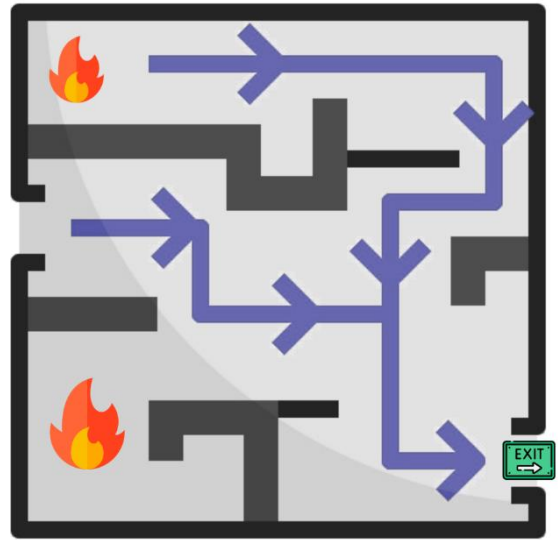




Building Evacuation Assistance System

Daniel Sousa Mendes

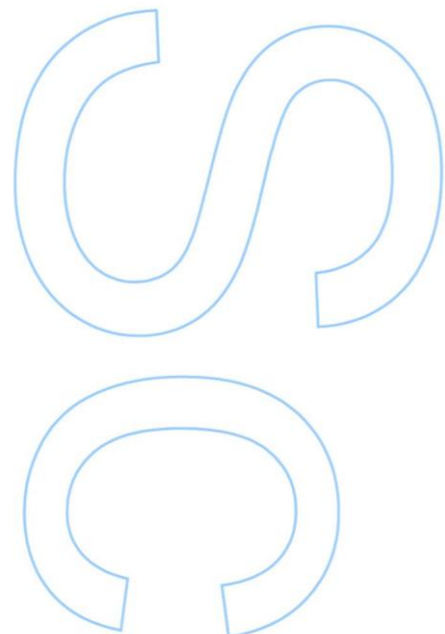
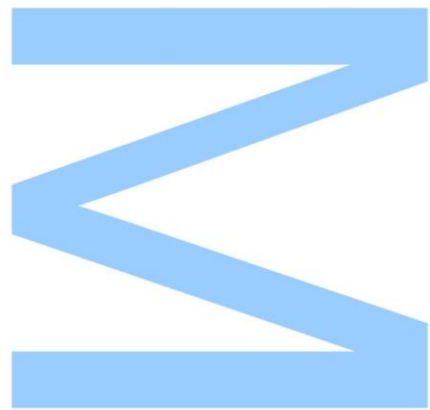
Mestrado em Engenharia de Redes e Sistemas Informáticos
Departamento de Ciências de Computadores
2022

Orientador

Rui Prior, Professor, Faculdade de Ciências da Universidade do Porto

Coorientador

Sérgio Crisóstomo, Professor, Faculdade de Ciências da Universidade do Porto

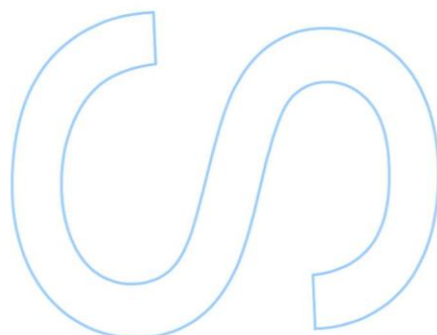
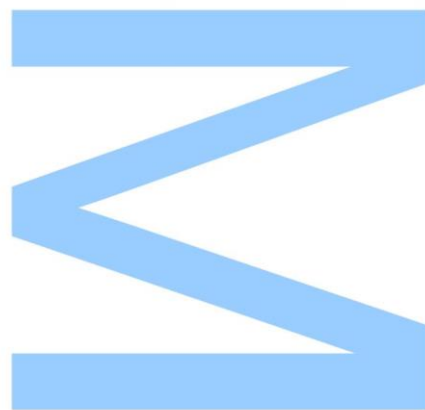




Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, ____/____/____



Dedicated to my mother, Silvia

Declaração de Honra

Eu, Daniel Sousa Mendes, inscrito no Mestrado em Engenharia de Redes e Sistemas Informáticos da Faculdade de Ciências da Universidade do Porto declaro, nos termos do disposto na alínea a) do artigo 14.º do Código Ético de Conduta Académica da U.Porto, que o conteúdo da presente dissertação reflete as perspetivas, o trabalho de investigação e as minhas interpretações no momento da sua entrega.

Ao entregar esta dissertação, declaro, ainda, que a mesma é resultado do meu próprio trabalho de investigação e contém contributos que não foram utilizados previamente noutros trabalhos apresentados a esta ou outra instituição.

Mais declaro que todas as referências a outros autores respeitam escrupulosamente as regras da atribuição, encontrando-se devidamente citadas no corpo do texto e identificadas na secção de referências bibliográficas. Não são divulgados na presente dissertação quaisquer conteúdos cuja reprodução esteja vedada por direitos de autor.

Tenho consciência de que a prática de plágio e auto-plágio constitui um ilícito académico.

Daniel Sousa Mendes

Porto, 30 de Setembro de 2022

Acknowledgements

This work was carried out under the "SafeCities - Inovação para Construir Cidades Seguras" project, with reference POCI-01-0247-FEDER-041435, co-financed by the Fundo Europeu de Desenvolvimento Regional (FEDER), through the Programa Operacional Competitividade e Internacionalização (COMPETE 2020) and Portugal 2020.

In this section, I want to express my thanks to all the people who were involved in my journey and who helped me complete a very important stage of my life.

First of all, I want to thank my family, namely my mother, father, brother and grandparents who have always supported me and helped me whenever I needed them. Without you, your company and support, I would never have gotten this far. You guys are the most important thing in my life.

I would also like to thank my supervisors, professors Rui Prior and Sérgio Crisóstomo, for all their availability and support throughout the development of this project.

I thank the Institute of Telecommunications (UIDB/50008/2020) for hosting this work.

And finally, I would also like to thank my closest friends who have accompanied me throughout all these great years of college.

To all of you, without exception, thank you.

Resumo

Hoje em dia, os edifícios não são estruturas simples. A estrutura de cada edifício tem de ser tão eficiente quanto possível para maximizar a utilização de cada espaço e, como tal, navegar por um edifício desconhecido nem sempre é uma tarefa fácil, especialmente durante condições perigosas. Para tal, têm sido implementadas várias medidas para tentar ajudar o processo de evacuação, tais como o desenvolvimento e colocação de planos de emergência em áreas visíveis do edifício. No entanto, o problema é que em situações de pânico, estes planos raramente são úteis e, como são estáticos, podem por vezes indicar informações erradas, não indicando sempre o caminho óptimo para a saída.

O objectivo deste projeto é desenvolver um sistema dinâmico que permita o processo de cálculo de rotas de fuga seguras nos edifícios, durante situações de emergência. O trabalho propõe a criação de um sistema que possa receber e processar informação de um edifício definida em um formato padrão de mapeamento interior, que também seja capaz de estar à escuta de leituras monitorizadas enviadas pelos sensores instalados em todo o interior do edifício, e que possa desenvolver uma rede de navegação do edifício a fim de calcular e transmitir caminhos seguros para os utilizadores localizados no interior do mesmo, evitando que os utilizadores naveguem através de áreas afectadas por elevados níveis de perigo.

Abstract

Nowadays, buildings are not simple structures. The structure of each building has to be as efficient as possible in order to maximize the use of each space and, as such, navigating through an unknown building isn't always an easy task, specially during hazardous conditions. To this end, various measures have been implemented to try to assist the evacuation process, such as the development and placement of emergency plans in visible areas of the building. However, the problem is that in panic situations, these plans are rarely useful, and since they are static, they can sometimes indicate wrong information, not always indicating the optimal way out.

The objective of this project is to develop a dynamic system that allows the process of calculating safe escape routes in buildings during emergency situations. The work proposes the creation of a system that can receive and process information from a building defined in a standard indoor mapping format, that is also able to listen to the monitored readings sent by sensors installed throughout the building, and that can develop a building navigation network in order to calculate and transmit safe paths for users located inside the building, preventing them from navigating through areas affected by high levels of danger.

Contents

Declaração de Honra	iii
Acknowledgements	v
Resumo	vii
Abstract	ix
Contents	xiv
List of Tables	xv
List of Figures	xix
Listings	xxi
Acronyms	xxiii
1 Introduction	1
1.1 Motivation	1
1.2 Objectives	3
1.3 Dissertation structure	4
2 Preliminaries	7
2.1 System architecture	8
2.1.1 Architecture types	8

2.1.2	Sensor network	9
2.1.3	Database	10
2.2	Indoor mapping standards	13
2.2.1	IMDF	14
2.2.2	IndoorGML	16
2.3	Indoor positioning systems	18
2.3.1	RFID	18
2.3.2	Bluetooth Low Energy	19
2.3.3	Wi-Fi	19
2.3.4	Ultra-Wideband	20
2.3.5	Summary	20
2.4	Search algorithms	21
2.4.1	Dijkstra’s algorithm	21
2.4.2	Bellman-Ford algorithm	21
2.4.3	Floyd-Warshall algorithm	22
2.4.4	A* algorithm	22
2.4.5	Complexity comparison	22
3	Related Work	23
3.1	Time-based graphs approach	23
3.1.1	Building representation	23
3.1.2	Hazard graph	23
3.1.3	Navigation graph	24
3.1.4	Sensor exposure time	24
3.1.5	Safest path calculation	25
3.1.6	Network propagation and execution process	25
3.2	Distance and hazard intensity-based graphs approach	26
3.2.1	Network structure	26

3.2.2	Assumptions	26
3.2.3	Building representation	26
3.2.4	Distributed architecture	28
3.2.5	Safest path computation	28
3.3	Risk level-based graph approach	29
3.3.1	System structure	29
3.3.2	Assumptions	29
3.3.3	Building graph representation	30
3.3.4	Safest path computation	32
3.3.5	Evacuation process	33
3.4	SmartEscape	33
3.4.1	System architecture	34
3.4.2	Mobile app	34
3.4.3	Intelligent Routing Engine	35
3.5	Summary	40
4	Analysis and Development	41
4.1	System architecture	41
4.2	Building representation	42
4.3	Backend	45
4.3.1	IndoorGML file processing	45
4.3.2	Data structure	46
4.3.3	API	50
4.3.4	<i>Publish/Subscribe</i> service	52
4.3.5	Evacuation process	53
4.4	Frontend	58
4.5	Mobile application	58
5	Results	61

5.1	Building representation	61
5.2	System demonstration	62
6	Conclusion	67
6.1	Future Work	68
6.1.1	Computational representation of the building	68
6.1.2	Additional metrics	68
6.1.3	Danger level of a link	68
6.1.4	User's location node	69
6.1.5	API security	69
6.1.6	Fault tolerance	69
	Bibliography	71

List of Tables

2.1	Centralized systems' pros and cons [38]	9
2.2	Decentralized systems' pros and cons [38]	9
2.3	Distributed system's pros and cons [38]	9
2.4	List of <code>.geojson</code> feature files that constitute an IMDF archive	15
2.5	Comparison between indoor positioning systems	20
2.6	Algorithm complexities. V = number of vertices and E = number of edges	22
3.1	Temperature index values [9]	31
3.2	Carbon monoxide index values [9]	31
3.3	Carbon monoxide index values [9]	31
3.4	Density index values [9]	31
4.1	Building table fields and descriptions	47
4.2	Node table fields and descriptions	48
4.3	NodeTransitions table fields and descriptions	48
4.4	Transition table fields and descriptions	49
4.5	TransitionNodeConnection table fields and descriptions	49
4.6	Sensor table fields and descriptions	50
4.7	Defined danger levels for specific temperature intervals	50

List of Figures

1.1	Total number of buildings over 100 meters high around the globe [21]	2
1.2	Example of a building emergency plan [32]	3
2.1	Centralized vs Decentralized vs Distributed system architecture [8]	8
2.2	Wireless Sensor Network [46]	10
2.3	Simple relational database structure, retrieved from [14]	11
2.4	RDB vs NoSQL [27]	12
2.5	Example university emergency floorplan, retrieved from [7]	13
2.6	University indoor space graph, retrieved from [7]	13
2.7	Basic GeoJSON elements, retrieved from [43]	14
2.8	IMDF layer hierarchy	15
2.9	IndoorGML cellular space model topology, retrieved from [29]	16
2.10	IndoorGML Multi-Layered Representation (MLR) [29]	17
2.11	IndoorGML Multi-Layered Space Model (MLSM) [29]	17
2.12	RFID Architecture, retrieved from [4]	18
2.13	BLE indoor tracking through RSSI [5]	19
2.14	Comparison between UWB and other indoor positioning solutions's frequencies [18]	20
3.1	Example hazard graph [25]	24
3.2	Example navigation graph [25]	24
3.3	Calculated hazard times [25]	24
3.4	Example safety path calculated using the approach proposed by [25]	25

3.5	Building graph, retrieved from [11]	27
3.6	Example of a link between two decision nodes, ruled by one sensor [11]	27
3.7	Example graph of sensors and decision nodes, retrieved from [11]	27
3.8	Example safest path recommendation, retrieved from [11]	28
3.9	Example of a building floorplan, retrieved from [9]	30
3.10	Graph representation of the building floorplan retrieved from [9]	30
3.11	Graph transformation, retrieved from [9]	32
3.12	SmartEscape architecture, retrieved from [40]	34
3.13	SmartEscape mobile app user form, retrieved from [40]	35
3.14	Evacuee’s smartphone screenshots taken during live evacuation test. [40]	35
3.15	Risk factors used for the Artificial Neural Network model, retrieved from [40]	36
3.16	Risk factors considerations in case of physical disabilities, retrieved from [40]	36
3.17	Risk factor penalty results, retrieved from [40]	37
3.18	Risk groups, retrieved from [40]	37
3.19	ANN structure, retrieved from [40]	38
3.20	(a) - ANN based routing; (b) - evaluation of link states, retrieved from [40]	39
4.1	Proposed system architecture	42
4.2	Floor 0 [10]	43
4.3	Floor 1 [10]	43
4.4	IndoorGML base layer representation of Departamento de Ciências de Computadores (DCC)’s floor 0	44
4.5	IndoorGML sensor layer representation of DCC’s floor 0	44
4.6	Generated classes	45
4.7	Proposed Entity-Relationship (ER) model diagram	47
4.8	DCC’s measured length	48
4.9	DCC’s measured width	48
4.10	DCC’s floor 0	51

4.11	Apache Kafka <i>publish/subscribe</i> communication flow	52
4.12	Arrays of distances by danger level	55
4.13	Example of an evacuation route transmitted to an evacuee	57
5.1	IndoorGML base layer model	62
5.2	Building creation form	63
5.3	Database building list	63
5.4	Simulated sensor coverage areas	64
5.5	Building configuration section	64
5.6	Mobile app <i>views</i> (deployed in an Android Pixel 5 emulator)	65

Listings

4.1	Example of node Room 003 defined in IndoorGML	43
4.2	IndoorGML file unmarshalling	46
4.3	Example of an IndoorGML inter-layer connection	49
4.4	Example of an IndoorGML transition	49
4.5	Building Graph construction	54
4.6	Proposed implementation of Dijkstra	56
5.1	Kafka producer environment	65

Acronyms

API	Application Programming Interface	JVM	Java Virtual Machine
AP	Access Point	MLR	Multi-Layered Representation
AoA	Angle of Arrival	MLSM	Multi-Layered Space Model
BLE	Bluetooth Low Energy	NRG	Node-Relation Graph
DBMS	Database Managment System	NoSQL	Not Only SQL
DB	Database	OGC	Open Geospatial Consortium
DCC	Departamento de Ciências de Computadores	RDB	Relational Database
ER	Entity-Relationship	REST	Representational State Transfer
FCUP	Faculdade de Ciências da Universidade do Porto	RFID	Radio-Frequency Identification
GML	Geographic Markup Language	RSSI	Received Signal Strength Indicator
GPS	Global Positioning System	SDK	Software Development Kit
ID	Identifier	SPA	Single-Page Application
IMDF	Indoor Mapping Data Format	SQL	Structured Query Language
JAXB	Java Architecture for XML Binding	ToF	Time-of-Flight
JSON	JavaScript Object Notation	UWB	Ultra-Wideband
		WSN	Wireless Sensor Network

Chapter 1

Introduction

In recent years, with the help of all the technological growth that has followed the evolution process of our species, we have been able to witness the creation of massive and increasingly complex structures to accommodate essential purposes for the proper functioning of our daily lives. These structures range from large shopping centers with underground parking areas to buildings hundreds of meters high, continuously habitable, and with a vast number of floors (skyscrapers). Constructions of this magnitude must be prepared to be frequented by thousands of people constantly, which calls into question the efficiency of existing methods for evacuation in case of accidents, whether these are fires, earthquakes, gas leaks, among others. Thus, bearing in mind the complexity of the structure of these buildings, it is crucial to monitor and implement safety measures to protect as many victims as possible in the event of any type of accident.

1.1 Motivation

The occurrence of accidents, although occasional, is inevitable. Be these accidents caused by human error, or natural causes, however much we try to find ways to prevent the occurrence of a sinister event, we only have the ability to prevent and prepare ourselves for such situations. For this, various types of sensors, emergency plans and evacuation signaling devices have been placed in key areas of a building to not only detect the occurrence of a sinister event, but also to help evacuate people quickly out of the building, in the shortest amount of time possible.

In [1] we learn that, according to the estimates of the United Nations, by 2050, two thirds of people will be living in urban areas, meaning that in the future, most of us will be living or working in buildings designed to hold hundreds if not thousands of people. This also means that with limited space, we will increasingly build upwards, which is already occurring in major cities around the globe, as illustrated in figure 1.1.

With fires being one of the most frequent forms of hazardous events in building structures, the means of preventing and protecting fires have to be designed to reduce the possibility and severity of these fires, thereby ensuring the safe evacuation of people in buildings.

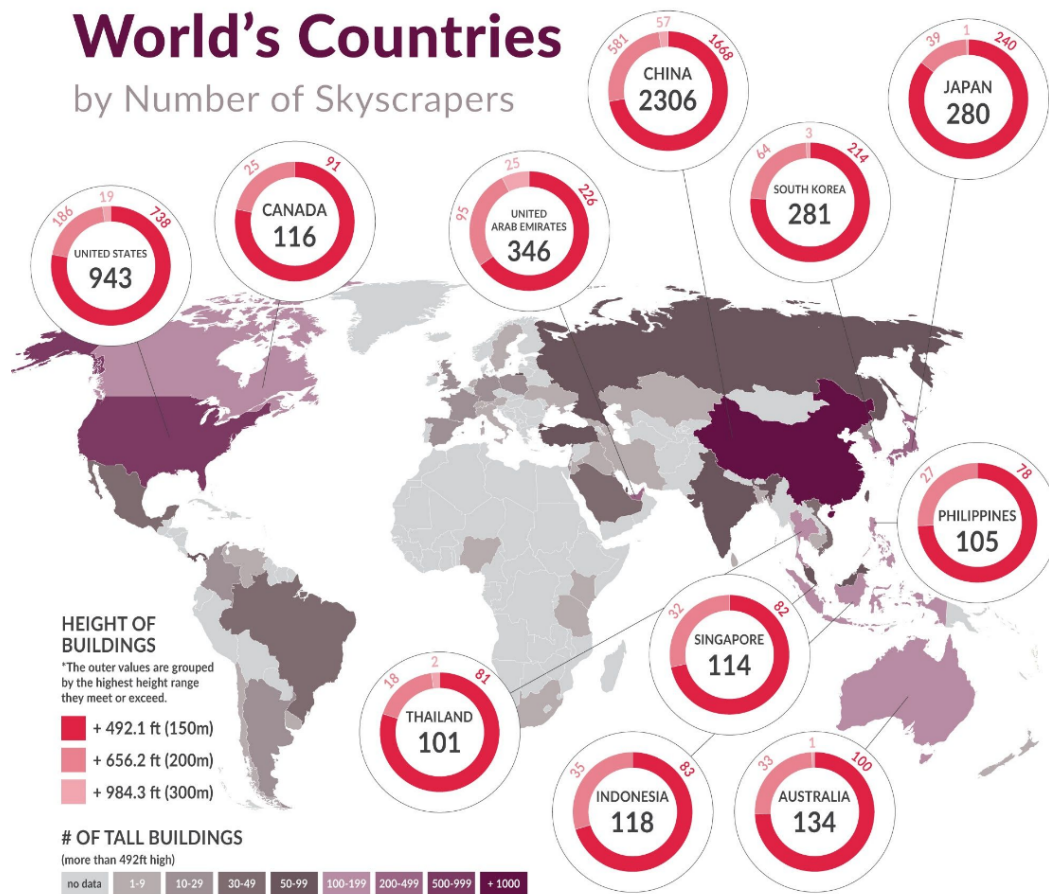


Figure 1.1: Total number of buildings over 100 meters high around the globe [21]

According to [6], the United States and the United Kingdom have been studying the effects of toxic gases and smoke on human behavior in fire situations, especially in family homes, since 1970. The conclusion is that most fire victims, whether fatal or non-fatal, show greater difficulty (or related to the cause of death) when exposed to toxic gases and smoke than when exposed to burns. This indicates that the fire itself might not be the biggest issue, but the time of exposure to the smoke resulting from it could be, meaning that an expedited and effective evacuation time is essential.

Another thing to keep in mind, is that in an apartment or familiar workplace it is expected that people are aware of their surroundings and are well informed about what to do in the event of a fire [1]. However, in places like hotels, this is not the case. Most of its occupants are transient and unfamiliar with the building layout, and could well be asleep when an emergency occurs. Also, other mass occupied buildings such as care homes or hospitals bring a whole host of new challenges for evacuation where a number of people may be less mobile and able to escape without assistance.

Most buildings have emergency plans in place to determine evacuation routes and procedures to be taken in dangerous situations, as illustrated in figure 1.2. The emergency plan is usually

located in strategic areas to ensure that it can be easily identified in a dangerous situation. It shows relevant information about the location of fire hydrants, fire extinguishers, alarm buttons and emergency telephones. However, since they are static, these evacuation routes can become unsafe if the path is affected during the event of a crisis. As such, thanks to advancements made on emergency evacuation and technology fields, it is possible to greatly decrease the amount of casualties by developing systems that are able to monitor the safety state of the building and calculate safe evacuation routes.



Figure 1.2: Example of a building emergency plan [32]

1.2 Objectives

The objective of this work is to propose an intelligent system capable of monitoring the security status of a building through the use of sensors. These sensors should always be alert to possible environment changes in the indoor space, in order to preemptively detect an emergency situation and compute safe evacuation paths.

In order to make this possible, firstly we need to be able to create a computational representation of the building, which is a crucial point in the implementation of the system. For this, we will be mapping the indoor characteristics of buildings using an open standard format designed for representing simple geographical features. This will allow the mapping of every office, hall or staircase in the building, which will be essential in order to calculate the safest pathway to the emergency exit.

Another important topic of this research is to allow the system to be alert to sudden changes of sensor readings. These readings can inform about temperature fluctuations or changes in smoke density that will trigger a new recalculation of the safest emergency routes. This is crucial

because, as fire spreads, temperatures rise and smoke becomes more dense, and, as such, initial calculated routes might become dangerous and inaccessible during the evacuation procedure. Therefore, a module will be developed to receive information in the form of events from the sensors, so that readings of environmental variables can be sent periodically. If one of these events contains data values that are considered unsafe, a new recalculation of the safest path will trigger, enabling the victims to avoid the area covered by that sensor.

So as to achieve the calculation route to exit the building, one more key topic during the development of this work is to find out the best possible algorithms that allow a quick and effective computation of the safest path towards the emergency exit. For this, a study will be conducted to analyze different safe path search approaches in order to choose the most appropriate one for the proposed system.

Finally, a mobile application will be developed in order to provide the means to guide hazard victims through the route they should take to reach the emergency exit unharmed. However, one thing to keep in mind is that, in order to pinpoint the location of each evacuee in a real situation, indoor real-time tracking techniques need to be studied and used, and, since its a rather complex and broad topic, it will not be inside the scope of this project. Therefore, only general research approaches that allow the overcoming of this problem will be mentioned in this dissertation.

1.3 Dissertation structure

Chapter 1 - Introduction Gives a brief introduction to the subject of evacuation systems, explains the motivation behind this project and the objectives of its development.

Chapter 2 - Preliminaries Mentions the main concepts that impact the development of an evacuation system. More specifically, important factors in designing the system architecture, standard formats for indoor area mapping, indoor positioning systems, and evacuation pathfinding algorithms are explained.

Chapter 3 - Related work Presents the study conducted beforehand on existing solutions to the indoor evacuation problem. The various approaches proposed by the authors and the divergences in the resulting system are discussed and explained.

Chapter 4 - Analysis and Development Describes the steps taken and the main decisions made during the development of the project. The whole structure of the system is described, namely the chosen architecture, the distinct components implemented and how they interact with each other, and the process of calculating and transmitting evacuation paths.

Chapter 5 - Results Presents the results obtained through tests performed on the system. Simulated temperature readings are sent from sensors to a specific area of the building, thus forming a danger region. Subsequently, an appropriate evacuation path is calculated and sent to the user's mobile application, in order to illustrate the route to take to the recommended emergency exit.

Chapter 6 - Conclusion Conclusion of the dissertation, presenting some general observations about the project and describing sections that are qualified for improvement in future work.

Chapter 2

Preliminaries

A hazard event can happen unexpectedly at any given time. From in-house fires to gas leaks to earthquakes, natural or non-natural causes can effectively change drastically and instantly the entire course of our daily routine. If no proper safety measures are taken, these type of hazards can result in massive life fatalities and building damages, which means researching and working on development solutions that can mitigate these effects will have huge impacts on public safety and can, in catastrophic scenarios, save dozens if not hundreds of lives.

Over the years, some solutions have been developed to try to minimize casualties in emergency scenarios, namely the creation and placement of emergency plans in visible areas inside the building. However, the problem with this solution is that emergency plans are static, so there may be instances where the path suggested by them passes through dangerous regions, exposed, for example, to fire, and therefore, as they do not retain the most up-to-date information on the safety status of all areas of the building, they are not able to indicate the safest path every time it is needed.

This is where evacuation systems come in. They are responsible for understanding which areas of the building are navigable or not, and which of them can be used safely in the event of hazards. They also need to monitor the environmental variables constantly in order to be alert to sudden changes in temperature, smoke density or toxic gas values. These systems should be able to calculate the safest path from any given location of the building to one of its emergency exits and transmit that information to every evacuee present inside the building.

This chapter will serve as a baseline for crucial topics that will be mentioned throughout this thesis or are related to the field of evacuation systems. Notions and mechanisms that inspired and impacted the development of the project, such as appropriate indoor mapping format standards for building representation, algorithms for searching shortest paths, sensor monitoring services, and system architectures methodologies will be explained in order to provide a better understanding of this project.

2.1 System architecture

In this section the different types of system architectures will be addressed and explained, exposing their different structures and corresponding advantages and disadvantages, shown in tables 2.1, 2.2 and 2.3.

2.1.1 Architecture types

Firstly, before the development phase of any system, it's important to have a firm idea of what the architecture design should be. Generally, there are three distinct type of architecture designs: centralized, decentralized, and distributed. And while all these types of systems can work effectively, some are better suited for specific use cases. Each systems' magnitude can be very small, connecting only a few devices and a few users, or they can be extensive, spanning all around the globe. However, all of them face the same challenges: fault tolerance, maintenance costs, and scalability [38].

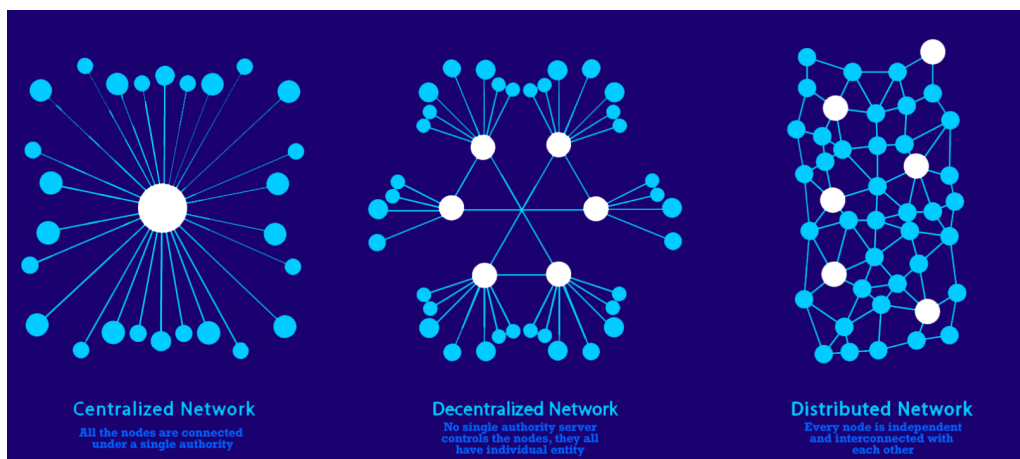


Figure 2.1: Centralized vs Decentralized vs Distributed system architecture [8]

2.1.1.1 Centralized systems

A centralized system, probably the most intuitive type of architecture, uses a client/server model, where one or more client nodes connect directly to a central server. This is the most common type of system in many organizations, where clients send requests to corporate servers and receive responses [13]. However, this system has an important limitation. If the server crashes or becomes otherwise unavailable, the system stops working properly and users cannot access data. Since centralized systems require a central owner to connect all other users and devices, the availability of the network depends on that owner and on the communications infrastructure. Table 2.1 summarizes the pros and cons of centralized systems.

Advantages	Disadvantages
Simpler to implement and deploy More cost-efficient for small systems Dedicated resources Practical when data needs to be controlled centrally	Prone to failures Highly dependent on the network connectivity Higher security and privacy risks for users Longer access times for users who are far from the server

Table 2.1: Centralized systems' pros and cons [38]

2.1.1.2 Decentralized systems

Decentralized systems are somewhat similar to centralized ones, differing in that they do not have one central owner. Instead, decentralized systems have numerous owners, where each one has a copy of the data in order for users to access. Even though it still is vulnerable to server crashes, a decentralized system is inherently more tolerant to faults since if one central owner fails, the others can still handle users' requests [38].

Advantages	Disadvantages
Less prone to failures Increased performance More autonomy and control over resources	Higher maintenance costs Require replication and coordination among multiple servers

Table 2.2: Decentralized systems' pros and cons [38]

2.1.1.3 Distributed systems

Finally, a distributed system is one so that is similar to the decentralized system in the fact that it does not have a single central owner, but removes the centralization factor altogether, meaning that there is no single entity that receives and responds to requests. As such, every node makes its own decisions and the final behaviour of the system is an aggregation of decisions individually taken by the nodes.

Advantages	Disadvantages
Fault-tolerant Increased security Highly scalable	Harder to deploy Higher maintenance costs

Table 2.3: Distributed system's pros and cons [38]

2.1.2 Sensor network

Given that the aim of this project is to develop a evacuation system, it is necessary to maintain a constant monitorization of all areas of a building's infrastructure in order for the evacuation

algorithm to guide each evacuee through safer pathways, avoiding dangerous spaces. One way to accomplish this is by implementing and deploying a Wireless Sensor Network (WSN) model that extends throughout the entire building's indoor region.

A wireless sensor network is an infrastructure-free wireless network that consists in the deployment of a broad number of wireless sensors that are responsible for monitoring environmental conditions of a given area. The sensors can communicate between each other using radio signals and transmit it's data through a base station that simply serves as an interface between the sensor network and the users that need to receive and analyse the data monitored by the sensors [35].

In the evacuation systems' use case, most common sensors should usually be able to monitor temperature and smoke density, in order to detect in-house fires, although a few sensors are also able to monitor different poisonous fumes originated from eventual gas leaks. The data retrieved by these sensors is crucial, especially if it is updated regularly, so that the evacuation procedure runs in the most efficient way possible by always transmitting the safest path to its evacuees at every time it is requested.

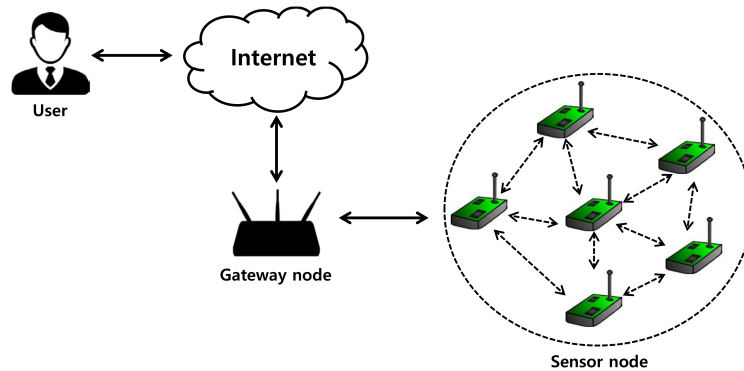


Figure 2.2: Wireless Sensor Network [46]

2.1.3 Database

In the previous subsection it was briefly explained how a Wireless Sensor Network works and how it allows for the deployment of a sensor network that enables constant monitorization of a building's indoor space to detect possible hazards. However, any evacuation system has the need to save the information monitored by sensors, as well as data concerning the building's navigable network or other relevant information related to the system's specific objective. Hence the need of a database. A Database (DB) is simply an organized collection of structured information [30], usually stored in a computer system and generally controlled by a Database Managment System (DBMS). The information stored in the most common types of databases is normally structured in a set of rows and columns throughout a number of tables, where each table stores information about a specific subject.

2.1.3.1 Relational databases

Relational databases are the most common type of database because they allow for organization of the information stored into a set of relationships where the data is saved into one or more tables. Each table has a predetermined set of columns, indicating each attribute of the relation (table) and a number of rows where the information is stored in. Usually, each table can have unique identifier (ID) column in order to distinguish each row of data from the same table, as well as different external tables in the database. A Relational Database (RDB) can establish relationships between information by joining tables, which allows for a quick and easy understanding and insight about the relationship between various data points [14]. Figure 2.3 illustrates a simple structure of a relational database.

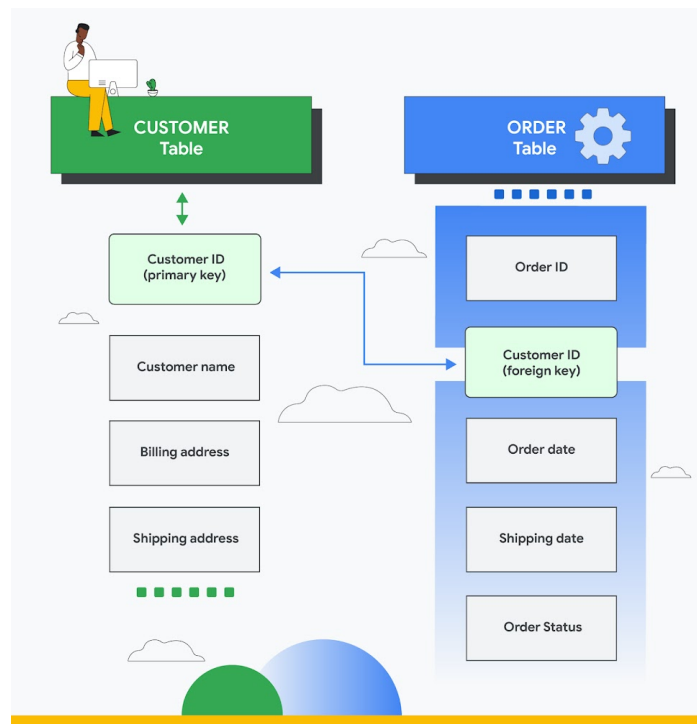


Figure 2.3: Simple relational database structure, retrieved from [14]

Relational databases typically use Structured Query Language (SQL) queries in order to fetch, insert, modify, delete or link information in tables. One of their main advantages and reason for their popularity is that they allow for simple and intuitive retrieval of data, which is beneficial for many corporations that need to store massive amounts of data and access it frequently [37]. A few common relational databases are MySQL, which is an open source management system that maintains high availability, scalability and performance, making it extremely useful for web and application development; PostgreSQL, also an open-source database manager that retains a high level of performance and flexibility, which is useful in large read/write speeds, specially through big amounts of data; and finally MariaDB which is based on MySQL but adds support for even more storage engines, allowing it to perform faster than MySQL while also running SQL and NoSQL in the same database [31].

2.1.3.2 Non-relational databases

As opposed to relational databases mentioned before, non-relational databases, also commonly referred as Not Only SQL (**NoSQL**), are more flexible because they allow for the storing of information in a non-tabular form. Rather, these types of databases use a storage model that is developed for the distinct requirements of the type of information being saved. As an example, the data stored in the database may be in the format of a JavaScript Object Notation (**JSON**) document, a key/value pair, or even as a graph structured as a set of edges and vertices [39].

Non-relational databases are frequently used when large amounts of complex and different data need to be organized. As an example, if a big store has the need to store data about each client, like the name, order history or address, all of this information, despite having different formats, can be saved in the same document in the database [26]. A Non-relational database is frequently faster than a Relational Database (**RDB**) because each query doesn't need to iterate through several tables in order to deliver an answer, as opposed to relational databases. This means that non-relational databases are thus ideal for systems that need to handle a vast amount of distinct types of data or systems that need to save information that may be modified constantly. A few of the most popular NoSQL databases are MongoDB, a cross-platform open-source document-oriented database with JSON-like documents in dynamic schemas; Apache Cassandra, which is a distributed database that excels at handling massive amounts of structured data; and Redis, one of the most popular key/value structured databases [42]. To summarize, figure 2.4 illustrates an example of the structural difference between a regular Relational Database (**RDB**) and a MangoDB NoSQL database.

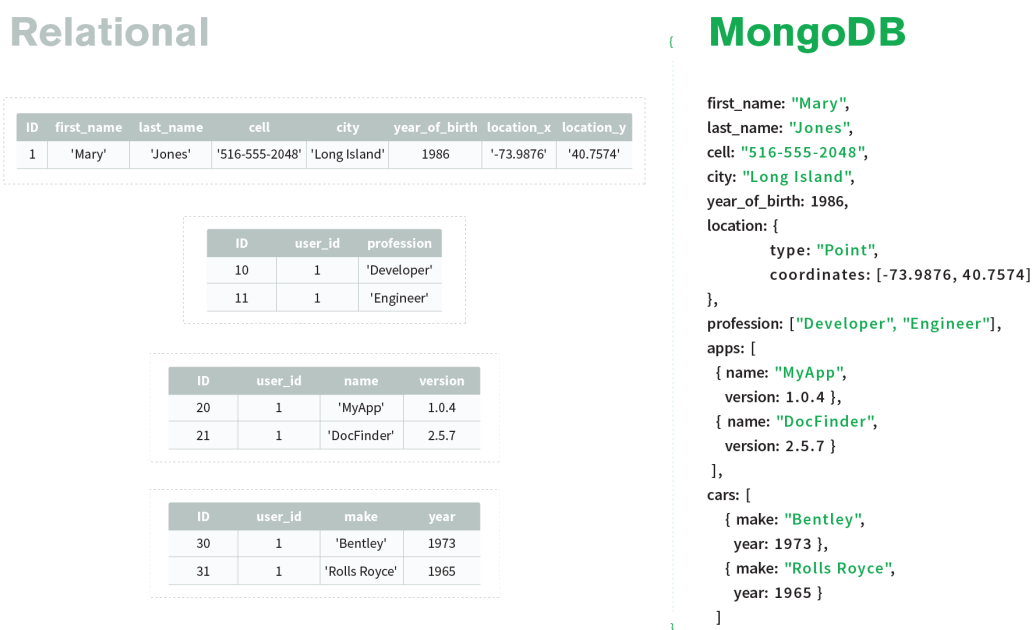


Figure 2.4: RDB vs NoSQL [27]

2.2 Indoor mapping standards

Another important aspect in the development of an evacuation assistance system is how to represent the interior space of a building. Although there are a few ways to exhibit indoor spaces and their features, most approaches usually gravitate towards a graph-based representation. A graph depiction typically has a set of nodes that define indoor spaces, such as rooms, restrooms, auditoriums, corridors, staircases, elevators and so on, as well as a set of edges that represent the connection of areas that have a direct pathway between each other, thus creating a representation of a walkable network of indoor locations. Each small/regular sized indoor space could be portrayed by one single node, while larger spaces should have multiple nodes in order to represent a more realist and complete navigable network. Figure 2.5 and 2.6 illustrate an example of an university's emergency floorplan and it's corresponding navigable graph representation.

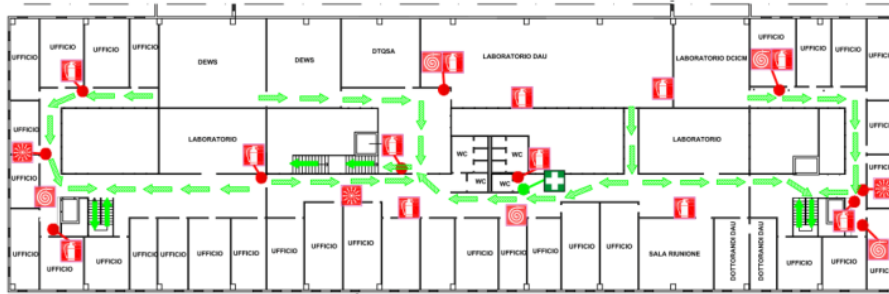


Figure 2.5: Example university emergency floorplan, retrieved from [7]

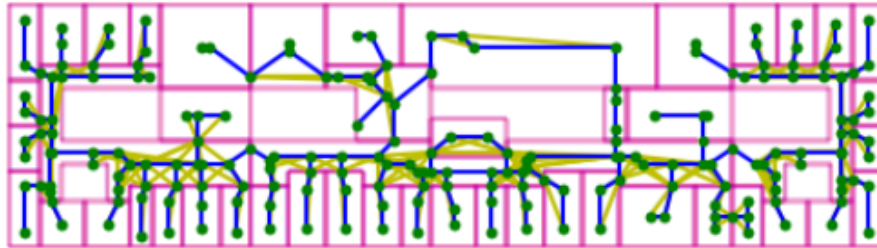


Figure 2.6: University indoor space graph, retrieved from [7]

Although a simple graph representation of the building would be enough to apply search algorithms for an evacuation scenario, there have been developments of indoor mapping format standards that allow for a better and more specified characterization of each indoor space. As such, this section will mention a few popular indoor mapping standards and explain briefly each of their main features in order to provide a better understanding of how they work and how they allow for the description of any area within the building's internal structure.

2.2.1 IMDF

Indoor Mapping Data Format (**IMDF**) is an indoor mapping standard developed by Apple Inc. and approved by the Open Geospatial Consortium (**OGC**) that provides a generalized, yet comprehensive, model for any indoor location, providing a basis for orientation, navigation and discovery [3]. IMDF's structure is expressed as an archive of GeoJSON files and each of them describe sections of a indoor space. All of the GeoJSON files must meet specific data model criteria in the specification of IMDF, as well as the validation requirements of GeoJSON itself.

GeoJSON is a **JSON**-based open standard format designed for representing simple geographical features, which include *Points* that can serve as addresses and locations; *LineStrings* to illustrate roadways or boundaries; *Polygons* that can be used to represent countries, businesses, or other land areas; as well multipart collections of these primitives (*FeatureCollections*) [43]. Figure 2.7 shows the basic elements of the GeoJSON format as well as their respective notation.

Type	Examples	
Point		<pre>{ "type": "Point", "coordinates": [30.0, 10.0] }</pre>
LineString		<pre>{ "type": "LineString", "coordinates": [[30.0, 10.0], [10.0, 30.0], [40.0, 40.0]] }</pre>
Polygon		<pre>{ "type": "Polygon", "coordinates": [[[30.0, 10.0], [40.0, 40.0], [20.0, 40.0], [10.0, 20.0], [30.0, 10.0]]] }</pre>
		<pre>{ "type": "Polygon", "coordinates": [[[35.0, 10.0], [45.0, 45.0], [15.0, 40.0], [10.0, 20.0], [35.0, 10.0]], [[20.0, 30.0], [35.0, 35.0], [30.0, 20.0], [20.0, 30.0]]] }</pre>

Figure 2.7: Basic GeoJSON elements, retrieved from [43]

According to IMDF’s archive description found in [3], the representation of IMDF has three requirements in order to be considered valid:

- Datasets must be delivered as compressed `.zip` archives
- Archives must contain a Manifest object supplied in a dedicated `manifest.json` file
- Features must be packaged as homogeneous GeoJSON *FeatureCollections* in dedicated `.geojson` files where only the address and venue features are mandatory alongside the `manifest.json` file.

In order to provide a brief understanding of each feature, figure 2.8 illustrates the layer hierarchy of IMDF and table 2.4 shows the full list of *FeatureCollections* that are describe each important section of the indoor space.

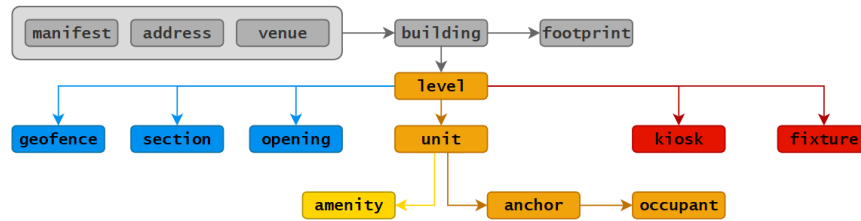


Figure 2.8: IMDF layer hierarchy

Feature	Presence	Description
manifest	Required	The Manifest is a container for metadata describing the content and origin of an archive
address	Required	An Address represents a postal address, of official or proprietary designation, associated with an element of a venue
venue	Required	A Venue models the presence, location and approximate extent of a place
amenity	Optional	An Amenity models the physical presence and approximate point location of a pedestrian amenity that serves a utilitarian purpose
building	Optional	A Building describes a physical building structure associated with a venue
detail	Optional	A Detail models the presence, location and extent of a physical object
fixture	Optional	A Fixture models the presence and approximate physical extent of a moveable or semi-permanent physical asset
footprint	Optional	A Footprint models the approximate physical extent of one or more referenced Buildings
kiosk	Optional	A Kiosk models the presence and physical extent of furniture that facilitates the distribution of products and/or services
level	Optional	A Level models the presence, location and approximate physical extent of a floor area
occupant	Optional	An Occupant models the presence and location (via Anchor) of a business entity that trades goods and/or services
opening	Optional	An Opening models the presence, location and physical extent (width) of an entrance
relationship	Optional	A Relationship models the physical or conceptual association between two map elements as an edge
section	Optional	A Section models the presence and approximate extent of a theme. The boundary of a theme may be a threshold or an indeterminate border
geofence	Optional	A Geofence models the extent of a geofence
unit	Optional	A Unit models the presence, location and approximate extent of a space

Table 2.4: List of `.geojson` feature files that constitute an IMDF archive

With the IMDF archive fully created, Apple’s proprietary IMDF Sandbox provides tools and mechanisms to visualize each of the described indoor sections as well as validate all of the GeoJSON files and the IMDF archive itself as a whole. Lastly, although Apple does not provide any public API for navigation with IMDF, there are a few commercialized third-parties that support navigation, such as MappedIn [24] and Esri ArcGis Indoors [16].

2.2.2 IndoorGML

IndoorGML is an Open Geospatial Consortium (OGC) standard, defined as an application schema of OGC Geographic Markup Language (GML), for an open data model and XML schema for indoor spatial information. IndoorGML aspires to implement a framework to intuitively represent and exchange indoor spatial data by providing means to locate stationary or mobile features in the indoor space and give spatial information services referring their positions in indoor space, rather of representing structure architectural factors [29]. The initial IndoorGML version aims to fulfil indoor navigation requirements due to strong and urgent standardization demands, such as indoor routing and location-based services, as well as emergency control applications.

IndoorGML defines the indoor space as a set of cells, each having its own unique Identifier (ID), where each cell may share boundaries with other cells, but cannot overlap them. Also, although the position of a cell in the cellular space can be specified by its ID, (x, y, z) coordinates are added to each cell in order to specify more precise locations.

Semantic representation is used in IndoorGML in order to provide cell classification and to determine the connectivity between cells, so that it is intuitively simple to detect and characterize the cells that are crucial for navigation purposes. In addition, IndoorGML depicts the topological structure of the indoor space as a Node-Relation Graph (NRG), which illustrates the connectivity and adjacency between indoor elements and, therefore, enables efficient implementation for complex computational problems within the fields of indoor navigation. Figure 2.9 shows an example of the topology of the IndoorGML space model.

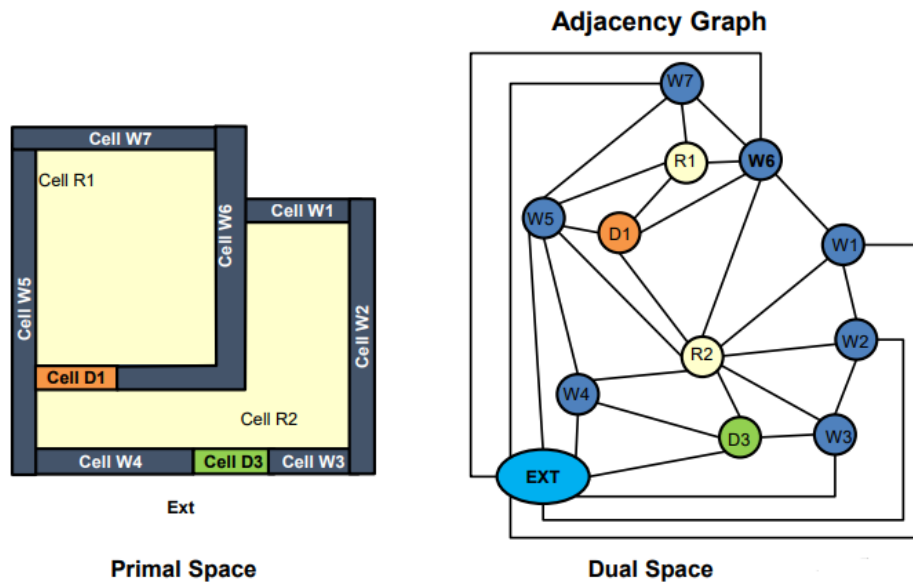


Figure 2.9: IndoorGML cellular space model topology, retrieved from [29]

IndoorGML also supports the representation of multiple layers, formally named as Multi-Layered representation, since a single indoor space can be semantically interpreted into different cellular spaces. As an example, figure 2.10 illustrates an indoor space that is constituted by

rooms and a corridor but is also represented by WiFi and Radio-Frequency Identification (**RFID**) sensor coverage areas. The Multi-Layered Representation (**MLR**) allows the decomposition of the pictured indoor space into different semantic interpretations and grants the possibility of creating inter-layer relations between these layers, formulating a Multi-Layered Space Model (**MLSM**), shown in figure 2.11. In this **MLSM**, the nodes can represent the either rooms and corridors (topographic space), WiFi sensors (WiFi sensor space) or RFID sensors (RFID sensor space) and the edges represent intra-layer transitions between these nodes.

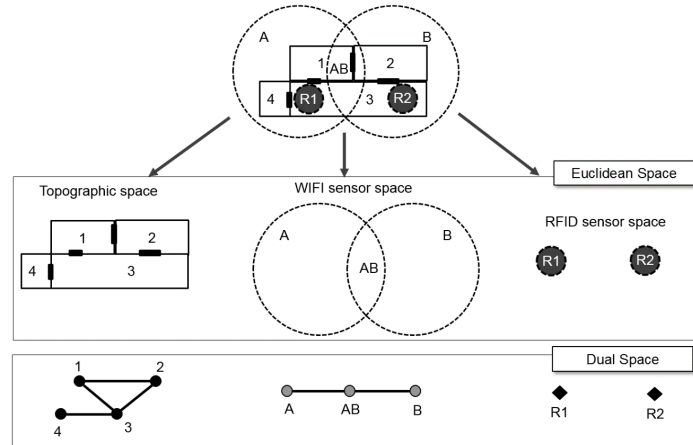


Figure 2.10: IndoorGML Multi-Layered Representation (**MLR**) [29]

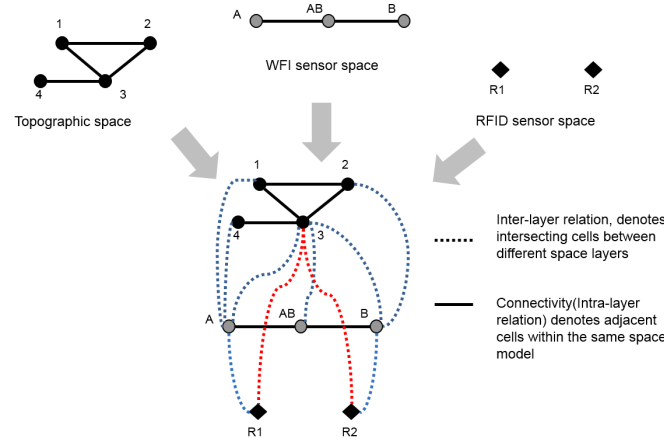


Figure 2.11: IndoorGML Multi-Layered Space Model (**MLSM**) [29]

Given that IndoorGML is based on XML, it is possible to validate the integrity of the IndoorGML files against the defined XML schema found in [29] with appropriate software tools such as *val3dity* [22]. Finally, although no current commercial indoor positioning system support IndoorGML, the team that is responsible for maintaining this standard, *STEMLab* from Pusan National University, provides open-source tools to create, visualize and validate IndoorGML files.

2.3 Indoor positioning systems

Global Positioning System (**GPS**) technology has achieved breakthrough advancements in outdoor localization and navigation, however, since it requires line-of-sight to a number of satellites, which is blocked by the walls and ceiling of buildings, it is rendered unsuitable for indoor use. In the subject of evacuation systems, indoor positioning is essential because it is the mechanism that allows the system to locate each person inside the building's indoor space and, thus, grant the possibility of applying search algorithms to calculate pathways for the evacuees to reach safe zones. There are several technologies that present different approaches to the process of indoor tracking, such as *dead reckoning*, *image processing*, *Li-Fi*, *ultrasounds*, etc. However, in this section we will mention only some of the most popular indoor positioning systems in order to provide a better understanding of each of the solutions and how they differ from each other.

2.3.1 RFID

Radio-Frequency Identification (**RFID**) is a form of wireless communication that uses electromagnetic radio-frequency waves in order to identify objects or living beings. Usually, RFID systems make use of RFID readers, tags and antennas, making them similar to barcoding but with major advantages, such as being able to read hundreds of tags within seconds and not needing to be within direct line-of-sight [23].

In an RFID system, the reader continuously sends radio waves of a specific frequency. If an RFID tag is within the range of the radio waves, it transmits feedback to the RFID reader, allowing it to identify the object or user. Distinct RFID tags have different applications and frequencies but can generally be categorized as *passive* or *active*. Passive tags can operate without a battery and are less expensive, but have a smaller range (up to 15 meters), whereas active tags need a dedicated power source to operate and are more expensive, but exhibit a significantly longer range (up to 300 meters) [4].

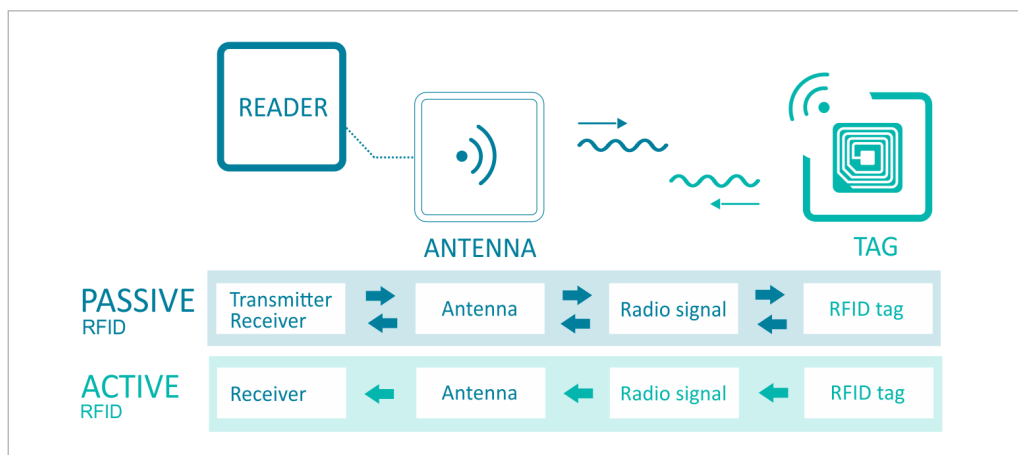


Figure 2.12: RFID Architecture, retrieved from [4]

2.3.2 Bluetooth Low Energy

Bluetooth Low Energy (**BLE**) is a radio-frequency wireless communication technology that allows the detection and localization of wireless devices, such as tracking tags or smartphones, within an indoor space. BLE indoor positioning uses BLE *beacons* that are deployed in fixed positions of an indoor space to transmit regularly bluetooth low energy pulses containing the *beacon*'s unique identifier (ID). The device recognizes the wave signal from the *beacon* and is able to create a rough estimation of the distance between itself and the *beacon*. If several *beacons* are placed in key areas of an indoor space, communication between three or more *beacons* and a wireless device can be used to position the device via Received Signal Strength Indicator (**RSSI**) multilateration [17]. With the introduction of Bluetooth 5.1, it is also possible to measure the Angle of Arrival (**AoA**) of the signal, which enables the use of triangulation in order to pinpoint locations [28].



Figure 2.13: BLE indoor tracking through RSSI [5]

2.3.3 Wi-Fi

Wi-Fi is a radio-frequency wireless communication technology that can also be used in indoor positioning systems similarly to BLE *beacons*. Wi-Fi positioning systems operate through pre-existing access points (AP) in order to detect and pinpoint transmitting Wi-Fi devices within indoor areas. Indoor positioning systems based on Wi-Fi are able to use several distinct positioning techniques that will not be explained in-depth since they are not in the scope of this thesis. However, similarly to BLE, most of the techniques are based on Received Signal Strength Indicator (**RSSI**) methods, such as RSSI multilateration, RSSI fingerprinting, Time-of-Flight (**ToF**) or Angle of Arrival (**AoA**) [19].

Although Bluetooth Low Energy (**BLE**) can achieve higher degrees of location precision and requires significantly less power, Wi-Fi can transmit over longer ranges and many buildings have already pre-existing Wi-Fi infrastructures that can also be utilized for indoor positioning purposes.

2.3.4 Ultra-Wideband

Finally, the last indoor positioning system that will be mentioned in this section is Ultra-Wideband (**UWB**). UWB is a short-range radio-frequency wireless communication technology that has unmatched precision for locating devices in indoor spaces. It consists of fixing UWB anchors in indoor corners and attaching a UWB tag to the device or asset that needs to be located. The ultra-wideband tag transmits radio pulses frequently, allowing the anchors to measure the distance by calculating the time taken for the signals to travel between devices and, therefore, pinpoint the position with high levels of accuracy [18]. Figure 2.14 shows a comparison between previous positioning system's frequencies and spectral density.

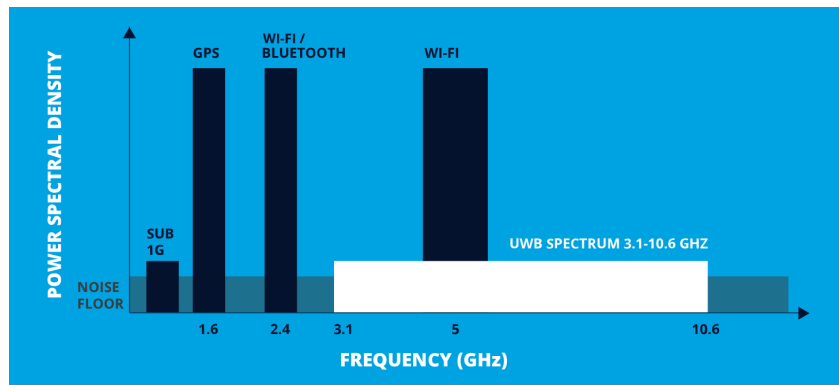


Figure 2.14: Comparison between UWB and other indoor positioning solutions's frequencies [18]

2.3.5 Summary

Table 2.5 makes a brief side-by-side comparison between each of the solutions mentioned in this chapter. In general, all of the solutions previously mentioned are viable and offer means for indoor positioning. Radio-Frequency Identification (**RFID**) is an older technology with a small range, however, it has advantages such as not needing direct line-of-sight. Bluetooth Low Energy (**BLE**) and Wi-Fi operate in similar way, but with BLE having lower range, higher accuracy and added infrastructure costs, since most buildings already have pre-installed Access Point (**AP**)'s. Lastly, Ultra-Wideband (**UWB**) has the most accuracy while also being able to maintain low values latency, which makes it a great option for real-time navigation purposes.

	RFID	UWB	BLE	Wi-Fi
Accuracy	< 10cm	10-50cm	< 5m	< 10m
Range	up to 3m	up to 150m	up to 75m	up to 150m
Latency	1 second	< 1ms	3-5 seconds	3-5 seconds
Data rate	4-8kbps	up to 27Mbps	up to 2Mbps	up to 1 Gbps
Power consumption	Low	Low	Low	High

Table 2.5: Comparison between indoor positioning systems

2.4 Search algorithms

To close this chapter, it is important to mention one last key element of an indoor evacuation system, which is the choice of a search algorithm for calculating appropriate evacuation paths. When the environmental conditions are normal and there is no hazard detected, unless an evacuee is physically impaired, the safest route to the exit is usually the shortest route. However, during an emergency situation, this might not always be the case. If the shortest path coincides with a hazard epicenter, for example a fire source, the evacuee cannot travel through that pathway. As such, the evacuation algorithm, through sensor monitorization, must account for environmental variables and guide each evacuee through a safer route towards the exit.

As stated in section 2.2, most building representations are based on a graph, meaning that each node in the graph usually represents possible locations an evacuee can be in, and the edges between these nodes represent direct connections from one location to another. Nevertheless, before delving any further, it is important to note that besides nodes and edges, a graph also has other characterizations that need to match the area it is representing. For example, graphs can be either directed or undirected. An undirected graph is one where edges are bidirectional, whereas a directed graph may have a direct connection between node A to node B but not from B to A . Graphs can also be unweighted or weighted, in the sense that each transition from A to B can have an associated cost. This cost depends on the use case of the application, but in emergency systems, these values are usually related to, for example, the time it takes to travel from A to B , the distance between A and B , the danger level of A to B or even a more complex formula that combines some these metrics to define a safety value of an edge.

This section will mention some of the most common algorithms that are able to solve the shortest path problem [45], while also taking into consideration the possible different metrics that can arise in emergency scenarios, such as temperatures, distances or danger levels.

2.4.1 Dijkstra's algorithm

Dijkstra's algorithm is an algorithm developed by Edsger W. Dijkstra for finding the shortest path between nodes in a graph. Given a source node, the algorithm utilizes the weights of each edge in the graph to compute the path that minimizes the total weight sum between the source node and all of the other nodes [33]. In short, Dijkstra's algorithm begins in the source node and determines the shortest path between the origin and all of the other nodes the network. Then, for every visited node, it calculates the shortest path from itself to the source and sets the node as visited, while keeping track of the currently recognized shortest paths from all nodes to the source, updating them if a better route is found and repeating this process until all the nodes are visited.

2.4.2 Bellman-Ford algorithm

Similarly to Dijkstra's, the Bellman-Ford algorithm is also used to tackle the single-source shortest path problem. However, unlike Dijkstra, the Bellman-Ford algorithm is also capable of handling negative weights in graph edges, whereas Dijkstra's algorithm can only handle positive weight values [41]. In short, the Bellman-Ford algorithm works by overestimating the length of

the path from the source to all of the other nodes in the graph and then iteratively "relaxing" those estimates by finding alternative paths that are shorter than the previously overestimated routes. Though if no negative edges are present in the graph, the Dijkstra algorithm manages to keep a better time complexity with the use of heap priority queues, this method allows the Bellman-Ford algorithm to be applied to a wider range of input scenarios than Dijkstra's.

2.4.3 Floyd-Warshall algorithm

Contrasting from the previous search algorithms, the Floyd-Warshall algorithm is used to solve the all-pairs shortest paths problem, meaning it is able to find the shortest path between every pair of nodes x and y in the graph [34]. This algorithm works for both directed and undirected graphs, however it does not work on graphs that possess negative weight cycles. In summary, the Floyd-Warshall starts by creating a matrix containing the weight values of each edge in the graph and states that any given path $\mathbf{X} \rightarrow \mathbf{Z}$ is not optimal if there exists a node $\mathbf{Y}$ such that the path $\mathbf{X} \rightarrow \mathbf{Y} \rightarrow \mathbf{Z}$ is shorter. In order to ensure this, it then tries to recursively break down any path into sub-paths to find the shortest route between a pair of nodes.

2.4.4 A* algorithm

The A*, or A-star, algorithm is considered an extension of the Dijkstra's algorithm, and is widely used for solving the single-pair shortest path problem by means of utilizing specific heuristics in order to make a more efficient search. The heuristic introduced by this algorithm essentially plans each step ahead so more a more optimal decision can be made [12]. Similar to Dijkstra's, the A* algorithm creates a shortest-path tree from the source node to all of the other nodes, however, for each node, this algorithm calculates an estimate of the total sum of weights of the path through that node, according to the formula $f(n) = g(n) + h(n)$, where $f(n)$ is the total estimated weight of the path through node n ; $g(n)$ is cost to reach n , so far; and $h(n)$ is the estimated cost from n to the end node.

2.4.5 Complexity comparison

Table 2.6 shows a brief comparison in complexity of the algorithms mentioned above. The Dijkstra and Bellman-Ford algorithms are very similar, however only the latter can support negative weight values, allowing it to be applied in a larger range of cases. Nevertheless, with an implementation using priority queues, Dijkstra can achieve a better time complexity compared to Bellman-Ford. The Floyd-Warshall algorithm is a simple and versatile dynamic programming algorithm that can calculate the shortest path between a pair of nodes in the graph, however, since it uses a three-cycle implementation, it becomes asymptotically worse than the previous algorithms. Finally, the A* algorithm is also based on Dijkstra and it can be used to solve complex search problems while ensuring that an optimal solution is found. Its complexity is, however, highly dependent on the heuristic used for the decision process.

Dijkstra	Bellman-Ford	Floyd-Warshall	A* (A-star)
$O(E * \log(V))$	$O(V * E)$	$O(V^3)$	heuristic dependant

Table 2.6: Algorithm complexities. V = number of vertices and E = number of edges

Chapter 3

Related Work

In the previous chapter, key elements of evacuation systems were mentioned, such as the system's architecture, suitable indoor mapping standards, indoor positioning systems and search algorithms that enable safest path calculation, in order for providing a basis for a discussion of the main important aspects constituting a sound indoor evacuation system. This chapter will present the state of the art in this field, discussing some of the more relevant previous research studies and analyzing the different metrics and methodologies between these works and the divergences between each other.

3.1 Time-based graphs approach

One of the main reference research studies, [25], proposed by Barnes and Leather, consists of a proposal for a distributed evacuation system using Wireless Sensor Networks. This approach uses evacuee movement times and hazard propagation times as metrics for calculating evacuation paths for an emergency exit.

3.1.1 Building representation

In this solution, the building's environment is represented by two weighted graphs. A 'hazard graph' with information related to the sensors location along the interior of the building, and a 'navigation graph' that illustrates the complete network that contains all possible routes of pedestrian navigation within the building's internal structure.

3.1.2 Hazard graph

The vertices or nodes of the 'hazard graph' are connected by an edge to represent the cases where the hazard can proliferate directly from one node to another. These links have an associated weight that represents an estimate of the fastest time for the hazard to travel the link distance between the two sensors, in other words, the worst case scenario. The edges can also be directed, implying that, for example, the spread rate of a fire in a given direction can be different from

the spread rate in the opposite direction, a possible instance of this would be on long staircases where flames can travel upstairs faster than downstairs.

3.1.3 Navigation graph

The set of nodes of the 'navigation graph' correspond to the same sensor locations represented by the 'hazard graph', however, the weight of links between these vertices in the 'navigation graph' are now estimates of the time taken by evacuees to travel from one node to another. Similarly to the 'hazard graph', the edges on the 'navigation graph' can also be directed since, for example, one-way escalators allow for only one navigable route, up or down, and the expected time navigating upstairs as opposed to downstairs can be significantly different, therefore, the weights of the links will diverge.

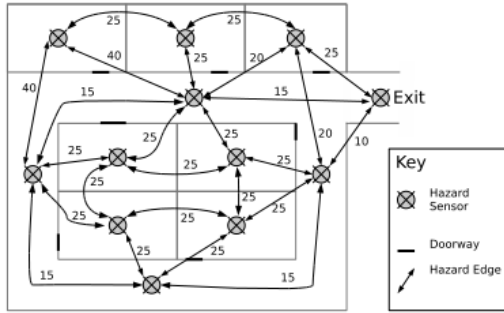


Figure 3.1: Example hazard graph [25]

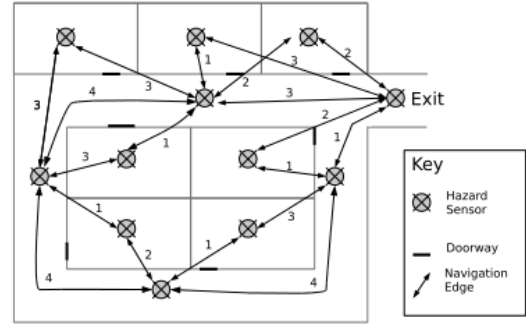


Figure 3.2: Example navigation graph [25]

3.1.4 Sensor exposure time

In [25], in order to calculate the estimated time for the location of sensor x to be exposed to the hazard, the following formula should be applied.

$$H_x = \min_{p \in P, p = \langle p_1, p_2, \dots, p_n = x \rangle} \left(\sum_{i=1}^{n-1} F_{p_i, p_{i+1}} \right)$$

Where P is the set of all possible paths beginning in one node of the network and ending in x , and $F_{p_i, p_{i+1}}$ is the amount of time it takes for an hazard to spread from p_i to p_{i+1} . This allows the usage of a breadth-first search algorithm to trivially calculate the hazard times.

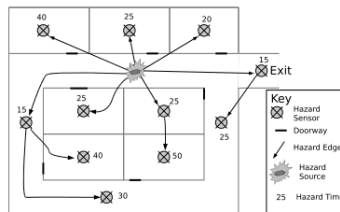


Figure 3.3: Calculated hazard times [25]

3.1.5 Safest path calculation

In regards to the search of the safest path, the approach proposed by [25] is so that the safety of any route to an emergency exit node is the minimum of the differences in time between the hazard reaching each node on that route and an evacuee starting at the beginning of the route reaching those nodes. This means that the safety metric is the amount of time an evacuee has to reach each node before it becomes exposed to the hazard. Formally, given that the safety of an empty path on a emergency exit is equal to the time it takes for the hazard to reach that specific exit, we have

$$S(< v_{exit} >) = F_{v_{exit}}$$

So for a given route p where

$$p = < v_i, v_{i+1}, \dots, v_{exit} >$$

And adding a new vertex at the beginning, results in the route

$$p' = < v_{i-1}, v_i, v_{i+1}, \dots, v_{exit} >$$

The formula for the calculation of the safety of the path, proposed in [25], is

$$S(p') = \min_{S(p) - R_{v_{i-1}, v_i}, H_{v_{i-1}}}$$

Where R_{v_{i-1}, v_i} represents the time for a human to travel from v_{i-1} to v_i .

The first term indicates that any delay of an evacuation process makes a pathway less safe by that exact amount of time delay and the second term represents the understanding that regardless of how safe an evacuation pathway is, the hazard can always reach the evacuee first.

3.1.6 Network propagation and execution process

Finally, as for the implementation of said formulas, the idea is so that both algorithms for the propagation of the hazard times and the safety times run simultaneously, following the event of an emergency. This means that both hazard and safety times are updated continuously as soon as new information is collected. Every node stores the identifiers of all adjacent nodes to which it can lead evacuees, and stores the safety of each node's exit path.

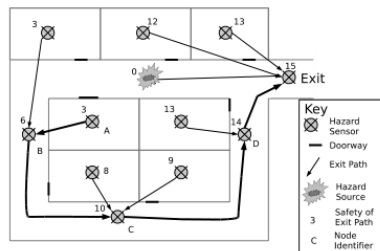


Figure 3.4: Example safety path calculated using the approach proposed by [25]

3.2 Distance and hazard intensity-based graphs approach

Another research study that highly impacted this thesis is presented by Filippoupolitis and Gelenbe in [11]. The authors proposed a distributed decision support system to calculate the best possible evacuation routes in real time, during the spreading of an active hazard, while using the distance and the intensity of an hazard as metrics to compute possible evacuation paths.

3.2.1 Network structure

The system itself has a network of decision and sensor nodes, that are located in explicit locations of the building. The role of the decision nodes is to compute, recommend and broadcast to the evacuees the following path to follow in order to reach a safe zone as soon as possible.

3.2.2 Assumptions

One of the main assumptions stated by [11] is that the building layout is already known and that the decision support network is already pre-installed and deployed in the buildings internal structure. This is crucial because the computation of the safest path relies on the information broadcasted by the decision nodes.

Another assumption is the existence of a set of decision nodes also pre-installed in key locations of the building. The role of these nodes, according to [11], is to calculate the best route that the evacuee should follow in order to reach the exit. The synergy between these nodes can be done via a wireless communication device or via a visual indicator.

The last supposition stated by Filippoupolitis and Gelenbe is the existence of a sensor graph that will provide to the pre-installed decision nodes temperature or smoke density readings that can be found in specific locations of the building.

3.2.3 Building representation

In the solution proposed by [11], a known building layout is used in order to create a graph to representation of it, as shown in the image below. The vertices of the graph correlate to possible pedestrian locations inside the building's indoor space and the links representing direct navigable connections between these vertices.

In this navigable graph, there are two values that are crucial for the computation of the actual weight of each link. The first one is the navigable pedestrian distance between a pair of nodes, represented by $l(i, j)$ in the paper, and the second one is the hazard intensity, represented by $H(i, j)$, which is given by the sensor that rules the transition between nodes i and j (shown in image 3.6). Initially the hazard intensity is 1 and it will increase as the hazard progresses throughout the building, or, in other words, the closer the hazard is to a node, the higher the hazard intensity value will be.

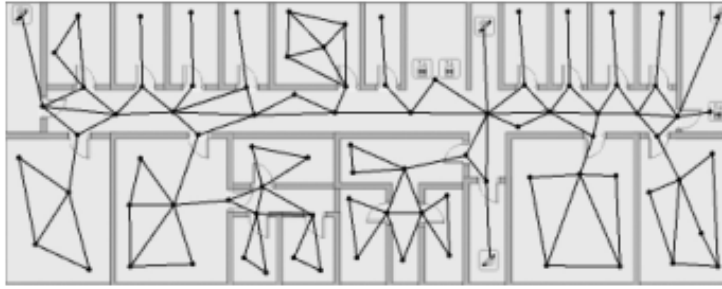


Figure 3.5: Building graph, retrieved from [11]

With these two functions, the formula proposed by [11] to represent the actual weight (referred to as 'effective length' in the paper) associated with each link between two decision nodes is

$$L(i, j) = l(i, j) * H(i, j)$$

This means that the higher the value of $L(i, j)$ the more dangerous it is for an evacuee to travel between node i and node j .

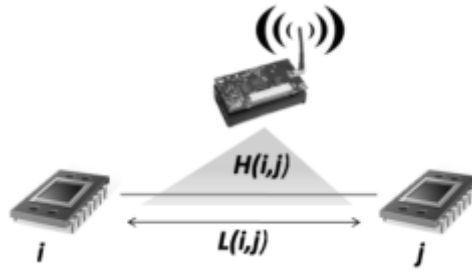


Figure 3.6: Example of a link between two decision nodes, ruled by one sensor [11]

Figure 3.7 shows an example graph with sensors and nodes stationed in key points in the building. Since it is often impractical to have one sensor monitoring each link between two nodes, some sensors can be in charge of multiple adjacent areas of the building.



Figure 3.7: Example graph of sensors and decision nodes, retrieved from [11]

3.2.4 Distributed architecture

The architecture proposed by the authors is a distributed one, which means that there is no central unit that processes every decision possible in order to find the safest path available. Instead, every decision node stores enough information in order to direct the user to the next best decision node in the route to the exit. So, for a decision node x , the information stored is:

- The actual weight ('effective length' - L) of each transition of x
- The effective length of all neighbouring transitions of x to an exit e
- The effective length of the shortest path from x to an exit e
- The recommendation for the next best decision node

3.2.5 Safest path computation

As stated before, in the absence of an hazard, the hazard intensity of a link between two nodes i and j is equal to 1 ($H(i, j) = 1$), which means that, in normal environmental conditions, the effective length of a transition between two nodes is equal to the physical distance between those locations ($l(i, j)$). And since the decision node network is pre-installed before any hazard event occurs, each decision node will already have stored the information of the shortest path for an available exit.

In the event of a hazard, the algorithm proposed and explained in depth by [11] will be executed periodically at each decision node, who will send to every neighbour the effective length of the path from itself to the exit. It will update the effective length of shortest path to the exit and suggest the next decision node that minimises that effective length in order to find the route with the total minimum sum of all its links actual weight, given that each transition weight is represented by $L(i, j) = l(i, j) * H(i, j)$.

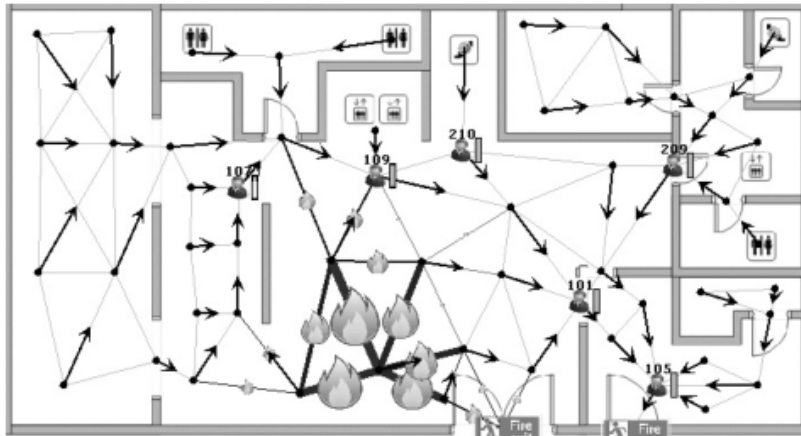


Figure 3.8: Example safest path recommendation, retrieved from [11]

3.3 Risk level-based graph approach

An additional research study that also highly impacted this dissertation is presented by Kamoun, Barachi, Belqasmi and Hachani in [9]. In this paper, the authors proposed a real-time dynamic evacuation system that accounts for updates in risk levels associated with specific building sections based on distance, heat, toxic gases and density.

3.3.1 System structure

The system proposed by [9], consists of a few key ideas, some of which have already been mentioned in one of, if not both, previous research works discussed above, however, it also presents a few new additions. The list below compiles the key points of the solution proposed in [9].

- A graph model representation of the building, with nodes serving as possible evacuee locations and links representing navigable pathways between nodes.
- An algorithm to determine a safe escape route, without relying only on distance or time until reaching the exit. Instead, the risk level of each section after the start of the hazard is periodically measured and the safest path that minimises the exposure to higher levels of risk recalculated.
- Addition of different metrics of risk level besides environment temperature, such as gas exposure and overcrowding.
- A mechanism that updates the safest path dynamically throughout the spreading of the hazard.
- Use of visual directional indicators located in key areas of the building.

3.3.2 Assumptions

Similarly to the previous research works, the solution proposed by the authors also mention some assumptions before the implementation phase, such as:

- The building floorplan and the locations of sensors/visual directional indicators are known beforehand.
- Each sensor is capable of monitoring temperature, carbon monoxide, hydrogen cyanide and crowd density levels and communicates with the central server through a base station.
- The server receives data from the base stations, computes the safest path to the exit and transmits the suggested direction to the corresponding visual direction indicator.

3.3.3 Building graph representation

Same as previous works, in the graph representation of the building, each of the nodes in the represent possible locations for an evacuee to follow in order to reach the exit, however, contrary to previous works, the location of these nodes is not conditioned by the location of the sensors. The figures below illustrate an example of a exhibition center floorplan along with its respective graph representation.

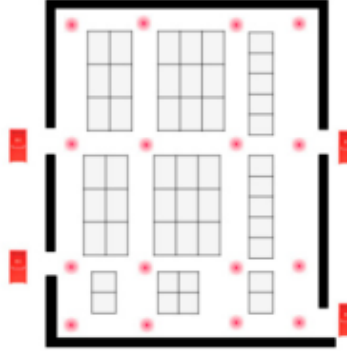


Figure 3.9: Example of a building floorplan, retrieved from [9]

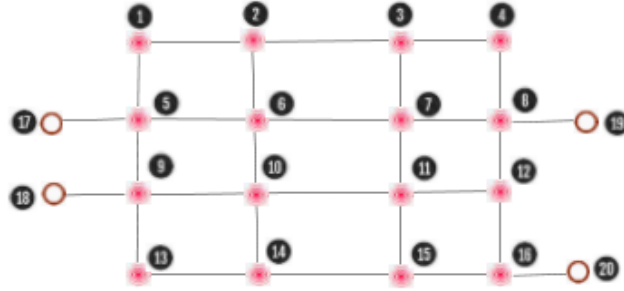


Figure 3.10: Graph representation of the building floorplan retrieved from [9]

However, in this approach, the formula that calculates a given weight or cost of a transition between two nodes, i and j , or formally, $W_t(i, j)$, is:

$$W_t(i, j) = a_1 L(i, j) + a_2 P_T(i, j) + a_3 P_{CO}(i, j) + a_4 P_{HCN}(i, j) + a_5 P_D(i, j)$$

Where:

- $L(i, j) \in [0, 1]$ represents the relative length of the segment in relation to the longest segment in the graph
- $a_i \in [0, 1]$ is a coefficient that represents the relevance of the associated variable in the decision making computation
- $P_X \in [0, 1]$ is the risk index associated with the corresponding variable

According to the authors, the values of the a_i coefficients are based on fire and rescue data, as well as expert reports. The stated values given to each of the coefficients are as follows:

$$a_1 = 0.05 \mid a_2 = 0.19 \mid a_3 = 0.28 \mid a_4 = 0.38 \mid a_5 = 0.10$$

Furthermore, the paper provides table mappings of each sensor reported variable values and their respective risk classifications and indexes, shown below. More detailed explanation of each table in [9].

T - Temperature (°C)	Risk classification	Risk index
$T < 48$	Low	0
$48 \leq T < 50$	Medium	0.5
$50 \leq T < 150$	High	0.7
$T \geq 150$	Very high	1

Table 3.1: Temperature index values [9]

CO - Carbon monoxide (ppm)	Risk classification	Risk index
$CO < 150$	Low	0
$150 \leq CO < 420$	Medium	0.5
$420 \leq CO < 600$	High	0.7
$CO \geq 600$	Very high	1

Table 3.2: Carbon monoxide index values [9]

HCN - Hydrogen cyanide (ppm)	Risk classification	Risk index
$HCN < 36$	Low	0
$36 \leq HCN < 108$	Medium	0.5
$108 \leq HCN < 135$	High	0.7
$HCN \geq 135$	Very high	1

Table 3.3: Carbon monoxide index values [9]

D - Density (person/m^2)	Risk classification	Risk index
$D < 4$	Low	0
$4 \leq D < 5$	Medium	0.5
$5 \leq D < 7$	High	0.7
$D \geq 7$	Very high	1

Table 3.4: Density index values [9]

3.3.4 Safest path computation

The algorithm proposed by [9] is one that allows for the minimisation of total exposure time to all possible hazard variables, which means choosing the path with the minimum sum of all its link weights, given by the formula of $W(i, j)$. So, given a weighted graph representation of a building, with n possible exits, a naive solution would be to simply apply a Dijkstra algorithm to find the shortest path from the node where the person is located to one of the exits. This approach would work, however, in the worst case, the algorithm would have an $O(n)$ complexity since we would need to run Dijkstra n -times, one for each exit. In order to improve the complexity, it's possible to transform the graph by simply adding a hidden dummy node D that is linked to each of the emergency exits. This will allow for only one execution of Dijkstra with D as the source node in order to get the path from D to the node where the evacuee is located. In their case, the transitions do not need to be reversed because they are always bidirectional.

Finally, one other imposed constraint that can be found in [9] is the minimum safety of a link, which simply allows for the *pruning* of a transition that has an associated risk index of value 1. This grants a way to skip the computation of the shortest path algorithm in edges that are completely unsafe for evacuees to navigate through. Figure 3.11 shows an illustration of the graph transformation mentioned above. Algorithm 1 and 2 show the pseudo-code proposed by the authors for the periodic search of safer paths.

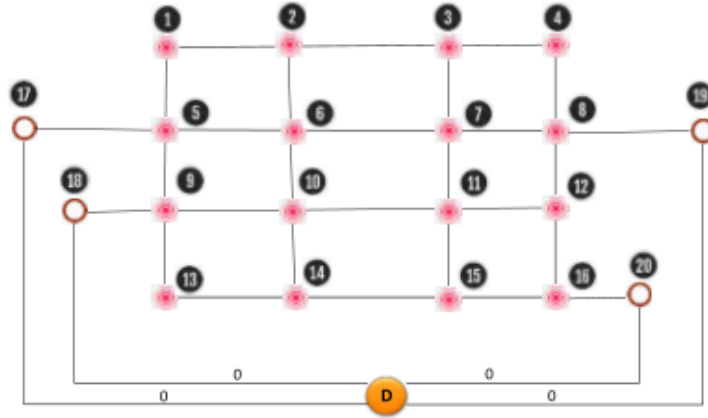


Figure 3.11: Graph transformation, retrieved from [9]

Algorithm 1 CSP (Constrained Shortest Path), retrieved from [9]

Require: Input graph $G'(N', E')$ with weights $W_t(i, j)$

- 1: Prune links that don't meet safety constraint
 - 2: Set vertex D as source vertex
 - 3: Apply Dijkstra
 - 4: Find shortest distance from D as source node
 - 5: Output shortest path from each regular node to the closest exit node
-

Algorithm 2 Evacuation, retrieved from [9]

Require: Input graph $G'(N', E')$

- 1: Collect sensor data
 - 2: Calculate the weight $W(i, j)$ for each link
 - 3: Apply **CSP Algorithm**
 - 4: **if** new routes are found **then**
 - 5: Send directions to visual direction indicators
 - 6: **end if**
 - 7: Wait for t seconds and go to 1
-

3.3.5 Evacuation process

In the event of a hazard detection inside the building's indoor space, the central server demands more frequent reporting of information readings from all available sensors in order to have the the latest updated values of each reading variable. After gathering the monitored information of all sensors, the centralized server calculates the weight formula $W(i, j)$ for every pair of linked nodes i and j , executes the CSP algorithm mentioned above and transmits the updated direction suggestions to the visual directional indicators. Finally, to make sure the evacuation process flows smoothly, every t seconds, the *Evacuation* algorithm (algorithm 2) runs again in order to bypass possible crowd bottlenecks, given by the density readings. This will clear out all blockages in overcrowded sections of the building, consequently reducing the total evacuation time.

3.4 SmartEscape

To finalize this related work chapter, it is also worth to mention perhaps one of the most complete smart fire evacuation systems, developed by members of the Department of Computer Engineering in the Karabuk University in Turkey, *SmartEscape* [40]. Although slightly different than the previous works mentioned in previous sections, this solution still touches some of the same bases of the above mentioned research studies, while providing a different evacuation approach.

Similarly to the previous solutions, *SmartEscape* is a real-time, dynamic evacuation system that collects environment information via sensor monitoring. However, it also takes into account individual features from each user in order to evaluate the potential risk before choosing the appropriate evacuation route (further explained below). *SmartEscape* uses an artificial neural network (ANN) to calculate the risk of each edge inside the building's navigable path for every user and compute the best possible route under hazardous conditions. Each user's location is monitored by RFID (Radio-Frequency Identification) and is sent to the ANN in order to calculate the safest path. The system makes use of visual and vocal suggestions through a developed smartphone app to guide it's users throughout the whole evacuation process.

3.4.1 System architecture

The architecture of *SmartEscape* is shown in figure 3.12. It consists of four main components: the positioning system, the sensor network, the mobile app GUI and the intelligent routing engine. It is assumed that the RFID system transmits periodical locations of each evacuee and the sensor network is already pre-installed in the building. The intelligent routing engine is responsible for receiving the sensor and user individual data to feed the ANN in order to calculate the risk level of every navigable transition, calculate the safest route for the user and send the appropriate directions to his smartphone.

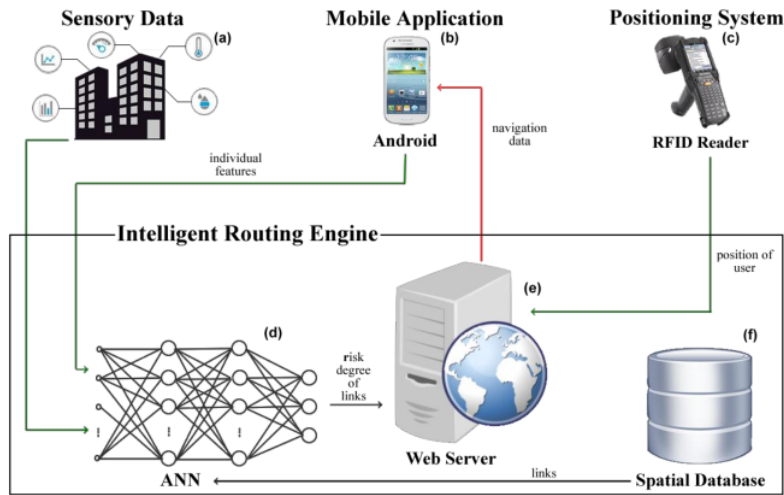


Figure 3.12: SmartEscape architecture, retrieved from [40]

3.4.2 Mobile app

One of the aspects that makes the *SmartEscape* solution unique is the user interface in the mobile application that allows each evacuee to fill out a form (shown in figure 3.13) that provides the routing engine the necessary information about the user's physical condition in order to calculate the best path for that person to follow. For example, an evacuee that has respiratory diseases must avoid navigating through smoke at all costs, whereas someone without any respiratory disease could probably manage to navigate somewhat safely through a sparsely smoked areas.

In the event of a hazard, the Intelligent Routing Engine will calculate the safest path for the user, keeping in mind his/her individual characteristics, and provide the correct navigation suggestions to the best exit. Figure 3.14 shows an example output of an evacuee during an evacuation scenario.

The figure displays two screenshots of the SmartEscape mobile app user form. The left screenshot shows the main form with the following fields and options:

- Age:** Select age range
- Gender:** Select gender
- Body Structure:** Select your body type
- Health Problem:** Do you have a heart problem? ☐
- Joint-Muscle Problem:** Do you have a joint-muscle problem? ☒
- Respiratory Disease:** Do you have a respiratory disease? ☐
- Physical Disability:** Do you have physical disability? ☐
- Fire Protection Wear:** Do you have fire protection wear? ☐

The right screenshot shows a modal dialog for **Body Structure** with the following options:

- ☐ Athletic
- ☒ Normal
- ☐ Fat

A **CANCEL** button is visible at the bottom right of the modal.

Figure 3.13: SmartEscape mobile app user form, retrieved from [40]

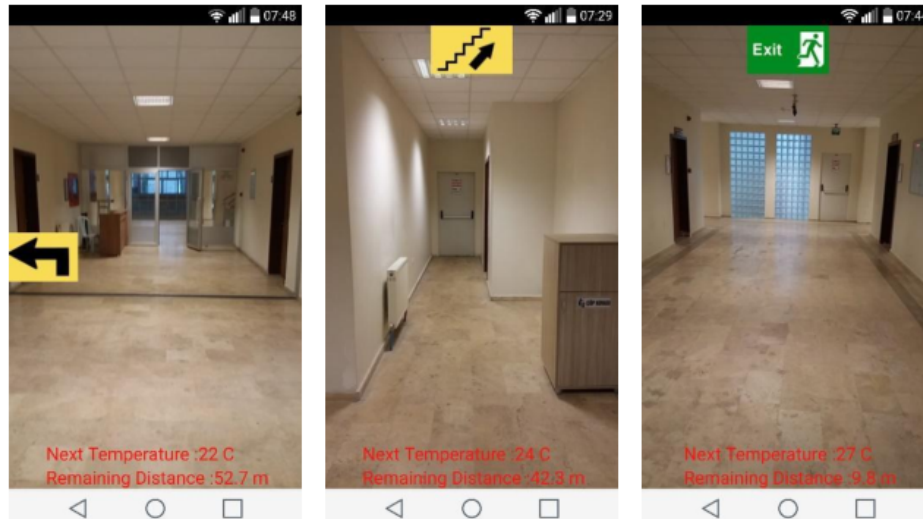


Figure 3.14: Evacuee's smartphone screenshots taken during live evacuation test. [40]

3.4.3 Intelligent Routing Engine

The routing engine is responsible for collecting the data from sensors located inside the building as well as each user individual features. According to [40], it uses an Oracle Network (ON) model that presents many methods to analyze and process spatial data, such as shortest path and nearest neighbors, while also keeping a persistent data model saved in the database. The Oracle Network model provides volatile model copies for each evacuee that has filled the mobile app form, allowing all the navigation services to use the correct user copy in order to run the evacuation process.

3.4.3.1 Risk factors

As is thoroughly explained in the paper, there are several input factors that highly impact the routing process of the intelligent routing engine. To keep the explanation brief, figure 3.15 illustrates each risk factor and level of the dataset, with 5 indicating the highest level of risk and 1 indicating the lowest value of risk.

No	Group	Factor Name	Real Values (Transformed Risk Values)
1	Environmental Factors	Temperature °C	0–45 (1), 45–75 (2), 75–150 (3), over 150 (4)
2		Fire Growth Rate (kW/s ²)	0 (1), 0.0001–0.0029 (2), 0.0029–0.012 (3), 0.012–0.047 (4), over 0.047 (5)
3		Visibility (meter)	0–10 (3), 10–30 (2), over 30 (1)
4		Carbon monoxide concentration (ppm—particle per million)	0–50 (1), 50–100 (2), 100–3200 (3), 3200–12,600 (4), over 12600 (5)
5		Human density (human/m ²)	0–0.8 (1), 0.8–1.8 (2), 1.8–4 (3), over 4 (4)
6		Alternative Ramp for Disabled People	No (2)/Yes (1)
7		Link Type	Corridor (1)—Stair (2)—Elevator (3)
8		Link Length (meter)	0–10 (1), 10–30 (2), over 30 (3)
9	Individual Factors	Age	18–40 (1), 40–60 (2), over 60 (3)
10		Gender	Male (1)/Female (2)
11		Body Type	Athletic (1), Normal (2), Fat (3)
12		Hearth Disease	No (1)/Yes (2)
13		Respiratory Disease	No (1)/Yes (2)
14		Joint-Muscle Disease	No (1)/Yes (2)
15		Physical Disability	No (1)/Yes (2)
16		Familiarity with Building Geometry	No (2)/Yes (1)
17		Fire Protection Wear	No (2)/Yes (1)
18		Gas Mask	No (2)/Yes (1)

Figure 3.15: Risk factors used for the Artificial Neural Network model, retrieved from [40]

However, if an evacuee is physically handicapped, the *SmartEscape* solution adopts other risk factors in order to circumvent possible pathway issues. For example, if an elderly evacuee is physically disabled and the building doesn't have support ramps, staircases are evaluated as a riskier option for that person to follow. Figure 3.16 depicts these additional risk values to help the decision process during such scenarios.

Physical Disability	Alternative Ramp	Link Type	Transformed Risk Values
No	No	Stair	1
No	No	Corridor	1
No	No	Elevator	1
No	Yes	Stair	1
No	Yes	Corridor	1
No	Yes	Elevator	1
Yes	No	Stair	4
Yes	No	Corridor	2
Yes	Yes	Stair	3
Yes	Yes	Corridor	2

Figure 3.16: Risk factors considerations in case of physical disabilities, retrieved from [40]

To correctly calculate the risk score of each link, all of the factors named in the figures above should have a defined order of priority. In order to accomplish this, the authors interviewed 78 people in order to determine the values of each risk factor for every single user surveyed, where each person had to sort the factors in descending order of importance by giving points to each of them from 16 to 1. After that, a *PenaltyWeight(PW)* of each factor is calculated by adding all the points of that factor from every survey collected, which resulted in the values shown in figure 3.17.

Order	Parameter	Penalty Weight
1	Physical Disability	1061
2	Respiratory Disease	921
3	Visibility	893
4	Fire Growth Rate	851
5	Temperature	811
6	Gas Mask	773
7	Carbon monoxide Concentration	766
8	Fire Protection Wear	750
9	Human Density	667
10	Link Length	640
11	Familiarity with Building	586
12	Geometry	517
13	Joint–Muscle Disease	517
14	Body Type	366
15	Age	272
16	Heart Disease	258
	Gender	124

Figure 3.17: Risk factor penalty results, retrieved from [40]

Subsequently, the next step of the ANN model is to calculate the total risk score for the user, which given by the weighted sum of risk values based on the following formula:

$$TotalRiskScore = \sum_{i=1}^{16} (PW_i * TRV_i)$$

where PW is the penalty weight illustrated in the table above and TRV is the transformed risk value of the i^{th} factor that is listed in table 3.15. This grants the splitting of data into five distinct risk groups, based on their total risk score, shown in the image 3.18.

Total Risk Score	Risk Group
$\leq 12,000$	VERY LOW
$>12,000$ AND $\leq 13,000$	LOW
$>13,000$ AND $\leq 14,000$	MEDIUM
$>14,000$ AND $\leq 15,000$	HIGH
$>15,000$	VERY HIGH

Figure 3.18: Risk groups, retrieved from [40]

3.4.3.2 Artificial Neural Network model

According to [40], the proposed ANN model has 16 inputs, listed in figure 3.15, which include personal and environmental factors that will result into 5 distinct risk group output categories. Since the study of the behaviour of a neural network is a complex subject which is not covered in the scope of this thesis, it is highly recommended to explore the much more in depth explanation of the ANN model and its proposed training method that be found in the paper. However, a brief overview of the ANN model is shown in figure 3.19.

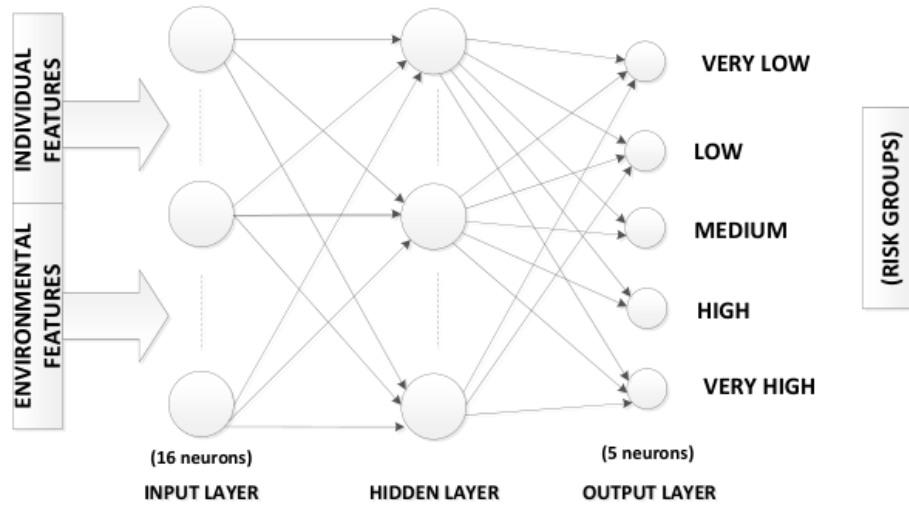


Figure 3.19: ANN structure, retrieved from [40]

3.4.3.3 Routing process

Finally, when a hazard is detected by the sensors, the intelligent routing engine sets the tolerable risk level for evacuation routes to *LOW* and the system starts a real-time navigation service to begin transmitting the shortest *LOW* risk evacuation path to the users' smartphone. Every two seconds, the mobile app sends requests to the *SmartEscape* server in order to receive updated navigation information. At each request, the server evaluates the risk levels of each link, enables or disables transitions in the network according to the acceptable risk level and transmits navigation instructions to the evacuee's mobile app, along with an image with indications of the path to follow.

In case a person is stuck in a transition with the current tolerable risk level and there is no way out using the present network model, the system will increase the tolerable risk and recalculate the appropriate path for the user to follow. Lastly, in the worst case, if the user is stuck in a transition with the highest acceptable risk level, *SmartEscape* will ignore all the remaining risk levels and simply find the shortest path to the nearest exit. The image 3.20 shows a simple diagram explanation of *SmartEscape*'s intelligent routing process.

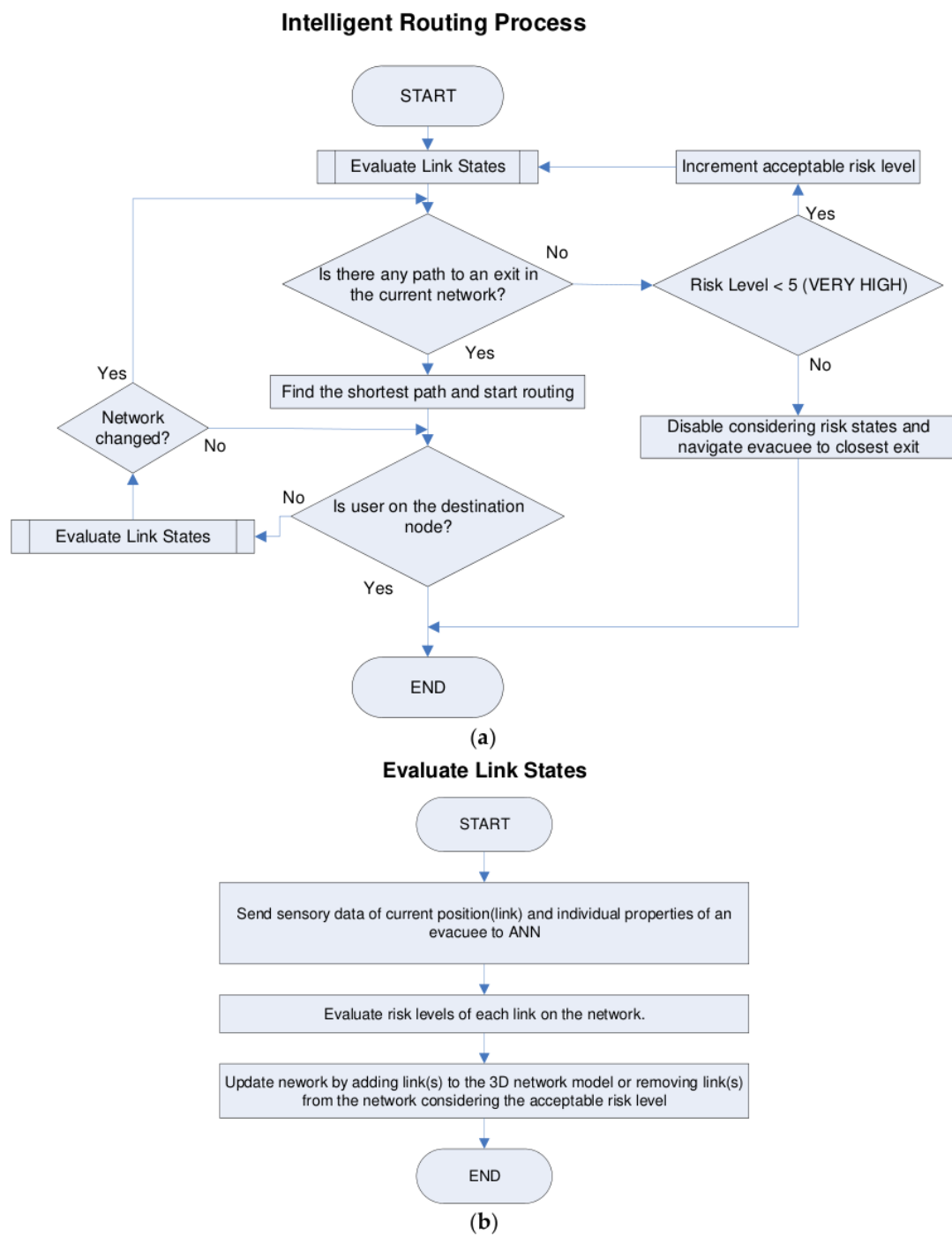


Figure 3.20: (a) - ANN based routing; (b) - evaluation of link states, retrieved from [40]

3.5 Summary

In this chapter, four important research works that highly impacted and supported the development of this thesis were mentioned. As such, this section will serve as a summary of the key components of each proposed solution cited.

All of the solutions use a graph in order to represent the building's navigable network. Some use more than one graph, such as the solution proposed in [25], in order to represent the sensor grid. In the majority of the graphs, the nodes represent locations where evacuees can be located and the links represent the walkable path between two locations. However, in both papers [25] and [11], the nodes that constitute the navigation graph are conditioned by amount the sensors placed inside the building.

As for the system architecture type, papers [9] and [40] specify a centralized architecture where the the computation of the evacuation process is done entirely by a backend server, whereas papers [25] and [11] propose a distributed system architecture in order to achieve better fault tolerance in chaotic scenarios.

Regarding the different metrics for the safest path calculation, the approach [25] uses an exclusively time based formula, where the safety of a link (i, j) is given by the minimum of the time taken for an evacuee travel from i to j and the hazard spreading from i to j . In [11], the authors propose that the safety of a link (i, j) is given by the product of the distance between i and j and the *hazard intensity* of that edge. The approach [9] introduces a risk parameter that, as the name suggests, correlates a 'risk' index to a link between a pair of nodes. It then defines the safety of a transition (i, j) as the sum of the distance between i and j and each risk parameter between i and j , giving distinct weight coefficients for each risk parameter to indicate different priority levels. Lastly, the solution [40] also uses risk values to calculate the safety of a link but the resulting transformed risk values are personalized for each user, since that they are dependant of each evacuee's physical features.

Finally, as mentioned before, approaches [25] and [11] use sensors placed inside the building as nodes for the formation of the navigation graph, which is not very convenient since it either does not allow the formation of a very complete navigable network, due to the existence of few sensors, or it will have high costs of acquisition, installation and maintenance in the case of the existence of several sensors. Papers [11] and [9] make use of visual direction indicators to guide evacuees along the calculated way out, which is not an ideal solution since, depending on the situation, they can be difficult to visualize and follow, specially in cases of high smoke density, panic, and overcrowding. On the other hand, proposal [40], is a well documented, complete and very personalized solution for each user that needs to be evacuated, since it is able to calculate different paths for two people located in the same place. However, it may end up being a bit impractical, given that it requires each user to always fill out a form in the mobile application in order to calculate the appropriate path, making it a bit complex from the evacuee's perspective. As such, the objective of this project will be to try to develop a system based on these works, but capable of overcoming some of the limitations previously mentioned.

Chapter 4

Analysis and Development

In the previous chapters, important topics that must weigh in the decisions to be made in the process of developing an evacuation system have been mentioned, as well as research works related to the field of indoor evacuation, which propose different approaches and solutions to be addressed during the development of the system. This chapter will serve to document, analyze and justify the main decisions made in the both the overall system structure, as well as each component of the architecture of this proposed solution, while also explaining in-depth all the factors of the implementation process of this project.

This work proposes an implementation of an evacuation system consisting of a frontend component for configuring important data related to each building, a backend component with a server capable of processing all the information given by the frontend module and responsible for calculating safe paths, a database to store the necessary information about the building, an asynchronous *publish/subscribe* communication service to establish sensor/server communication in order to send monitored readings from different sections of the internal area of the building and, finally, a mobile application for guiding the evacuees in a emergency evacuation scenario.

4.1 System architecture

Before delving into more technical details of the implementation of this system, it is crucial to keep in mind that one of the most important points in the development phase of any project is the design of its structural architecture, which will define how all components of the system interact with each other.

As mentioned in chapter 2, there are three main types of architectures: centralized, decentralized and distributed. Each one has its advantages and disadvantages. For the purposes of this project, a centralized architecture was chosen, since it is simpler to control, which facilitates the implementation of a single server that is able to perform the data processing and handle all requests sent by both the front-end component and the mobile application. This server will have a dedicated database, which will be responsible for storing both information relevant to added buildings, as well as a structure of nodes and edges between them, which will enable the construction of a indoor navigation graph and allow the application of path-finding algorithms.

With the system architecture type specified, it is also important to define how each component will communicate with the central server in order to satisfy each processing request needed. In this sense, it was decided to use a Representational State Transfer (**REST**) architecture to establish the communication between the central server with both the frontend configuration and mobile application components, since it offers a flexible structure of API's that can handle many types of requests and transmit data several distinct formats of data. To achieve communication between the sensors and the backend server, a *publish/subscribe* service was used to allow the server to be actively listening for new readings sent by the sensors. Figure 4.1 illustrates the representation of the architecture defined for this proposed solution.

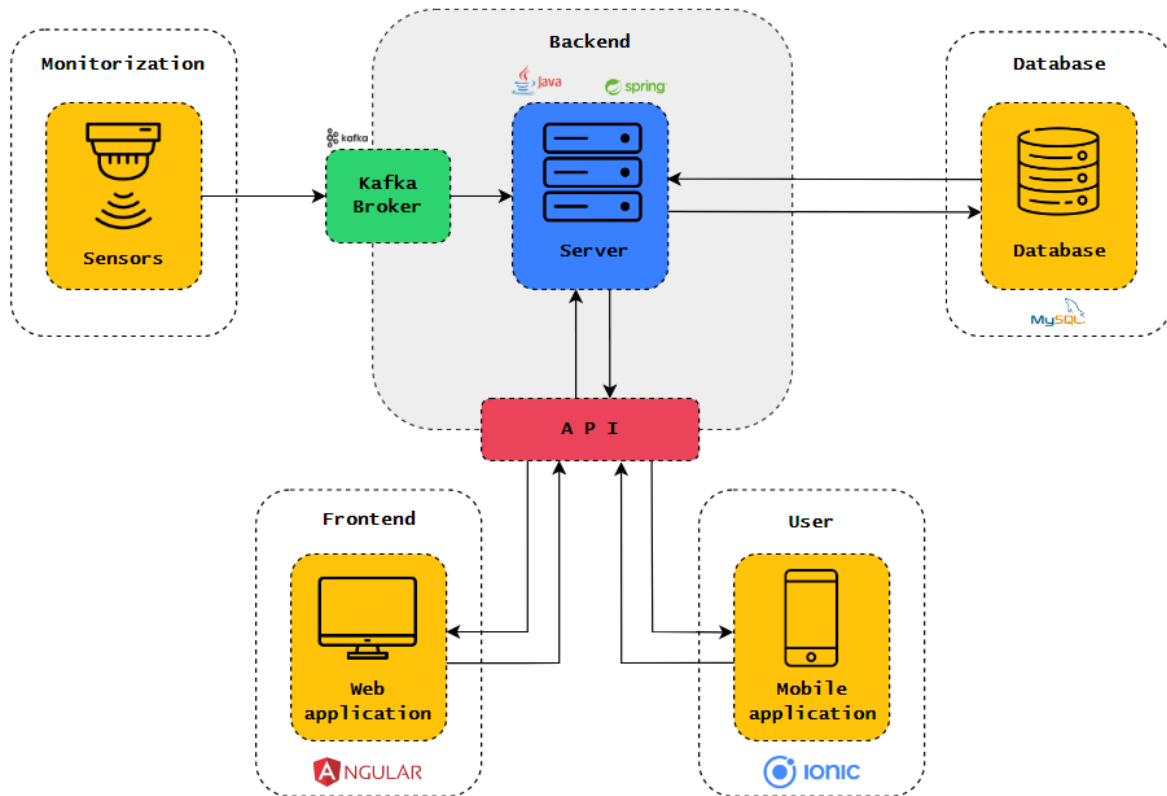


Figure 4.1: Proposed system architecture

4.2 Building representation

In order to be able to perform frequent tests during the development phase of all the components of the system architecture, it was determined to focus this analysis process around the Departamento de Ciências de Computadores (**DCC**) of Faculdade de Ciências da Universidade do Porto (**FCUP**), since it is a familiar building, with moderate structural complexity, and with quite common characteristics present in most modern buildings of our everyday life. This building is composed of two floors, with stairs and elevator to move between them, several classrooms, offices, laboratories, corridors, large open areas and six main emergency exits. Figures 4.2 and 4.3 illustrate both floor plans of this department.

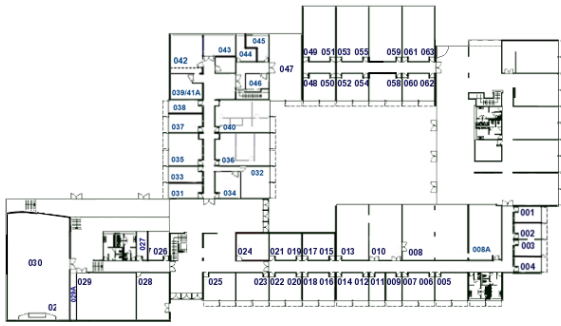


Figure 4.2: Floor 0 [10]

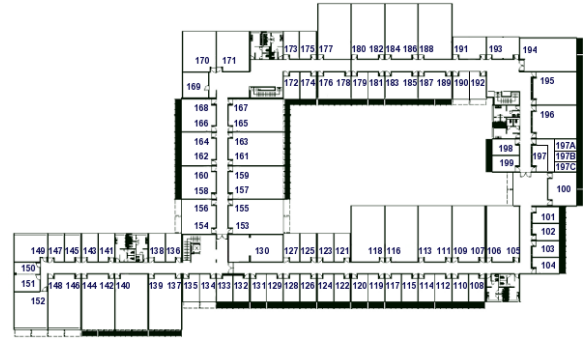


Figure 4.3: Floor 1 [10]

With the building to be represented chosen, one of the main requirements of this work was to use an indoor mapping format standard to represent the characteristics of its internal area. As such, section 2.2 of chapter 2 explains in-depth the two main indoor mapping formats, Apple's IMDF and IndoorGML, that were considered for this purpose. However, since time was of the essence and creating an indoor model of a building with this magnitude is a considerable time consuming task in itself, it was decided to use IndoorGML as the indoor mapping format to represent Departamento de Ciências de Computadores (DCC), since it is simpler to use and also allows the definition of an indoor navigation network of the building. In this sense, a simple representation of floor 0 of the DCC was built, which is illustrated in figure 4.4. The IndoorGML file contains the essential information about all of the defined nodes, namely their name, coordinates and description tags, while also specifying a set of transitions that define links between these nodes, which allows the construction of a navigation network of the building. In order of the backend server to distinguish which nodes represent building exits, all of the exit nodes were assigned a *emergency-exit* description tag.

As was stated in 2.2.2, IndoorGML enables the representation of several layers, formally referred as Multi-Layered Representation (MLR), since a single indoor space can be semantically interpreted into different cellular spaces. As such, a `sensor` layer was created in IndoorGML in order to simulate the representation of four different sensor coverage areas, shown in figure 4.5. Finally, inter-layer connections between sensors and the nodes of the base layer inside the coverage area were also formed to indicate that, for example, a sensor *X* is responsible for monitoring the environment around node *Y*.

```
<core:State gml:id="S4">
  <gml:description>floor -level="0":type="room":</gml:description>
  <gml:name>Room 003</gml:name>
  <!-- ... -->
  <core:connects xlink:href="#T111"/>
  <core:connects xlink:href="#T111-REVERSE"/>
  <core:geometry>
    <gml:Point gml:id="SG-S4">
      <gml:pos srsDimension="3">
        377.07082643554287 75.33587603118775 0.0
      </gml:pos>
    </gml:Point>
  </core:geometry>
</core:State>
```

Listing 4.1: Example of node Room 003 defined in IndoorGML



Figure 4.4: IndoorGML base layer representation of DCC's floor 0

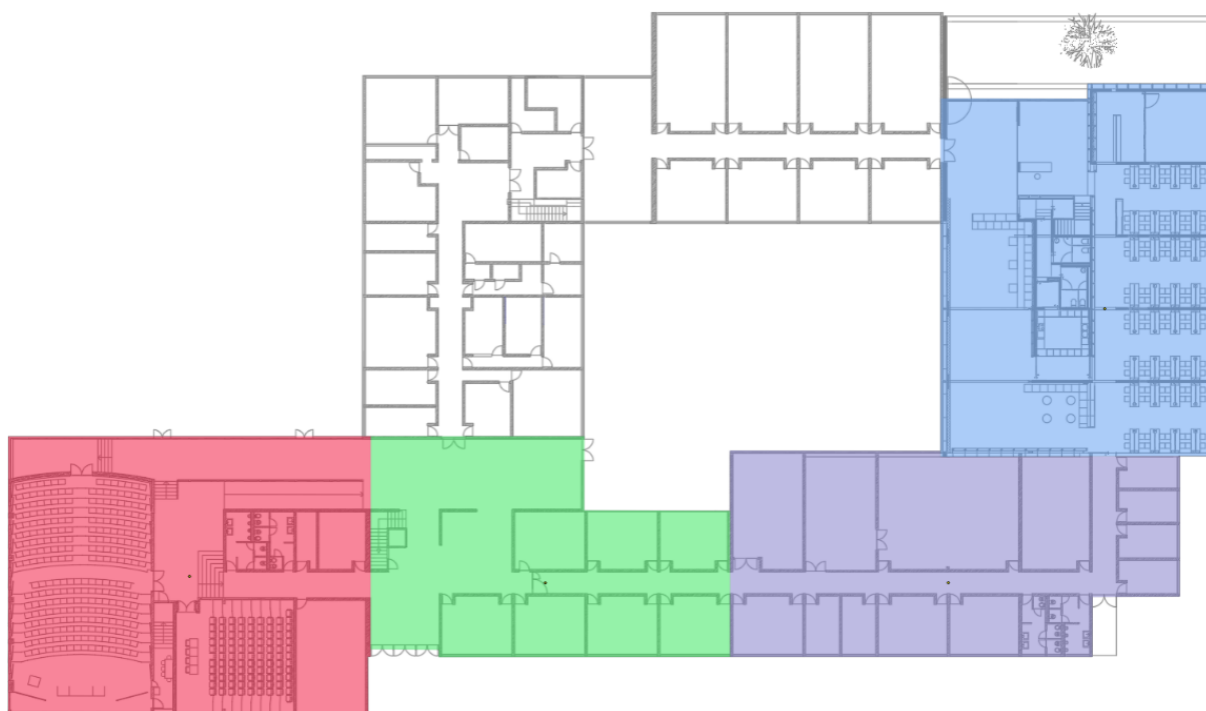


Figure 4.5: IndoorGML sensor layer representation of DCC's floor 0

4.3 Backend

As already mentioned in the architecture section of this proposed system, 4.1, the backend component will aim not only to process all the configuration steps done by the frontend module, such as processing the IndoorGML file, but also to store values of environmental variables monitored by the sensors, construct a graph to represent the building's navigation network, calculate safe paths and transmit them to the user. For this backend module, it was decided to use Spring Boot [36], which is an open source framework that allows creating standalone, production-grade applications that run on the Java Virtual Machine (JVM), while also enabling integration with asynchronous transmission services and Representational State Transfer (REST) architectures.

In this section the structure of the backend module will be detailed, starting with the building creation procedure by processing IndoorGML files, the database data structure, the API specification for client-server communication, the *publish/subscribe* service used for asynchronous communication between the sensors and the server, the application of a safe path search algorithm and, finally, the communication between the mobile application and the server for transmission of evacuation routes.

4.3.1 IndoorGML file processing

The IndoorGML format is based XML, as mentioned in section 2.2.2, therefore, it was necessary to search Java libraries and frameworks that allow the reading of XML to process the important information about the building. For the purpose of this project, it was decided to use the Java Architecture for XML Binding (JAXB) framework, which offers tools for reading and mapping XML to Java objects. This framework uses a XML Schema to generate several Java classes and is capable of *unmarshalling* an XML file, which will map each element in the IndoorGML file to a Java object.

As such, a set of Java classes that map each of the data types defined in the IndoorGML schema [15] were generated through running the `xjc` compiler, using the core XML schema as a command argument. The amount of files generated from the IndoorGML schema were quite large because the schema invokes the Geographic Markup Language (GML) namespace, therefore, all GML data types were also generated automatically by the JAXB framework. The image 4.6 shows a small subset of the Java classes generated.

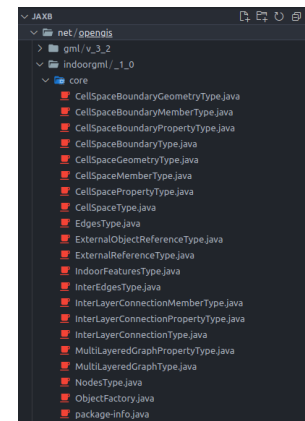


Figure 4.6: Generated classes

With the elements stipulated in the IndoorGML schema defined in Java classes, it is possible to proceed to the *unmarshalling* process through the JAXB framework. This process is performed by the code excerpt transcribed in 4.2, which reads the IndoorGML file in its entirety and maps all element values defined in the file to the corresponding Java object. The root element defined in IndoorGML is `IndoorFeaturesType` which, after the *unmarshalling* process, will now contain all other children elements in its data structure.

```
private ResponseEntity<?> processIndoorGMLFile(MultipartFile file , Building
    building) throws JAXBException, IOException {

    // Read IndoorGML file and transform to Java objects
    JAXBContext jc = JAXBContext.newInstance(IndoorFeaturesType.class);
    Unmarshaller unmarshaller = jc.createUnmarshaller();
    InputStream inputStream = file.getInputStream();
    Source source = new StreamSource(inputStream);
    JAXBElement<IndoorFeaturesType> root = unmarshaller.unmarshal(source ,
        IndoorFeaturesType.class);
    IndoorFeaturesType indoorGML = root.getValue();
    // ...
```

Listing 4.2: IndoorGML file unmarshalling

Finally, after all the information contained in the IndoorGML file is mapped to a Java data structure, it is only necessary to iterate through the root object, `IndoorFeaturesType`, to search for all the children classes that contain important data to store in the database, namely the nodes of the navigation graph drawn in the *base layer*, the sensors defined in the *sensor layer* and the inter-layer connections between sensors and nodes of the *base layer*.

4.3.2 Data structure

During emergency scenarios, the stability and smooth operation of the evacuation system can always be in question. For instance, there may be times when the system needs to be restarted, and if all the data about the building is only stored in server memory, the information given by the indoor mapping format (IndoorGML) will be lost and need to be reprocessed when the backend server is up and running again. To avoid these types situations, it is possible to resort to a database, which, as mentioned in section 2.1.3, is a structure that allows to store a collection of information in an organized and persistent way.

A MySQL relational database management system was chosen, since it is open-source, stores data in tables in a structured and easily accessible way and also offers a familiar database environment, which allowed for a more expedited development process. Figure 4.7 represents the proposed Entity-Relationship (ER) diagram of the implemented database model, which contains a set of tables responsible for maintaining a persistent model of all system data, namely the information defined in the IndoorGML format, and the updated sensor reading values.

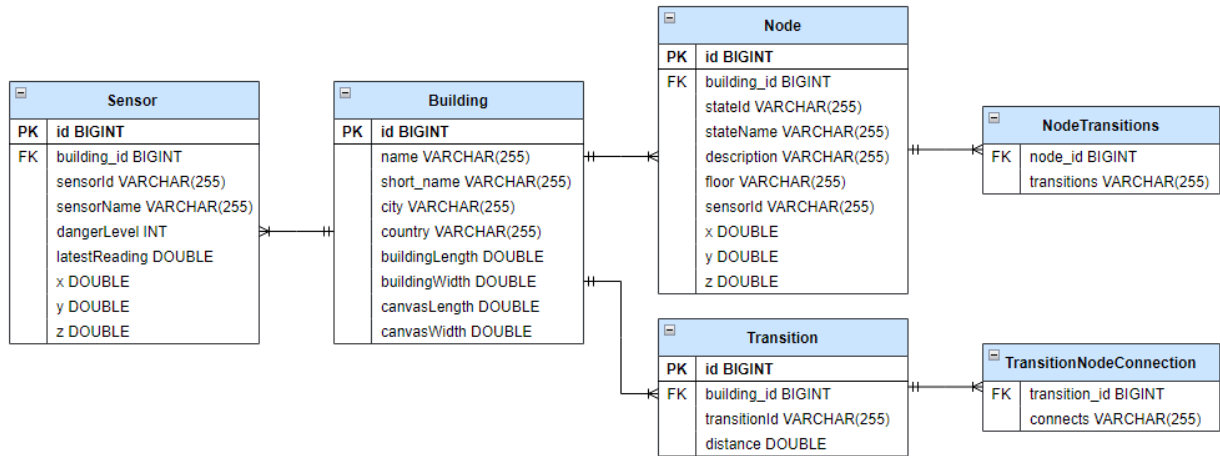


Figure 4.7: Proposed Entity-Relationship (ER) model diagram

4.3.2.1 Building

In order to be able to store all the basic information of the building, the Building table was created. This is the main table of the database and is responsible for storing essential variables whose brief description can be found in table 4.1.

Field	Datatype	Description
id	Integer	Unique identifier of Building
name	String	Building name
short_name	String	Building name abbreviation
city	String	City where the building is located
country	String	Country where the building is located
buildingLength	Double	Building length in meters
buildingLength	Double	Building width in meters
canvasLength	Double	IndoorGML canvas length
canvasWidth	Double	IndoorGML canvas width

Table 4.1: Building table fields and descriptions

The bottom four variables found in table 4.1 are necessary because the coordinates of each node defined in the IndoorGML file are represented in canvas units, with the bottom left corner of the canvas indicating the origin point (0, 0) and the top right corner indicating the point (canvasLength, canvasWidth). In this use case example, the department's floorplan shown in figure 4.4 was designed using the default IndoorGML canvas dimensions of 400 × 300, however, the building itself is approximately 101.09 meters long and 60.03 meters wide, as can be seen in figures 4.8 and 4.9. As such, during the processing of the IndoorGML file, before storing the building information in the database, it was decided to convert every node's canvas coordinates to metric coordinates through the application simple rule of three. This means that the updated coordinates of each node present in the navigation graph, which will be stored in the database, now reside between the (0, 0) and (101.09, 60.03) interval, corresponding respectively to the lower left and upper right corner of the department's floorplan.

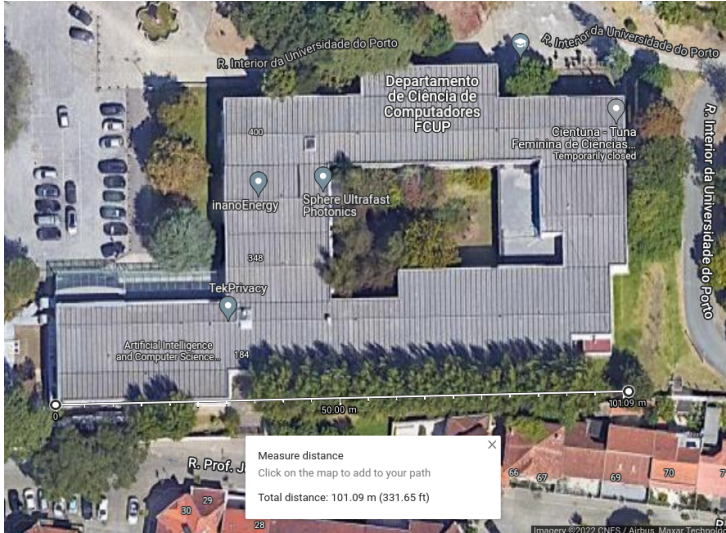


Figure 4.8: DCC's measured length

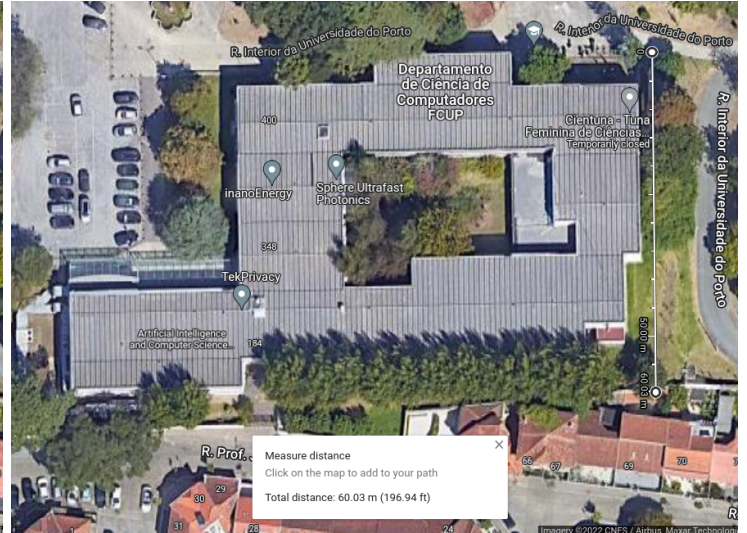


Figure 4.9: DCC's measured width

4.3.2.2 Nodes and Edges

The nodes and edges form the navigation graph of the building will be used for the path-finding process. For this purpose, the tables Node, NodeTransitions, Transition and TransitionNodeConnection were created to store essential information about the nodes and their connections, in order to enable the server to form the navigation network with only persistent data from the database. Tables 4.2, 4.3, 4.4 and 4.5 describe all columns of the respective database table and provide a brief description of each.

Field	Datatype	Description
id	Integer	Unique identifier of Node
building_id	Integer	Building ID
stateId	String	IndoorGML node ID (e.g. S4)
stateName	String	IndoorGML node name (e.g. Room 003)
description	String	IndoorGML type tag (e.g. room)
floor	String	IndoorGML floor-level tag (e.g. 0)
sensorId	String	IndoorGML sensor ID that monitors this node (inter-layer connection)
x	Double	Location coordinate X (m)
y	Double	Location coordinate Y (m)
z	Double	Location coordinate Z (m)

Table 4.2: Node table fields and descriptions

Field	Datatype	Description
node_id	Integer	Node ID
transitions	String	IndoorGML transition ID (e.g. T111)

Table 4.3: NodeTransitions table fields and descriptions

The sensorId attribute of the Node table is given by an indicating inter-layer connection between a sensor node of the sensor layer, and one regular node of the base layer. Code excerpt 4.3 transcribes the definition of an Inter-Layer Connection defined in the IndoorGML file between sensor S108 (purple area of 4.5) and the node S4 (Room 003), mentioned in 4.1.

```

<core:InterLayerConnection gml:id="I97">
  <gml:boundedBy xsi:nil="true"/>
  <core:typeOfTopoExpression>CONTAINS</core:typeOfTopoExpression>
  <core:interConnects xlink:href="S108"/>
  <core:interConnects xlink:href="S4"/>
  <core:ConnectedLayers xlink:href="sensor"/>
  <core:ConnectedLayers xlink:href="base"/>
</core:InterLayerConnection>

```

Listing 4.3: Example of an IndoorGML inter-layer connection

```

<core:Transition gml:id="T111">
  <!-- ... -->
  <core:connects xlink:href="S4"/>
  <core:connects xlink:href="S91"/>
  <core:geometry>
    <gml:LineString gml:id="TG-T111">
      <!-- canvas coordinates of S4 and S91 -->
    </gml:LineString>
  </core:geometry>
</core:Transition>

```

Listing 4.4: Example of an IndoorGML transition

Field	Datatype	Description
id	Integer	Unique Identifier of Transition
building_id	Integer	Building ID
transitionId	String	IndoorGML transition ID (e.g. T111)
distance	Double	Euclidean distance between both nodes that constitute the Transition (m)

Table 4.4: Transition table fields and descriptions

Field	Datatype	Description
transition_id	Integer	Transition ID
connects	String	IndoorGML node ID (e.g. S4)

Table 4.5: TransitionNodeConnection table fields and descriptions

The Transition table stores the basic information of a link in the graph, namely its identification and the distance between two nodes that constitute it, which is calculated through the application of the following Euclidean distance formula

$$AB = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

where point $A(x_1, y_1, z_1)$ and $B(x_2, y_2, z_2)$ form a direct navigable link between each other. A transition defined in IndoorGML can also have coordinates of an intermediate point that intersects a *CellSpaceBoundary* responsible for delimiting boundaries between cells (doors). In these cases, the total distance calculated is the sum of the Euclidean distances of the two components, i.e. $AX + XB$, where X is the intersection point with a *CellSpaceBoundary*.

Finally, the table TransitionNodeConnection is just an auxiliary table that was created to list the nodes that establish a Transition. For example, for transition T111, showed in code listing 4.4, two entries would be inserted, one for node S4 (Room 003) and another for node S91 (Room 003 door).

4.3.2.3 Sensors

After structuring the nodes and edges of the building's navigation graph, it is important to organize and correctly represent the information regarding sensors and environment monitoring values. The sensor nodes are defined just like regular nodes in IndoorGML. However, they are in a different layer, in this case, the `sensor` layer. As such, the `Sensor` table was created, which contains basic information about the sensor itself, as well as data regarding the most recent monitored reading. Table 4.6 lists the columns of the `Sensor` table and gives a brief description of each one.

Field	Datatype	Description
<code>id</code>	Integer	Unique identifier of Sensor
<code>building_id</code>	Integer	Building ID
<code>sensorId</code>	String	IndoorGML sensor ID (e.g. S108)
<code>sensorName</code>	String	IndoorGML sensor name (e.g. VIRTUAL_SENSOR_3)
<code>dangerLevel</code>	Integer	Danger level of the sensor
<code>latestReading</code>	Double	Latest monitored environment value
<code>x</code>	Double	Location coordinate X (m)
<code>y</code>	Double	Location coordinate Y (m)
<code>z</code>	Double	Location coordinate Z (m)

Table 4.6: Sensor table fields and descriptions

Although there may be several metrics for defining hazard levels in a building area, only danger level calculations based on temperature readings have been implemented. Therefore, the `latestReading` variable corresponds to the last reading monitored temperature by the sensor and the `dangerLevel` field corresponds to the assigned danger index associated to the `latestReading` temperature value, which will be later used to calculate safer paths that try to avoid nodes with higher danger zones. Table 4.7 maps the defined temperature ranges monitored inside the building to their respective danger level.

Temperature Interval	Danger level
$T \leq 30^{\circ}\text{C}$	0
$30^{\circ}\text{C} < T \leq 40^{\circ}\text{C}$	1
$40^{\circ}\text{C} < T \leq 50^{\circ}\text{C}$	2
$T \geq 50$	3

Table 4.7: Defined danger levels for specific temperature intervals

4.3.3 API

For the communication between the frontend/mobile application components and the backend server, an Application Programming Interface (**API**) was developed to allow the exchange of information and initial configuration of the building, namely building creation by the upload of an IndoorGML file, initialization of evacuation paths for the graph representation of the building, transmission of simulated sensor reading values and communication of safe paths to the mobile application. The list below identifies each method of the implemented **API** and a brief description of its operation.

- **`http://backend/api/building/createNewBuilding`** - This endpoint allows the configuration frontend component to create a building from a brief information form and an IndoorGML file.
- **`http://backend/api/building/getBuildings`** - Endpoint that retrieves all the buildings stored in the database.
- **`http://backend/api/building/initializePaths`** - This endpoint initializes the calculation of evacuation paths for all the nodes in the building graph.
- **`http://backend/api/building/updateSensorReadings`** - The process of sending the simulated sensor readings is performed by the *publish/subscribe* service defined in section 4.3.4. However, this endpoint allows the frontend component to send a REST request to the server, in order to perform a *publish* operation to the *pub/sub topic*, with the most recent monitored reading value of a sensor.
- **`http://backend/api/building/uploadFloorplanImg`** - This endpoint allows the submission of an image in PNG format illustrating a particular floor of the building, which will later be used to guide evacuees through the suggested safe path. For the use case studied in this dissertation, the image 4.10 was uploaded, corresponding to floor 0 of Departamento de Ciências de Computadores (DCC).
- **`http://backend/api/building/getBuildingFloorplans`** - This endpoint returns all of the floorplan images submitted to the server for illustration in the mobile application.
- **`http://backend/api/building/evacuate`** - This endpoint method returns an ordered list with all the nodes that constitute the calculated safest path the user has to follow, from their current location to the emergency exit chosen by the evacuation algorithm.

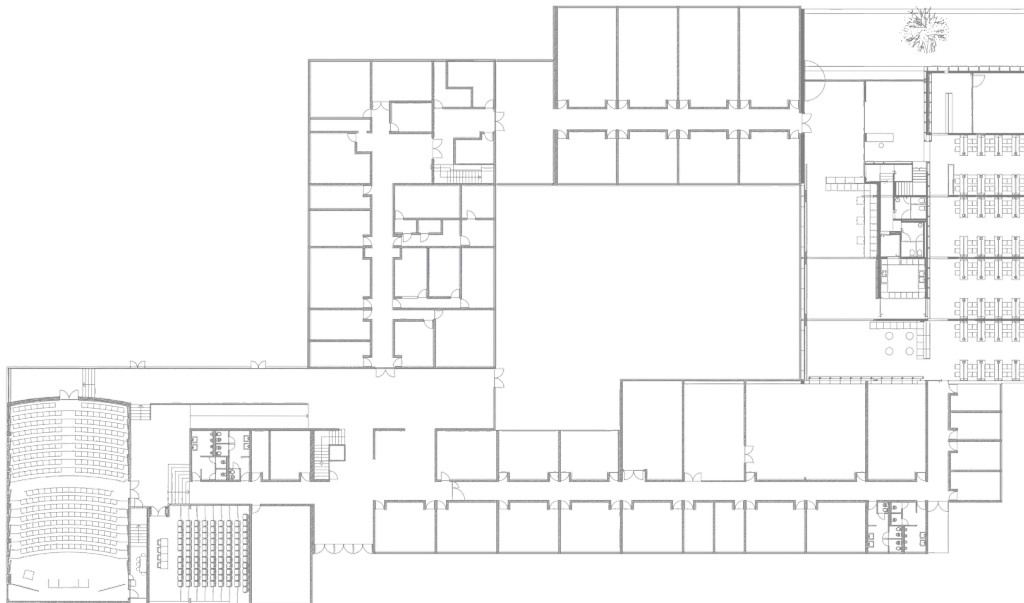


Figure 4.10: DCC's floor 0

4.3.4 Publish/Subscribe service

To be able to establish communication between the sensor network and the backend component, an asynchronous *publish/subscribe* communication service is used. A *pub/sub* service uses a message *broker* to establish a structure for event transmission between publishers and subscribers, allowing the host (*publisher*) to publish events in a *topic*, so that all components that are subscribed to this *topic* are able to access the published information.

In this use case, since the communication model is asynchronous, the sensor readings can be published to a *broker topic*, allowing the data to be accessible to the server that is subscribed to it. In more complex system contexts, there may be more subscribed components, in case more intermediate calculations need to be performed in the evacuation pathfinding process.

For the development of this work, the *publish/subscribe* platform chosen for sensor-server communication was Apache Kafka [20]. This event streaming platform is open-source, developed in Java and Scala, and has an easy integration with applications developed in Spring Boot, which was the framework chosen for the implementation of the backend server. A *sensors topic* was created in a Kafka *broker*, to which all sensor readings will be published to. This *topic* will be subscribed by the backend component, which will then process these readings by assigning a danger level to it, update the information stored in the *Sensor* table of the database, construct a new building graph from the latest updated danger indexes and calculate new evacuation paths from all nodes in the graph. Figure 4.11 shows the implemented Apache Kafka architecture for asynchronous sensor-server communication.

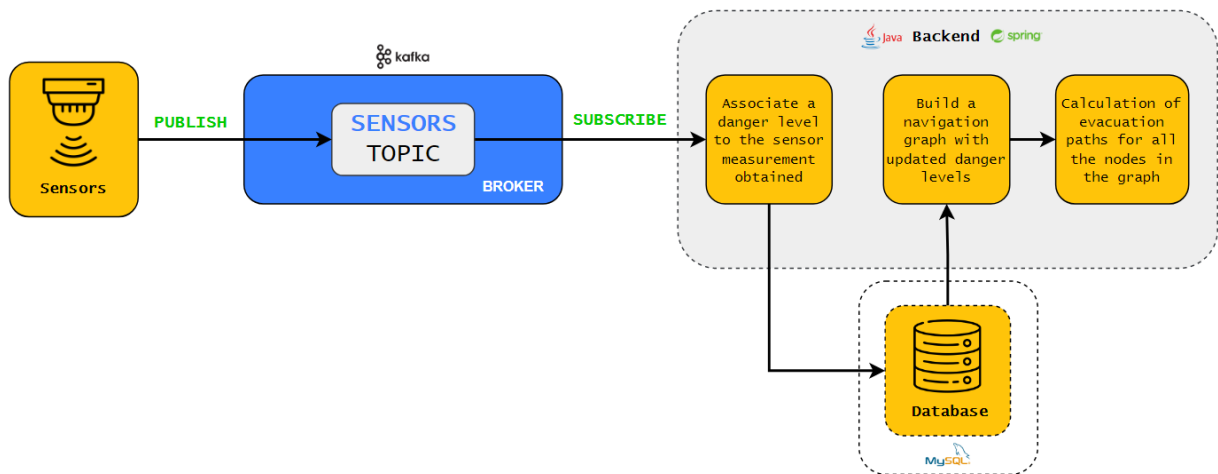


Figure 4.11: Apache Kafka *publish/subscribe* communication flow

4.3.5 Evacuation process

As mentioned in section 2.4, there are several search algorithms for solving safe pathfinding problems, such as Dijkstra, A*, Floyd-Warshall, and Bellman-Ford, each of which has its respective advantages and disadvantages in terms of implementation simplicity, use cases, and time complexity. However, before delving further, it is important to understand how the navigation network information can be prepared for the application of such algorithms.

In the scope of this project, it was stipulated that all navigation nodes defined by the IndoorGML format will have associated danger levels, given by reading values monitored by sensors. More specifically, each node entry in the Node table has an associated sensor that is responsible for sending temperature readings monitored inside the building, which will later be processed by the backend server that will assign a danger level to that sensor, and consequently, to all nodes associated with it. For example, if node S4 (Room 003) is under supervision of sensor S108 (VIRTUAL_SENSOR_3), whose most recent temperature reading was 55.8 degrees, equating to a danger level of 3 (4.7), then all nodes under S108's supervision, including node S4, will also be associated with a danger level of 3.

That said, it was defined that the danger level of a link between two nodes will equal to the highest danger level of those two nodes. In other words, if node X has a danger level of 2 and node Y is at danger level 0, the danger level associated with the link XY will be 2, according to the following formula:

$$DangerLevel_{XY} = \max_{(DL_X, DL_Y)}$$

where DL_X and DL_Y are the independent danger levels of node X and Y , respectively, given by the sensors that monitor each one.

4.3.5.1 Graph construction

Although the information contained in the IndoorGML file, further processed by the server and stored in the database with the data structure represented in section 4.3.2, implicitly allows for the analysis of the building's navigation network, it was necessary to specifically create a graph class, composed of a set of vertices and edges, in order to be able to fetch all the data stored across all the tables in the database and condense this information into a single graph object to be able to apply path-finding algorithms through it. Listing 4.5 shows the Graph, Vertex and Edge classes that were created for this purpose, as well as the algorithm implemented in order to populate the Graph object with all the information contained in the database tables. This algorithm simply iterates through all the data objects stored in the database, namely Node and Transition, and for each Node N listed in the database, the algorithm creates a Vertex V from N 's information, iterates through N 's list of transitions to discover the nodes it is connected to, and creates a reversed Edge for those neighbors. This reversal is further explained in the next subsection.

```

1 public class Graph {
2     private Set < Vertex > vertices; // Collection of vertices
3     // ...
4 }
5 // ...
6 public class Vertex {
7     private String indoorGMLId, indoorGMLName, description, sensorId, floor;
8     private Double x, y, z;
9     private boolean visited, isEndVertex;
10    private Set < Edge > edges;
11    private Building building;
12    // ...
13 }
14 // ...
15 public class Edge {
16     private Vertex destination;
17     private Double distance;
18     private Integer dangerLevel;
19     private Double[] dangerDistance = new Double[4];
20     // ...
21 }
22 // ...
23 private Graph createBuildingGraph(Building building) {
24     List < Node > nodes = building.getNodes();
25     List < Transition > transitions = building.getTransitions();
26     List < Sensor > sensors = building.getSensors();
27
28     // Hashmaps with IndoorGMLId as key
29     Map < String, Node > nodesMap = building.getNodesMap();
30     Map < String, Transition > transitionsMap = building.getTransitionsMap();
31     Map < String, Sensor > sensorsMap = building.getSensorsMap();
32
33     Graph graph = new Graph();
34     Map < Node, Vertex > nodeToVertexMap = new HashMap < > ();
35
36     for (Node node: nodes) {
37         Vertex vertex = new Vertex(node);
38         nodeToVertexMap.put(node, vertex);
39     }
40
41     for (Node node: nodes) {
42         Vertex vertex = nodeToVertexMap.get(node);
43         List < String > stringTransitionList = node.getTransitionsList();
44         String vertexIndoorGMLId = vertex.getIndoorGMLId();
45
46         // Iterate through transitions list (ex: [T1, T2, ...]);
47         for (String currentTransitionID: stringTransitionList) {
48             Transition transition = transitionsMap.get(currentTransitionID);
49             List < String > connects = transition.getConnects();
50             Double distance = transition.getDistance();
51
52             // Iterate through connection list which will always have size 2
53             // (ex: connects=[S1, S2] where either S1 or S2 = nodeIndoorGMLId
54             for (String possibleNeighborId: connects) {
55                 if (!possibleNeighborId.equals(vertexIndoorGMLId)) {
56                     Node neighborNode = nodesMap.get(possibleNeighborId);
57                     Vertex neighborVertex = nodeToVertexMap.get(neighborNode);
58
59                     Sensor nodeSensor = sensorService.getSensorByName(node);
60                     Sensor neighborSensor = sensorService.getSensorByName(neighborNode);
61
62                     Integer nodeSensorDL = nodeSensor.getDangerLevel();
63                     Integer neighborSensorDL = neighborSensor.getDangerLevel();
64
65                     neighborVertex.addEdge(new Edge(vertex, distance, Math.max(nodeSensorDL,
66                                             neighborSensorDL)));
67                 }
68             }
69             graph.addVertex(vertex);
70         }
71     }

```

Listing 4.5: Building Graph construction

4.3.5.2 Path-finding algorithm

Since all information about the building's navigable network is now stored in a Graph object and the only metrics impacting the evacuation path calculation process in this proposed system are the total distance to be traveled and the danger level, the application of a Dijkstra algorithm is a reasonable choice, given that it allows the calculation of all of the best paths from one source to all the other nodes in the graph. However, instead of using the minimum total distance to discern the best route, an array of distances per level of danger was implemented, as shown in image 4.12. This metric will update the objective of the algorithm, which will now be to choose paths with shorter distance traveled in higher levels of danger (higher indexes of the array), thus minimizing the user's exposure to more hazardous areas during the evacuation process.

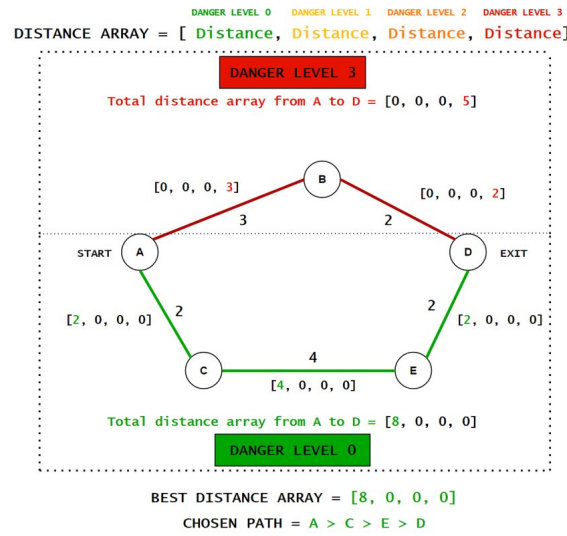


Figure 4.12: Arrays of distances by danger level

As mentioned before, the Dijkstra algorithm needs a source node to initialize the path calculation from, so a possible naive solution would be to apply the algorithm with the node closest to the user's location being the source node, thus forming a set of best paths from the evacuee's location to all nodes of the graph. The problem with this solution is that the Dijkstra algorithm would need to be executed every time the server receives a new location of a user who needs to be evacuated, which will result in poor system performance, especially in instances of big navigable networks. Another possible solution is to apply the algorithm backwards, that is, by reversing every edge in the graph and use one of the emergency exits as the source node, allowing the server to know the distance from the user's location node to the source. This solution would be more effective, however it would still require the execution of the algorithm once for each possible emergency exit to be able to compare which one outputs the optimal route. In order to improve the system performance, the approach proposed in subsection 3.3.4 was used, which consists on creating a dummy node that has an Edge to all of the *emergency-exit* nodes in the Graph, and execute Dijkstra's algorithm starting on the dummy node. This way, it is possible for the server to maintain in memory the set of best paths from the dummy node to all nodes of the graph and return the best path to the evacuee's node whenever requested, without having to re-execute the algorithm. Listing 4.6 shows the proposed implementation of Dijkstra.

```

1 public class DijkstraAlgorithm {
2     private Building building;
3     private List < Vertex > nodes;
4     private Set < Vertex > settledNodes;
5     private Set < Vertex > unsettledNodes;
6     private Map < Vertex, Vertex > predecessors;
7     private Map < Vertex, Double[] > dangerDistance;
8
9     public void execute(Vertex source) {
10         settledNodes = new HashSet < Vertex > ();
11         unsettledNodes = new HashSet < Vertex > ();
12         distance = new HashMap < Vertex, Double > ();
13         predecessors = new HashMap < Vertex, Vertex > ();
14         distance.put(source, 0.0);
15         unsettledNodes.add(source);
16
17         dangerDistance = new HashMap < Vertex, Double[] > ();
18         dangerDistance.put(source, new Double[] { 0.0, 0.0, 0.0, 0.0 });
19
20         while (unsettledNodes.size() > 0) {
21             Vertex node = getMinimumDD(unsettledNodes);
22             settledNodes.add(node);
23             unsettledNodes.remove(node);
24             findMinimalDistancesDD(node);
25         }
26     }
27
28     private void findMinimalDistancesDD(Vertex node) {
29         List < Vertex > adjacentNodes = getNeighbors(node);
30         for (Vertex target: adjacentNodes) {
31             Double[] sum = add(getShortestDistanceDD(node), getDistanceDD(node, target));
32             if (lessThan(sum, getShortestDistanceDD(target))) {
33                 dangerDistance.put(target, sum);
34                 predecessors.put(target, node);
35                 unsettledNodes.add(target);
36             }
37         }
38     }
39
40     private Double[] getShortestDistanceDD(Vertex destination) {
41         Double[] dd = dangerDistance.get(destination);
42         if (dd == null) {
43             return new Double[] {
44                 Double.MAX_VALUE, Double.MAX_VALUE, Double.MAX_VALUE, Double.MAX_VALUE
45             };
46         } else return dd;
47     }
48
49     private Double[] getDistanceDD(Vertex node, Vertex target) {
50         for (Edge edge: node.getEdges()) {
51             if (edge.getDestination().equals(target)) {
52                 return edge.getDangerDistance();
53             }
54         }
55     }
56
57     private Vertex getMinimumDD(Set < Vertex > vertexes) {
58         Vertex minimum = null;
59         for (Vertex vertex: vertexes) {
60             if (minimum == null) {
61                 minimum = vertex;
62             } else {
63                 if (lessThan(getShortestDistanceDD(vertex), getShortestDistanceDD(minimum))) {
64                     minimum = vertex;
65                 }
66             }
67         }
68         return minimum;
69     }
70     // The lessThan(A, B) function compares danger distance arrays A and B
71     //     returns true if A is smaller (has less distance in higher levels of danger)
72     //     returns false otherwise
73     // The add(A, B) method adds two danger distance arrays A and B and returns the result
74     // ...
75 }

```

Listing 4.6: Proposed implementation of Dijkstra

4.3.5.3 Safe paths communication

After developing the previously mentioned algorithm that calculates all the safest paths and returns the best path from a given node in the graph to the recommended exit, it was necessary to create an API for direct communication between the mobile application and the server to transmit the suggested path to the user's location.

However, since no indoor localization system could be developed, as it is in itself a complex problem requiring further research and implementation time which could not be covered by this project, it was necessary to resort to simulations of user location. For this, the evacuate endpoint mentioned in section 4.3.3 was implemented, which receives as input the floor where the user is and its location coordinates in (x, y) format in meters (with reference to the lower left corner of the building floorplan as the origin) and searches for the node of the graph that is at the closest Euclidean distance. After finding the closest Vertex, this method returns the path suggested by the algorithm to one of the emergency exits and uses the image submitted by the frontend configuration component to highlight the path that the user has to follow. Figure 4.13 shows an example of a calculated path to a user's location at coordinates $(91, 20)$ of floor 0, with a current danger level of 3 in the area where the evacuee is in (purple section illustrated in figure 4.5). The blue dot represents the user's current location, the red dot shows the closest node, the yellow dots are the locations of all other navigation nodes to follow in the path and the green dot indicates the suggested emergency exit.



Figure 4.13: Example of an evacuation route transmitted to an evacuee

4.4 Frontend

As already mentioned in previous sections, the frontend module has the objective of performing the initial configurations for the evacuation process. These configurations include visualizing the list of all buildings stored in the system's database, creating new buildings through upload of a IndoorGML file and the indication of basic information about the building, upload of floorplan images in *PNG* format for evacuation path illustrations, initialization of safe path calculations and also sending simulated sensor monitoring reading values in order to update danger levels in specific areas of the building (for testing purposes).

For this, it was decided to develop a web application in Angular, which is an client-side open-source component-based framework developed with TypeScript for creating a Single-Page Application (*SPA*). The architecture of Angular relies on a set of components that are organized as *NgModules*, can define *views*, which are sets of screen elements, and use *services* which provide specific functionalities by being injected into components as dependencies [2].

Two main components were developed in the Angular project. The first one presents a form for filling information and submission of the IndoorGML file of the building, along with a list of buildings currently stored in the database. The second component, visible only after the selection of a building in the first, allows building configurations to be performed, such as submitting floor plans for path illustration purposes, sending sensor monitoring readings, and initializing evacuation paths.

4.5 Mobile application

Once the frontend and backend components had been implemented and all necessary configurations for the calculation of the escape routes had been performed, it was necessary to develop a mobile application to guide the evacuees of the building through the safest pathway to the recommended exit.

For this, a mobile application was developed in Ionic, which is an open-source Software Development Kit (*SDK*) constituted by a set of web components that enable the construction of services and mechanisms for hybrid mobile, desktop and progressive web application development [44]. Ionic also allows the use of the Angular framework for the app development which simplified the overall implementation process, since it shares the same framework with the frontend component.

In order to be able to transmit the location of the evacuee's smartphone so that the backend component can map that location to one of the nodes of the building's graph, it would be necessary to implement an indoor tracking system. However, since this is a rather complex problem in itself, and would require not only much more time to implement, but also further research in the respective area of study, it could not be included in this project. The API

mentioned in 4.3.5.3 was developed for direct communication with the server to send simulated location coordinates in meters, so that the backend component is able to map the user's current location to the nearest node, allowing it to return the suggested safe path to the appropriate exit.

As such, two *views* have been developed. The first *view* sends a request for all of the building floor plan images when the app starts and stores them in memory for illustration purposes. It also presents inputs for the user's (x, y) coordinates in meters, along with an input to indicate the floor the user is currently on, in order to request an evacuation path from the backend module. The second *view*, visible only after the application receives an safe path, highlights the route received in the Ionic canvas over the floor plan image in the background, and lists all of the nodes constituting the route to the recommended emergency exit.

Chapter 5

Results

In this chapter, the results obtained through tests ran on the developed system will be documented. A simulated temperature reading indicating a high level of danger was published into the `sensors topic`, in order to illustrate the proper evacuation path calculation process that aims to choose evacuation routes that avoid passing through dangerous regions of the building.

5.1 Building representation

Before proceeding to the practical demonstration of the system, it was necessary to create a model of the building in IndoorGML. In this sense, a simple representation of the department floor 0 was created, which is illustrated in figure 5.1. A cell was designed for each interior space, containing nodes and links between them to represent direct navigable connections. For each interior section of larger magnitude, such as long corridors, halls or large open areas, a few extra nodes were added to form a more complete and realistic navigation network. In addition, since IndoorGML also allows the creation of description tags to describe objects present in the model, for each cell and node present in the indoor space, a name identifies it was given, as well as two description tags, `floor-level` and `type`, which indicate the floor this cell/node is on, and the type of area it represents, respectively. Examples of values given to the `type` tag are: *area*, *auditorium*, *corridor*, *door*, *emergency-exit*, *hall*, *room*, *sensor*, *stairs* and *wc*. After drawing all the cells representing the internal areas of the department floor, direct transitions between nodes were also created, thus forming navigable links between them.

Given that IndoorGML accepts Multi-Layered Representation (**MLR**), a `sensor layer` was also created to simulate the representation of sensor coverage areas. As such, four sensor monitored regions were developed, which are illustrated in figure 4.5. Finally, inter-layer connections were added between the sensor nodes of the `sensor layer` and the navigation nodes of the base layer that are located within the coverage area of their respective sensor. It is important to note that the four simulated coverage regions are for simplified demonstration purposes only. In an eventual deployment of the system for a building of this magnitude, a significantly larger number of sensors would be required which would result in smaller coverage areas and more accurate evacuation paths.

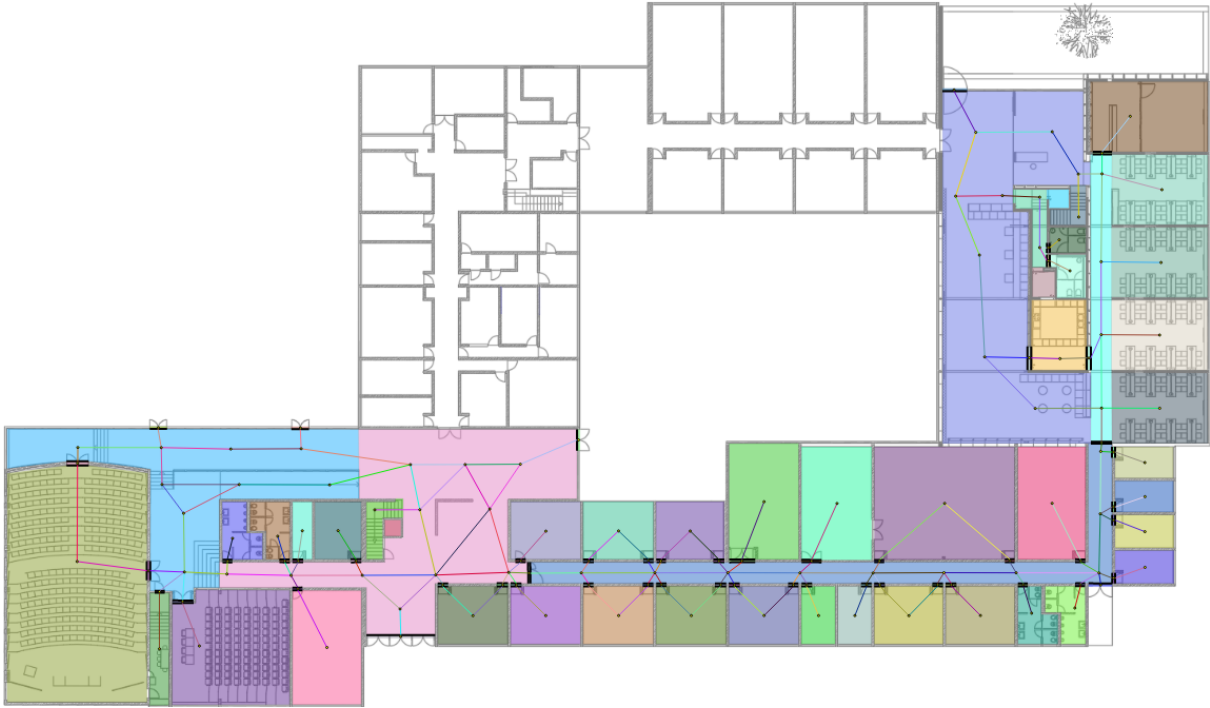


Figure 5.1: IndoorGML base layer model

5.2 System demonstration

Since testing the system by placing physical sensors in the department creates complex problems with too high acquisition costs and installation time, which is not compatible with the limited development time of the work, the transmission of readings monitored by the sensors was simulated, in order to facilitate testing procedures.

The initial process of setting up a building involves creating it in the frontend component by filling out a short form with trivial information about the building, along with the IndoorGML canvas width/length measurements, the actual measurements in meters of the building, and finally the submission of an IndoorGML file defining an internal navigation network for the building. After the building has been created, or if there is one already saved in the database, it is necessary to select one of the buildings shown in the building list to be able to start the evacuation simulation process. Figures 5.2 and 5.3 show the process of creating and selecting a building in the frontend component.

The screenshot shows the 'WAY OUT - Configuration' interface. At the top, there's a red header with the application logo and name. Below it, a yellow bar offers two options: 'SELECT BUILDING FROM DATABASE' or 'CREATE NEW BUILDING'. The main area is split into two panels. The left panel, titled 'BUILDING LIST', displays 'No buildings Found'. The right panel, titled 'NEW BUILDING FORM', contains several input fields: 'Building name' (filled with 'Departamento de Ciências de Computadores'), 'Building name abbreviation' (filled with 'DCC'), 'Country' (filled with 'Portugal'), 'City' (filled with 'Porto'), 'Canvas length' (filled with '400'), 'Canvas width' (filled with '300'), 'Building length' (filled with '101.09'), and 'Building width' (filled with '60.03'). At the bottom of the form are three buttons: a red 'Reset Form' button, a yellow 'Upload IndoorGML file' button, and a green 'Create Building' button.

Figure 5.2: Building creation form

This screenshot shows the same 'WAY OUT - Configuration' interface after a building has been successfully created. A green success message box in the top right corner reads '[SUCCESS] Successfully created a new building!'. In the 'BUILDING LIST' panel, the text 'No buildings Found' has been replaced with details for the 'DCC' building: 'Departamento de Ciência de Computadores', 'Length: 101.09m | Width: 60.03m', and 'Nodes: 136 | Sensors: 4 | Emergency exits: 6'. A green button with a right-pointing arrow is visible below this information. The 'NEW BUILDING FORM' panel remains the same as in the previous figure, with all input fields filled and the same three action buttons at the bottom.

Figure 5.3: Database building list

After selecting a building by clicking the green button, the web application redirects to the building's configuration page, shown in image 5.5, to prepare the evacuation process. This new page allows the user to submit a *PNG* image of a building floor (for path illustration purposes in the mobile application), simulate temperature readings for the building's sensors and initialize safe path calculations. In order to test the evacuation process with one area of the building with a higher danger level, a high temperature monitoring was simulated in the region monitored by `VIRTUAL_SENSOR_3` (red area in figure 5.4), indicating a danger level 3 to the server. All the other areas remain with danger level 0.



Figure 5.4: Simulated sensor coverage areas

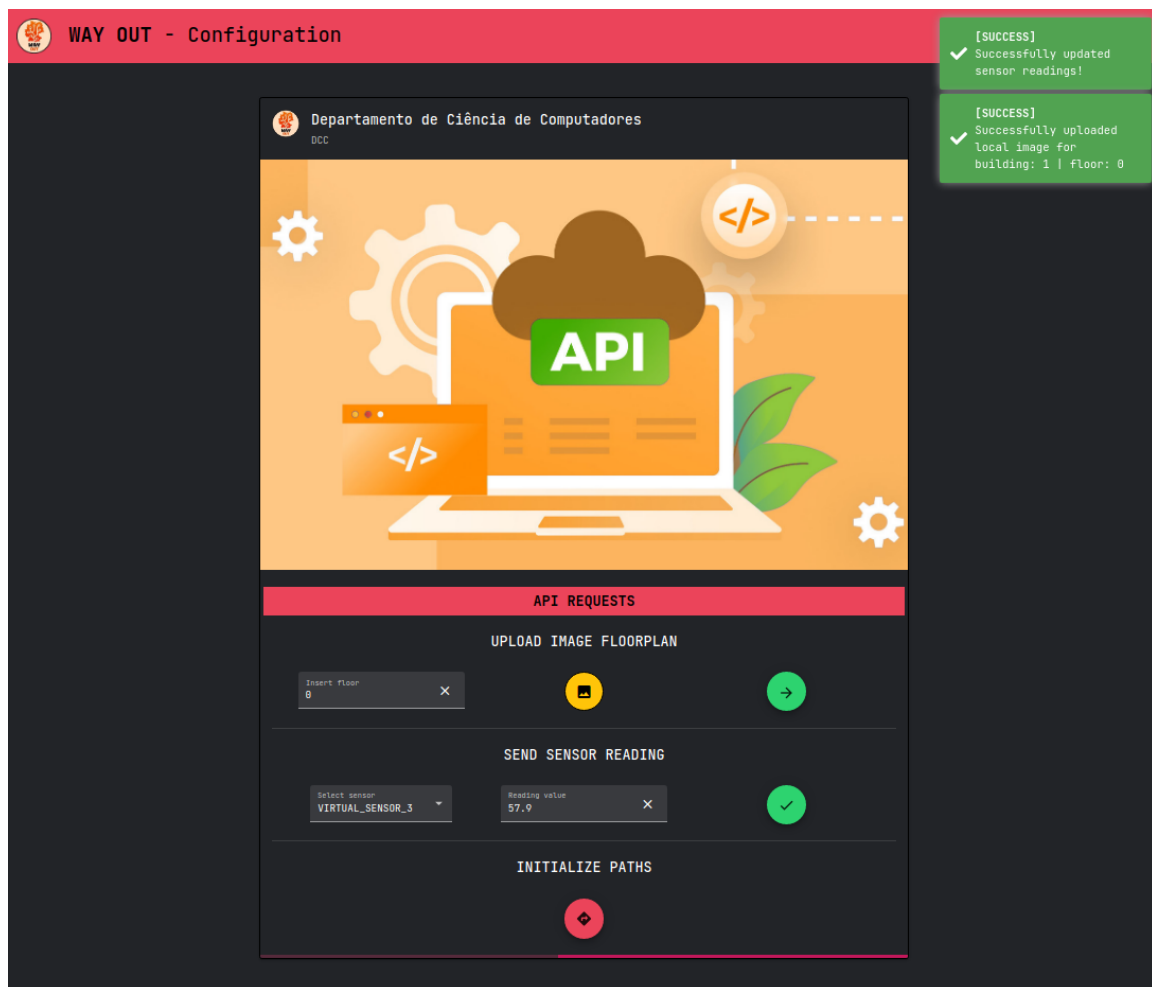


Figure 5.5: Building configuration section

Note that the transmission of simulated monitoring can be sent either via the configuration page shown in figure 5.5, which will send a request to the server that will later publish the sensor reading in the Kafka *broker topic*, or directly from a system terminal running the Kafka producer client, using the following command and sensor information:

```
$ bin/kafka-console-producer.sh --topic sensors --bootstrap-server localhost:9092
> { "sensorId": 447, "data": 57.9 }
```

Listing 5.1: Kafka producer environment

Whenever there are updates of danger levels given by the sensors, the evacuation path search algorithm will reset, so that the server always has in memory the current safest paths for all nodes of the building's navigation network.

When the mobile app starts, it first makes a request to the server to receive all the images of the submitted floors and stores them in memory. Then, after the initialization of the evacuation paths, the user's mobile application can send a request to the server, indicating what floor the user is on and its simulated coordinates in meters, to receive the safest route recommended by the backend component, which is then illustrated over the image canvas. Figure 5.6 shows the *views* of the user's mobile application.

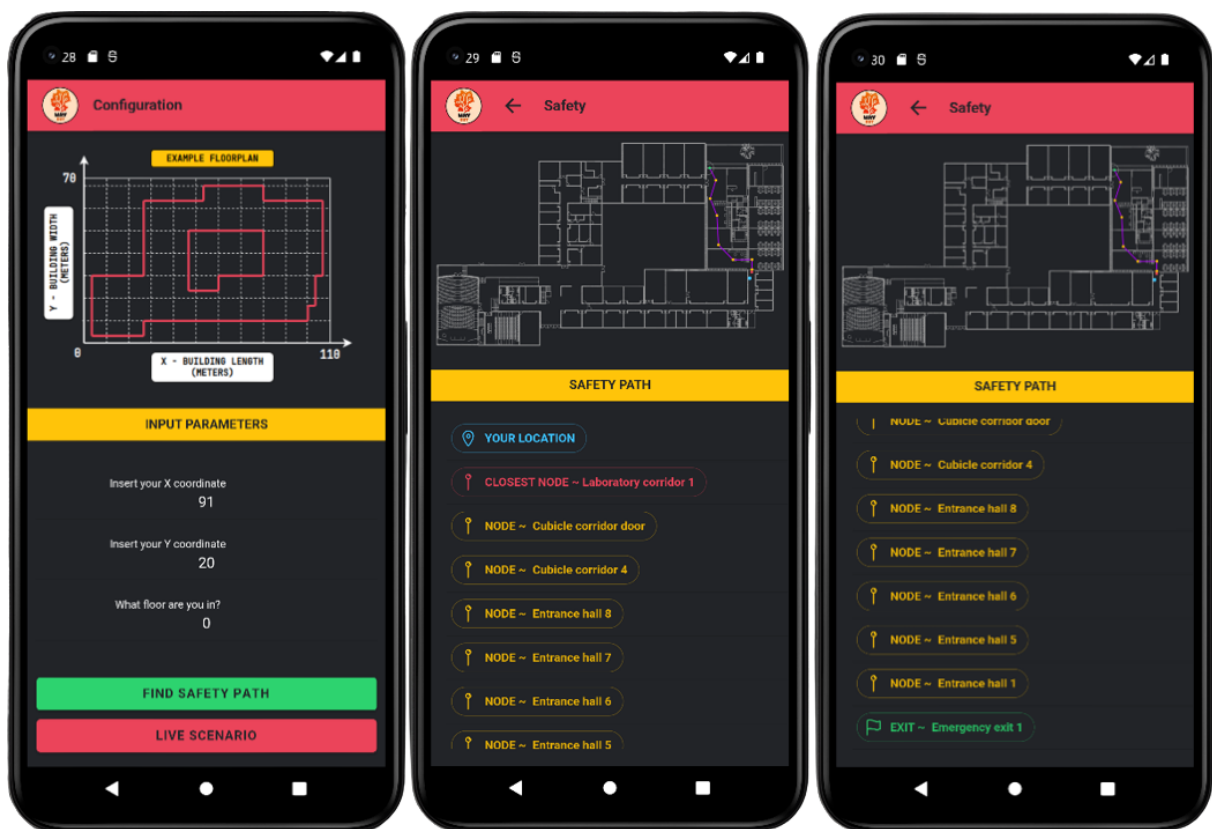


Figure 5.6: Mobile app *views* (deployed in an Android Pixel 5 emulator)

Chapter 6

Conclusion

In this work, a dynamic system for building evacuation was developed, based on the processing of building information given by a standard indoor mapping format. This system allows the simulation of sensor monitored temperature readings, enabling the assigning of danger levels inside the building. Subsequently, an algorithm was developed to search for the best evacuation paths, with the goal to find the safest path from the location of a user to one of the building's emergency exits.

During the first phase of the work, an exhaustive research on existing solutions for this subject was carried out, which made it possible to form the basis for the beginning of this project. Existing standard formats for indoor mapping were also researched, which allowed for the definition and characterization of the various areas existing in the building's internal structure.

The next step was to analyze which indoor mapping format would be most suitable for the use case of this project. After more in-depth research, the IndoorGML format was chosen. This standard allowed to define internal areas of the building in an intuitive way and also enabled the formation of a connectivity network of the building. The information extracted from an IndoorGML file allowed the developed server to form a corresponding graph of the building's navigation network, where each node indicates a possible location of the user and each edge indicates a path between two nodes that the evacuee can travel through. IndoorGML also supports Multi-Layered Representation (MLR), since an indoor space can have semantically different interpretations. This allowed the creation of a sensor layer to indicate coverage areas for virtual sensors, responsible for sending simulated environment temperature readings. The simulated values given by these sensors allowed the server to assign a certain danger level to the entire coverage area of the respective sensor, and consequently to all nodes located within that area. This information is mapped onto the graph representation of the building, so that the chosen algorithm is able to compute evacuation paths that try to prevent the user from navigating through dangerous zones in the building's internal area.

Next, a study was conducted on the appropriate best-path search algorithms, taking into account the metrics to be used. The chosen implementation was based on the Dijkstra algorithm, which allowed to create a tree of best paths for all nodes in the tree. However, instead of choosing the path with the shortest distance, preference was given to paths constituted by nodes with the lowest level of danger, thus ensuring that the user is exposed to danger for the shortest possible time.

Finally, a frontend component for building creation and configuration was developed, as well as a mobile application capable of communicating with the server to illustrate the evacuation path calculated by the server from the user's simulated location to one of the emergency exits of the building.

6.1 Future Work

Although the project presented satisfactory and pleasant results, it would be possible to develop improvements that could not be integrated into the project due to lack of time for implementation. Thus, some of the possible improvements for future work will be mentioned in this subsection.

6.1.1 Computational representation of the building

Although this project relies on the use of an IndoorGML format, it would also be interesting to integrate the possibility of accepting the Indoor Mapping Data Format (**IMDF**) format, developed by Apple, for the process of creating a building and developing navigable networks for it.

6.1.2 Additional metrics

The current proposed system only associates danger levels to coverage areas monitored by simulated temperature sensors. However, as mentioned in some of the related works, it would be interesting to create additional sensors for other environmental variables, such as smoke density or presence of toxic gases, that also impact, with more or less weight, the danger level assigning process, thereby creating more accurate and safe evacuation paths.

6.1.3 Danger level of a link

The calculation of the danger level of an area monitored by a sensor is given by an analysis performed of various temperature intervals. This danger level is then associated with all nodes that are located within the coverage area of a sensor, and thus it was defined that the danger level of a connection between two nodes is given by the highest danger level of these two nodes. However, this metric is not effective because there may have cases where the algorithm chooses a longer path than necessary only because the transition that would actually be better has a small

portion of its path affected by a higher danger zone. Therefore, a possible better solution, which was not implemented due to lack of time, would be to partition each transition by distance under each hazard level, thus allowing a more accurate calculation of evacuation paths.

6.1.4 User's location node

In this work, during the communication process between the user's mobile application and the server, the app sends the coordinates in meters of the user's location. The server consequently checks the coordinates received, iterates the list of existing nodes in the graph and associates the node that is at the shortest Euclidean distance to the user. However, this approach does not present appropriate results in instances where the user and the closest chosen node are separated by a wall. Therefore, it would be relevant to consider other possible solutions to circumvent this issue in the future.

6.1.5 API security

Another improvement to be done would be the implementation of authentication tokens to enable the access to the API. At the moment it was not possible to integrate this protocol and authentication due to lack of time and, therefore, the API's are not properly protected.

6.1.6 Fault tolerance

Finally, another possible improvement would be in choosing a different type of system architecture. The proposed system was developed with a centralized architecture in mind, since it is simpler to implement and control. However, in case of fires, it may happen that the central server, the sensor network or even the telecommunication infrastructure become compromised, so it would be relevant to consider other solutions, for example a distributed architecture, in order to make the system more robust and fault tolerant.

Bibliography

- [1] Fire safety in high risk buildings – preventing the next tragedy. *EATON*, 2019.
- [2] Angular. [The modern web developer’s platform](#). Last accessed in August 2022.
- [3] Apple. [Indoor mapping data format overview](#). Last accessed in May 2022.
- [4] Aucxis. [Rfid – identify, read and process unique information](#). Last accessed in September 2022.
- [5] BehrTech. [A new approach to indoor localization in large-scale environments](#), May 2021. Last accessed in September 2022.
- [6] Richard G. Gann Nelson P. Bryner. Combustion products and their effects on life safety. *SFPE Handbook of Fire Protection Engineering*, 3rd ed., 2002.
- [7] Eliseo Clementini and Andrea Pagliaro. The construction of a network for indoor navigation. pages 254–261, 01 2020.
- [8] Blockchain Engineer. [Centralized vs decentralized vs distributed](#), Jan 2019. Last accessed in August 2022.
- [9] Fatna Belqasmic Faouzi Kamouna, May El Barachib and Abderrazak Hachani. A smart dynamic crowd evacuation system for exhibition centers. 2021.
- [10] FCUP. [Fc6 edifício de ciência de computadores \(fc6\)](#). Last accessed in September 2022.
- [11] Avgoustinos Filippoupolitis and Erol Gelenbe. A distributed decision support system for building evacuation. 2009.
- [12] GeeksforGeeks. [A* search algorithm](#). Last accessed in August 2022.
- [13] GeeksforGeeks. [Comparison - centralized, decentralized and distributed systems](#), Sep 2021. Last accessed in August 2022.
- [14] Google. [What is a relational database \(rdbms\)?](#) Last accessed in July 2022.
- [15] IndoorGML. [Indoorgml core xml schema](#). Last accessed in May 2022.

-
- [16] ArcGIS Indoors. [Arcgis indoor mapping, wayfinding; space planning software](#). Last accessed in June 2022.
 - [17] Inpixon. [Ble \(bluetooth\) location tracking and positioning](#), . Last accessed in September 2022.
 - [18] Inpixon. [Ultra-wideband \(uwb\) positioning and sensor technology](#), . Last accessed in September 2022.
 - [19] Inpixon. [Wi-fi location tracking and positioning systems](#), . Last accessed in September 2022.
 - [20] Apache Kafka. [Apache kafka overview](#). Last accessed in June 2022.
 - [21] Yuka Kato. [A visualization of the world's countries by number of skyscrapers](#), Dec 2019. Last accessed in January 2022.
 - [22] Hugo Ledoux. [Val3dity: Geometric validation of 3d primitives](#). Last accessed in April 2022.
 - [23] Adalja Disha M. A comparative analysis on indoor positioning techniques and systems. *International Journal of Engineering Research and Applications (IJERA)*, 2013.
 - [24] MappedIn. [Mappedin indoor maps](#). Last accessed in June 2022.
 - [25] Hugh Leather Matthew Barnes. Emergency evacuation using wireless sensor networks. 2007.
 - [26] MongoDB. [What is a non-relational database?](#), . Last accessed in July 2022.
 - [27] MongoDB. [Relational database vs mongodb](#), . Last accessed in September 2022.
 - [28] Navigine. [Using angle of arrival for direction finding with bluetooth 5.1](#). Last accessed in September 2022.
 - [29] IndoorGML OGC. [Ogc standard for indoor spatial information](#). Last accessed in July 2022.
 - [30] Oracle. [What is a database?](#) Last accessed in July 2022.
 - [31] phoenixNAP. [What is a relational database?](#) Last accessed in July 2022.
 - [32] CAD Pro. [Building evacuation design plans: Building evacuation guidelines](#). Last accessed in January 2022.
 - [33] Programiz. [Dijkstra's algorithm](#), . Last accessed in August 2022.
 - [34] Programiz. [Floyd-warshall algorithm](#), . Last accessed in August 2022.
 - [35] V. Rajaravivarma, Yi Yang, and Teng Yang. [An overview of wireless sensor network and applications](#). pages 432–436, 2003. doi:10.1109/SSST.2003.1194607.
 - [36] Spring. [Spring boot framework overview](#). Last accessed in May 2022.
 - [37] SQLCourse.com. [What is sql \(structured query language\)?](#), Feb 2022. Last accessed in July 2022.

-
- [38] Bertly Technologies. [Centralized vs decentralized vs distributed systems](#) · bertly technologies, Jun 2019. Last accessed in August 2022.
 - [39] Zoiner Tejada. [Non-relational data and nosql - azure architecture center](#). Last accessed in July 2022.
 - [40] Kasim Ozacar. Emrullah Demiral Umit Atila, Yasin Ortakci and Ismail Rakip Karas. Smartescape: A mobile smart individual fire evacuation system based on 3d spatial model. *International Journal of Geo-Information*, 2018.
 - [41] Soni Upadhyay. [Bellman-ford algorithm: Pseudocode, time complexity and examples: Simplilearn](#), Nov 2021. Last accessed in August 2022.
 - [42] Upwork. [Nosql vs. sql: Important differences and which one is best for your project](#). Last accessed in July 2022.
 - [43] Wikipedia. [Geojson](#), . Last accessed in May 2022.
 - [44] Wikipedia. [Ionic \(mobile app framework\)](#), . Last accessed in July 2022.
 - [45] Wikipedia. [Shortest path problem](#), Sep 2022. Last accessed in August 2022.
 - [46] SungJin Yu and YoungHo Park. [Slua-wsn: Secure and lightweight three-factor-based user authentication protocol for wireless sensor networks](#). *Sensors*, 20(15), 2020. ISSN: 1424-8220. doi:10.3390/s20154143.