatively low compared to its peers. On the other hand, the data sizes of Indoor OSM and SVG are relatively small and, therefore, easy to encode and decode. In contrast, the complexity of the other standards is reasonably large, resulting in a lower efficiency when using them. Only IndoorGML of the above options explicitly describes the topology and navigation network. CityGML, as mentioned before, does not really provide any advanced visualization support. It must often be converted to SVG or other formats for visualization purposes. It is important to remember that CityGML and IndoorGML can be integrated.

2.3 Conclusion

In the current years, as the demand for indoor maps and indoor navigation systems increases, as shown by the growing interest of prominent companies like Apple, Google and Microsoft, it is natural that several different formats and standards are being developed and constantly evolving, with each inevitably having their weaknesses and strengths.

To develop an indoor navigation application, we must select one that should be the most appropriate for the intended application's requirements. In this document, several standards were analysed, specified their differences, benefits and downsides, and made a small comparison between them, each presenting one of the many possible options available out there.

However, there are a few potential areas of continued development to support this effort, such as better integration of indoor and outdoor [CRSs](#), a more refined concept of [LOD](#) for indoors, and the use of alternative representations for viewings for future solutions, which should be kept in mind since these improvements could appear as new standards rise in popularity.

For this project, creating an extension of IMDF for university campi has been the chosen option for the data format. Considering that Apple has provided guidance on using this format for Airports, Shopping Centers and Transit stations, applying this solution to University Campi should prove to be a pretty similar process.

Additionally, due to its usage of GeoJSON, it also presents an easy option to visualize the data on a map due to the high availability of tools and support for GeoJSON visualizations.

Its major upside is the fact that it is a mobile-friendly and compact solution. Since the whole purpose of

this project is to develop a mobile application, its mobile-friendliness presents a major advantage compared to its competitors.

Finally, its relatively small size, in tandem with the amount of information it can carry, contributes towards a fast information transfer process, minimizing any possible delay during server requests and maximising the amount of useful information to present to the end-user.

Due to the advantages presented, IMDF seemed like the best data format choice for the specific use case of this project.

Floorplan Service Development

At the beginning of development, there was no usable information available to provide the system with the functionality relative to its floorplan representation to the end-user. For this reason, it was necessary to develop and utilize a procedure to gather the existing floorplan data and transform them from their current state into usable information.

For this, a methodology was developed to obtain and convert all of the SVG floorplan files present in the web platform <https://whereis.uminho.pt>, georeference them and finally convert these into an usable GeoJSON format. It is important to remember that for this project, the decision taken in the previous chapter was to utilize a simpler and Android-friendly adaptation of IMDF, which would lead to using GeoJSON files to represent the geometrical information.

In the following sections, the whole process of developing the conversion and georeferencing method will be explained, along with the reasoning behind the choice for the database system; afterwards, the interface definition for the service will be introduced, and finally, the successive changes to the database model will be presented and discussed.

3.1 Process to obtain the geographically referenced GeoJSON files

Before proceeding to a more detailed explanation, it is important to outline the main steps required for this process to be executed and for the desired result to be achieved, these being:

1. Extracting the SVG document from the website <https://whereis.uminho.pt>.
2. Opening the resulting SVG file in Inkscape and ungrouping all elements.
3. Removing all unnecessary elements.
4. Using Inkscape, export the clean file to PDF.
5. Still using Inkscape, export the PDF file to DXF format.

6. Importing the DXF file into QGIS.
7. Creating two shapefile layers, one for ungeoreferenced control points and one for georeferenced control points.
8. Mapping four points in the imported DXF file to their georeferenced versions, adding them to their respective shapefile layers.
9. Exporting both shapefiles as CSV files containing the id of the points and their respective coordinates.
10. Importing the DXF files using the AnotherDXFImporter plugin and mapping the four ungeoreferenced and georeferenced points accordingly.
11. Setting the Coordinate Reference System of the resulting layer to Pseudo-Mercator.
12. Exporting the Layer as a GeoJSON file whilst selecting the WGS 84 Coordinate Reference System.

Some important things to note before going into even further detail are that:

- This whole process is required because initially the files are in SVG, with a significant amount of unnecessary elements that cannot be removed or even georeferenced in QGIS; therefore, they need to be cleaned up in Inkscape and converted to a CAD format that can be georeferenced in QGIS, such as is the case of DXF.
- Unfortunately, the direct conversion between SVG and DXF in Inkscape produces an erroneous outcome. This was circumvented by using PDF as an intermediate format.
- QGIS was the final tool of choice because it was the simplest open-source option that could quickly import a CAD file and, after a small process, export the desired georeferenced GeoJSON result.
- The shapefiles created are used to establish control points that map the SVG coordinates to latitude and longitude pairs of coordinates. These pairs can then be imported into the AnotherDXFImporter plugin in QGIS, which automatizes the remaining steps of the floorplan mapping and georeferencing process.
- Ideally, especially for more complex floorplans, four control points should always be used; otherwise, there is a higher chance for error and inconsistencies in the resulting outcome, but in simpler geometries, three control points are sufficient.

Having said all this, it is time to go into further detail and present the settings and configurations used for each step.

As mentioned previously, all of the SVG files were obtained through the implementation of a webscraper implemented using the Beautiful Soup library of the python language. This webscraper ran through

3.1. PROCESS TO OBTAIN THE GEOGRAPHICALLY REFERENCED GEOJSON FILES

all of the URLs corresponding to the buildings and floors of the Azurém and Gualtar campi, available on the <https://whereis.uminho.pt> website. It extracted and downloaded the SVG files built into them that represented the floorplans in their respective directories. A directory was created for each campus and, inside that directory, another one was also made for each building number which would contain the given building's floorplan SVG files.

After obtaining all of the required SVG files, to finalize this method, it was necessary to, through the usage of Inkscape [14] and QGIS [26], produce a georeferenced GeoJSON file for each SVG image. Firstly, the SVG file is imported into Inkscape using the settings in Figure 5:

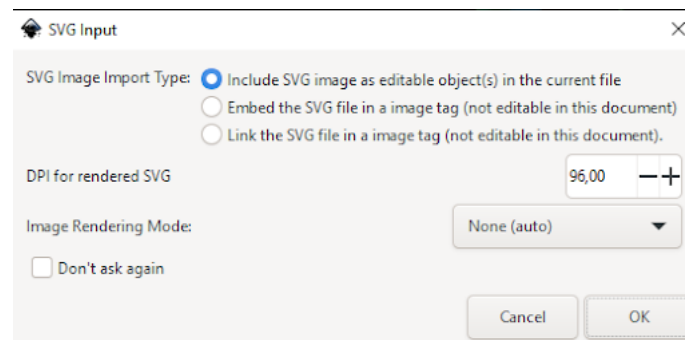


Figure 5: Settings to import SVG file into Inkscape

Following that, the SVG file elements must be ungrouped to remove the extra shapes that do not contain relevant geometrical information. This is possible by selecting the image, right-clicking and selecting the option “ungroup”, as shown in Figure 6:



Figure 6: Ungrouping SVG file in Inkscape

Afterwards, all extra elements should be removed until the production of a result similar to Figure 7.

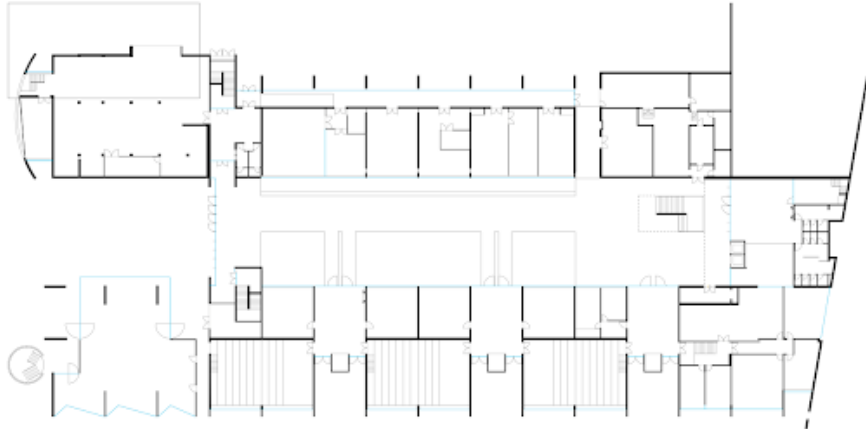


Figure 7: Cleaned up and usable SVG file

After this step, to obtain a correct DXF file representation, it is first needed to use the PDF format as an intermediate format to correct some inaccuracies that happen from the direct conversion from SVG to DXF, but do not exist when PDF is used as an intermediate step. Therefore, the edited SVG should be exported to PDF with the following settings in Figure 8.

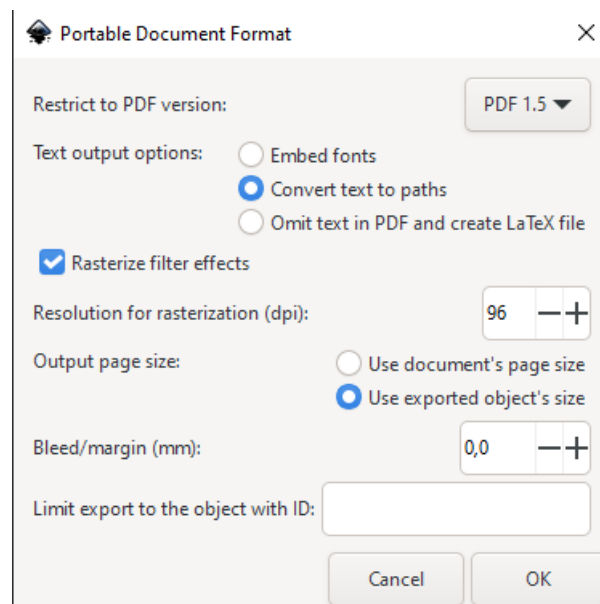


Figure 8: settings to correctly export to PDF

Afterwards, it should be imported once again into Inkscape with the required settings presented in Figure 9.

3.1. PROCESS TO OBTAIN THE GEOGRAPHICALLY REFERENCED GEOJSON FILES

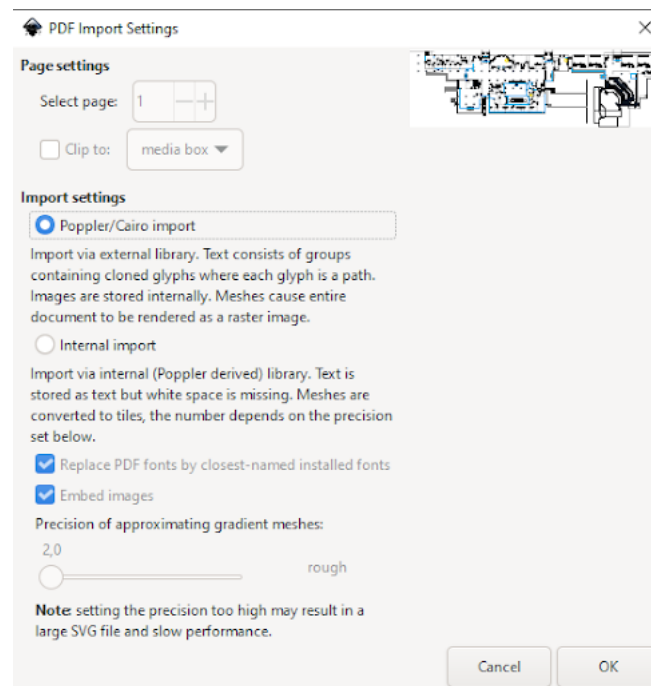


Figure 9: Settings to import a PDF file into Inkscape

Finally, it is time to export the PDF file as a DXF file that can later be used in QGIS to complete the georeferencing process. The settings shown in Figure 10 are used during the exportation process to achieve this.

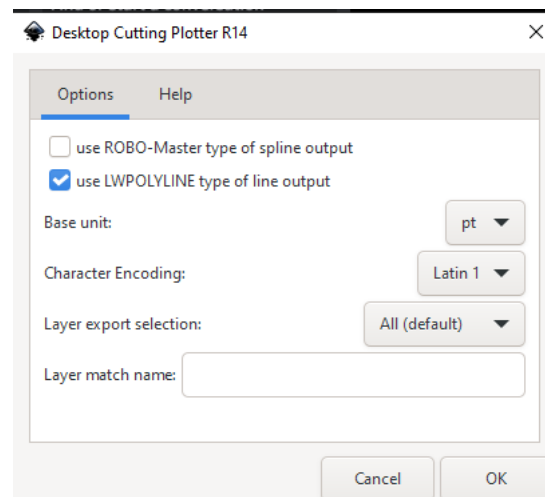


Figure 10: Settings to export a PDF as DXF to use in QGIS

In the next step, it is simply necessary to drag the DXF files, corresponding to the building that is intended to be georeferenced into QGIS, and subsequently two shapefiles should be added to the project with the steps described in Figure 11 and Figure 12, one for the ungeoreferenced points and one for the georeferenced equivalent points:

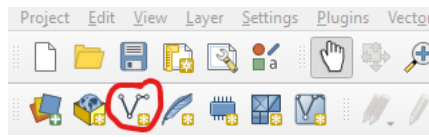


Figure 11: Button to create a shapefile

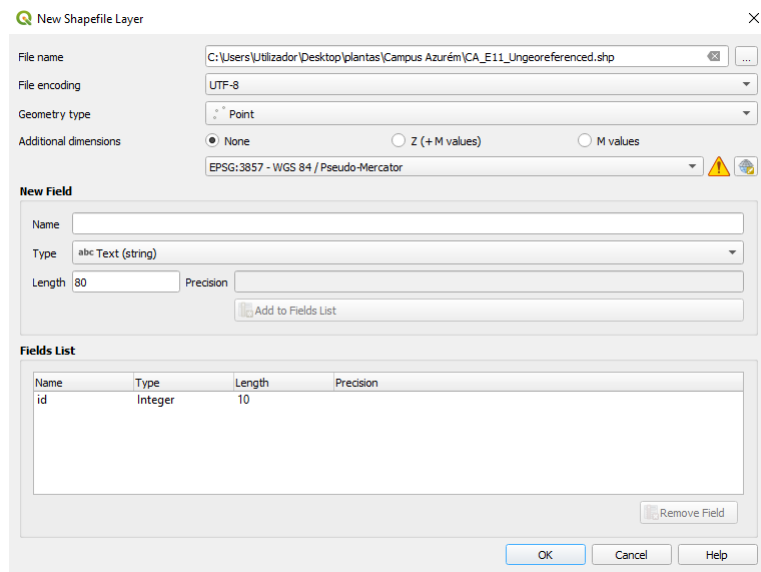


Figure 12: Shapefile creation settings

Finally, four control points per floor should be set (if the desired control points overlap in more than one floorplan, these can be reutilized) in both the imported floor plan and the final georeferenced destination intended for the floorplan to move to. An example of this process is shown in Figure 13 and Figure 14.

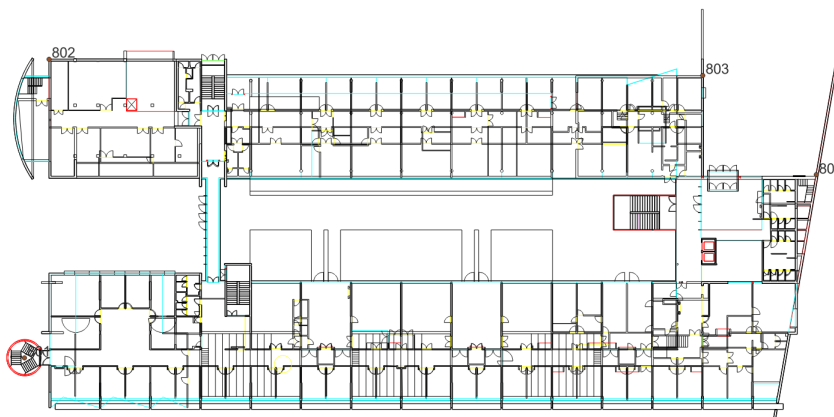


Figure 13: Four example control points in an ungeoreferenced floorplan

3.1. PROCESS TO OBTAIN THE GEOGRAPHICALLY REFERENCED GEOJSON FILES



Figure 14: Four equivalent control points in the desired georeferenced location

Following the definition of the control points, both shapefile layers should be exported as CSV files that will contain the latitude, longitude, and ids of the defined points using the settings visualized in Figure 15:

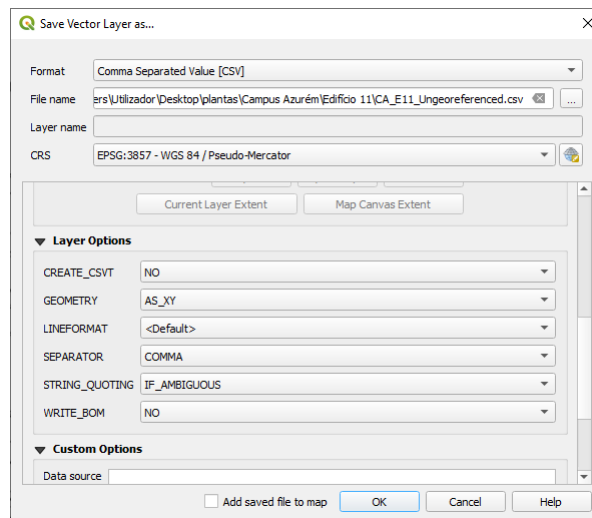


Figure 15: Shapefile exportation settings

X	Y	id
21.9388070727749	76.1867718634117	801
51.855566972097	440.645121316437	802
849.517418116501	420.78323715686	803
987.871808162394	299.467565096914	804

Figure 16: Example of a CSV containing the Ungeoreferenced control point coordinates

The georeferenced and ungeoreferenced CSV file results are similar to the ones shown in Figure 16; the ungeoreferenced one contains the X and Y coordinate pairs correspondent to the initial SVG coordinates, while the georeferenced contains the matching Latitude and Longitude pairs in the real world location.

The QGIS Plugin called AnotherDXFImport must be used using the settings in Figure 17; the table should be filled out with the matching ungeoreferenced and georeferenced points that were previously saved in the exported CSV files as seen in the example mentioned above. On the same row the matching points should be placed. The ungeoref columns for the ungeoreferenced points and the georef for the georeferenced ones.

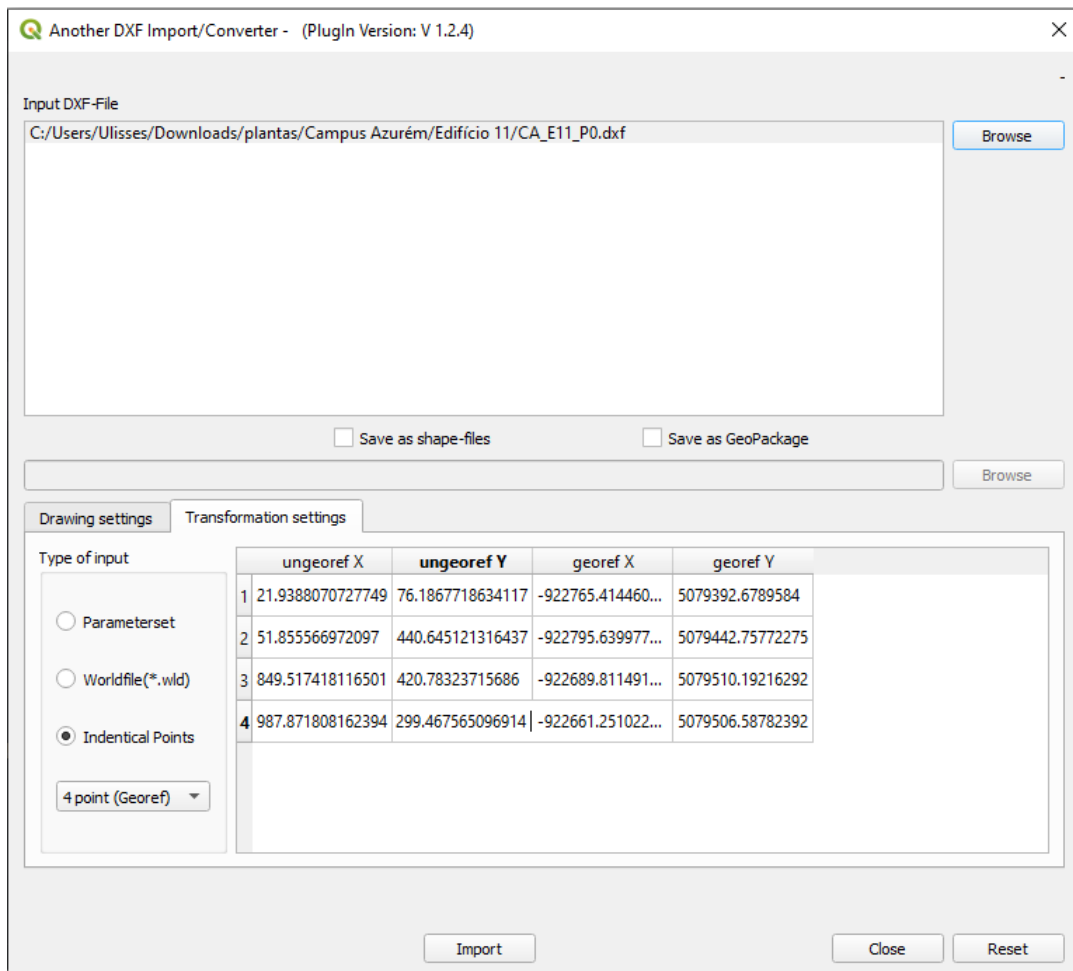


Figure 17: AnotherDXFImport settings

With this exportation done, the DXF file is now georeferenced with minimal error and in the required location, as seen, for example, in Figure 18. It is important, nevertheless, to mention that a vast majority of the buildings have been georeferenced utilizing the Satellite Layer of Google Maps; this may cause a slight misalignment when the floorplan is put over any of the other layer types provided by map visualization tools.

3.1. PROCESS TO OBTAIN THE GEOGRAPHICALLY REFERENCED GEOJSON FILES

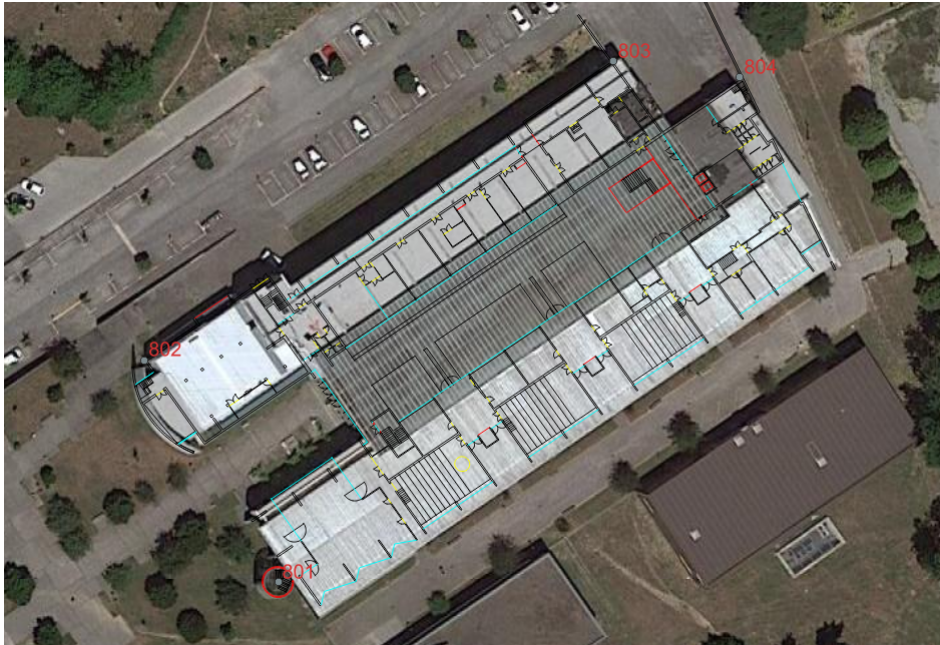


Figure 18: Georeferenced GeoJSON result

To export the layer to GeoJSON, it is first required to make sure that the layer's CRS is defined as the correct one. Throughout this guide, the one used has been Pseudo-Mercator since it provides a more accessible point of view for the manual finding of control points throughout the whole process. As such, the layer CRS should be set to this, as shown in Figure 19.

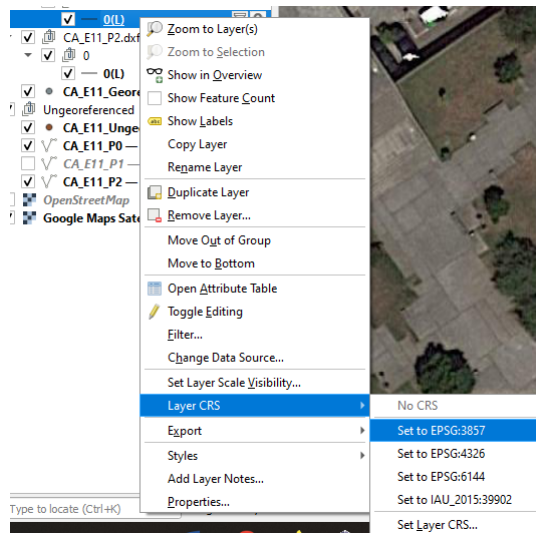


Figure 19: Setting the appropriate CRS on the GeoJSON layers

Next, the export layer option is selected as shown in Figure 20, and set to export in GeoJSON format.

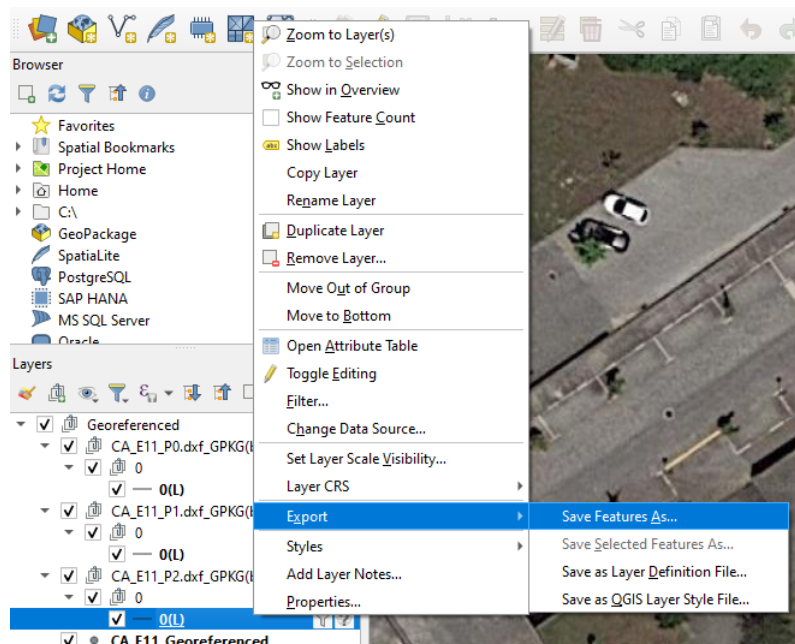


Figure 20: Saving the layers to GeoJSON

In this step it, is important to remember that the exportation must be done in WGS 84 format, since that is the CRS that most map visualization tools, such as leafletJS, use to read and represent GeoJSON. With the settings presented in Figure 21, it is simply required to finally save the file. This final step concludes the format conversion steps of the procedure.

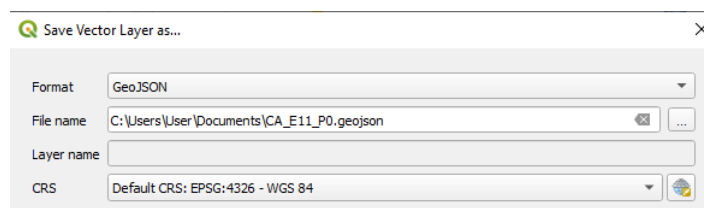


Figure 21: Settings to correctly save as GeoJSON

This process was repeated for every single floor and building of the Azurém and Gualtar campi.

Additionally, after the completion of the georeferencing of both campi, it was later discovered that, during format conversion, QGIS adds, for each feature, a large number of properties that have no use for either the Mobile or Web Applications developed, which ultimately means it will lead to bigger files and therefore longer request loading times without any particular benefit. To fix this detail, a short python script was created, which is possible to see it in Annex I. This script runs through every GeoJSON file and cleans it up, removing the unnecessary properties from each of them.

```
"properties": {
  "fid": 1716,
  "Layer": "0",
  "PaperSpace": null,
  "SubClasses": "AcDbEntity:AcDbPolyline",
  "RawCodeValues": "(2:30 0.0,30 0.0)",
  "Linetype": null,
  "EntityHandle": "7d6",
  "Text": null,
  "ogr_style": "PEN(c:#000000)",
  "font": null,
  "angle": null,
  "size": null,
  "size_u": null,
  "anchor": null,
  "color": "#000000",
  "underline": null,
  "plaintext": null,
  "fcolor": null,
  "flnum": null,
  "bold": null,
  "italic": null
},
```

Figure 22: Removed properties from the GeoJSON files

This led to a drastic reduction in the total size of the floorplan files; more specifically, the combination of all the files went from 126MB to 50MB.

Additionally, it was perceived that each coordinate value contained had fifteen digits of precision, which is unnecessary. With this in mind, using the library `geojson-precision` [16], this number was reduced to six digits of precision, which reduced the size of the combination of all the files to only 40MB.

3.2 Server interface definition

The floorplan web service is fundamental for the healthy functioning of several services from the overall project. Different services required different functionalities and data and therefore needed different endpoints for their specific use cases.

First, an endpoint that would require a specific campus id, building and floor number denominated “getByIds” was developed to satisfy developer needs, as well as simplify some requests in the mobile application, since it presents an easy endpoint to request to verify whether a certain floorplan is available in the system and, if so, to obtain it.

The mobile application developed for the project needed an endpoint that, given a coordinate pair of latitude, longitude along with a floor number, was able to evaluate whether there was a floorplan available for that location or not, and therefore the endpoint “getByLocation” was created.

Afterwards, the web application developed to visualize the radio mapping progress of the project required being able to receive all of the floorplans for a given floor number and campus id at once. With this in mind, the endpoint “getByFloor” was developed, this endpoint aggregates all of the floorplans for a given campus and a given floor number, all in a single feature collection, to then be returned to the web application.

The mentioned mobile and web applications are discussed in later chapters of this document.

In this section the interfaces of the possible requests that can be sent to the developed server are defined, explaining their methods, parameters, possible errors and expected responses.

3.2.1 “getByFloor“ endpoint

This endpoint can answer requests of the type GET to obtain all of the floorplans for a specific floor number. For this, it is required that the user provides it with the request parameters listed in Table 2, respecting their specified rules.

Field	Type	Nullable?	Description	Possible values
floorNumber	Integer	False	Number of the floor for all the floorplans	The integer value of any existing floor be it negative for underground floors or 0 and above for surface levels.
campusId	String	True	Id of the campus for all the floorplans	The string id of any existing campus.

Table 2: Request parameters for the getByFloor GET request

Possible returning values of this request: In case of success this request returns a JSON object that contains only a GeoJSON feature collection that describes all of the features of every floorplan available in the system with the given floor number for the relevant campus (if the campus ID is not specified it will return all of the floorplans in the system).

In case of an error, any of the errors listed in Table 3 is a possible scenario.

Value	Error code	Description
No Floormap Found	404 - Not Found	Error that happens when there are no floormaps associated with the requested floor number.
Invalid Floor Number	400 - Bad Request	Error that happens when the floor number provided is an invalid value.
Invalid Campus Id	400 - Bad request	This error will occur whenever the user provides any value other than “CA” or “CG” for the campus field in the request body
Badly Formatted Request Error	400 - Bad request	Error that happens when the request is not built properly, due to parameters missing.

Table 3: Possible errors for the getByFloor GET request

Example of valid request parameters:

An example request to this endpoint would be something like:

“getByFloor?floorNumber=0&campusId=CA”.

When successful, this request returns a JSON object that describes the GeoJSON feature collection that represents all of the floors numbered 0 available in the system that are located in the Azurém Campus.

3.2.2 “getBylds” endpoint

This endpoint can answer requests of the type GET to obtain a specific floormap. For this, it is required that the user provides it with the request parameter’s listed in Table 4, respecting their specified rules.

Field	Type	Nullable?	Description	Possible values
campusId	String	False	Unique identifier of the campus that the desired building’s floor plan belongs to	“CA” – Meaning Campus Azurém. “CG” – Meaning Campus Gualtar.
buildingNumber	String	False	Identifier of the desired building’s number, must always be double digit to properly specify	“01”, “02”, “03”, ..., “18”, “19”, Along with any other two digit combination that represents an existing building for the desired campus.
floorNumber	Integer	False	Number of the floor for the desired floorplan	The integer value of any existing floor for the desired building, be it negative for underground floors or 0 and above for surface levels.

Table 4: Request parameters for the getBylds GET request

Possible returning values of this request: In case of success, the request returns a JSON object that is the GeoJSON feature collection that describes the floorplan of the requested floor for the desired building and campus.

In case of an error, any of the errors listed in Table 5 is a possible scenario.

Value	Error codes	Description
Invalid Campus Error	400 - Bad request	This error will occur whenever the user provides any value other than “CA” or “CG” for the campus field in the request body
Invalid Building Error	400 - Bad request	This error will occur whenever the user provides the building number of a non existing building or does not provide it in the specified two digit format
Invalid Floor Number Error	400 - Bad request	This error will occur whenever the user provides the floor number for a floor that does not exist
Badly Formatted Request Body Error	400 - Bad Request	This error will occur whenever the request body object is not properly built, be it missing fields or wrongly named ones.

Table 5: Possible errors for the getBylds GET request

Example of valid request parameters:

An example request to this endpoint would be something like this:

“getByFloor?campusId=CA&buildingNumber=11&floorNumber=2”

This request, when successful, should return a JSON object that describes the GeoJSON feature collection that represents the features for floor 2 of the building 11 of the Azurém Campus.

3.2.3 “getByLocation“ endpoint

This endpoint can answer requests of the type GET to obtain the floor plan of a given location. For this, it is required that the user provides it with the following request parameters respecting their specified rules.

Field	Type	Nullable?	Description	Possible Values
Lat	Double	False	The latitude of the position where we wish to get a floor plan for.	Any valid latitude values.
Lng	Double	False	The longitude of the position where we wish to get a floor plan for.	Any valid longitude values.
Floor	Integer	False	The number of the floor we wish to get the floorplan of in the requested location.	Valid floor number for the requested location.

Table 6: Request parameters for the getByLocation GET request

Possible returning values of this request:

In case of success, the request returns an object of the type JSON with the fields:

- “floorplan” containing the GeoJSON feature collection depicting the floorplan of the building found at the given location and floor number.
- “buildingNumber“ containing the ID of the building found.
- “campusId“ containing the ID of the campus the building belongs to.
- “floorNumber“ contains the number of the floor the floorplan belongs to.
- “floorlist“ containing the array with the numbers of the floors that the building also has available.
- “buildingBounds“ containing a Polygon that defines the bounding box of the building found at the given location.

In case of an error, any of the errors listed in Table 7 is a possible scenario.

Value	Error codes	Description
Invalid Latitude Error	400 - Bad request	This error will occur whenever the given value is not a possible Latitude value in the WGS84 format.
Invalid Longitude Error	400 - Bad request	This error will occur whenever the given value is not a possible Longitude value in the WGS84 format.
Invalid Floor Number Error	400 - Bad request	This error will occur whenever the user provides the floor number for a floor that does not exist in the given location.
Not Inside Building Error	404 - Not Found	This error will occur whenever the given location is not within the premises of any of the buildings with available floorplans in the system.
Badly Formatted Request Body Error	400 - Bad Request	This error will occur whenever the request body object is not properly built, be it missing fields or wrongly named ones.

Table 7: Possible errors for the getByLocation GET request

Example of valid request parameters:

A valid request for this endpoint is, for instance, the request:

`"/getByLocation?lat=41.56155&lng=-8.3973202&floorNumber=0"`.

These request parameters being sent to the endpoint should return an object of the type JSON with the field "floorplan" containing the GeoJSON feature collection of the requested building, the field "buildingNumber" containing the ID 07, the field "campusId" containing the ID of the campus the building belongs to, in this case CG, the field "floorlist" containing the array with the numbers of the floors that the requested building also has available and the field "buildingBounds" containing the bounding box of the building found. This example request contains then the information relative to the building 7 of the Gualtar Campus.

Even though some information that is too extensive is omitted, Figure 23 represents a possible response given by this endpoint.

```

{
  "floorplan": {
    "type": "FeatureCollection",
    "features": [
      {
        "type": "Feature",
        "geometry": {
          "type": "LineString",
          "coordinates": [
            [-8.397242, 41.561561],
            [-8.397233, 41.561562],
            [-8.397232, 41.561555],
            [-8.39724, 41.561554],
            [-8.397242, 41.561561]
          ]
        },
        "properties": {}
      }
    ]
  },
  "buildingBounds": [
    [-8.397123366594315, 41.56107157866442],
    [-8.397202491760254, 41.56167767781346],
    [
      [-8.397175669670105, 41.56106355082298],
      [-8.397123366594315, 41.56107157866442]
    ]
  ],
  "buildingNumber": "07",
  "campusId": "CG",
  "floorNumber": 0,
  "floorList": [0, 1, 2, 3, 4]
}

```

Figure 23: Example response of the getByLocation endpoint

3.3 Possible database systems

In this section, four different possibilities that present an option for this server's database system will be discussed along with the pros, cons and viability of every one of them further explained.

Two options are open source, already implemented possibilities while the latter two are technologically more straightforward strategies that present their value as a solution.

Ultimately the final choice will be discussed as well, along with a short explanation of why it stood above the remaining possibilities.

3.3.1 Systems already implemented

Currently, there are a few DaaS (Data-as-a-service) options, such as Azure SQL Database and DocumentDB which support Geometry objects and are compatible with open-source technologies that fit this type of use

case such as GeoServer, but for the scope of this project the solution intended is meant to be deployed on a local server so the comparison made will be instead between PostGIS (a PostgreSQL extension made to host GIS data) and MongoDB (a document based NoSQL database that easily supports GeoJSON objects) as the possible geospatial database solutions.

3.3.1.1 PostGIS

Relational databases are systems extensively used to store and query large structured data sets. They are based on the relational model, which consists of tables representing each set of objects. A variety of different solutions have been proposed to increase their efficiency, such as the case of indexes and partitioning.

PostGIS [25] turns PostgreSQL into a spatial database by adding support for three features: spatial types(e.g.,Point,Polygon), spatial functions(e.g., Distance, Within, Intersects), and spatial indexes (eg. B-trees,R-trees). It follows the Simple Features for SQL specification from the [OGC](#) and is the most used state-of-the-art relational database system for the majority of geospatial use cases.

Additionally, it is compatible with state-of-the-art geographic information systems (GIS), such as, ArcGIS and with map server software such as GeoServer.

Sadly, it possesses no direct support for the FeatureCollection data type, the GeoJSON type in which the bulk of the floorplan data is stored. This would quickly become an inconvenient, and tasks that would otherwise be simple in systems with direct support for this type, such as MongoDB, would quickly become more complex and inefficient.

3.3.1.2 MongoDB

NoSQL databases are often used to manage unstructured data. They are designed to scale horizontally since they have been proposed to manage big data sets using commodity servers. However, they sacrifice the standard ACID (Atomicity, Consistency, Isolation, Durability) properties that relational databases possess to obtain horizontal scalability.

MongoDB [24] is an open-source NoSQL document-oriented database, based on JSON-like documents and one of the few NoSQL systems that supports geospatial data. To perform queries and for storage purposes on geospatial data, MongoDB requires an initial specification of the surface type to be used when running operations on its data. It supports two surface types: Spherical (2dsphere), which involves a calculation based on an Earth-like sphere and Flat (2d), which considers a Euclidean plane with 2d coordinates, which are stored as pairs of longitude and latitude values. It supports a set of standard GeoJSON data types (e.g. Point, LineString, Polygon) and implements a small amount of basic spatial operations (inclusion, intersection, and proximity). Hence, the types of possible queries are in short supply compared to its SQL peers such as PostGIS.

However, it supports horizontal scalability by using the previously mentioned commodity servers, which enables the distributed execution of queries by utilizing the sharding technique. The sharding technique consists of partitioning the input data collection into chunks and storing each chunk in a different server. Upon a query's execution, each server executes the query on its respective slice of data, parallelizing its execution. This partitioning of data depends on the value of the selected sharding attribute; therefore, the choice of this attribute is fundamental to achieving a balanced distribution of the data in the servers. It is important to mention that it is possible to use MongoDB's supported geospatial attributes as sharding attributes. The selection of the best attributes is based on the responses to the queries expected to be done more frequently. MongoDB provides no automatic support to change the sharding attribute after sharding a collection. Thus, a good estimation of the expected workload at design time is important to avoid complications later on.

3.3.1.3 Comparison between PostGIS and MongoDB

Database	Supported Geometry objects	Main supported geometry functions	Supported Spatial indexes	Horizontal scalability
PostGIS	Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, GeometryCollection	PostGIS supports the Open GeoSpatial Consortium (OGC) methods on geometry instances	B-Tree index. R-Tree index, GiST index	No
MongoDB	Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, GeometryCollection	Inclusion, Intersection, Distance/Proximity	2dsphere index, 2d index	Yes (sharding)

Table 8: Comparison between MongoDB and PostGIS [6]

The main difference highlighted by the information displayed in Table 8 is that the relational database PostGIS implements more geospatial functionalities than the NoSQL one, MongoDB. It is also more tightly integrated and supported by the GeoServer software. The plug-in that is used to connect GeoServer with MongoDB is in the unsupported branch of the current version of GeoTools. The interoperability with the

GeoServer, or with similar software, is indispensable for large-scale geospatial applications that need to visualise maps and geolocated objects with high quantities of data, especially when mobile devices are involved. Hence, in terms of functionalities, a spatial relational database is preferable (it is a more mature technology) because it allows the performance of complex geospatial queries and is well integrated with geographic information systems (GIS) and with map server software such as GeoServer. On the other hand, since this particular use case makes such a strong use of the Feature Collection type that is not directly supported by spatial relational databases, NoSQL databases still present a strong competitor .

So far in this section, the discussion has assumed the existence of complex geospatial queries or humongous amounts of data being transferred between systems. But after having converted the data to GeoJSON as explained in the previous section, it is worth noting that most of the files produced do not contain more than a meagre thousand features per file and as seen in articles such as [17] or [2] the performance in obtaining and querying geospatial data at such a small scale even in a worst case scenario is hardly noticeable. Therefore, this led to the question “Is there any need to implement a more technologically complex solution, or are there simpler and equally viable solutions?”

This question leads to the next subsection of this chapter, where two more technologically simple solutions are presented and discussed.

3.3.2 File system based floorplan storage solutions

The options involving databases with geospatial capabilities that are depicted above are only some of the alternatives. It is still possible to manage the floor plan data using either a file system hierarchy solution or a relational database model that can be used to store the metadata and paths to the files in the file system. But like everything else, there are advantages and disadvantages to using each of these methodologies.

3.3.2.1 File System Hierarchy

The primary benefit of using a File System Hierarchy solution is its inherent simplicity. With a well-structured file system hierarchy, obtaining a response for a query that desires a GeoJSON file becomes a trivial process by specifying its Campus, Building and Floor number because it only requires accessing the file system on the necessary directory and returning the requested file. However, suppose the query requires a file specified by the user’s position and floor number. In that case, this is not a viable solution since it would imply opening and then reading every single GeoJSON file until one matches the given location, which, with the expected delays of all the file system reads, would quickly prove an inefficient and unviable solution.

3.3.2.2 Database tables with metadata

The idea behind this solution is to mitigate the problems of the File System Hierarchy solution by storing the metadata of both the buildings and floors in two relational database tables. With this implementation

it is possible to identify which building a given user is located in through their location since, with each row of the buildings table containing a polygon that defines the bounding box of the specified building, when latitude and longitude values are given to the API, applying a “within” function implemented on the server on every row, it is possible to determine which is the building that the user is in the premises of.

The upsides of this solution are that it is a comparatively straightforward solution to implement, and the organised storage of the necessary meta-data of the buildings and floors also can prove helpful to meet other ends. The downsides include having to define the within function manually, which is already defined and available in the geospatial database solutions, as well as not being as efficient speed wise when put against the aforementioned solutions since it is not feasible to think it will be as well optimised as the already well studied implementations in the state of the art open source options or that it will not be slower due to the still required usage of the file system.

3.3.3 Conclusion and chosen solution

Ultimately the solution selected was to store the floorplan metadata in tables on a PostgreSQL database, this decision provided reasonable querying times and the performance impact of using the file system was mitigated since it is only used once per query in time-sensitive endpoints.

While the file system hierarchy proved unviable, both MongoDB and PostGIS presented good cases. Still it was decided that this system could make good use of a technologically more straightforward solution and did not require the added complexity of these two. In the future, if the system were to take bigger proportions this decision would need to be revised, but with the current predicted size of the system, this is sufficient.

In a later section, the exact layout of the database tables will be presented, and the thought process and reasoning beyond its’ design will also be introduced.

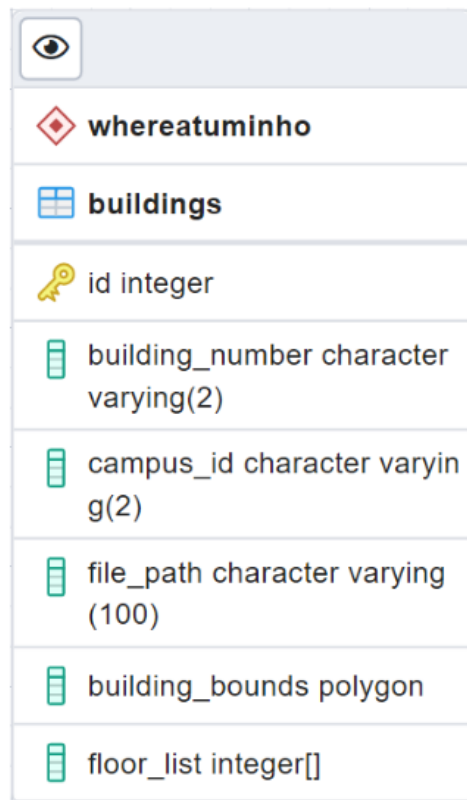
3.4 Server database model definition and description

In this section of the present document, the several iterations of the database model will be discussed. Throughout its subsections, all the versions, from version 1 until the final version, will be presented and discussed along with the changes between them and their previous iteration, alongside the reason for said changes.

3.4.1 Version 1

In the first version of the database system, only one table that was deemed as necessary. This table stored the metadata related to the buildings of both the Gualtar and Azurém campi and the file path to their respective floorplan data files.

A strict and specific file system hierarchy was needed in the first few iteration of this storage system. Every building had its folder where the data of the floorplans of their respective floors was contained, and these files followed a strict naming convention containing the Id of the campus they belonged to, the building number and the floor number. An example would be a file named “CA_E01_P0” that would indicate the floorplan file of Floor 0 of Building 01 of the Azurém campus, while this was contained in the E01 folder that was contained inside of the CA folder. This was seen as a solution that was too restricting and therefore was discarded.



whereatuminho	
id	integer
building_number	character varying(2)
campus_id	character varying(2)
file_path	character varying(100)
building_bounds	polygon
floor_list	integer[]

Figure 24: Version 1 of the floorplan’s server database model

In the table represented in Figure 24 it was stored the building number, the campus id, the file path to the folder of the building, the boundaries of the building represented as a polygon and an array of integers that represented the numbers of the floors said building contained. Through this structure the path to the desired file was then internally built in the server, following the strict naming conventions mentioned previously.

With these inefficiencies having been found, there was the need to think of a better and more sophisticated strategy. Thus the database model progressed onto version 2.

3.4.2 Version 2

For version 2, the logic was divided into two separate tables, one containing the building logic, and the other containing the logic for each floor. The purpose of this change was to erase the file system hierarchy and naming dependencies discussed in the last subsection. With this structure, the floorplan files can all be contained in the same folder without specific naming conventions.

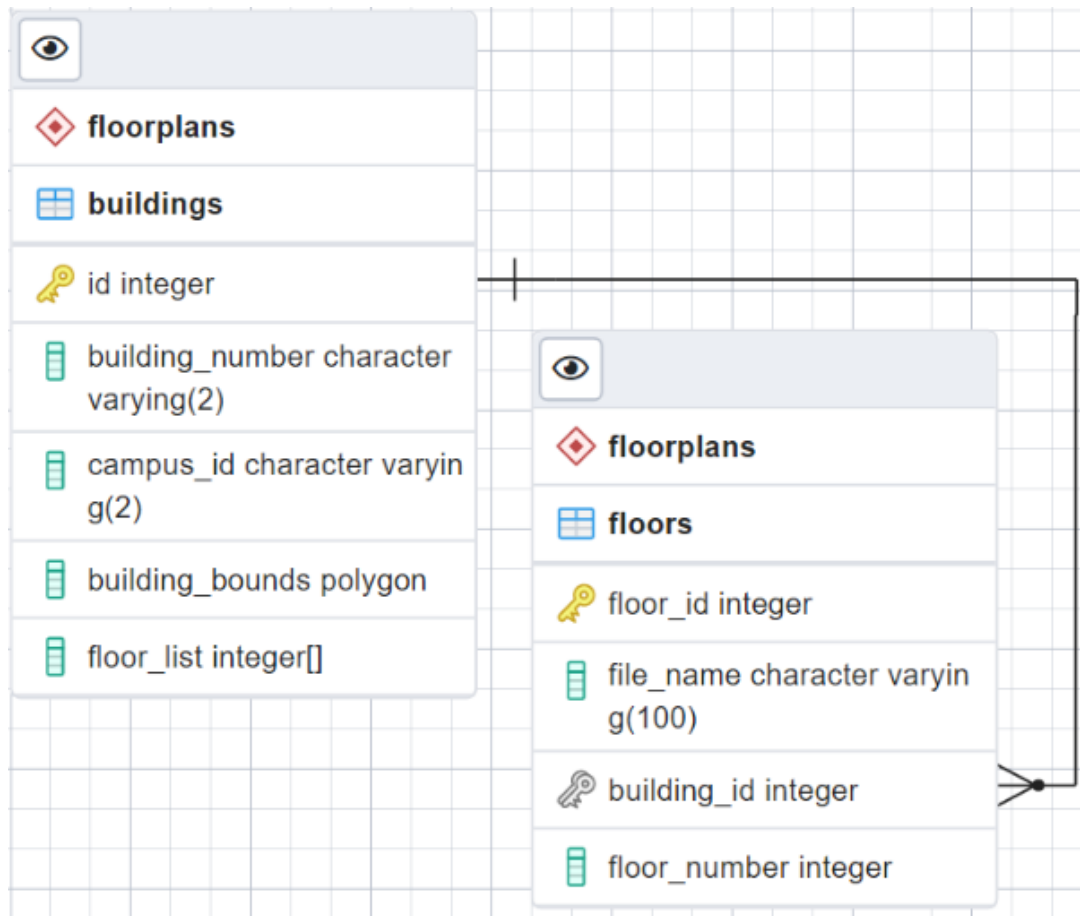


Figure 25: Version 2 of the floorplan's server database model

In this new version, the tables in Figure 25 are then structured so that the buildings table only stores the array of valid floor numbers, the boundaries of the building, the building number and the id of the campus it belongs to. Whilst the floors table contains the name of the floorplan file, the floor number and the foreign key of the building it belongs to. Nevertheless, this version still presented one last flaw: the floor numbers array contained in the buildings table was prone to error.

This means, it was easily possible for there to be inconsistencies between this array and the floor's table. In scenarios where a floor number is added to the array but its' floor is not added to the floor's table, or vice versa. Therefore this last flaw leads the database model to its' third and final version.

3.4.3 Version 3

The third and final version of the floorplan's server database model, as seen in Figure 26, no longer contains the floor numbers array on the buildings table. Through a simple join operation, it is possible to have all of the required information of a building associated with its floors.

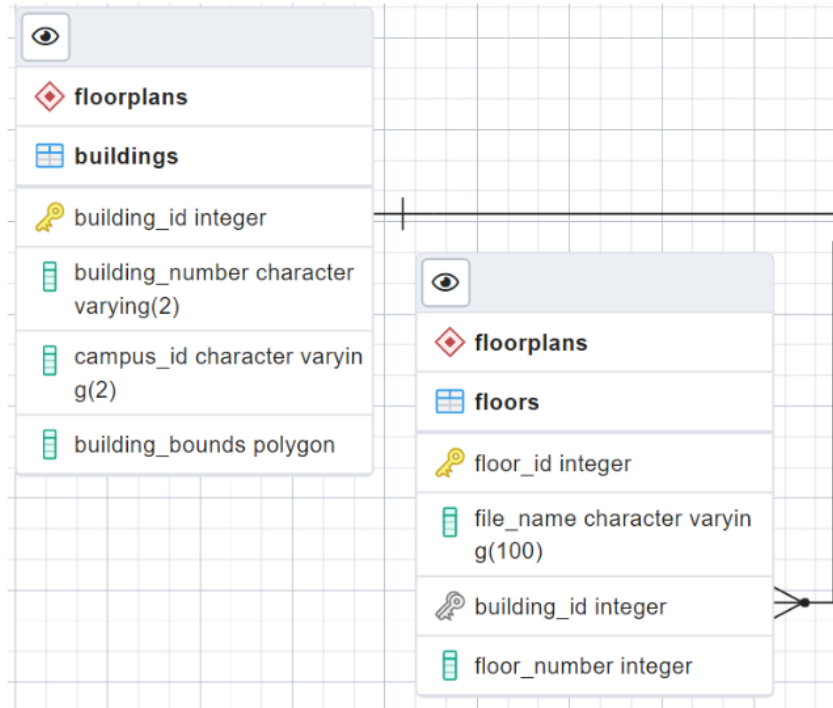


Figure 26: Version 3 of the floorplan's server database model

In this final version, the buildings table stores the building number, the id of the campus it belongs to, and the boundaries of said building, while the floors table contains the file name of the floorplan of the said floor, the foreign key that represents the id of the building it belongs to and the number of the floor it represents in said building.

3.5 Technologies used and file structure

For this server, the technologies of choice were the *Spring Boot* Framework, along with a *PostgreSQL* database system while the *Object Relational Mapper (ORM)* used was *Hibernate*.

Other technologies were seen as possibilities, such as *Node.js* and many different database systems, SQL and NoSQL. Still, for the server framework, the choice was *Spring Boot*, since other project components had already been built using this technology. The other options did not present any major advantage that would justify introducing a new technology to the Tech stack.

As for the database system, the reasoning behind why PostgreSQL was picked has been discussed in an earlier section of this chapter.

To deploy this server, a docker container produced through a docker-compose file was used for the database system and for the server itself.

In Figure 27, it is possible to see the overall server architecture.

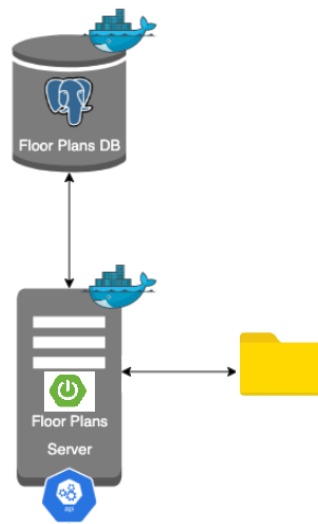


Figure 27: Floorplan service architecture

The file system of the server's container is used to store the GeoJSON files that contain the floorplans that are returned to the user. These files include a singular floorplan each and are all stored in the same directory. In Figure 28, it is possible to see a snippet of one of these files; they are made of a GeoJSON file that represents a feature collection which contains a plethora of LineStrings that represent the details of a floorplan.

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "LineString",
        "coordinates": [
          [-8.288543, 41.453463],
          [-8.288544, 41.453463],
          [-8.288594, 41.453515],
          [-8.288596, 41.453514],
          [-8.288546, 41.453462],
          [-8.28857, 41.453449],
          [-8.288569, 41.453448],
          [-8.288542, 41.453462],
          [-8.288543, 41.453463]
        ]
      },
      "properties": {}
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "LineString",
        "coordinates": [
          [-8.288629, 41.453496],
          [-8.288632, 41.453495],
```

Figure 28: Snippet of a GeoJSON file

3.6 Implemented Solution Validation

In this section, the implemented solution will be validated through the use of load tests. These load tests were made using the Apache JMeter tool. With this tool, thousands of requests were sent to the server within a short period of time. Results such as transactions per second, percentage of success and average speed of response, among others, have been collected and will therefore be analysed. The strategy used to test all endpoints was the same, a certain amount of users were created, each sending 10 requests. These users had a ramp-up time of 100 seconds, meaning that, in this case, a new user started his workload every second.

It is important to note that every time this load proved too small to test the system effectively it was adjusted to find what is the respective endpoint's limit.

Another thing to keep in mind is that different endpoints will sometimes have significantly different le... (14 KB restante(s))

Mobile application

The center point of the project, where every developed system converges, is a mobile application called “Where@UM”. As mentioned in the introduction of this dissertation, this application is meant to navigate and guide its users through complex indoor locations. Several web services, such as floorplans, data collection, and navigation services, have been developed to support this functionality. Of the services that support the mobile application, the ones that are the focus of this dissertation can be seen in Figure 29, surrounded by red rectangles.

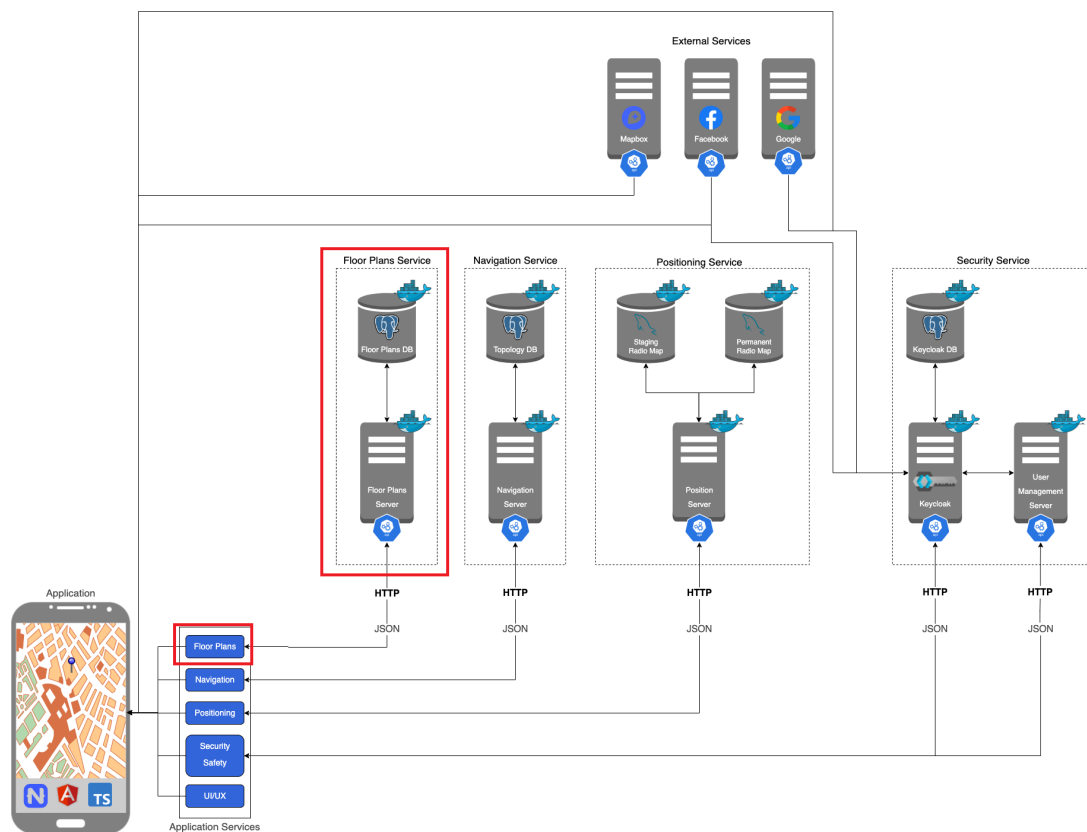


Figure 29: System architecture that supports the mobile application

The conversion methodology and floorplans web service developed so far have been created to support this application's functionality. Having this in mind, discussing the functionalities supported by this development, the components used to achieve this goal, and the strategies implemented to succeed in acquiring a well functioning system is fundamental.

4.1 Objective of the integration of the floorplan service

The floorplan service was devised to introduce a fundamental functionality into the mobile application: the representation of indoor locations. Up to this point, the application differed very little from any other mapping solution currently out on the market, since it only had the capability of representing outdoor locations provided by third-party APIs. Integrating the floorplans service makes it possible to achieve the representation of indoor blueprint layers on top of the current application. This allows the user to better perceive where they are located while traversing through a mapped indoor location since the user will be able to see their location on the map on top of the floormap.

This perception was useful for two use cases inside the mobile application "Where@UM"; one is for the user to perceive through their location where they are on the campi, indoor or outdoor. This can be visualized in Figure 30.

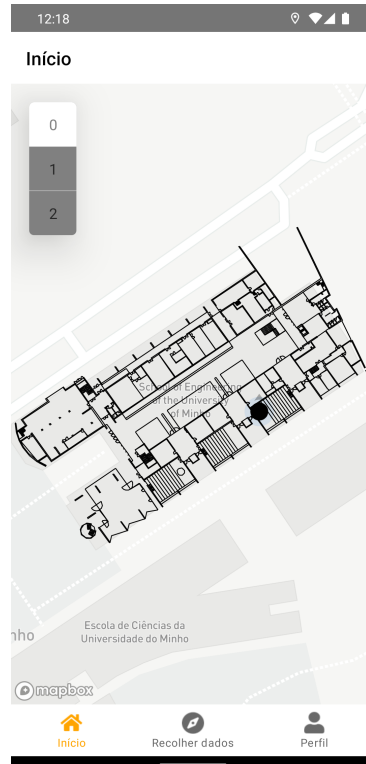


Figure 30: Mobile application home page

As shown in Figure 30 on this screen, a user can see where they are both on the map, and with

the floorplan integrated into its correct geographic location. They can also perceive where exactly on the floorplan their location is.

The second use case that required the floorplans service is, in reality, made in conjunction with another dissertation, being part of one of the crowdsourcing methodologies. This crowdsourcing methodology involves the user volunteering their position by indicating it with a marker on the map. To ease this process, since there will be some location errors due to GPS not working as well indoors, the floorplan service will aid the user by showing them the floorplan of the building they are located in. An example of this process can be seen in Figure 31

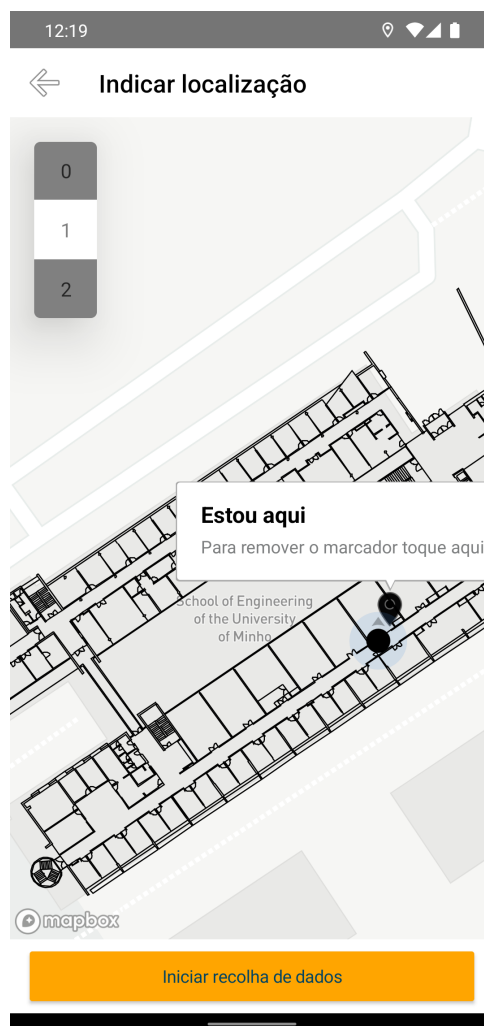


Figure 31: Pick Location method of crowdsourcing

4.2 Components used

To achieve the use cases presented in the previous section, it was necessary to select the right tools and customize them appropriately so they are used in the best way possible to reach this goal.

The choice of components needed for this specific part of the mobile application presented a challenge. Due to other parts of the application requiring the use of NativeScript for their well-functioning, it was necessary to constrain this project to the tools and plugins available for such a framework. Sadly the map visualization tools for NativeScript are both scarce and still quite early in their development stages, thus having incomplete implementations.

The first attempt at a solution was to use the Google Maps Native Script plugin. Unfortunately, this plugin does not possess the ability to show GeoJSON layers on it, which is a must for the success of this representation. The application intends to show the user's indoor location on a map by seamlessly integrating GeoJSON layers that represent the floorplan of the building the user is located in. Therefore, since this is not a possibility with the Google Maps plugin, the plan B was to use the Mapbox plugin implementation, and complement it with some features to better approach it to the requirements of this application. While not up to par with its other versions available in different frameworks, the NativeScript Mapbox plugin still can represent and manipulate GeoJSON and other such basic functionalities that make it sufficient for the use case it has in this application.

Additionally, since map visualization was needed in multiple places of the application, it was deemed necessary to develop a single component that would fit the map visualization needs of all of the application and not several more specialized components; this way, it is possible to keep code duplication to a minimum since a lot of these features still share some traits and functionalities therefore making this the more reasonable decision.

For this purpose, a custom component was developed based on the Mapbox component, satisfying all of the needs relating to map visualization for the entire application. Not all the said needs will be explained in the present document since these clarifications belong in different dissertations better fit for the subject, only the ones relating to the floorplan presentation and user location will.

To ease the transition between floors, a floor picker was added; this part of the interface shows the user which floors are available for the building they are located in, the possible floors to display are stored in the component as an array that is received and updated every time a request to the floorplan service is made. Additionally a subscription and polygon are kept in the component as parts of the strategy that keeps the displayed floorplan updated. This strategy is discussed and explained further in the next section.

4.3 Strategy

In implementing this component, a strategy was created to periodically update the state of the floorplan being presented to the user. A subscription object is created and stored on the component's initialisation. This subscription runs every five seconds and is connected to the floorplan service. It causes the mobile application to, periodically, check if there have been any significant changes in the user's position. For this purpose a polygon is kept and saved in the component's data. This polygon represents the boundaries of the building the user is currently located in, and is provided by the web service upon receiving the data of

the floorplan, being used every time the subscription executes to verify whether the user is still inside the same building or not. This verification is done through the use of a *within* function that calculates whether the user is inside the boundaries of the same building, on the boundary, or outside of it. The arguments for the *within* method are the user's location and the building's boundaries stored inside of the developed component for the map.

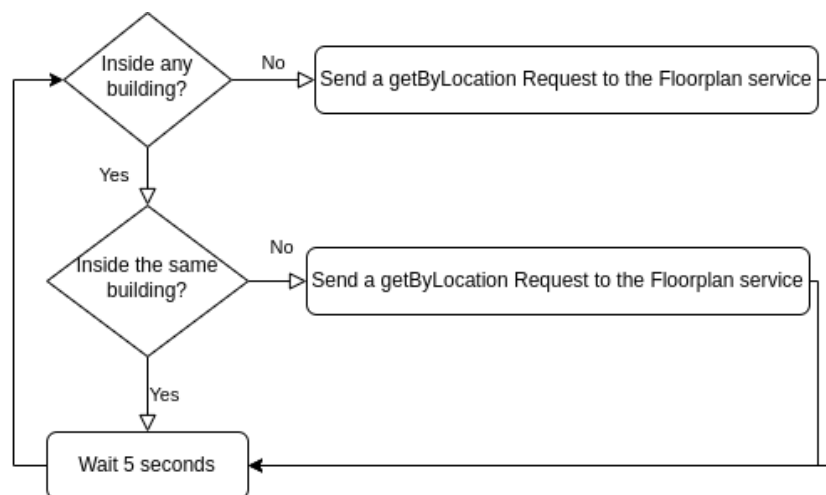


Figure 32: Representative Diagram of the Strategy to keep the floorplan being displayed updated

As seen in the diagram in Figure 32, if the user is detected to be inside the same building or on its boundary, nothing happens, and the application keeps running normally, tracking the user's position. If the user has moved outside of the boundaries, a request to the web service is sent with the user's current position and floor number to check whether they are inside a new building or simply in an outdoor location. When the building's boundaries variable is set to null, the user is not in an indoor location mapped in the system.

This strategy was set in place to minimize the number of requests to the server and place some of the processing load on the mobile application. Otherwise, the floorplan service would easily be overloaded by the constant, and sometimes unnecessary, stream of requests from all the users. It is quite a simple strategy, but its simplicity does not reduce its effectiveness.

Alongside this strategy an array is kept as a cache of sorts, its purpose is to further minimize the number of requests to the service as well as help the application cope with possible connection problems. This array has a size of three and every time a new floorplan is loaded it is added to it, and in case of it being full the oldest floorplan is removed from the system.

This cache system helps in the case where a user selects a new floor number that they wish to visualize; before sending a request to the service, the application verifies whether this floorplan is already present in cache before sending a request to the service. This helps reduce the number of requests to the service and in cases where the connection is lost, as long as the user had already loaded the desired floorplan they are not dependent on the service to be able to visualize the desired floorplan.

Web application for collected data

In this chapter, the development of a web application created in the context of this system will be presented. Throughout the following sections, subjects such as the reason for developing this web application for this project, the choices made during its creation, the technologies used and the interface definition and description of the backend service that had to be implemented for the system's well functioning will be approached and discussed.

5.1 Objectives of the web application development

The purpose of this web application is to provide a platform where users and administrators of the system may be able to check and perceive which locations are mapped by the crowdsourcing process and which require increased attention. Considering crowdsourcing is an essential foundation for this project's success, it is important to be able to track its progress in some way. This web application seeks to make simple and accessible the visualization of this information.

Not only does it help track the crowdsourcing progress, it also facilitates the future implementation of new features such as gamification mechanisms in the mobile application. For instance, if points are assigned to each contribution towards the mapping, and contributions in areas with fewer samples reward an individual with more points, it will be helpful for the user to be able to see which parts of the campi have more samples and which would need more and therefore simplify the process of knowing which areas should be mapped for a maximum amount of points, benefiting both the system and the user.

For administrative purposes, having such a view also simplifies the detection of any outlier samples that may have been introduced into the system by ill-willed users. If samples show outside the campi or outside any indoor spaces they might indicate malicious attempts at sabotaging the system. This way, it is possible to keep a cleaner, safer and more accurate system implementation running.

5.2 Requirements of the web application

To accomplish its purpose, this application must include the floorplans of the buildings, so the user can better perceive which rooms and sections of the building are mapped, as well as give the option of which floor they wish to see the data for. This way, the data is easier to understand since it will allow the user to check individual floors and locations rather than just a cluster of information that includes the entirety of a building and would, therefore, not make much sense since it would make it impossible to show the accurate floorplan.

Additionally, through the filtering of dates and the smart processing of the crowd sourced data, the application needs to be able to show the user the fingerprinting samples pertaining only to set time frames with the intent of adding some flexibility to the data visualization and making possible the verifying of which areas have been recently sampled and which have the oldest fingerprints and may need more user input to stay relevant and updated. Example time frames would be the last year, last six months, last three months, and so on.

In future implementations, there is also the possibility of implementing authentication, so instead of just a view of the contributions of every user, each individual may be able to view only their own contributions; this particular feature would be a necessity if gamification mechanisms were to be implemented.

5.3 Ways of representing the data

For representing hundreds or even thousands of samples, a simple marker representation would quickly prove to be an inefficient system, simply due to the fact that such a huge amount of markers being loaded every time new information was needed would cause the overall application to be slow to load and use.

With this fact in mind, it was necessary to research and seek other options to solve this problem and display the data in a way that would prove more useful and readable to the end user. Two main options were found that would prove useful and the most appropriate for this kind of application.

5.3.1 Marker Clustering

The first option, and the most similar to the initial implementation, is known as Marker Clustering. Essentially, this consists of, within a certain radius, clustering all of the nearby markers into a single cluster that shows the user the absolute number of markers it aggregates. The difference in efficiency between this option and several individual markers is huge, since it requires a significantly lower amount of markers to be loaded at once. This amount varies with zoom level; the more the user zooms in, the more the markers separate from each other and become new clusters, as seen in figures 33 and 34.



Figure 33: Example of a more clustered Marker Clustering representation of the crowdsourcing samples

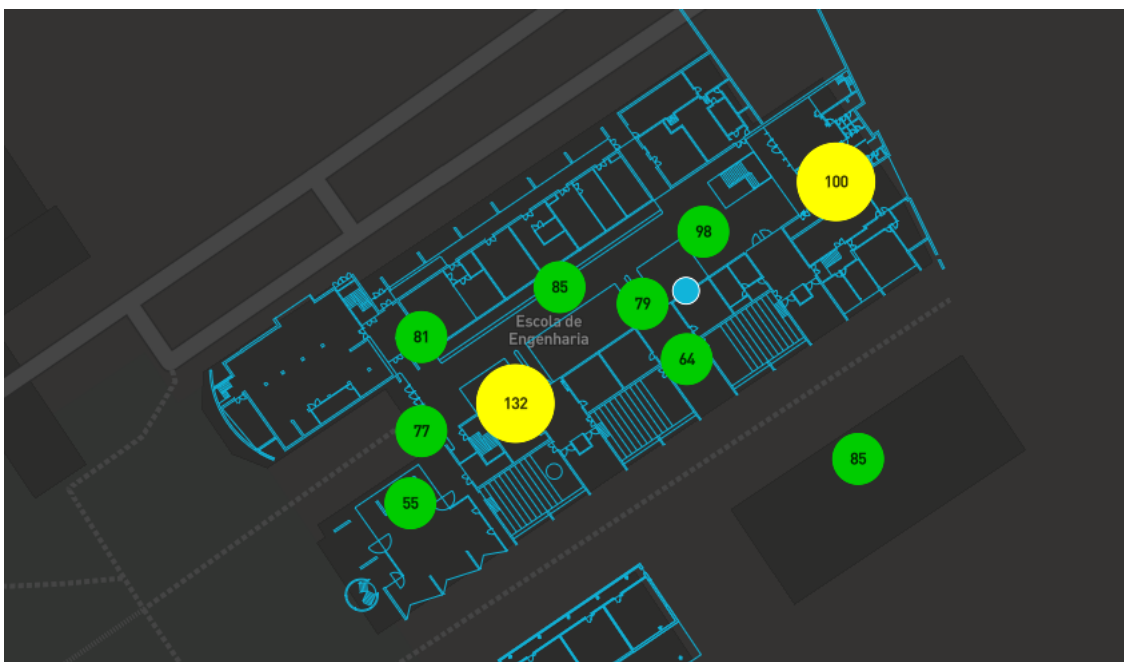


Figure 34: Example of a less clustered Marker Clustering representation of the crowdsourcing samples

This solution allows the user to perceive the exact amount of samples located in a general area and, the more the user increases the zoom, these start spreading out to their exact locations. So it is a great option since it allows the ability to choose which level of detail an individual wishes to see.

5.3.2 Heatmap

The second option is to use a heatmap layer on the application's map; this layer should hold equal weights for each distinct crowdsourcing contribution. This method makes it impossible to see the exact number of contributions within a given location; nevertheless, it is still visible how condensed or dispersed they are among the space visible in the map due to the variable intensity of colors within the heatmap. These colors also change with any zoom the user does to adjust themselves to the new visible area.

The main disadvantage of this option, versus the marker clustering, is that it does not represent the absolute value of how many samples are contained within a given area. The map visualization tools do not possess a scale to represent how many samples a given color represents because these change with a plethora of different values and not just the number of samples. On the other hand, it is an easier and more immediate way of perceiving where the highest concentration of samples is due to the nature of this representation.

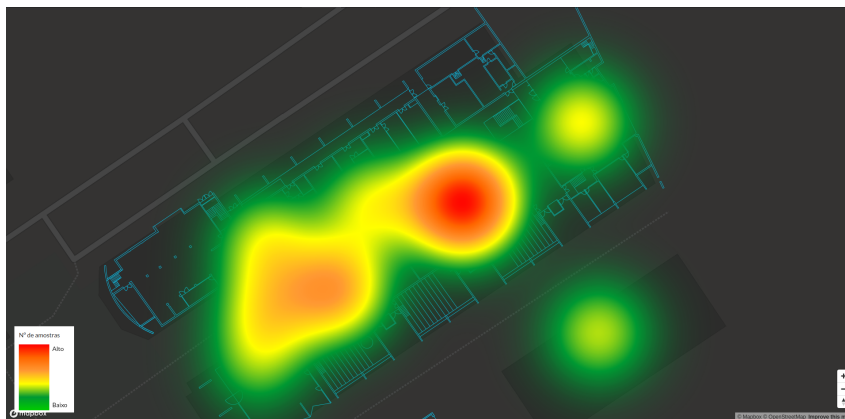


Figure 35: Example of an heatmap representation of the crowdsourcing samples

5.3.3 Conclusions

At the end of the day both options present their ups and downs. While the Marker Cluster allows the visualization of the absolute number of samples on the map, and is able to give the user a notion of where they are mostly concentrated, the heatmap on the other hand will not allow the user to know exact values but will provide, in a more easily readable fashion, the information of where the samples are concentrated.

Ultimately, the choice made was to simply implement both options and allow the user to pick which they want to visualize. This allows the application to have some flexibility to better fit the needs of each individual user at no cost.

If an individual prefers a simple more immediate and without numbers associated solution they have the heatmap available, if numbers are required and a higher level of detail, the Marker Cluster option is available. Both are relatively simple to implement, therefore there was no need to pick only one, the

research done previously sought only to validate if either, or even both, were valid and viable options for this use case.

5.4 Technologies used

For the development of this web application React was the technology of choice for the frontend and Spring Boot for the backend.

The conversion server in Spring Boot was implemented to process the data available in the Data Collection Service database, convert it into a GeoJSON feature collection of points and structure it in such a way that it becomes more useful and easily processable for the web application frontend to consume. The definition and description of the interfaces made available by this webservice will be discussed in the next section, while this web application's architecture can be seen in detail in Figure 36.

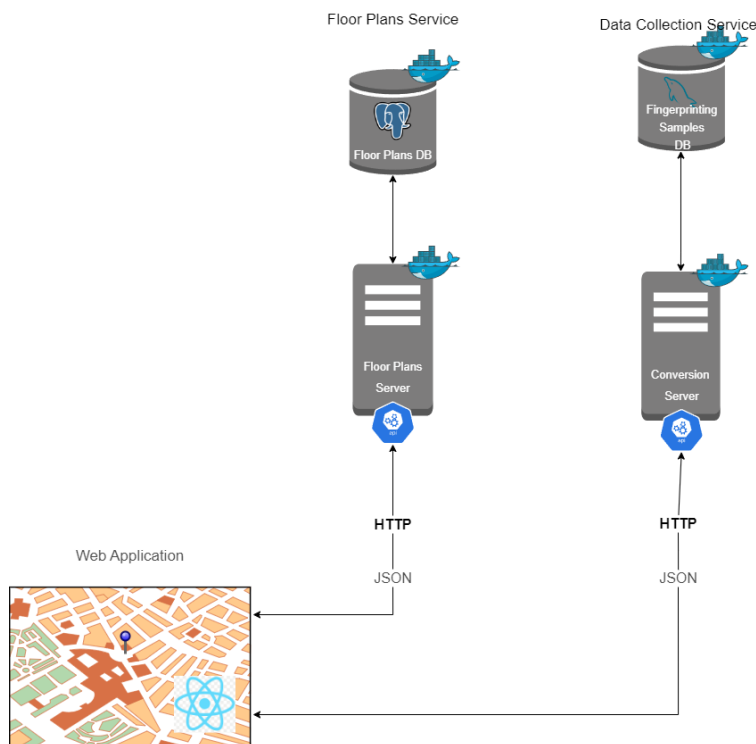


Figure 36: Web Application Architecture

The map visualization library of choice was react-mapbox-gl; this constitutes an implementation that uses the official mapbox-gl plugin and wraps it with extra functionalities such as is the case of the heatmap and marker clustering layers. react-mapbox-gl was developed by vis.gl[12], a part of the Urban Computing Foundation[11], to implement some extra functionalities that the base implementation did not possess. This whole development is available as open source and was made with the support of the official Mapbox team as seen on the library's official GitHub[27].

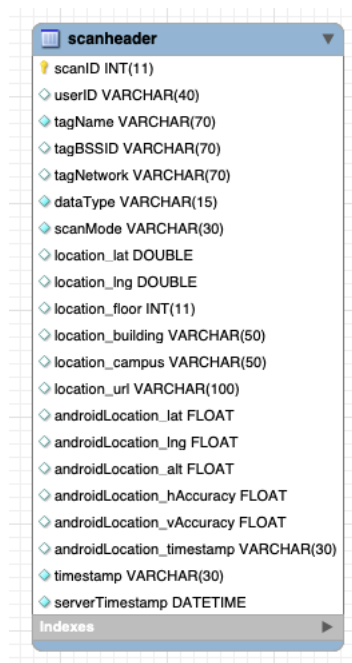
Other libraries were added to improve the user experience, such as React router for the routing of the application and Semantic UI React for the styling of the application. Semantic UI React is a library consisting of a plethora of pre-defined and already styled visually appealing and lightweight components; this leads to a huge time save which allowed for a quicker production of a prototype of the application.

5.5 Interface definition and description

This section will detail the endpoints provided by the web application's backend and their functionalities and possible errors. It is split into two subsections, one that details the behaviour and possible errors of an endpoint which obtains all samples in the system, and a second subsection that details the endpoints that handle requests that only return data within a given time frame.

It is necessary to remind that this server was introduced to obtain the data from the database that contains the crowdsourcing data obtained by a colleague from the same project, and to then transform and reproduce it into data that will be useful for the viewing purposes of the web application.

The schema of the database table that is useful for this particular purpose is depicted in Figure 37. The information for the endpoints of this service is all obtained from it, processed by the server and transformed from this original format into a GeoJSON feature collection containing points that each define a fingerprinting sample.



Field Name	Data Type
scanID	INT(11)
userID	VARCHAR(40)
tagName	VARCHAR(70)
tagBSSID	VARCHAR(70)
tagNetwork	VARCHAR(70)
dataType	VARCHAR(15)
scanMode	VARCHAR(30)
location_lat	DOUBLE
location_lng	DOUBLE
location_floor	INT(11)
location_building	VARCHAR(50)
location_campus	VARCHAR(50)
location_url	VARCHAR(100)
androidLocation_lat	FLOAT
androidLocation_lng	FLOAT
androidLocation_alt	FLOAT
androidLocation_hAccuracy	FLOAT
androidLocation_vAccuracy	FLOAT
androidLocation_timestamp	VARCHAR(30)
timestamp	VARCHAR(30)
serverTimestamp	DATETIME

Figure 37: Table containing the crowdsourced samples relevant to the Web Application

Table 9 contains the request parameters for all of the endpoints of this API. These are represented by a single table due to the fact that they all share the same information needs to be able to provide the

application with a response.

Field	Type	Nullable?	Description	Possible values
floorNumber	Integer	False	Number of the floor for all the fingerprinting samples	The integer value of any existing floor be it negative for underground floors or 0 and above for surface levels.
campusId	String	True	The id of the campus for all the fingerprinting samples	CA or CG for the respective campus.

Table 9: Request parameters for all Web Application GET requests

5.5.1 All samples endpoint

This endpoint's purpose is to, as the name says, provide all of the samples available in the system respective to the given floor number, its usage is to visualize the data alongside the floorplans in the system through Marker Clustering or an Heatmap. It can respond only to requests of the type GET that contain the request parameter presented at the beginning of this section.

An example request to this endpoint would be "all?floorNumber=0&campusId=CA", in case of success, this request should return a JSON object that contains a feature collection depicting all of the floorplans identified by a floor number of zero for the Azurém campus. Represented in Figure 38 is a portion of the response to this endpoint. All of the other endpoints presented in this chapter respond in this same format.

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [-8.360047, 41.639484] },
      "properties": {}
    },
    {
      "type": "Feature",
      "geometry": { "type": "Point",
        "coordinates": [-8.360047, 41.639484] },
      "properties": {}
    },
    {
      "type": "Feature"
    }
  ]
}
```

Figure 38: Example of an all samples endpoint response

In case of an error any of the ones mentioned in the Table 10 is a viable possibility with each having their conditions described in this table.

Value	Error code	Description
No Samples Found	404 - Not Found	Error that happens when there are no fingerprinting samples associated with the requested floor number.
Invalid Floor Number	400 - Bad Request	Error that happens when the floor number provided is an invalid value.
Invalid Campus	400 - Bad Request	Error that happens when the campus id provided is an invalid value.
Badly Formatted Request Error	400 - Bad request	Error that happens when the request is not built properly, due to parameters missing.

Table 10: Possible errors for the all samples GET request

5.5.2 Temporal endpoints

In this subsection, the endpoints presented will be the temporal ones, which specify a time frame for which the user may want to see the data. The time frames available in the system are the last year, last three months and the last month. An example of each and every one of these endpoints will be presented. This endpoint's utility is to give the user, through the web application, the ability to visualize the areas of the map with only the fingerprinting samples within the specified time frame. All of the following endpoints can reply to requests of the type GET that contain the request parameter specified at the beginning of this section.

5.5.2.1 Less than a year endpoint

An example request for this endpoint would be "lessThanOneYear?floorNumber=0&campusId=CA", in case of success, this request should return a JSON object that contains a feature collection depicting all of the fingerprinting samples identified by a floor number of zero in the Azurém campus with less than a year of age.

5.5.2.2 Less than three months endpoint

An example request for this endpoint would be "lessThanThreeMonths?floorNumber=0&campusId=CA", in case of success, this request should return a JSON object that contains a feature collection depicting all of the fingerprinting samples identified by a floor number of zero in the Azurém campus with less than three months of age.

5.5.2.3 Less than one month endpoint

An example request to this endpoint would be "lessThanOneMonth?floorNumber=0&campusId=CA", in case of success, this request should return a feature collection depicting all of the fingerprinting samples identified by a floor number of zero in the Azurém campus with less than one month of age.

In case of an error any of these three endpoints can return the same type of errors, all specified in Table 11

Value	Error code	Description
No Samples Found	404 - Not Found	Error that happens when there are no fingerprinting samples associated with the requested floor number.
No Samples within Timeframe Found	404 - Not Found	Error that happens when there are no samples found for the floor number provided within the given time frame.
Invalid Floor Number	400 - Bad Request	Error that happens when the floor number provided is an invalid value.
Invalid Campus Id	400 - Bad Request	Error that happens when the campus id provided is an invalid value.
Badly Formatted Request Error	400 - Bad request	Error that happens when the request is not built properly, due to parameters missing.

Table 11: Possible errors for the temporal samples endpoints GET request

5.6 Implemented Solution Validation

For this chapter, the validation is done for the conversion server developed in Spring Boot. But the load tests will only be done for the endpoint relative to all the samples, since all of the samples available in the data collection database are from the last month, therefore all of the temporal endpoints would return the same. It is important to keep in mind that, at the time these tests were done, the number of fingerprinting samples in the database was 3039.

This service includes the same gzip compression system as the floorplan web service, but does not include a caching system. Eventually, a cache could be added, but it would need to have a reduced period between updates to better fit the use case of this service since it is important to have the tracked progress be as updated as possible.

The first test, similarly to the floorplan web service, was done with 500 users and 5000 requests in a span of 100 seconds, the results can be seen in Figure 39.

Requests	Executions			Response Times (ms)							Throughput
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s
Total	5000	0	0.00%	84.02	50	966	79.00	108.00	124.00	164.99	49.78

Figure 39: Results of the test for 500 users 5000 requests

As can be seen in Figure 39 under this load the conversion server worked perfectly, being able to successfully take care of all transactions with an average response time of 84ms. This means that the service is still not at its limit, therefore an increase to 750 users and 7500 requests in the same time span was done.

Requests	Executions			Response Times (ms)							Throughput
Label ^	#Samples ↕	FAIL ↕	Error % ↕	Average ↕	Min ↕	Max ↕	Median ↕	90th pct ↕	95th pct ↕	99th pct ↕	Transactions/s ↕
Total	7500	0	0.00%	2542.04	63	9803	2552.50	4341.00	4494.00	4759.90	59.10

Figure 40: Results of the test for 750 users 7500 requests

As seen in Figure 40 there was now an increase in the average response time to 2542ms and an increase in the transactions per second to 59, which indicates that the service is reaching its load limit for this endpoint. But for the sake of limit testing the user number was increased to 1000 and the total requests increased to 10000.

Requests	Executions			Response Times (ms)							Throughput
Label ^	#Samples ↕	FAIL ↕	Error % ↕	Average ↕	Min ↕	Max ↕	Median ↕	90th pct ↕	95th pct ↕	99th pct ↕	Transactions/s ↕
Total	10000	0	0.00%	6702.24	177	17178	7071.00	10597.90	10934.90	11324.86	58.51

Figure 41: Results of the test for 1000 users 10000 requests

With now an average response time of 6702ms as can be seen in Figure 41 it is clear that the service's limit for this endpoint has been reached, since even this much waiting time can prove frustrating to some users. While not a time sensitive task and not an endpoint that will receive requests as frequently as some others, it is still clear to see that the average number of requests per 100 second should be kept under 7500. This number makes sense since this endpoint is meant to be executed alongside the "getByFloor" query and they do have similar performances. Its main limitation is the amount of processing it has to do, converting all of the usable samples available in the database to a GeoJSON feature collection that contains all of the points that represent the usable samples.

Conclusions and future work

The main focus of this dissertation was the development of a floorplan web service that would allow the integration of georeferenced floorplans into the mobile application “Where@UM”, having said development been completed with success.

This mobile application, as mentioned previously, is being developed by multiple people, and its other supporting systems are the focus of different dissertations.

With this integration complete the problem of the space’s geometry was resolved. To achieve this result, many challenges were faced and successfully overcome with the steps described in the several sections and chapters of this dissertation.

One of the main challenges faced in this project was that the existing floorplans were not in a format appropriate for our applications. Therefore, looking for the state-of-the-art solutions available was necessary to find the best format possible. The identified format was IMDF, but the use of this format is only directly supported for iOS and the intent was to build an android application. Thus, it lead to the challenge of adapting this format in the simplest and most efficient way to this operating system.

As mentioned in previous chapters, this format heavily depends on GeoJSON; therefore, a conversion method between the available SVG data and GeoJSON was needed and accomplished. Both were the main challenges of this dissertation that required the most time to solve.

Afterwards, having the required GeoJSON files, it was necessary to design a web service that would have an interface the mobile application could make requests to, to obtain these files whenever needed. The endpoints made available by this service had to fulfill, in the most efficient and scalable way, the needs of the mobile application, and therefore were improved several times throughout the development process.

Finally, some slight adjustments had to be made to the web service so it could satisfy the needs of a web application that was later developed to support the tracking of the crowdsourcing process.

Once all of the web service development was complete, it was necessary to develop the components required for the mobile and web applications, which would seamlessly integrate the floorplans received from the floorplans web service into their respective map components and functionalities. Some challenges were faced during this step in particular with the mobile application integration, due to a limited

availability of map visualization libraries that possessed the required functionalities. This limited availability was derived from a need to use NativeScript as the application's framework and this framework does not possess many map visualization tools.

With all of the above mentioned systems and methods having been developed and implemented with success, it is possible to say that all of the main requisites and objectives of this dissertation were fulfilled.

6.1 Future work

Although the main objectives of this dissertation have been accomplished, there is still a lot that can be done to improve the overall system further since this was the first prototype.

6.1.1 Mobile application

About the mobile application, additional functionalities can be added. Eventually, topology graphs and an entire navigation system that provides routes and guides the user to the desired destination, while working in tandem with the floorplans service.

6.1.2 Web application

In relation to the web application, it is still possible to add a notion of users. Hence, each user can perceive which data collections belong to them. This functionality helps implement gamification mechanisms that further incentivise users to contribute to the crowdsourcing. Further administrative functionalities could also be added eventually with the purpose of simplifying the monitoring process from an administrator's point of view. Additionally, the backend could be optimized by implementing a cache system that perfectly balances having updated data as well as reducing the processing needed in case a plethora of requests are made in a short period of time.

6.1.3 Floorplans web service

In relation to the floorplans web service, eventually, when the platform hits a big enough scale, it will be necessary to scale the service horizontally to be able to efficiently provide the floorplan operations to more users. More changes to the structure of the stored files could also be done to approximate further the structure of these GeoJSON files to the implementation of IMDF for iOS.

Bibliography

- [1] T. Adams. “Using SVG and XSLT to display visually geo-referenced XML”. In: *Visualizing Information Using SVG and X3D* (Jan. 2005), pp. 256–265. doi: [10.1007/1-84628-084-2_12](https://doi.org/10.1007/1-84628-084-2_12) (cit. on p. 10).
- [2] S. Agarwal and K. Rajan. “Performance analysis of MongoDB versus PostGIS/PostgreSQL databases for line intersection and point containment spatial queries”. In: *Spatial Information Research* 24 (Nov. 2016). doi: [10.1007/s41324-016-0059-1](https://doi.org/10.1007/s41324-016-0059-1) (cit. on p. 31).
- [3] Apple. “Indoor Maps Program”. In: (2021). url: <https://register.apple.com/indoor> (cit. on p. 1).
- [4] Apple and O. G. Consortium. “Indoor Mapping Data Format”. In: (2021). url: <https://docs.ogc.org/cs/20-094/index.html> (cit. on p. 9).
- [5] ArcGIS. “GeoJSON”. In: (2021). url: <https://doc.arcgis.com/en/arcgis-online/reference/geojson.htm> (cit. on p. 10).
- [6] E. Baralis et al. “SQL versus NoSQL databases for geospatial applications”. In: Dec. 2017, pp. 3388–3397. doi: [10.1109/BigData.2017.8258324](https://doi.org/10.1109/BigData.2017.8258324) (cit. on p. 30).
- [7] T. Becker, C. Nagel, and T. H. Kolbe. “A multilayered space-event model for navigation in indoor spaces”. In: *3D geo-information sciences*. Springer, 2009, pp. 61–77 (cit. on p. 7).
- [8] F. Biljecki, H. Ledoux, and J. Stoter. “An improved LOD specification for 3D building models”. In: *Computers, Environment and Urban Systems* 59 (2016), pp. 25–37. issn: 0198-9715. doi: <https://doi.org/10.1016/j.compenurbsys.2016.04.005>. url: <https://www.sciencedirect.com/science/article/pii/S0198971516300436> (cit. on p. 8).
- [9] G. Brown et al. “Modelling 3D Topographic Space Against Indoor Navigation Requirements”. In: Oct. 2013, pp.1–22. isbn: 978-3-642-29792-2. doi: [10.1007/978-3-642-29793-9_1](https://doi.org/10.1007/978-3-642-29793-9_1) (cit. on p. 8).
- [10] O. G. Consortium. “CityGML”. In: (2021). url: <https://www.ogc.org/standards/citygml> (cit. on p. 8).

- [11] U. C. Foundation. “Urban Computing Foundation”. In: (2022). url: <https://uc.foundation/> (cit. on p. 47).
- [12] U. C. Foundation. “vis.gl”. In: (2022). url: <https://vis.gl/> (cit. on p. 47).
- [13] M. Goetz and A. Zipf. “Extending OpenStreetMap to indoor environments: Bringing volunteered geographic information to the next level”. In: Sept. 2011, pp. 51–62 (cit. on p. 6).
- [14] InkScape. “InkScape”. In: (2022). url: <https://inkscape.org/> (cit. on p. 15).
- [15] B. S. International. “IFC”. In: (2022). url: <https://technical.buildingsmart.org/standards/ifc/> (cit. on p. 9).
- [16] jczaplew. “geojson-precision”. In: (2022). url: <https://github.com/jczaplew/geojson-precision> (cit. on p. 23).
- [17] D. Laksono. “Testing Spatial Data Deliverance in SQL and NoSQL Database Using NodeJS Fullstack Web App”. In: *2018 4th International Conference on Science and Technology (ICST)*. 2018, pp. 1–5. doi: [10.1109/ICSTC.2018.8528705](https://doi.org/10.1109/ICSTC.2018.8528705) (cit. on p. 31).
- [18] J. Lee et al. “OGC IndoorGML 1.1”. In: (2019). url: <https://docs.ogc.org/is/19-011r4/19-011r4.html> (cit. on pp. 6, 7).
- [19] K.-J. Li et al. “Survey on Indoor Map Standards and Formats”. In: *2019 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. 2019, pp. 1–8. doi: [10.1109/IPIN.2019.8911796](https://doi.org/10.1109/IPIN.2019.8911796) (cit. on p. 11).
- [20] J. M. Lourenço. *The NOVAthesis \LaTeX Template User’s Manual*. NOVA University Lisbon. 2021. url: <https://github.com/joaomlourenco/novathesis/raw/master/template.pdf> (cit. on p. ii).
- [21] O. S. Maps. “Open Indoor”. In: (2022). url: <https://wiki.openstreetmap.org/wiki/OpenIndoor> (cit. on p. 6).
- [22] O. S. Maps. “Open Station Map”. In: (). url: <https://openstationmap.org/> (cit. on p. 6).
- [23] O. S. Maps. “Open Street Maps”. In: (2021). url: <https://www.openstreetmap.org/> (cit. on p. 5).
- [24] MongoDB. “MongoDB”. In: (2022). url: <https://www.mongodb.com/> (cit. on p. 29).
- [25] PostGIS. “PostGIS”. In: (2022). url: <https://postgis.net/> (cit. on p. 29).
- [26] QGIS. “QGIS”. In: (2022). url: <https://www.qgis.org/> (cit. on p. 15).
- [27] vis.gl. “React-map-gl GitHub”. In: (2022). url: <https://github.com/visgl/react-map-gl/> (cit. on p. 47).

Annex 1

This annex lists the python script developed to remove the properties that the georeferencing process added to the final GeoJSON file that serve no purpose for our application.

```
1 import os
2 import json
3 directory = 'Insert path here where geojson files are located'
4
5 for filename in os.listdir(directory):
6     f = os.path.join(directory, filename)
7     if os.path.isfile(f):
8         file = open(f, 'r')
9         geojson = json.load(file)
10        for i in geojson['features']:
11            i['properties'] = {}
12        file.close()
13        output = open(filename, 'w')
14        json.dump(geojson, output)
15        output.close()
```

Listing I.1: Script to remove extra properties

