

ΑΝΑΠΤΥΞΗ ΠΛΑΤΦΟΡΜΑΣ ΨΗΦΙΑΚΗΣ ΚΑΘΟΔΗΓΗΣΗΣ

του

ΠΑΞΙΜΑΔΑΚΗΣ ΔΗΜΗΤΡΙΟΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

υποβληθείσα σε μερική εκπλήρωση των απαιτήσεων για το πτυχίο

ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ



ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΕΛΛΗΝΙΚΟ ΜΕΣΟΓΕΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

2023

Εγκεκριμένη από:

Δρ. Βιδάκης Νικόλαος

DEVELOPMENT OF A DYNAMIC VIRTUAL GUIDE PLATFORM

by

PAXIMADAKIS DIMITRIOS

A THESIS

submitted in partial fulfillment of the requirements for the degree

BACHELOR OF INFORMATICS ENGINEERING



DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SCHOOL OF ENGINEERING

HELLENIC MEDITERRANEAN UNIVERSITY

2023

Approved by:

Dr. Vidakis Nikolaos

## Περίληψη

Η πλοήγηση σε πολύπλοκες διατάξεις μιας πανεπιστημιούπολης παρουσιάζει σημαντικές προκλήσεις για φοιτητές, καθηγητές και επισκέπτες. Παρά τις εξελίξεις στην τεχνολογία πλοήγησης, εξακολουθεί να υπάρχει ανάγκη για μια ολοκληρωμένη λύση προσαρμοσμένη στις ειδικές απαιτήσεις των πανεπιστημιούπολεων. Σε αυτή τη πτυχιακή εργασία, παρουσιάζουμε μια πλατφόρμα εικονικού οδηγού σχεδιασμένη ειδικά για την πλοήγηση στις πανεπιστημιούπολεις. Αυτή η πλατφόρμα συνδυάζει τεχνολογία πλοήγησης αιχμής με μια φιλική προς τον χρήστη διεπαφή, επιτρέποντας στους χρήστες να βρίσκουν αβίαστα προορισμούς και να αποκτούν πληροφορίες σχετικά με το περιβάλλον τους. Αντιμετωπίζοντας τόσο τις λειτουργικές όσο και τις συμφραζόμενες πτυχές της πλοήγησης στην πανεπιστημιούπολη, αυτή η καινοτόμος προσέγγιση επιδιώκει να βελτιώσει τη συνολική εμπειρία της πανεπιστημιούπολης για όλους τους χρήστες. Θα διερευνήσουμε το σχεδιασμό, την υλοποίηση και τα πιθανά πλεονεκτήματα αυτής της πλατφόρμας, χρησιμοποιώντας την Πανεπιστημιούπολη του Ελληνικού Μεσογειακού Πανεπιστημίου ως μελέτη περίπτωσης για να αναδείξουμε τις δυνατότητές της στη βελτίωση των αλληλεπιδράσεων και στην εξερεύνηση των πανεπιστημιούπολεων. Μέσα από αυτό το πραγματικό παράδειγμα, θα δείξουμε την αποτελεσματικότητα της προτεινόμενης λύσης στην αντιμετώπιση των μοναδικών προκλήσεων της πλοήγησης στην πανεπιστημιούπολη, προωθώντας τελικά μια πιο απρόσκοπτη και εμπλουτισμένη ακαδημαϊκή εμπειρία.

**Λέξεις κλειδιά:** εικονικός οδηγός, σύστημα εντοπισμού θέσης, GPS, πλοήγηση, διεπαφή χρήστη, αλληλεπίδραση χρήστη-υπολογιστή (HCI)

## Abstract

Navigating complex university campus layouts presents significant challenges for students, faculty, and visitors. Despite advancements in navigation technology, there is still a need for a comprehensive solution tailored to the specific requirements of university campuses. In this thesis, we introduce a virtual guide platform designed specifically for navigating university campuses. This platform combines cutting-edge navigation technology with a user-friendly interface, enabling users to effortlessly find destinations and acquire contextual information about their surroundings. By addressing both functional and contextual aspects of campus navigation, this innovative approach seeks to enhance the overall campus experience for all users. We will explore the design, implementation, and potential advantages of this platform, employing the Hellenic Mediterranean University Campus as a case study to showcase its potential in improving interactions with and exploration of university campuses. Through this real-life example, we will illustrate the efficacy of our proposed solution in tackling the unique challenges of campus navigation, ultimately promoting a more seamless and enriching academic experience.

**Keywords:** virtual guide, positioning system, GPS, navigation, user interface, human-computer interaction (HCI)

## **Acknowledgements**

I'm grateful to my supervising committee, Dr. Nikolaos Vidakis and Ilias Logothetis, for their exceptional guidance and unwavering support. Their expertise and encouragement were crucial to the success of this project. Thanks to Hellenic Mediterranean University's staff and faculty, especially Alexandros, for providing resources and facilities that helped me to achieve my goals. I'm grateful for my family's unwavering love and support that allowed me to pursue my dreams and achieve this milestone. I would also like to recognize the support of my friends and colleagues from NILE Lab, whose encouragement and motivation have been instrumental in helping me reach this important milestone. Special recognition goes to Harris, Myron, and Lorida, whose support was invaluable throughout this journey. Finally, I would like to express my appreciation to all those who have played a part in this thesis, either directly or indirectly.

# Table of Content

Περίληψη.....	iii
Abstract .....	iv
Acknowledgements .....	v
Chapter 1 - Introduction .....	1
1.1 Objectives .....	1
1.2 Outline .....	2
Chapter 2 - Background Work.....	3
2.1 Outdoor Positioning Methods .....	3
2.2 Indoor Positioning Methods.....	4
2.3 Shortest path algorithms .....	6
Chapter 3 - Similar Work .....	9
3.1 URWalking .....	9
3.2 ARCore Augmented Reality Navigation System.....	10
3.3 Bluetooth Beacon-based Indoor Navigation System for Android .....	12
3.4 BookMark .....	12
Chapter 4 - System Analysis and Design .....	13
4.1 Problem Description.....	14
4.2 System Analysis .....	14
4.2.1 Platform’s scenarios of use .....	15
4.3 System Design.....	18
4.3.1 Component Diagram.....	18
4.3.2 Flow Diagrams .....	20
Chapter 5- Implementation.....	25
5.1 Technologies used.....	25
5.1.1 Quantum Geographic Information System (QGIS).....	25
5.1.2 – PostgreSQL with PostGIS extension .....	25
5.1.3 – Node / Express.....	26
5.1.4 – Typescript.....	26

5.1.5 React Native with React Native Maps .....	26
5.2 Software Implementation.....	27
5.2.1 Database .....	27
5.2.2 Backend.....	39
5.2.3 Frontend .....	43
Chapter 6 - Results / Demo .....	57
6.1 Loading Screen.....	57
6.2 Home Screen.....	58
6.3 Map screen .....	59
6.4 Settings screen.....	62
6.5 Information screen.....	63
6.6 Help screen.....	64
Chapter 7 - Conclusion and future work .....	65

## List of Figures

Figure 1: Triangulation technique [5].	3
Figure 2: Hierarchical classification of indoor navigation systems based on adopted positioning technology [7].	5
Figure 3: Screenshot from URWalking [27].	10
Figure 4: Interface collections of the proposed system (a–d) and Gaode AR system (e–h) in outdoor [3].	11
Figure 5: One of BookMark screenshots, showing the system's various capabilities, and the steps involved in finding items within the library [30].	13
Figure 6: Component diagram.	20
Figure 7: Search location flow diagram.	21
Figure 8: Navigation flow diagram.	22
Figure 9: Location's information access flow diagram.	23
Figure 10: User with accessibility needs navigation flow diagram.	24
Figure 11: QGIS Browser and Layers panels.	28
Figure 12: Preview of Hellenic Mediterranean University's campus filled with geospatial data.	29
Figure 13: Add connection panel.	30
Figure 14: POINT table structure.	30
Figure 15: CONNECTION table structure.	31
Figure 16: POI table structure.	31
Figure 17: Visual representation of a directed graph.	32
Figure 18: Representation of editing geospatial data of a floor.	34
Figure 19: Selecting the data with rubber-banding.	35
Figure 20: After the moving the geospatial data.	35
Figure 21: Connection to database code snippet.	36
Figure 22: Queries of the node.js script.	37
Figure 23: Function that update geospatial data code snippet (a).	38
Figure 24: Function that update geospatial data code snippet (b).	38
Figure 25: An example of a DAO code snippet. In this case RoomDAO module is shown.	42
Figure 26: Map screen of the platform – (a) Search input field. (b) Navigation input fields.	44



Figure 27: Function showMarker code snippet. ....	45
Figure 28: Geospatial data formation function formatDoorData code snippet. ....	46
Figure 29: Geospatial data formation function formatFloorData code snippet. ....	46
Figure 30: Geospatial data formation function formatRoomData code snippet. ....	47
Figure 31: Navigation use cases. ....	48
Figure 32: The path symbols. ....	51
Figure 33: Function createPath code snippet(a). ....	52
Figure 34: Function createPath code snippet(b). ....	53
Figure 35: Function PlotPathWithClosestNode code snippet. ....	54
Figure 36: Function filterIconNodes code snippet(a). ....	54
Figure 37: Function filterIconNodes code snippet(b). ....	55
Figure 38: Loading screen screenshots. ....	58
Figure 39: Home screen screenshot. ....	59
Figure 40: Map screen iteration screenshots when user searches a destination. ....	60
Figure 41: Map screen iteration screenshots when user wants to navigate. ....	61
Figure 42: Map screen screenshot when user has pressed the marker of a searched location. ....	61
Figure 43: Settings screen screenshot. ....	62
Figure 44: Information screen screenshot the room is reservable. ....	63
Figure 45: Information screen when the room is not reservable and has no data. ....	64
Figure 46: Help screen screenshots. ....	65

## List of Tables

Table 1: Folder structure of the backend part of the platform.

39

# Chapter 1 - Introduction

In today's modern society, navigation has made remarkable advances. The widespread use of mobile phones and the expansion of the internet [1], have aided in the creation of various virtual guide platforms, such as [2], making them widely available to the public. However, navigating particular spaces, like sprawling university campuses, still remains a challenge. As the number of university students continues to rise, campuses have expanded, with more buildings and facilities being constructed to accommodate the growing population. Consequently, newcomers, including first-year students and visitors, often find it difficult to locate specific buildings or areas within these vast and intricate spaces[3]

Furthermore, both students and visitors may wish to gain an understanding of what to expect and how to safely navigate these new environments. Beyond the practical aspects of navigation, individuals might also be interested in exploring the contextual aspects of the places they encounter on campus. This could involve learning about various campus buildings, as well as obtaining information on class schedules and events taking place within those facilities. By providing a more comprehensive understanding of their surroundings, users can better appreciate and engage with the campus community and its rich array of offerings.

In light of these challenges, there is a clear need for a comprehensive navigation solution that caters to the unique requirements of university campuses. In response, we propose a virtual guide platform designed specifically for this purpose. This solution would ideally combine advanced navigation technology with a user-friendly interface, enabling users to seamlessly locate their destinations while also offering insights into the campus. By addressing both the practical and contextual aspects of campus navigation, this innovative approach has the potential to significantly enhance the overall campus experience for students, faculty members, and visitors alike.

## 1.1 Objectives

The primary objective of this thesis is to design and develop a comprehensive virtual guide platform that not only offers a free of cost solution to develop and publish but also delivers

seamless guidance for users navigating from one location to another. By leveraging cutting-edge technology and user-centered design principles, the platform aims to empower individuals to make choices, enhance their overall experience, and navigate their surroundings with ease and confidence.

To achieve this, the platform will provide users with essential information about searched locations, accessibility features, and other relevant details that contribute to a more informed and enjoyable navigation experience. By doing so, users can better plan their routes, make efficient use of their time, and discover new places with minimal hassle. Moreover, a key consideration in the development of this platform is data scalability and ease of maintenance. The system will be designed to accommodate the addition of new information effortlessly, ensuring that the platform remains up-to-date and relevant.

In summary, this thesis aims to create a versatile and user-friendly virtual guide platform that not only offers free creation and publication but also provides users with a seamless and informative navigation experience. By prioritizing data scalability and ease of maintenance, the platform will remain adaptable and relevant, ultimately enhancing the overall navigation experience for a diverse range of users and environments.

## **1.2 Outline**

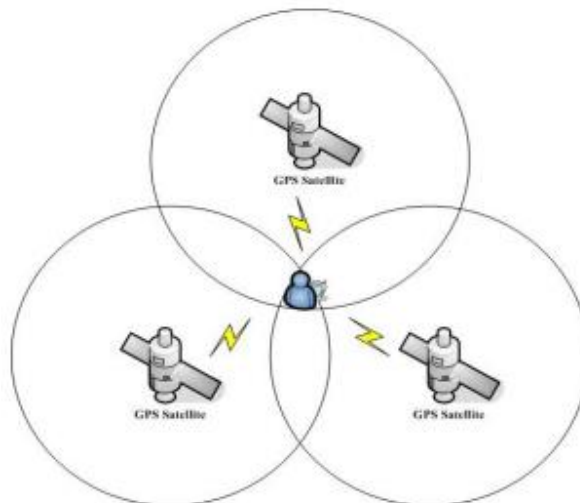
The current thesis report is organized into seven main chapters, beginning with Chapter 1, which introduces the virtual guide platform. Chapter 2 covers the theoretical and technological background, while Chapter 3 discusses related work and comparisons. Chapter 4 delves into the platform design process and rationale, and Chapter 5 focuses on implementation, technologies used, and software details. Chapter 6 showcases the final outcomes of the thesis through a brief demo, and finally, Chapter 7 concludes the report with a discussion of the findings and implications.

## Chapter 2 - Background Work

In this section of the thesis, the objective is to build a solid foundation by presenting the necessary theoretical and technological concepts that are fundamental to the research. We will explore key topics such as outdoor positioning, indoor positioning, and shortest path algorithms. This thorough examination will facilitate a more profound understanding of the subject matter.

### 2.1 Outdoor Positioning Methods

Navigation is the field of study concerned with the methods and techniques used to guide a person or vehicle from one location to another[4]. Outdoor navigation refers to navigation in outdoor environments. The issue of outdoor positioning has been addressed for over a decade now, thanks to the Global Positioning System (GPS) developed by the U.S. government. GPS uses a network of 24 satellites orbiting the Earth to transmit signals. When a GPS receiver on the ground receives signals from at least three of these satellites, it calculates the distance between the receiver and each satellite using the TOA (Time of Arrival) method, with the satellite as the center of a circle. The receiver then determines its location by finding the intersecting points of three or more circles, using triangulation. GPS is the most commonly used technology for outdoor location services [5][6], allowing the implementation of outdoor navigation systems.



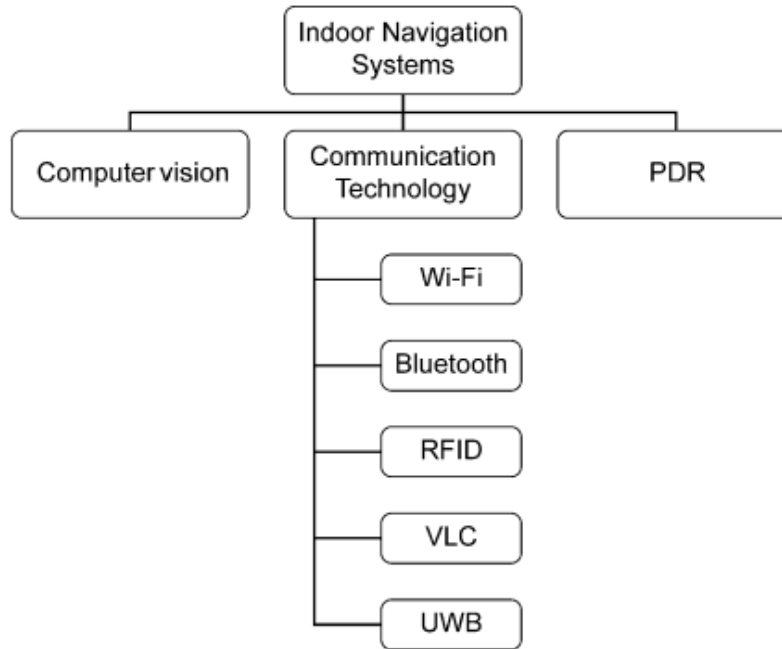
**Figure 1:** Triangulation technique [5].

## 2.2 Indoor Positioning Methods

Indoor positioning has become an essential research domain due to the constraints of GPS in such environments. While GPS is effective and dependable for outdoor positioning and navigation purposes, its performance diminishes indoors as building materials, walls, and other structures obstruct direct line-of-sight signals [6][7][8][9][10]. As a result, GPS fails to provide the necessary accuracy for indoor tracking and positioning. This has prompted the investigation and advancement of alternative technologies and approaches to address these challenges and improve indoor positioning capabilities, ultimately rendering indoor navigation more accurate and robust.

Systems based on computer vision rely on an array of cameras, such as omnidirectional, 3D, and embedded smartphone cameras, to collect data about indoor settings. Methods like Speeded Up Robust Feature (SURF), Gist features, and Scale Invariant Feature Transform (SIFT) are utilized for extracting and comparing features. Additionally, vision-based positioning and navigation systems often incorporate clustering and matching algorithms. In recent developments, deep learning strategies have been applied to these systems, enhancing the efficiency of object identification and classification. Moreover, egomotion-based techniques have been employed to estimate a camera's relative position within its environment [7].

Pedestrian Dead Reckoning (PDR) methodologies have emerged as a promising approach for determining a user's position by carefully analyzing previous locations and drawing on data gathered from a variety of sensors such as accelerometers, gyroscopes, and magnetometers [11][12]. However, the inherent drift in PDR methods frequently results to positional inaccuracies [12]. To address this issue, researchers and engineers are actively creating advanced navigation systems that integrate supplementary positioning technologies or make better use of the power of machine learning algorithms [11]. It is anticipated that by constantly refining these methods, PDR-based systems will become more accurate and reliable, providing a robust solution for a variety of applications that require precise positioning.



**Figure 2:** Hierarchical classification of indoor navigation systems based on adopted positioning technology [7].

Communication-based methodologies for indoor positioning encompass technologies such as RFID, Wi-Fi, visible light communication (VLC), UWB, and Bluetooth. RFID systems are composed of an RFID reader and tags affixed to objects, with two distinct types of tags available: active and passive. The majority of contemporary RFID-based navigation systems employ passive tags due to the elimination of the need for an external power source. Position estimation in RFID-based systems relies on parameters such as Received Signal Strength (RSS), Angle of Arrival (AOA), Time of Arrival (TOA), and Time Difference of Arrival (TDOA). RFID technologies are widely implemented in navigation systems because of their simplicity, cost efficiency, and long effective ranges.

Wi-Fi-based indoor navigation systems leverage RSS fingerprinting or triangulation/trilateration methods for positioning [7]. Wi-Fi-based indoor positioning aims to identify and track locations in intricate settings by leveraging a wireless local area network consisting of wireless access points, such as wireless routers. This approach integrates hands-on experiments and signal propagation models to ascertain the position of connected mobile devices, taking into account the spatial data of network nodes [13]. Consequently, this technique is particularly well-suited for application within building infrastructures where Wi-Fi access points are readily available and easily accessible.

Following that, Bluetooth-based systems are also employed for positioning purposes. iBeacon technology, a prominent Bluetooth positioning solution, relies on Bluetooth 4.0 and boasts low power consumption, accelerated connection speeds, high data transfer rates, secure and stable signal transmission, and a lack of interference. In an indoor positioning system utilizing low-power Bluetooth technology, the Bluetooth beacon is initially placed within the designated area, where it continuously broadcasts signals and data packets to its surroundings. The Received Signal Strength Indicator (RSSI) value is then harnessed to determine the precise location using the mobile phone's integrated positioning algorithm [13][14]. Leveraging a user's mobile phone for position calculation BLE offers a more cost-effective approach than other methods [14][15], while the simplicity of Bluetooth positioning implementation and its accuracy, which is closely tied to the deployment density and transmission power of Bluetooth beacons, serve as additional advantages [13].

VLC-based systems take advantage of the ubiquity of LED or fluorescent lamps present within a building's infrastructure. These lighting sources are becoming increasingly widespread in indoor settings [7]. A VLC-based system can be implemented using sensors available in modern smartphones. Owing to the pervasive presence of LED lamps and smartphones, VLC-based systems have emerged as a cost-effective alternative for indoor positioning systems, accompanied by additional benefits such as license-free spectrum utilization and immunity to RF interference [16].

Ultra-wideband is a wireless technology that uses radio frequency pulses with a very short time duration (picoseconds to nanoseconds) to transmit information. Short pulses result in a large bandwidth and, therefore, spread the energy of the signal over a large frequency band. Advantageous properties of UWB include its excellent robustness against shadowing, good multipath resolution, large channel capacity (high data rates) and low energy consumption. UWB technology is quickly becoming the dominant technology in the indoor positioning market [17].

### **2.3 Shortest path algorithms**

To enable navigation within the platform, an algorithm must identify the path connecting the user's two positions, with a preference for selecting the shortest path. These algorithms play a pivotal role in determining the most efficient route between two points, optimizing travel time and



resource usage. Route planning is a critical aspect of navigation systems that has become increasingly important in today's fast-paced world. There are several methods that are commonly used for route planning, each with its own strengths and weaknesses [7], which will be discussed below.

One widely used algorithm for route planning is A\* (pronounced "A star"), which is a heuristic search algorithm commonly used in route planning and pathfinding [18][19]. A\* uses heuristics to guide the search process towards the goal, reducing the number of nodes that need to be explored. The algorithm maintains a priority queue of nodes to be explored, where each node is assigned a cost based on its distance from the start node and an estimated cost to the goal. A\* is known to be both complete and optimal under certain conditions, and the heuristic function used in A\* must be admissible, meaning that it must never overestimate the cost to reach the goal [20].

Dijkstra's algorithm is a widely used technique for finding the shortest path between two points in a connected graph with nonnegative edge weights [21]. The algorithm maintains a set of vertices with known shortest-path distances from the source node and a min-priority queue of vertices, sorted by their distance values. It iteratively selects the vertex with the smallest distance estimate, adds it to the set, and relaxes all its outgoing edges. The algorithm begins by initializing the source node and an empty set. It then repeatedly chooses the lightest or closest vertex not yet included in the set and adds it to the set. Next, it relaxes each edge leaving the selected vertex, updating distance estimates and predecessors if a shorter path is discovered. This process continues until the shortest path to the target node is determined. By employing a greedy strategy and consistently updating distance estimates and predecessors, Dijkstra's algorithm effectively computes the shortest paths in a graph [22][23].

D\* (pronounced "D star") algorithm is a popular method for route planning that was designed to handle dynamic environments where the graph is constantly changing. The algorithm maintains a grid-based representation of the environment, where each cell represents a node in the graph. When the environment changes, D\* updates the costs of the affected nodes and re-plans the path. D\* uses heuristics to guide the search process towards the goal, similar to A\*, but also has a mechanism for handling changes to the graph[24]. Therefore, D\* is particularly useful in robotics applications, where the environment can be highly dynamic and unpredictable.

Finally, Floyd's algorithm, also known as the Floyd–Warshall algorithm, maintains a matrix of shortest distances between each pair of vertices and iteratively updates the matrix,

considering all intermediate vertices that could potentially shorten the path [25]. Floyd's algorithm is known to be both complete and optimal, but it has a higher time complexity than other algorithms such as Dijkstra's algorithm. Floyd's algorithm is particularly useful in cases where it is necessary to find the shortest paths between all pairs of vertices, rather than just between two specific vertices[26].

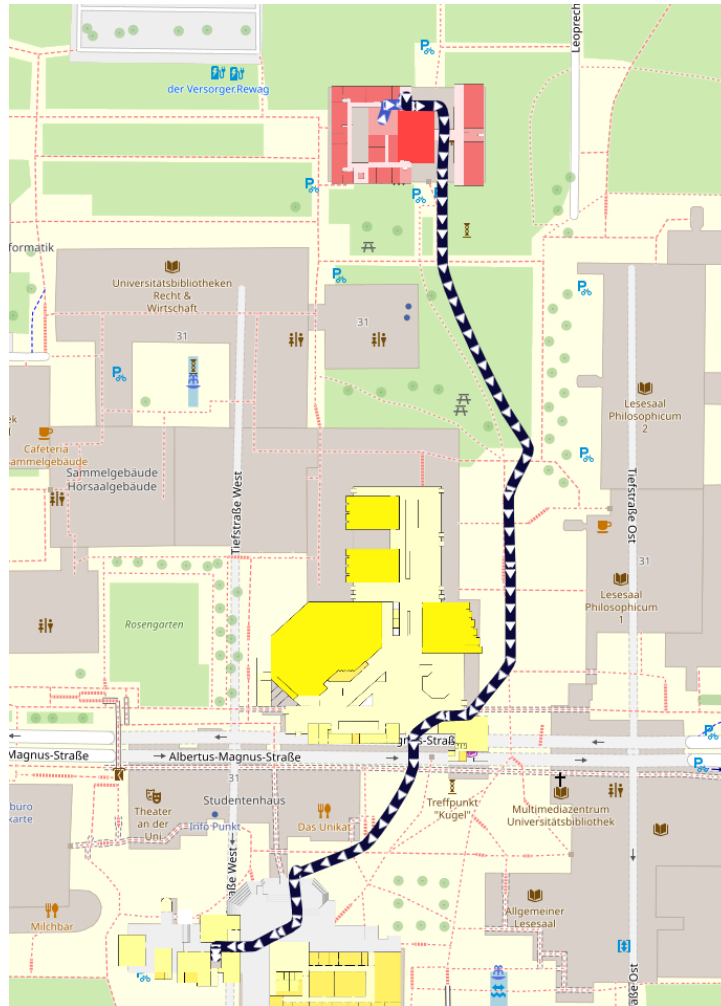
## Chapter 3 - Similar Work

This chapter examines the landscape of virtual guide platforms, with an emphasis on those tailored to particular environments such as university campuses, commercial complexes, and urban centers. We will look at studies and projects that used advanced technology, innovative design, and a user-centered strategy to improve navigation experiences.

This chapter's main goal is to provide a contextual knowledge of existing virtual guide platforms, identify successful features and functionalities, and identify potential areas for improvement. By reviewing similar works, we can gain insights into the challenges and opportunities in developing our virtual guide platform, as well as learn from previous efforts' overcomes and limitations.

### 3.1 URWalking

The URWalking project at the University of Regensburg is an indoor navigation system that combines practical navigation with wayfinding research, focusing on the University of Regensburg Campus. Consisting of a web server, a web client, an Android application, and the YAMA editing tool, the project has developed a hybrid mapping concept for indoor environments that can be customized for any space. Surprisingly, the system has shown that users value real-time indoor navigation, even in the absence of indoor positioning. Users appear to favor the absence of positioning rather than dealing with a subpar system. URWalking aims to collect large datasets to improve indoor positioning algorithms in the future. The system has been used in various studies to collect eye-tracking data, think-aloud protocols, and log data, offering valuable insights into human wayfinding behavior. Despite its success, real-time indoor navigation remains a challenge, particularly in understanding natural language descriptions of users' destinations and tracking human wayfinding behavior.



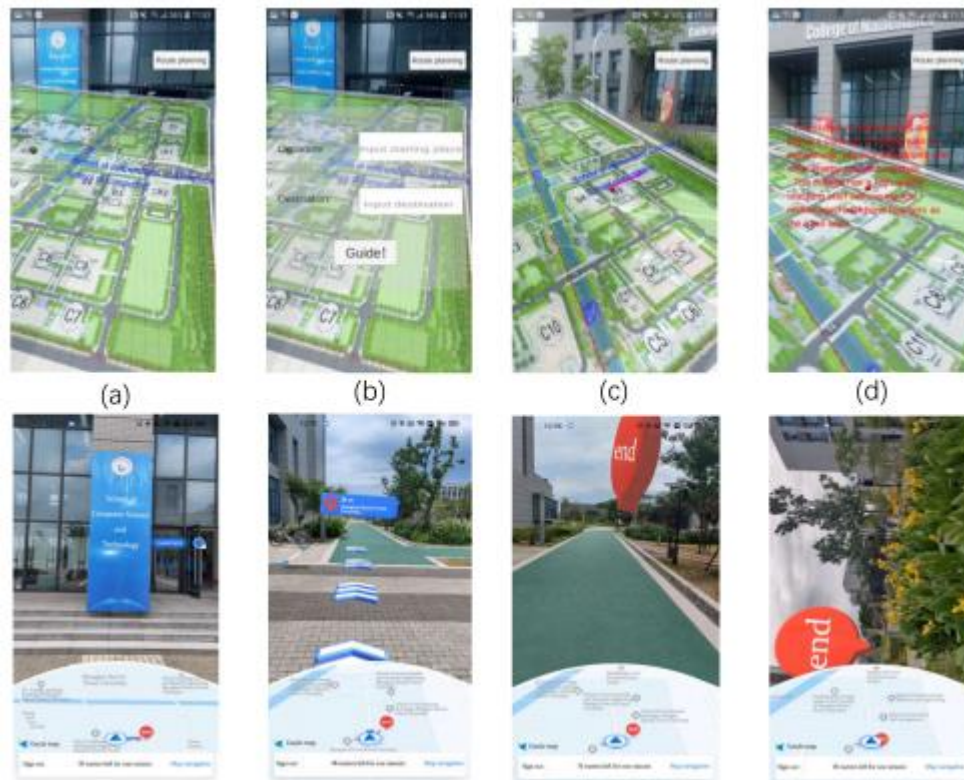
**Figure 3:** Screenshot from URWalking [27].

The URWalking project is working to improve indoor positioning algorithms, develop better prediction accuracy for destinations, and explore proxies for gaze data. By sharing log data with the community, integrating indoor tracking implementations, and comparing performance, the project aims to advance the GeoAI community and inspire the development of better AI-based wayfinding aids. In summary, the URWalking project demonstrates the potential of comprehensive indoor navigation systems that address both practical navigation and wayfinding research, offering valuable insights for the design and development of similar virtual guide platforms [27][28].

### 3.2 ARCore Augmented Reality Navigation System

This study presents an innovative ARCore-based augmented reality navigation system that outperforms conventional campus navigation systems that use 2D map to navigate the users. It

seamlessly integrates indoor and outdoor navigation to offer a more efficient and accurate experience for university campuses. The system employs visual inertial odometry (VIO) and the Unity development platform to enhance navigation accuracy and user experience. By synchronizing the position of the device camera with the sphere's position on the 3D map, the system accurately tracks the user's location. Additionally, it incorporates rich interactive components such as voice, 3D text, and 3D objects to enhance the overall experience.



**Figure 4:** Interface collections of the proposed system (a–d) and Gaode AR system (e–h) in outdoor [3].

The system's improved accuracy and scalability are demonstrated through experiments conducted at Shanghai University of Electric Power, with the indoor library test site showcasing its flexibility and efficiency in quick book location and enhancing the overall learning experience. Overall, this research provides valuable insights into a groundbreaking solution for improving campus navigation experiences[3].

### **3.3 Bluetooth Beacon-based Indoor Navigation System for Android**

This paper describes the Bluetooth Beacon-based Indoor Navigation System for Android that estimates the user's location using Bluetooth Low Energy and compass sensors and links the positions to predetermined coordinates inside the building using symbolic positioning. The system employs Dijkstra's shortest path algorithm for route planning and offers a user-friendly interface for route generation. As stated in the paper, the system's applicability is confirmed by experimental results from simulations and real-life scenarios. The system's accuracy was improved by using beacons within 5 meters and dedicated beacons in corners.

The Android-based application has two user interfaces that navigate the user: an image-based and a direction-based navigation interface, displaying the user's position, distance left in the journey, and the next position. The direction-based interface shows the destination room number, while the image-based interface displays pre-recorded images every 5 meters. Upon reaching the destination, the image or arrow changes, and a "You have arrived" message displays. The result was tested in the Institute of Information Science building at the University of Miskolc, covering three floors with Bluetooth beacons and images for image-based navigation [29]

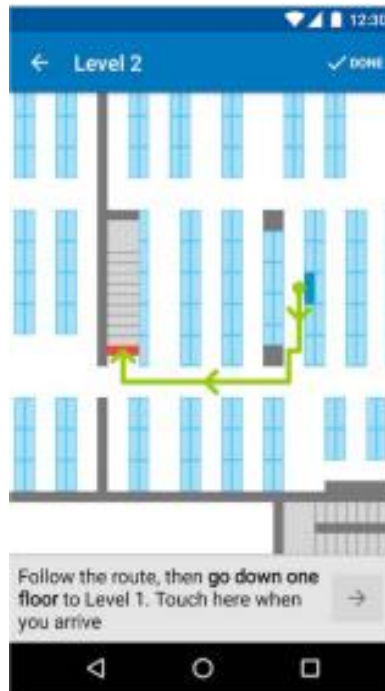
### **3.4 BookMark**

Utilizing barcodes on library books, BookMark serves as a self-contained indoor navigation system that aids visitors in locating their desired book by scanning the codes. As a versatile app compatible with any phone equipped with a camera, it provides users with an extensive library map that includes the locations of stairs, elevators, doors, exits, obstructions like pillars or interior walls, and every bookshelf. The map is subsequently converted into a scalable vector graphics format, with unique color codes signifying different locations.

To associate library books with the map, a database containing call numbers (unique alphanumeric identifiers) and corresponding locations is created. When users scan a book's barcode to find its location, BookMark's server-side retrieves the book's information through an existing library API, which includes the call number. The system then searches the call number within the book database to provide the location to the user.

BookMark employs the A\* algorithm to plan routes between two points of interest. Since it doesn't use a positioning technique, the system remains unaware of the user's current location until the navigation is completed or the next barcode is scanned. The primary limitations of this

system are the unavailability of barcodes on certain books and the potential for books to be placed on incorrect shelves [30].



**Figure 5:** One of BookMark screenshots, showing the system's various capabilities, and the steps involved in finding items within the library [30].

## Chapter 4 - System Analysis and Design

Virtual guide platforms are now integral to our everyday lives, offering crucial information and directing us to our desired destinations. These systems rely on intricate algorithms to deliver precise directions and points of interest. The development of a virtual guide platform necessitates thorough planning and evaluation of several factors such as the target audience, user needs, technical requirements, and overarching system structure. This chapter presents the problem description, which elucidates the foundational ideas behind the platform's design. It also covers system analysis, discussing the factors that need to be considered prior to design and implementation, and finally, it outlines the system design.

## 4.1 Problem Description

Considering the benefits, limitations, and objectives outlined in previous chapters, this section presents the primary structure of the virtual guide platform. Much of the Hellenic Mediterranean University in Heraklion is an outdoor area, with open-air access between campus buildings. However, for users navigating between indoor locations, an indoor positioning method is necessary for accurate tracking and direction.

Developing a reliable indoor positioning system that is cost-free is a challenging task since most of the indoor positioning methods described in the literature involve some cost. Moreover, implementing a low-cost but ineffective indoor positioning system is not desirable as users would prefer a system without indoor positioning over a poorly executed one [28]. Therefore, the virtual guide platform will utilize GPS to locate users and provide directions to their destination.

When it comes to determining the shortest path between two points on the campus, the literature suggests that the A\* algorithm is the fastest. However, the need for a heuristic function in the A\* algorithm can make it cumbersome to implement. The D\* algorithm is proposed as a solution for systems that have variable positions calculated, but it is not the most optimal solution for the virtual guide platform since it does not provide positioning. Similarly, the Floyd-Warshall algorithm is recommended for one-to-many shortest paths, which is not suitable for this project. Therefore, the Dijkstra's shortest path algorithm will be implemented, which always produces the correct answer and, for smaller-scale projects like this one, has the same speed to calculate the shortest path as the A\* algorithm [21].

Finally, users will be able to access information about specific locations when searching. A list of available locations will be provided for users to search. The university's MRBS already includes outlines of most campus buildings, rooms, and room names, which the platform will use to ensure maintainability and scalability - key objectives of the virtual guide platform.

## 4.2 System Analysis

In this section, we will focus on the system analysis of the virtual guide platform, examining the various components and considerations that need to be taken into account when designing such a system. We will begin by outlining the key requirements of the system and the target audience. We will then analyze the specifications required to build a reliable and efficient



platform. By the end of this section, we hope to provide a comprehensive overview of the system analysis of a virtual guide platform, and how it can be used to design effective and user-friendly systems.

The virtual guide platform will be able to provide users to navigate from one location to another, search for specific locations and points of interest, and access relevant information about these locations. In addition, users will be able to customize certain aspects of the map style and functionality, such as selecting a path that avoids stairs to accommodate individuals using wheelchairs or change the color of the path. The target audience for this platform will be the students and faculty members of the Hellenic Mediterranean University's Campus in Heraklion.

#### ***4.2.1 Platform's scenarios of use***

In agile software development, user stories are informal, yet concise summaries of software features written from the perspective of end users. They are useful in expressing the value of a product feature and understanding why users want it. Development and product management teams usually write user stories on index cards or sticky notes and use them in discussions during the development process. These discussions revolve around the users, resulting in a more robust product and better user experience. User stories are categorized based on the different roles each user can have in the application, such as new student, student, faculty member, or user with accessibility needs. Since each role requires different functionalities, every category of user stories has its own set of features. To be considered complete, each user story must have one or more acceptance criteria, which specifies the requirements that need to be fulfilled.

##### Role new student user stories

New students are users that will use the platform to navigate in an area that they see for the first time. Below is a list of the user stories for new students as well as their acceptance criteria.

- As a new student, I want to be able to use the platform to navigate from one building to another, so I can easily find my way around campus and get to my classes on time.
  - Given that I am a new student, by selecting my current location and desired destination from a list, the platform will provide me with the shortest route.
  - Given that I am a new student, when I search a location, there will be some suggested location in a list that I might be interested in going to.

### Role student user stories

Students are users that will use the platform to navigate to a class that they attend to for the first time or because the layout of the campus, just search for the location of the class. Below is a list of the user stories for students as well as their acceptance criteria.

- As a student, I want to be able to search for a specific room, such as a classroom or a lab, and see its location on the map, so I can easily find it and attend my scheduled classes.
  - Given that I am student, when I search for a specific room by entering its name, number, or building, and the platform will show me a list of possible rooms and when I select the desired room from the list, the platform will show its location on the map.
  - Given that I am student, by selecting my current location and the lab or class from a list, the platform will provide me with the shortest route.
- As a student, I want to be able to see the timetable and usage information of a specific room, such as which classes are held in it and at what times, so I can plan my schedule accordingly.
  - Given that I am a student, I searched for a specific room and the platform showed me its location, when I press the marker that shows its location, a panel will be displayed which can lead me to an informational page that shows the schedule of the room.
  - Given that I am a student, I open the informational page of an room and see the room's today's schedule, when I press a button, there is the possibility to change the date of the schedule that is being shown.

### Role faculty member user stories

Faculty members are users that will use the platform to search for the location of a room. Additionally, they want to see the schedules of a room that they teach or have a meeting in to confirm that it is correct. Below is a list of the user stories for faculty members as well as their acceptance criteria.

- As a faculty member, I want to be able to see the timetable and usage information of a specific room, such as which classes are held in it and at what times, so I can plan my schedule accordingly.
  - Given that I am a faculty member, I searched for a specific room and the platform showed me its location, when I press the marker that shows its location, a panel will be displayed which can lead me to an informational page that shows the schedule of the room.
  - Given that I am a faculty member, I open the informational page of an room and see the room's today's schedule, when I press a button, there is the possibility to change the date of the schedule that is being shown.

#### Role visitor user stories

Visitors to the university are users that will use the platform to navigate from their position to different areas of interest such as the library or the amphitheater where an inauguration is scheduled. Below is a list of the user stories for visitors as well as their acceptance criteria.

- As a visitor to the university, I want to be able to use the platform to navigate to different areas of interest on campus, such as the library or the student center, so I can navigate the campus and its facilities with ease.
  - Given that I am a visitor, I learned about the platform at the information board which I used to locate my position and help me navigate to my desired destination, when I use the platform, by selecting my current location and desired destination from a list, the platform will provide me with the path to get there.

#### Role student with accessibility needs user stories

Student with accessibility needs are users that will use the platform to navigate the campus without barriers such as staircases. Below is a list of the user stories for students with accessibility needs as well as their acceptance criteria.

- As a student with accessibility needs, I want to be able to use the platform to navigate to accessible entrances and facilities on campus, so I can move around the campus with ease and without barriers.

- Given that I am a student with accessibility needs, when I go to the settings page, I see an option that will allow me to navigate through the campus, when I select the option, in later use, the platform plots a path that is accessible by me with my wheelchair.
- Given that I am a student with accessibility needs, by selecting my current location and desired destination from a list, the platform will provide me with the shortest route without staircases.
- Given that I am a student with accessibility needs, by selecting my current location and desired destination from a list, and the route is not accessible by me, the platform will inform me.

### **4.3 System Design**

In this section, we will focus on the system analysis of the virtual guide platform, examining the various components and considerations that need to be taken into account when designing such a system. We will begin by outlining the key requirements of the system and the target audience.

We will then analyze the technical specifications required to build a reliable and efficient platform, including the data sources and algorithms. Finally, we will explore the overall system architecture, including software components, and how they interact to create a seamless user experience. By the end of this section, we hope to provide a comprehensive overview of the system analysis of a virtual guide platform, and how it can be used to design effective and user-friendly systems.

#### ***4.3.1 Component Diagram***

Figure 6 illustrates the component diagram of the application, which depicts the various components in the system. The detailed components are represented independently, without showing their dependencies, to provide better clarity.

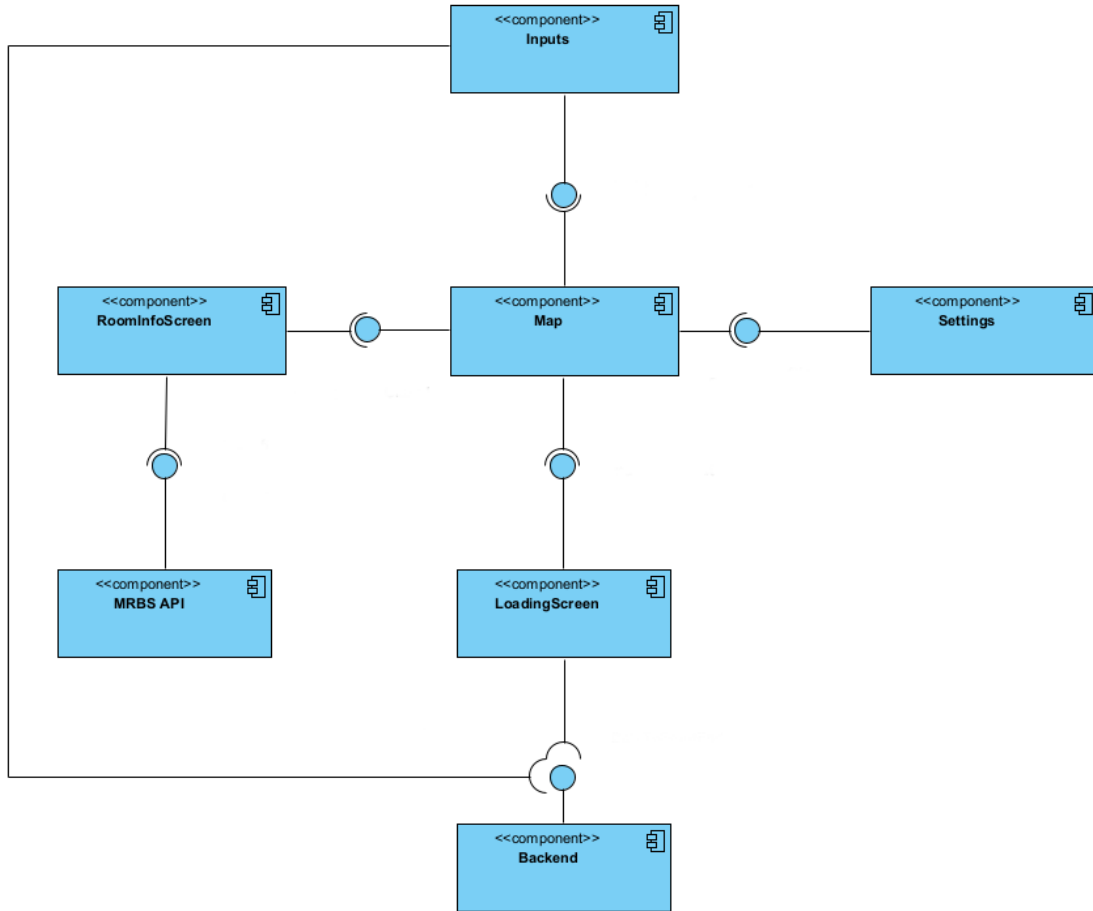
The "LoadingScreen" component is critical to the overall functionality of the virtual guide platform because it is responsible for retrieving a combination of informational and geospatial data from the "Backend" component or service. Once these important data have been acquired, it is

passed on to the "Inputs" component, which depends on these data to function properly and efficiently.

The "Map" component is at the heart of the platform, acting as the primary component responsible for geospatial data computation and offering essential features such as search/navigation functionality. The "Map" component depends on input, geospatial, and informational data from various sources to achieve this. The "Inputs" component handles the input data, while the "LoadingScreen" component handles the geospatial data required for the "Map" component to work.

Furthermore, the "Settings" component enhances the "Map" component's seamless functionality by giving the basic properties and configuration options needed for it to work properly. This ensures that the "Map" component can be tailored to suit the needs of individual users.

Lastly, the "RoomInfoScreen" component serves as a vital link between various components, utilizing data from both the "Map" component and the "MRBS API" component to present detailed room information to the user. This component plays an integral role in delivering a comprehensive and informative user experience, as it allows users to access relevant room data in a visually appealing and easily understandable format.



**Figure 6:** Component diagram.

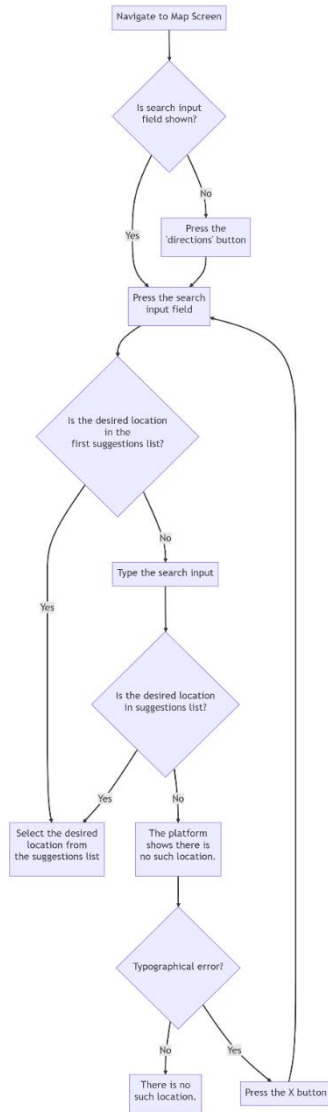
### 4.3.2 Flow Diagrams

Flow diagrams are diagrams that visually display interconnected information, such as process steps, events, functions, and more. This section will present the flow diagrams that have been developed based on the requirements outlined in the user stories. These diagrams will cover individual functionalities, rather than focusing on a particular user narrative, meaning that each diagram will encompass several user stories. In the following pages, the essential functionalities of the virtual guide platform will be discussed in detail.

#### 4.3.2.1 Search location flow diagram

Figure 7 outlines the steps involved in the process of searching a destination. Users must first access the map screen, enter the desired location into the search input field, and select the

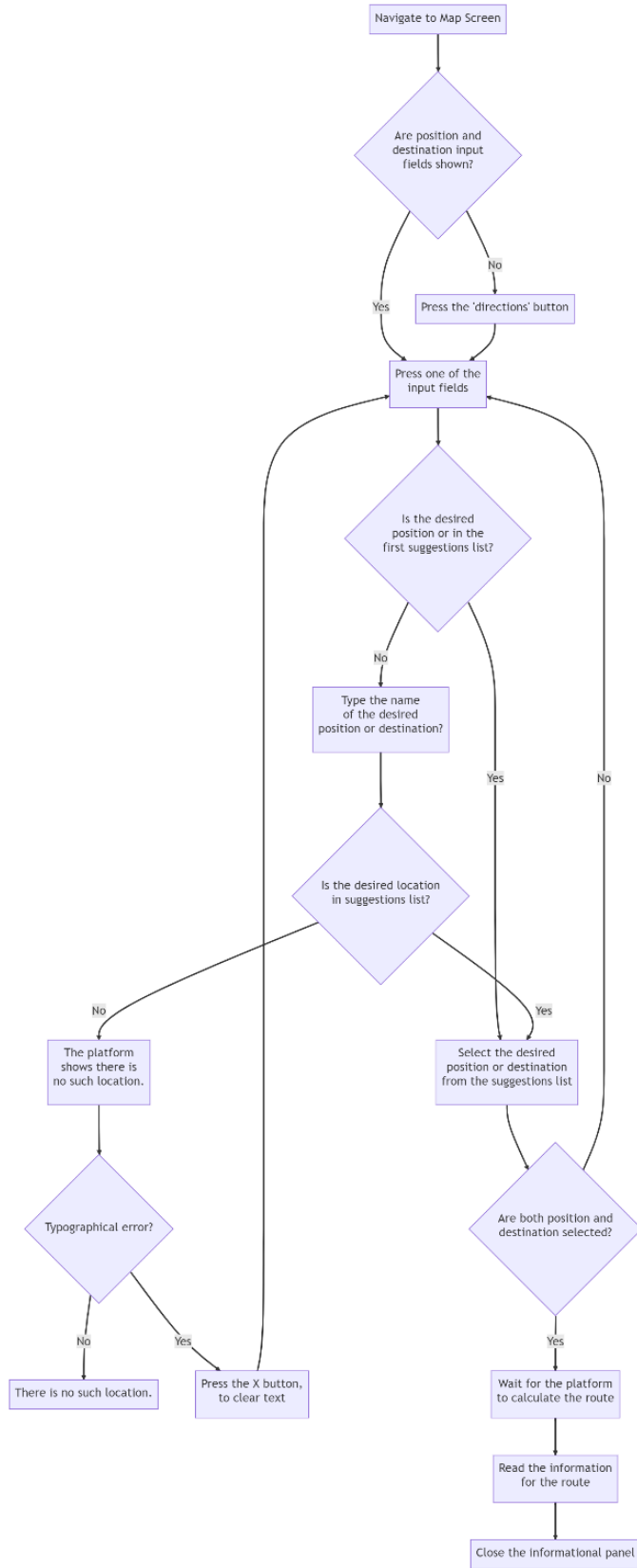
corresponding item from the suggestion list that appears. The platform will then indicate the location with a marker.



**Figure 7:** Search location flow diagram.

#### 4.3.2.2 Navigation flow diagram

Figure 8 shows how to navigate from a starting position to a destination. Users input both locations on the map screen, choose the relevant items from the suggestion lists, and the platform calculates the optimal path. An informational panel is displayed, which the user closes to view the navigation route.

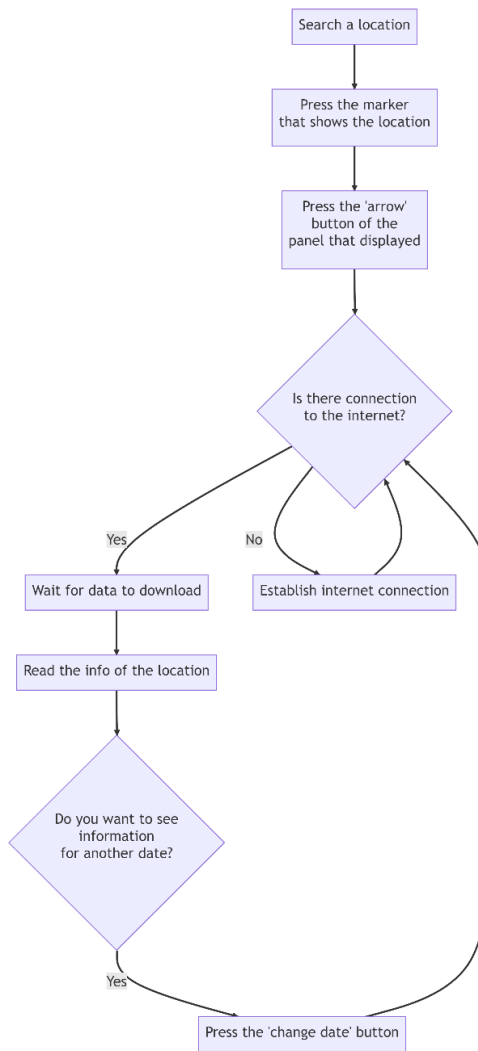


**Figure 8:** Navigation flow diagram.



### 4.3.2.3 Location's information access flow diagram

Figure 9 illustrates the process for accessing location information. After searching for a location and displaying a marker, the user must select the marker to access an information panel. From there, the user may navigate to the information screen by pressing the arrow button. Depending on internet connectivity, the screen may display today's information or require an internet connection to access. To modify the information display date, users may press the "change date" button.

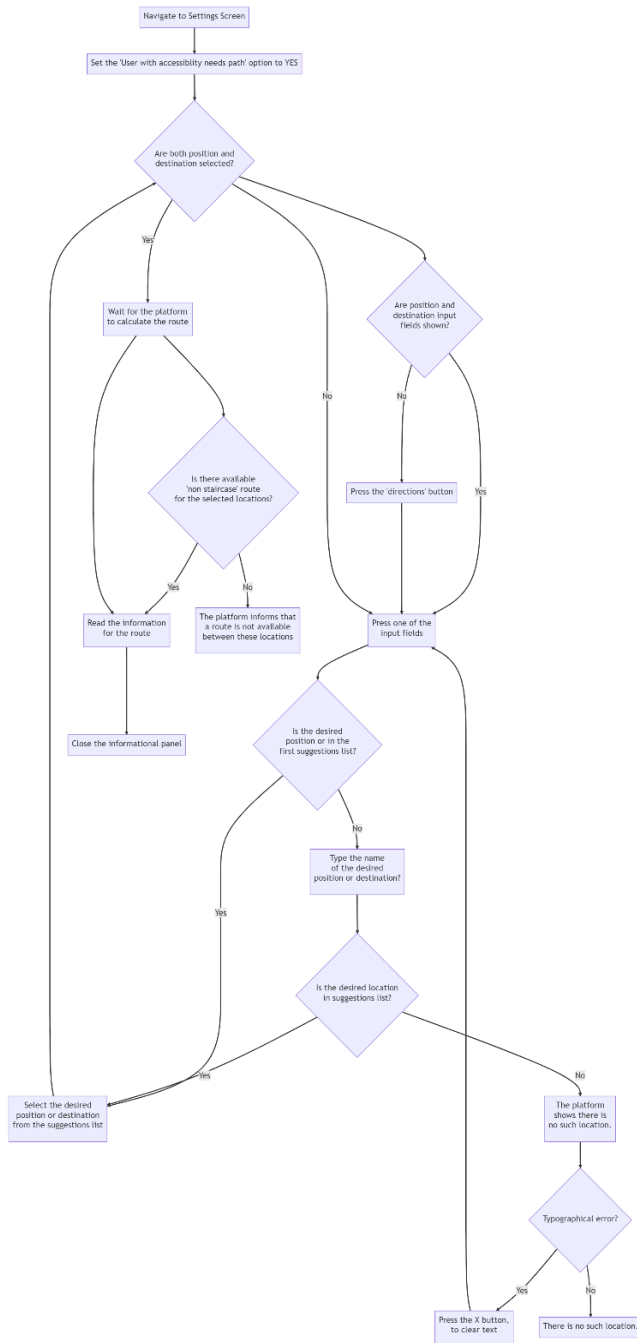


**Figure 9:** Location's information access flow diagram.

### 4.3.2.4 User with accessibility needs navigation flow diagram

Figure 10 presents the procedure for navigating a user with accessibility requirements. While the process is similar to that of a regular user, there are some key differences. To initiate the

accessibility mode, the user must first access the settings screen and enable the option for the platform to generate a path that suits their accessibility requirements. Once enabled, the user may follow the same steps outlined in Figure 8 for regular navigation. However, at the final stages of the process, there is a possibility that the path generated between the selected locations may not meet the user's needs. In such cases, the platform will notify the user accordingly.



**Figure 10:** User with accessibility needs navigation flow diagram.

## Chapter 5- Implementation

The virtual guide platform was implemented with three design concepts in mind. These concepts impacted the choice of the technologies that were used in the process. The first concept is expandability, support for new geospatial data and application features to be added in the future. The second one is versatility, the platform must be accessible to both Android and iOS operating systems, and the third one is maintainability, the software structure and design must be clear and the information about the development process precise.

### 5.1 Technologies used.

#### 5.1.1 *Quantum Geographic Information System (QGIS)*

The foundation of the virtual guide platform is involving the creation and manipulation of geospatial data related to a particular geographic region, and QGIS was chosen as the software tool of choice due to its ability to create and handle various types of geospatial data. QGIS is a free and open-source Geographic Information System (GIS) software that allows creation, editing, visualization, analysis, and publishing geospatial information [30]. Additionally, QGIS has a user-friendly interface, making it easy for both beginners and advanced users, and its active community provides valuable resources and support for troubleshooting and learning [31][32].

#### 5.1.2 – *PostgreSQL with PostGIS extension*

Geospatial data that are utilized by QGIS are stored in PostgreSQL database with PostGIS extension. PostgreSQL is a robust open-source relational database management system (RDBMS) that offers advanced data storage and management capabilities [32]. When integrated with the PostGIS spatial database extension, PostgreSQL becomes a potent geospatial database management system that enables users to store and manage vast volumes of geospatial data and conduct complex spatial queries and analyses [33][35]. The combination of QGIS and PostgreSQL with PostGIS enables users to harness the strengths of each tool to create comprehensive geospatial solutions. QGIS facilitates the effortless creation and visualization of maps, while PostgreSQL

with PostGIS permits the storage, management, and analysis of geospatial data on a significant scale.

### ***5.1.3 – Node / Express***

To access the geospatial data from the application, a server-side application that performs Read operations is needed. To implement these operations Node JS used, a platform for running JavaScript code beyond a web browser, which offers an asynchronous approach to construct scalable server-side applications [36][38]. Express JS is a minimalist web application framework for Node.js that simplifies the process of building fast, robust, and scalable server-side applications using JavaScript. It provides a simple, flexible, and unopinionated structure for creating APIs, handling HTTP requests, and managing middleware to handle complex application logic. [37][38]. With the use of Express JS, the service for providing access to geospatial data is implemented.

### ***5.1.4 – Typescript***

The aforementioned technologies have the potential to be utilized in conjunction with Typescript, a syntactic superset of JavaScript which incorporates static typing and supplementary syntax into JavaScript. By leveraging the strengths of both, it is possible to detect and rectify errors and bugs at an earlier stage of the development process. This is further augmented by a tighter integration with any code editor, which enables the system to catch issues before they are pushed into production [39][40].

### ***5.1.5 React Native with React Native Maps***

React Native is an open-source framework that enables developers to build mobile applications for multiple platforms, including iOS and Android, using JavaScript and React [41]. React Native utilizes a modular approach to building mobile apps by breaking down the user interface into small, reusable components. These components are written in JavaScript and compiled into native code, which allows the app to run smoothly on a given platform. This approach allows developers to write code that can be easily shared between different parts of the

app and across different platforms, reducing the time and effort required to develop and maintain mobile apps [41][42].

React Native Maps is a library that allows developers to integrate maps into their React Native applications. React Native Maps provides a simple and declarative API for developers to use, which includes features like markers, polylines, and custom map styling [43].

React Native Maps supports multiple map providers, including Google Maps, Mapbox, and Apple Maps. These features make it easy for developers to build highly customized and interactive maps within their applications [43].

## **5.2 Software Implementation**

In this section, the software implementation of the three main parts of this thesis will be presented: the database part, the backend, and the frontend. The database part involves creating and storing data. The backend is responsible for handling and retrieving data from the database, as well as interacting with other systems by sending and retrieving data. Finally, the frontend is responsible for displaying data and enabling user interaction with the application.

### ***5.2.1 Database***

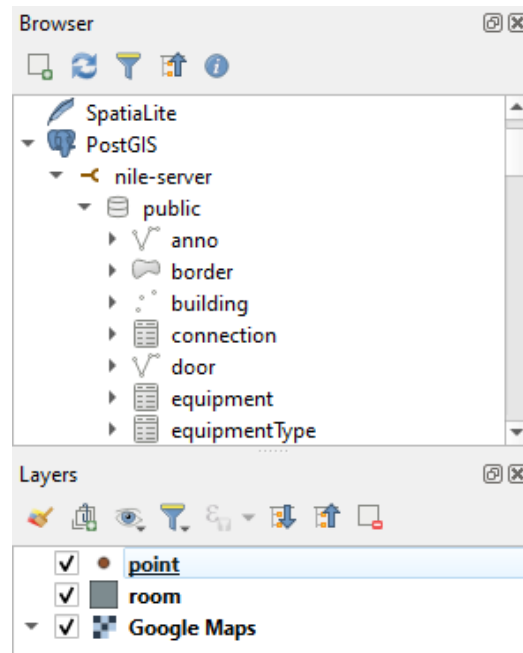
The database serves as a crucial foundation for any application, and in this case, we have utilized Quantum GIS and PostgreSQL with the PostGIS extension to create and manage data for the application. The data utilized in the application is derived from three distinct sources. Firstly, the application database was constructed and populated with data, including both geospatial and informational data, with QGIS. Secondly, we imported the geospatial data tables from an existing database utilized by Hellenic Mediterranean University's MRBS to the newly developed database for the application. Lastly, we included data that were fetched from the MRBS API containing information such as room names, descriptions, and other related details for certain rooms located within the Hellenic Mediterranean University's Campus. The combination of these data sources provides a comprehensive and precise representation of the campus environment.

### 5.2.1.1 Creating data

The creation of geospatial data starts with the use of QGIS. First, the connection to the PostgreSQL database, in which the geospatial data will be stored, is established.

The connection to the database can be established by right clicking the PostGIS entry in the browser and selecting the “New Connection” option. A window will pop-up in which the credentials of the database must be inputted.

Then, the database and its tables can be accessed by QGIS.

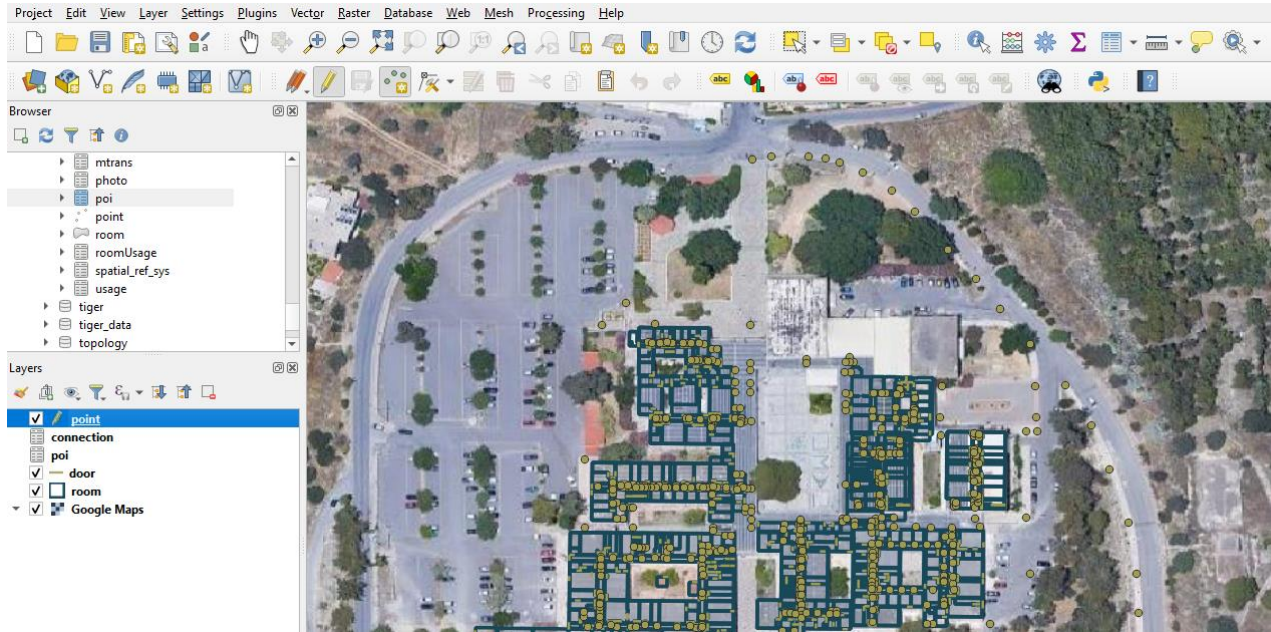


**Figure 11:** QGIS Browser and Layers panels.

By right-clicking at the schema name that will be the storage location of the new table and selecting the option “New Table”, a new window pops up. In this window, all the fields and qualities of the table can be defined, as well as the type of geospatial data we want to create. For the geospatial data, there are extra options that are hardcoded with the creation of the geospatial field of the table. These options are the geometry type, the name of the column in the table, the dimensions, the preferred coordinate reference system(CRS) that will be used in the application and whether to create spatial index.

Once these tables are in place, we move on to inserting geospatial data into them. To do this, we select the relevant table by clicking on it. Next, we enable editing in the table by clicking on the pencil icon.

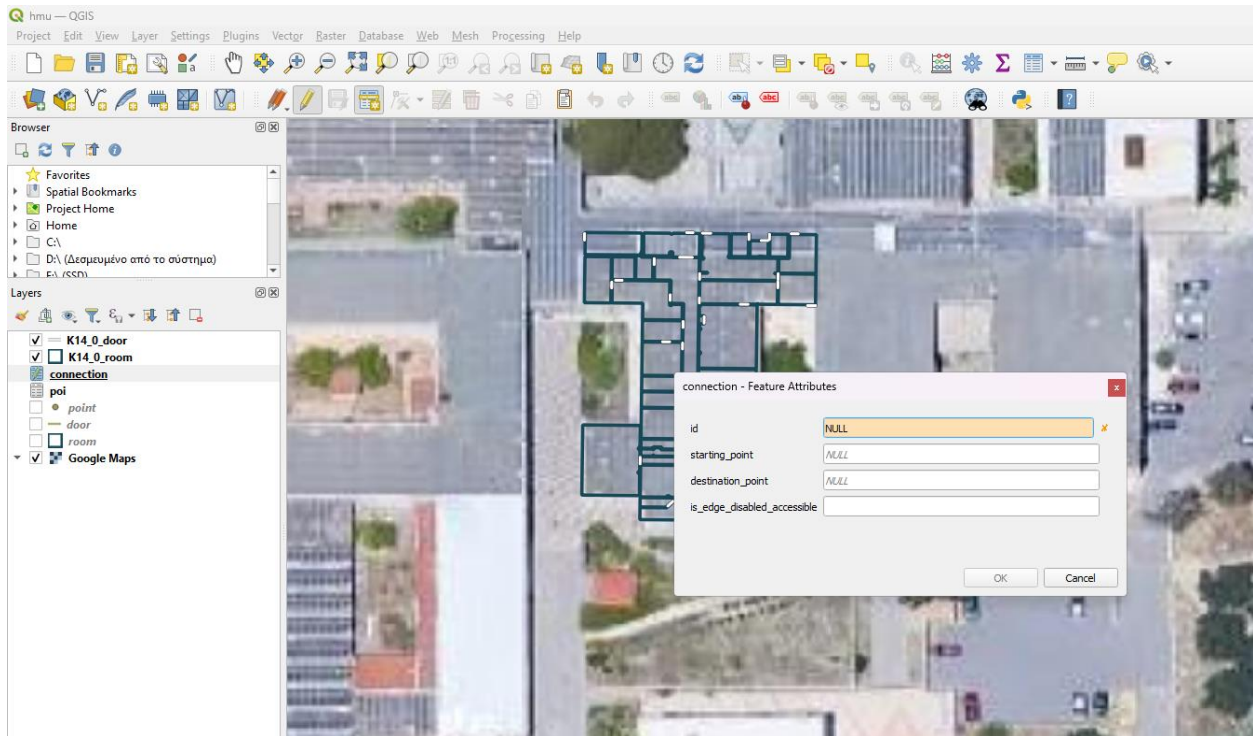
To add data to a table that contains geospatial data, such as a point, we click on the "Add Point Feature" button and place the desired geometry data on the map. Then, a pop-up window will appear where we can fill in the necessary data. Once all the required information is entered, we click on "OK" to save the changes.



**Figure 12:** Preview of Hellenic Mediterranean University's campus filled with geospatial data.

Otherwise, to add data into a table that contains informational data, such as connection, we click on the "Add Record" button. Fill the data and closing the window record. Then, a pop-up window will appear where we can fill in the necessary data. Once all the required information is entered, we click on "OK" to save the changes.

In the application, we fill three tables: point, connection, and POI. All the other data used in the application have already been created in the past to be used in MRBS. Some of this data is fetched from the MRBS API, while the rest is directly imported into the application database from the MRBS database.



**Figure 13:** Add connection panel.

Point table has the id, geom, latitude, longitude and is\_entrance fields. Each point represents a vertex in a graph. The id field serves as the unique identification of every point. The geom field contains the geospatial data of the points location in the map. Additionally, the latitude and longitude fields are used by the frontend of the application to render the location of the point. Finally, the is\_entrance field indicates whether the point represents an entrance to the university campus, and is set to 'true' if it does.

	TABLE ATTRIBUTES						
<b>POINT</b>	<i>int</i>	<i>geometry</i>	<i>double</i>	<i>double</i>	<i>varchar</i>	<i>varchar</i>	<i>bool</i>
	<b>id</b>	<b>geom</b>	<b>latitude</b>	<b>longitude</b>	<b>icon_type</b>	<b>floor_id</b>	<b>is_entrance</b>

**Figure 14:** POINT table structure.

The Connection table has the id, starting\_point, destination\_point, and is\_edge\_disabled\_accessible fields. Each connection represents an edge between two vertices in a graph. The id field serves as the unique identification of every connection. The starting\_point and destination\_point fields contain as values ids from point table, and they are depicting an edge in a directed graph (connection between two points). The starting\_point is the vertex where the edge is



starting, and the destination\_point is the vertex where the edge is ending, showing the direction of the edge. Lastly, the is\_edge\_disabled\_accessible field indicates whether the edge is accessible by disabled persons and is set to 'true' if it does.

	TABLE ATTRIBUTES			
<b>CONNECTION</b>	<i>varchar</i>	<i>varchar</i>	<i>varchar</i>	<i>bool</i>
	<b>id</b>	<b>starting_point</b>	<b>destination_point</b>	<b>isEdgeDisabledAccessible</b>

**Figure 15:** CONNECTION table structure.

The POI table includes the following fields: id, room\_name, description, capacity, projector, camera, sort\_key, room\_type, room\_admin\_mail, and is\_reservable. Each POI represents a point of interest on the map, such as a room or a special location. The POI table was designed to match the fields of room info data fetched from the MRBS API. It exists as a complement to the MRBS API data and adds data to each POI that is desired to be presented on the map but does not exist in the MRBS API. The id field is the unique identifier for each POI. The room\_name is the name of the POI, while the description provides a full description of the POI. The capacity, projector, and camera fields can have a value of zero or one. If the value is zero, it means that the POI does not have the feature described by the field name. If the value is one, it means that the POI has the feature described by the field name. The sort\_key field represents the ID of the geometry of each room fetched from the MRBS API. The room\_type field specifies the type of the POI, while the room\_admin\_mail provides the email address of each room's manager. Finally, the is\_reservable field indicates whether the POI has reservation information that can be fetched from the MRBS API and is the only additional field that does not exist in the MRBS API data.

	TABLE ATTRIBUTES									
<b>POI</b>	<i>int</i>	<i>varchar</i>	<i>varchar</i>	<i>int</i>	<i>int</i>	<i>int</i>	<i>varchar</i>	<i>varchar</i>	<i>varchar</i>	<i>bool</i>
	<b>id</b>	<b>room_name</b>	<b>description</b>	<b>capacity</b>	<b>projector</b>	<b>camera</b>	<b>sort_key</b>	<b>room_type</b>	<b>room_admin_mail</b>	<b>is_reservable</b>

**Figure 16:** POI table structure.

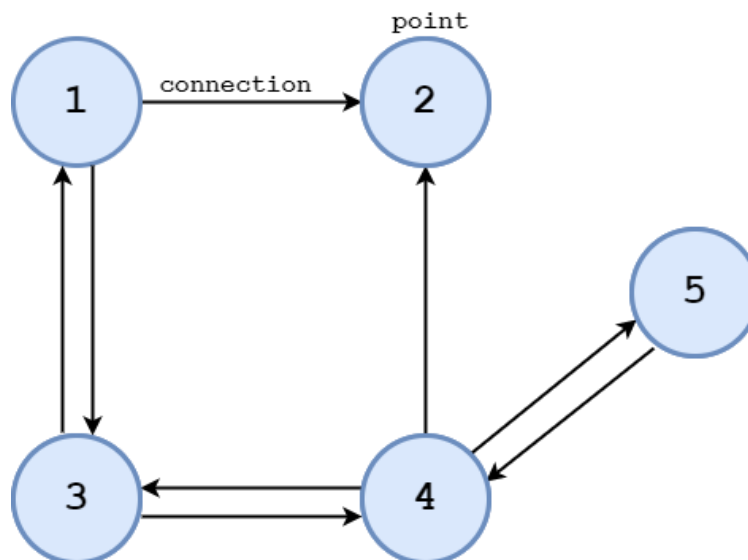
The POINT table and CONNECTION table, as stated above, are two tables that can be used together to represent a directed graph with weights. The point table contains information about the vertices or points of the graph, such as their names or labels, their coordinates, and any other relevant data that describes them. The connection table, on the other hand, contains information about the edges or connections between the vertices, such as the start and end points

of each edge, the distance between two points (weight of the edge), and any other relevant data that describes the edge.

When these two tables are used together, they form a directed graph with weights that can be used to portray any location as a directed graph with weights. This means that the graph contains all the necessary information about the location, including the positions of its points and the connections between them. By representing the location as a directed graph with weights, it becomes possible to use shortest path algorithms and other graph algorithms to navigate through it.

For example, if the location is a road network, the points in the point table could represent intersections, while the edges in the connection table could represent the roads between them. Using a shortest path algorithm, such as Dijkstra's algorithm, it is then possible to find the shortest path between two points on the road network, such as the shortest route between two cities [44].

The use of a directed graph with weights is a common approach to modeling locations and allows for the efficient use of graph algorithms to solve various problems, such as finding the shortest path.



**Figure 17:** Visual representation of a directed graph.

### 5.2.1.2 Modifying data

To modify data, we access the application's database, which includes both newly created tables in QGIS and pre-existing tables imported from the MRBS database. For newly created tables in QGIS, a special PostGIS query must be run to make them usable by the application. For the

imported tables of MRBS, we modify them to make them more precise and suitable for the specific needs of the application.

The initial step in modifying the data to ensure its proper display in the application involves running a PostGIS query. To execute this action, the user needs to access the point table layer in the Layers tab of QGIS and right-click on it. Next, the "Update SQL Layer" option should be selected, followed by typing the query "UPDATE point SET latitude = ST\_Y(geopoint), longitude = ST\_X(geopoint);" in the pop-up window. Upon pressing the "Execute" button, the query extracts the coordinates of the location where the point was placed on the map from the geom field, which was stored as geometry data. Subsequently, the latitude and longitude fields of the point table are populated with these coordinates. This procedure allows the application to correctly parse the geospatial data of the point table since raw geometry data cannot be processed by the application.

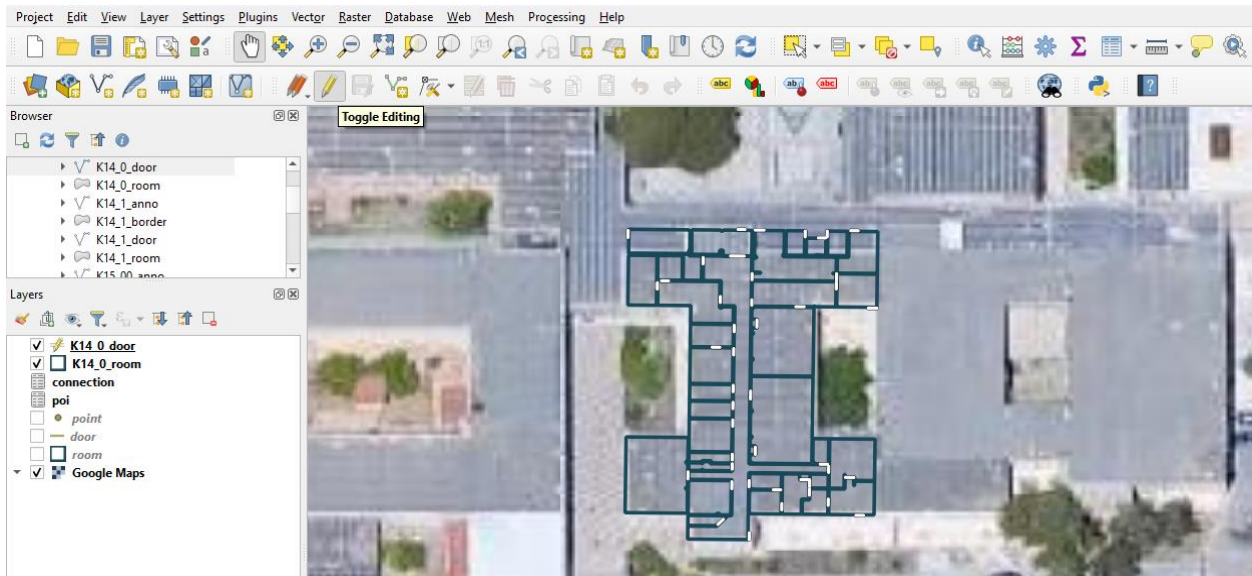
Further modifications to the application's database entailed adjusting the geospatial data contained in the imported tables from the MRBS database, specifically the outlines of each floor's rooms and door locations, to align with the Google Maps Satellite map used in the application. This change was deemed necessary as the original geospatial data were aligned with the OpenStreetMap, which was the map utilized in MRBS. To ensure compatibility with Google Maps, a thorough analysis of the original data was conducted, and necessary adjustments were made to align the outlines and door locations precisely with the map.

The MRBS database tables that were imported contain geospatial data, such as the layout of rooms on a floor of a building and the location of each room's doors within the campus of Hellenic Mediterranean University. The data consisted of geospatial information for several buildings listed in the MRBS database, including the following: K09, K10, K11, K12, K13, K14, K15, K16, K17, K18, K20, K24, K25, K26, K30, K31, K33, K37, K38, K39, K40, K43, K44, and K45.

All these buildings are separated in floors, for example, the building K14 has two floors, the ground floor and the first floor and they are represented as K14\_0 and K14\_1 respectively. The naming convention for floors in the application's geospatial data is based on the combination of the building's name and floor number, separated by an underscore or a period (e.g., K14\_0 or K14.0). This combination, referred to as a 'floor\_id', is used to identify and name the floors

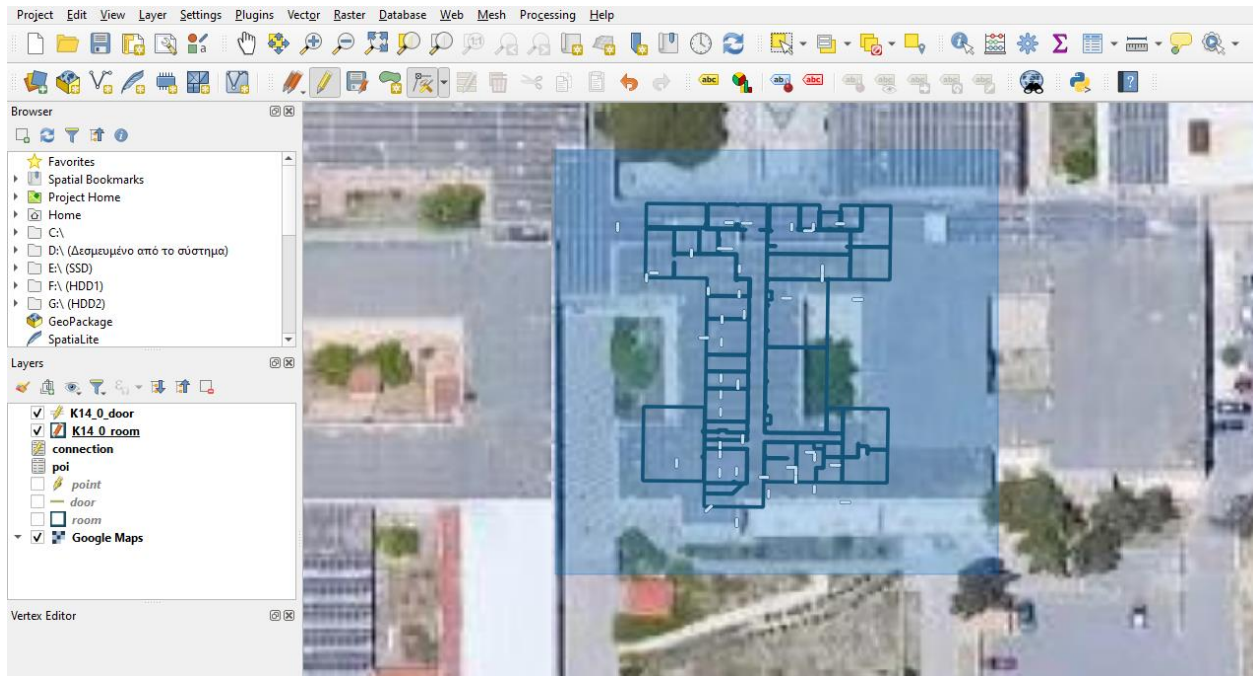
containing geospatial data. Moreover, the floor\_id is augmented with a type of data represented in the table to further differentiate between floors period (e.g., K14\_0\_room, K14\_0\_door).

As such, the naming of floors is standardized to ensure consistency and facilitate seamless integration of data into the application. Figure 18 displays the geospatial data depicting the room outlines and doors of the K14 building's ground floor as example.

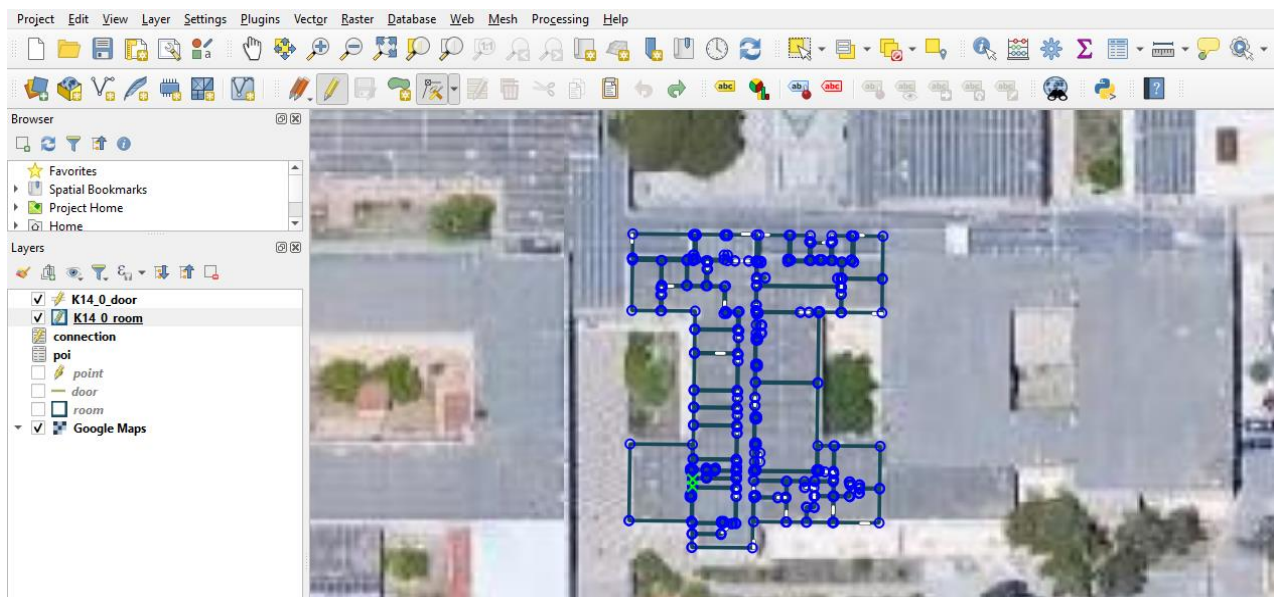


**Figure 18:** Representation of editing geospatial data of a floor.

To modify specific geodata data of a floor, it is necessary to select the appropriate floor\_id containing the data in question as a layer on the map, such as the K14\_0\_room layer. Once the layer has been selected, editing can be enabled by clicking on the pencil icon, followed by pressing the "Vertex Tool" button. At this point, the geospatial data in the table must be reallocated to match the Google Maps Satellite layer. This can be accomplished by using rubber-banding to select all relevant data in the table and reallocating it accordingly. Finally, to save the new modified data, press the SAVE button.



**Figure 19:** Selecting the data with rubber-banding.



**Figure 20:** After the moving the geospatial data.

The geospatial data for rooms and doors in the following buildings were modified to match the Google Maps Satellite layer: K10, K11, K12, K13, K14, K15, K16, K17, K18, K20, K24, K25, K33, K44, and K45.

The room and door data from each table are collected and stored in two other tables, named 'room' and 'door'. These tables are accessed by the backend of the application, which retrieves the necessary geospatial data to send to the frontend. Therefore, it was necessary to update these two tables with the new geospatial data adjusted to match the Google Maps Satellite layer.

To update the 'room' and 'door' tables, a node.js script was created to automate the procedure. This script significantly reduced the time required to complete the update. It works by connecting to the database using the database credentials and runs dynamic SQL queries using JavaScript functions.

To begin, the connection with the database needs to be established by providing the appropriate credentials in the corresponding fields in the Client object of node-postgres. Next, the connection with the database is established.

```
const { Client } = require('pg');

/* Connecting to the database using the needed credentials. */
const client = new Client({

  /* Connection Settings Config. */
  host: DATABASE_HOST,
  user: DATABASE_USER,
  port: DATABASE_PORT,
  password: DATABASE_PASSWORD,
  database: DATABASE_DB

});

client.connect((err) => {

  if (err) {
    console.error('connection error', err.stack)
  } else {
    console.log('connected')
  }

})
```

**Figure 21:** Connection to database code snippet.

Before proceeding, it's essential to define all the necessary queries. These may include a query to retrieve the entry ID for every record that needs to be updated, another to extract the required dynamic data, and a final one to store the updated data in the desired table. For each table, three queries must be executed, and each query is a promise.

```

/*===== ROOM QUERIES =====*/
const getRoomIds = () => new Promise((resolve, reject) => {
  client.query('SELECT id FROM room', (err, result) => {
    if (!err) {
      resolve(result.rows);
    } else {
      console.log(err);
    }
  });
});

const getRoomData = (getRoomDataQuery) => new Promise((resolve, reject) => {
  client.query(getRoomDataQuery, (err, result) => {
    if (!err) {
      resolve(result.rows[0]);
    } else {
      console.log(err);
    }
  });
});

const updateRoom = (updateRoomQuery) => new Promise((resolve, reject) => {
  client.query(updateRoomQuery, (err, result) => {
    if (!err) {
      resolve(result);
    } else {
      console.log(err);
    }
  });
});

/*===== DOOR QUERIES =====*/
const getDoorIds = () => new Promise((resolve, reject) => {
  client.query('SELECT id FROM door', (err, result) => {
    if (!err) {
      resolve(result.rows);
    } else {
      console.log(err);
    }
  });
});

const getDoorData = (getDoorDataQuery) => new Promise((resolve, reject) => {
  client.query(getDoorDataQuery, (err, result) => {
    if (!err) {
      resolve(result.rows[0]);
    } else {
      console.log(err);
    }
  });
});

const updateDoor = (updateDoorQuery) => new Promise((resolve, reject) => {
  client.query(updateDoorQuery, (err, result) => {
    if (!err) {
      resolve(result);
    } else {
      console.log(err);
    }
  });
});

```

**Figure 22:** Queries of the node.js script.

The main function of the script executes each query as an asynchronous function in conjunction with the relevant business logic. Asynchronous execution allows the script to handle multiple queries efficiently and ensure accurate and efficient updating of the data. This approach can also help prevent performance issues and reduce the risk of errors or data inconsistencies that could occur with synchronous execution.

During execution, the script outputs progress updates via console.log until it has completed processing. This feature can be useful for tracking the script's progress, especially when working with larger datasets or performing more complex updates. Once the script has finished processing, no further updates will be executed, and the console.log will stop.

Upon completion, the room and door tables of the database are updated with data that is aligned correctly with the application's map. This update ensures that the data is displayed in the correct format, providing an enhanced user experience. For example, if the application is used to display a map of a building, the updated data will ensure that the location of each room and door is accurately represented on the map.

```

let counter = 0;

const updateTables = async () => {

  /* Get the id each room */
  const roomIDArray = await getRoomIds();
  /* Get the id each door */
  const doorIDArray = await getDoorIds();

  roomIDArray.forEach(async (roomID) => {
    /* Splitting the ID into parts (e.g. BuildingID.FloorID.RoomID) */
    let splittedID = roomID.id.split(".");

    /* filter out of the function, the dorm rooms (e.g. BuildingID.FloorID.RoomID.Room) */
    if (splittedID.length === 3) {
      /* Formatting buildingID */
      let buildingIDTemp = splittedID[0].split("K");
      let buildingID = buildingIDTemp[1];

      /* Creating the dynamic query, that returns the geospatial data (geometry) of each room as GeoJSON */
      const getRoomDataQuery = `SELECT id, ST_AsGeoJSON(geom) as geom FROM "K${buildingID}_${splittedID[1]}_room" WHERE id='${roomID.id}'`;
      /* Running the query */
      const roomData = await getRoomData(getRoomDataQuery);

      /* If the previous query results are not undefined */
      if (roomData !== undefined) {
        /* Creating the dynamic query, that updates each room data in the room table */
        const updateRoomQuery = `UPDATE room SET geom = ST_GeomFromGeoJSON('${roomData.geom}') WHERE room.id='${roomID.id}'`;

        /* Counter to be shown in terminal*/
        counter+= 1;
        console.log("counter at room:", counter);
        updateRoom(updateRoomQuery);
      } else {
        console.log("undefined");
      }
    }
  });
};

```

**Figure 23:** Function that update geospatial data code snippet (a).

```

doorIDArray.forEach(async (roomID) => {
  /* Splitting the ID into parts (e.g. BuildingID.FloorID.RoomID) */
  let splittedID = roomID.id.split(".");

  /* filter out of the function, the dorm rooms doors (e.g. BuildingID.FloorID.RoomID.Room.doors) */
  if (splittedID.length === 4) {
    /* Formatting buildingID */
    let buildingIDTemp = splittedID[0].split("K");
    let buildingID = buildingIDTemp[1];

    /* Creating the dynamic query, that returns the geospatial data (geometry) of each room as GeoJSON */
    const getDoorDataQuery = `SELECT id, ST_AsGeoJSON(geom) as geom FROM "K${buildingID}_${splittedID[1]}_door" WHERE id='${roomID.id}'`;
    /* Running the query */
    const doorData = await getDoorData(getDoorDataQuery);

    /* If the previous query results are not undefined */
    if (doorData !== undefined) {
      /* Creating the dynamic query, that updates each door data in the door table */
      const updateDoorQuery = `UPDATE door SET geom = ST_GeomFromGeoJSON('${doorData.geom}') WHERE door.id='${roomID.id}'`;

      /* Counter to be shown in terminal*/
      counter+= 1;
      console.log("counter at door:", counter);

      updateDoor(updateDoorQuery);
    } else {
      console.log("undefined");
    }
  }
});
};

updateTables();

```

**Figure 24:** Function that update geospatial data code snippet (b).



In conclusion, the successful integration of Quantum GIS and PostgreSQL with the PostGIS extension has provided a firm foundation for the creation of a comprehensive and accurate database for our application. The combination of geospatial and informational data from multiple sources, including an external API and an existing database, has enabled us to create a precise representation of the Hellenic Mediterranean University's campus environment.

This project has demonstrated the critical role of a well-designed and efficiently managed database in supporting the functionality of any application. By leveraging the strengths of PostgreSQL and PostGIS, we were able to build a database that effectively manages and processes the data required for our application. The integration of QGIS allowed us to visualize and analyze the geospatial data efficiently, which was a critical aspect of the development process.

### 5.2.2 Backend

The backend component of our application plays a critical role in facilitating seamless communication between the frontend and database. It has been implemented using Node.js, Express.js, and TypeScript, while adhering to various design patterns and following the structural conventions used in all backend projects at Natural Interactive Learning Games and Environments Lab (NILE Lab). Its primary function is to query data from the database and provide an API for the frontend to access.

Our backend implementation has been designed with scalability, maintainability, and efficiency in mind. We have incorporated industry best practices and design patterns to ensure that it meets the requirements of our application. The backend has been structured in a modular fashion, which enables us to effortlessly incorporate new features and functionalities.

To adhere to the conventions followed by NILE Lab, we have implemented the following folder structure.

**Table 1:** Folder structure of the backend part of the platform.

Controllers	Should contain all the logic to transform data from and to the required form to store or send the data from the Database and guarantee the validity of this data.
DAO	Should contain the logic to communicate with the Database to store and retrieve data. (node-postgres)
DTO	Should contain the Objects to be transferred between layers of the service.

Middleware	Contains logic that should execute between specific layers of the service.
Models	Contains the persistence classes.
Routes	Should contain all the endpoints of the service.
Services	Contains extra logic required from the service.
Utilities	Contains small repeatable functions.

### ***5.2.2.1 Logging Service***

We added a logging service to record requests and errors in our application's backend. Errors are categorized into six levels, with Error being the most severe and Silly being the least. We use a middleware to call the `LoggingService` before passing requests to the router. The `LoggingService` uses the Chain of Responsibility design pattern with a `LoggingHandler` for each error level and a Root Handler. Handlers log messages with Winston logger or pass it to the next handler in the chain. Errors are logged to files named after the error level and date. A class diagram is shown in the code snippet.

### ***5.2.2.2 Error Handling***

In the backend of our application, the router plays a pivotal role in determining which controller method to invoke for processing incoming requests. However, the execution of controller code may result in errors, leading to severe consequences such as server crashes and disruptions in API usage for clients. To prevent such scenarios, an `ErrorHandler` middleware is employed to capture errors thrown by controller functions.

To accomplish this, each controller function is enclosed within a try-catch block that traps errors thrown during execution. In case of an error, the catch block triggers the `ErrorHandler` middleware to manage the error. The `ErrorHandler` middleware utilizes the logging service to record the error's details, including the status code, ensuring that the error is captured and processed for further analysis.

By employing the `ErrorHandler` middleware in conjunction with the logging service, we can ensure that any errors occurring within controller functions are intercepted, logged, and analyzed. This mitigates the risk of server crashes and ensures a seamless experience for API clients.

### 5.2.2.3 *Retrieving and sending data*

After the creation and modification of data, the retrieval of the data from a server is the next critical step. The backend retrieves data from the application database and the MRBS API. Retrieving data from a database involves querying the database using a query language such as SQL to specify the data to be retrieved. In our case, the queries are made using PostgreSQL with the PostGIS extension and the NodeJS library node-postgres. The data of MRBS are retrieved with the use of MRBS API.

DAO (Data Access Object) and DTO (Data Transfer Object) design patterns are used in retrieving the application's database data. These design patterns are commonly used to improve the organization and efficiency of data access and transfer inside the application. The DAO pattern separates the business logic of an application from its data storage mechanism and the DTO pattern, on the other hand, is used to transfer data between layers of an application, such as from the business logic layer(data retrieval and formatting) to the presentation layer(controllers and routing) [46][47].

The application's database contains six distinct data tables, each with its own unique model: Building, Room, Door, Point, Connection, and POI. To interact with each table, a separate model, DAO, DTO, controller and route functions are employed. The DAO of each table is a function that retrieves data from the table and it returns a Promise that resolves to an array of objects that are based in the table's model. It extends a generic DAO interface which contains all CRUD operation functions. CRUD is an acronym that stands for Create, Read, Update, and Delete. It is a set of basic operations that are commonly used to interact with data in a database or a user interface [45]. In this application, other CRUD operations besides the Read operation are not utilized. The DTO class of a table is used to filter data that is passed as an argument (in this case, data queried in a DAO function) and return only the data that is allowed by its properties defined in the constructor.

The controller module is importing and using the DAO to retrieve data from the database, and the DTO to filter the extracted data. Each table has its own controller. Once all controllers are created, the data can be accessed and sent to the user.

```

import { Error500 } from '../././config/Errors/models/error500';
import { IGenericDao } from './IGenericDao';
import { Room } from './models/room';
import { pool } from '../././config/db/connection';

export class roomDAO implements IGenericDao<Room> {

  /*****
  /***** NOT IN USE *****/
  /*****/

  async create(model: Room): Promise<Room> { throw new Error500("Not implemented", null); };
  async delete(id: string): Promise<boolean> { throw new Error500("Not implemented", null); };
  async update(id: string, model: Room): Promise<Room> { throw new Error500("Not implemented", null); };
  async getById(id: string): Promise<Room | null> { throw new Error500("Not implemented", null); };

  /*****
  /***** READ *****/
  /*****/

  async getAll(): Promise<Room[]> {
    let response;
    try {
      response = await pool.query(`SELECT 'Feature' as type,r.id as id,
      ST_AsGeoJSON(ST_Transform(ST_SetSRID(r.geom, 2100), 4326))::json as geometry,
      json_build_object('feat_type', 'line','feat_area', ST_Area(r.geom)) as properties
      FROM room r join building b on b.id = split_part(r."floorId",'.',1)
      left join mtrans t on t.target = 'osm' and t.id=split_part(r."floorId",'.',1)`);
      return response.rows;
    } catch (error) {
      throw (error);
    }
  }
}

```

**Figure 25:** An example of a DAO code snippet. In this case RoomDAO module is shown.

The information that we aim to provide to the frontend are point, connection, MRBS geospatial and MRBS room information data. To streamline the frontend data access process, the MRBSService combines all the MRBS geospatial data, as well as RoomDataService which combines all the needed room data from MRBS. In MRBSService, this process involves importing the controllers of the MRBS data tables and asynchronously fetching data from the MRBS API. Once obtained, all data is consolidated into a dictionary object using key-value pairs. For instance, to obtain the doors of room K14.0.24, one can use `geodata[K14][0][24].doors`. Geodata is the name of the dictionary, the first key is the building ID, the second is the floor ID, the third is the room ID and lastly, with the dot notation we access the properties of the specified room.

The backend of the application uses routing to manage incoming requests and direct them to the appropriate endpoint. The routing system is implemented using URL patterns and HTTP methods such as GET, POST, PUT, and DELETE. In this case, only the GET method is used.

Similarly, in RoomDataService, this process involves importing the controllers of the MRBS data table 'POI' and asynchronously fetching data from the MRBS API. Once obtained, all data is consolidated into a dictionary object using key-value pairs.

To transmit the point, connection, MRBS geospatial dictionary, room information data to the client, the application creates separate endpoints for each data type. Each endpoint imports the corresponding controller and executes it, then sends the resulting data back to the client as a response. This approach enables efficient and scalable data transmission between the backend and the client, while also providing flexibility and ease of use.

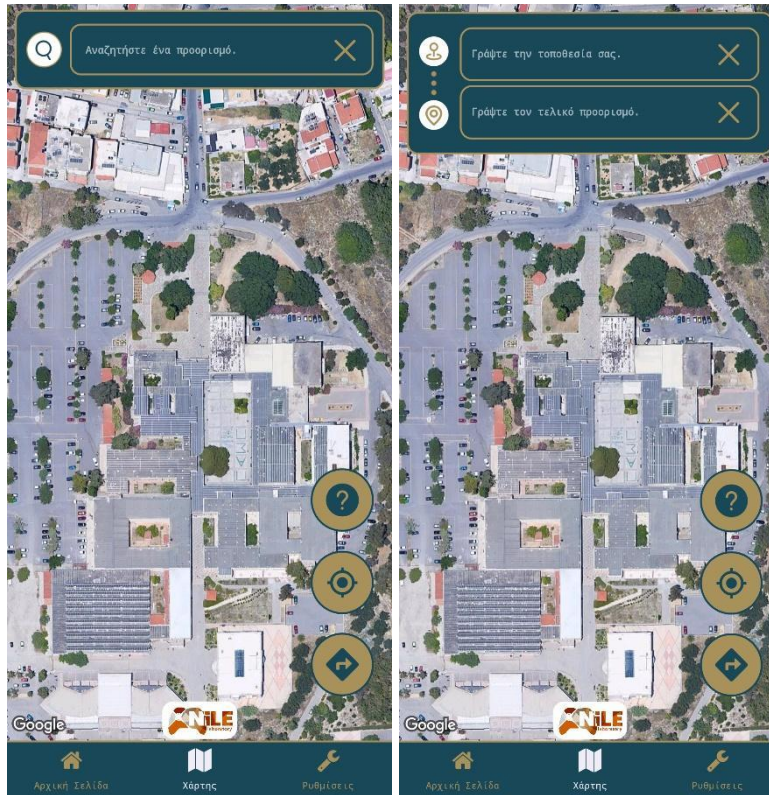
### ***5.2.3 Frontend***

This sub-chapter will focus on the frontend part of the platform, which is the visual and interactive component that users interact with. Additionally, the algorithms used to construct the application's functionality will be discussed.

#### ***5.2.3.1 Map Screen***

The Map Screen serves as the base component of most of the application, allowing users to interact with the map and access all geospatial data fetched from the application's backend. The map's default view is focused on the Hellenic Mediterranean University's Campus in Heraklion, as the application is designed for this location.

To search for specific locations within the campus and navigation from a position to a destination, users can utilize the SearchContainer component located at the top of the screen. This component provides functionality for two separate search actions, the search of a location and the input of a starting position and a final destination to navigate from one to another. The starting state of the SearchContainer is a single input field for the search action. To access the second option of navigation the user must press the button the DirectionsButton that is located at the right side of the bottom of the Map.



**Figure 26:** Map screen of the platform – (a) Search input field. (b) Navigation input fields.

The input fields are implemented as a custom component called `AutoCompleteInput`. This component allows users to input text, and as they type, it searches for relevant location names and displays them as a list of items. The list is updated in real-time as the user continues to type, helping them find the most relevant location quickly and easily.

As mentioned earlier, there are three `AutoCompleteInput` components, each of which returns a separate selected item ID. For instance, the `AutoCompleteInput` component used for searching returns `searchItemID`, the one for selecting the user's starting position returns `positionItemID`, and the one for selecting the destination returns `destinationItemID`. To access the data, we import the geodata dictionary from the backend of the application, which contains all the names that we search for.

Moreover, there is a history list of previously selected items. Every time an item is selected it is pushed to the list. The list persists throughout the use of the application and has a maximum length of four items.

In addition to the `AutoCompleteInput` components, another component utilized on this screen is the `Buttons Component`, which comprises the floating button components displayed on

the map. The component includes a Button that redirects the user to the Help screen. A Button focuses the position to the GPS position of the device when pressed, and the DirectionsButton expands/retracts the SearchContainer, allowing the user to use the navigation functionality of the application. Pressing the DirectionsButton also resets the geodata of search (marker) and navigation (path, icons, markers) that are plotted on the map.

### 5.2.3.2 Search Functionality

Upon the selection of an item in the AutoCompleteInput Component, which triggers a search, the Boolean flag searchFlag is set to true. Following this, a useEffect hook is implemented on the Map Screen, which monitors changes to the value of searchFlag. Once searchFlag is updated and its value is true, the showMarker function is invoked as the first action, which displays a marker at the location of the user's search, then the searchFlag is set to false.

Initially, the showMarker function checks if the searched location is a room, which is indicated by an ID containing the Greek letter 'K'. The letter 'K' specifies the building in which the room is situated. If the search pertains to a room, the function imports and executes formatDoorData. This function extracts the coordinates of the room's door, and its code snippet can be found in Figure 27. Using the setMarkerCoordinates function, the extracted data is then utilized to set the marker's coordinates.

If the search doesn't pertain to a room, the showMarker function proceeds to search for the searchItemID string in the points array, which contains the coordinates of all rooms on the map. If the ID is found, the function sets the marker coordinates via the setMarkerCoordinates function. If the ID is not present in the array, an error message is logged to the console.

```
if (searchItemID.includes('K')) {
  const markerCoordinatesObject = formatDoorData(searchItemID, MRBSData)[0][0];
  setMarkerCoordinates(markerCoordinatesObject);
} else {
  const markerCoordinatesObject = points.find((point: Point) => point.id === searchItemID)
  if (markerCoordinatesObject !== undefined) { // If the name exists, then
    setMarkerCoordinates({ latitude: markerCoordinatesObject.latitude, longitude: markerCoordinatesObject.longitude });
  } else { // If the name doesnt exist, then,
    console.log("The place you search was not found.");
  }
}
```

**Figure 27:** Function showMarker code snippet.

```

const splittedID = id.trim().split(".");

const myRoomDoors = MRBSDictionary[splittedID[0]][splittedID[1]][splittedID[2]];

let doorsOfRoom: LatLng[] = [];

/* Setting Doors of Room Coordinates in order to be parsed by react-native-maps */
myRoomDoors.map((door: any, index: number) => {
  if (door.json_build_object.maindoor === 1) {
    doorsOfRoom[index] = door.json_build_object.geometry.coordinates.map((coordinate: number[]) => {
      return { latitude: coordinate[1], longitude: coordinate[0] };
    });
  }
});

/* filter out undefined doors. */
doorsOfRoom = doorsOfRoom.filter((e: LatLng) => e !== undefined);

/* then return them to component */
return doorsOfRoom;

```

**Figure 28:** Geospatial data formation function formatDoorData code snippet.

In addition, when a user initiates a search for a particular room, the system automatically generates plots for all rooms located on the same floor as the searched room. In order to highlight the searched room, an additional red outline plot is added to the generated plots. The FloorRooms and HighlightedRooms components are responsible for providing this functionality, which are activated when the values of searchItemID, positionItemID, and destinationItemID are not null.

The FloorRooms component utilizes the react-native-maps library to plot polygon geometries that define each room on the floor where the searched room is located, thereby defining the floor outline. To format the data extracted from the mrbs dictionary into a coordinates array that can be parsed by the react-native-maps library, the formatFloorData function is used.

```

const splittedID = id.trim().split(".");

const myRoomsOfFloor = MRBSDictionary[splittedID[0]][splittedID[1]].rooms;

const roomsOfFloor: LatLng[][] = [];

/* Setting Doors of Room Coordinates in order to be parsed by react-native-maps */
myRoomsOfFloor.map((room: any, index: number) => {
  roomsOfFloor[index] = room.geometry.coordinates[0][0].map((coordinate: number[]) => {
    return { latitude: coordinate[1], longitude: coordinate[0] };
  });
});

return roomsOfFloor;

```

**Figure 29:** Geospatial data formation function formatFloorData code snippet.



Similarly, the HighlightedRooms component functions in a similar way, but utilizes the formatRoomData function to return the coordinates array of the room being searched. By incorporating these components, the system can provide a comprehensive visualization of all rooms on the same floor as the searched room, while also highlighting the searched room in a visually distinct manner.

```

const splittedID = id.trim().split(".");

const myRoomsOfFloor = MRBSDictionary[splittedID[0]][splittedID[1]].rooms;

const roomCoordinates: LatLng[] = [];

/* Setting Room Coordinates in order to be parsed by react-native-maps */
myRoomsOfFloor.forEach((room: any) => {

    if (JSON.stringify(room.id) === JSON.stringify(id)) {
        room.geometry.coordinates[0][0].map((coordinate: number[], coordinateIndex: number) => {
            roomCoordinates[coordinateIndex] = { latitude: coordinate[1], longitude: coordinate[0] };
        });
    }
});

return roomCoordinates;

```

**Figure 30:** Geospatial data formation function formatRoomData code snippet.

### 5.2.3.3 Navigation Functionality

The core functionality of the virtual guide platform is centered around navigating through the HMU campus, hence the name of the application itself. To achieve the navigation feature, the positionItemID and destinationItemID parameters are utilized. When a user selects both a starting position and a destination in the SearchContainer, the plotPathFlag is set to true, which in turn initiates the necessary actions to plot the path.

Within the Map Screen, a useEffect function is employed to listen for changes in the pathPlotFlag. When the Boolean value is true, the getPathVariables function is executed. This function is responsible for checking and defining the starting point and destination point of the path, which are then used as arguments in the functions that plot the paths. The function returns the PathArgs object, which includes the starting\_point and destination\_point attributes, e.g., {starting\_point: 'K14.0.14', destination\_point: '144'}.

In addition, another useEffect function is utilized to detect changes in the pathArgs. When a change is detected, the selectAndDrawPath function is triggered. This function selects the appropriate type of path required based on the pathArgs. There are three types of locations: rooms,

which have geospatial data associated with them; points in a grid with a name, which do not have geospatial data associated with them; and GPS locations, which are temporary points that change over time.

To ensure a structured and comprehensible codebase, we utilized use cases to split the functionality. These use cases are visible in Figure 31 below, and they provide a clear and modularized implementation of the navigation feature. The result is a codebase that is easier to comprehend, maintain, and scale.

<b>starting_point</b>	<b>destination_point</b>
GPS	Node
GPS	Room
Node	GPS
Node	Room
Node	Room
Room	GPS
Room	Node
Room	Room

**Figure 31:** Navigation use cases.

Regarding Figure 31, it is worth noting that there are eight use cases, each represented by a distinct function that plots a different type of path. In each function, the primary operation is the `shortestPath` function, which is responsible for calculating the shortest path between two points on the grid. However, there are other functions that are involved in the process in most cases.

To provide a more comprehensive understanding of the navigation feature, let us discuss each of the eight use cases and the actions they take to plot the path:

- **GPS to Room Path:**

This use case is responsible for plotting the shortest path from a GPS location to a room on campus. It uses the current GPS location of the user as the starting point, and the selected room as the destination. The GPS location is a temporary point that doesn't belong to the grid, therefore it must be connected momentarily with the grid with the use of a function called `closestNode`. There are 3 types of GPS locations, the location that is inside a building that has polygon geometry in the database, the location that is not inside a building but inside the campus and the location that it is outside of the campus. Depending on the type of the GPS location the path differs, if it is the

first type, the path connects the GPS location point inside of the building to the entrance of the building which belongs to the grid, so the path starts from the entrance of the building. That is done because although there is grid inside the building we cannot know the floor of which the GPS location is located. If GPS location is inside the campus but not inside a building, user then can easily connect with the closest node of the grid, so the GPS is connected with the closest point of the grid. For the last case, if the GPS location is outside of the campus then it connects it with the closest point that is also an entrance. As for the room, the geospatial data of its door are retrieved and connected to the closest point of the grid that is located at the same floor. The GPS location connection to the grid is called GPSToGridPath and the room to grid connection is called DestinationToGridPath and that's because the room is set as the destination point(destination). Then calculates the shortest path from that point to the selected grid point using the shortestPath function.

- Room to GPS Path:

This use case is similar to the previous one, but it plots the path from a selected room to the user's current GPS location. It works with same manner as the previous use case. The only change is that instead of DestinationToGridPath, the room to grid connection is called PositionToGridPath and that's because the room is set as the destination point. Then calculates the shortest path from that point to the selected grid point using the shortestPath function.

- Room to Room Path:

This use case plots the shortest path between two selected rooms on campus. It identifies the closest point on the grid of the floor that is shared with the floor of each room and connects to it to the doors of the rooms, and then calculates the shortest path between those two points using the shortestPath function. The connections of the two rooms to their closest points are stored in the states PositionToGridPath and DestinationToGridPath.

- Node to Node Path:

This use case plots the shortest path between two points on the grid that do not correspond to any particular room. It directly calculates the shortest path between the two points using the shortestPath function.

- GPS to Node Path:

This use case plots the shortest path from a user's GPS location to a point on the grid. It first identifies the closest point on the grid to the corresponding GPS location type as explained in the first use case, stores the path GPSToGridPath and then calculates the shortest path from that point to the selected grid point using the shortestPath function.

- Node to GPS Path:

This use case is the inverse of the previous one, plotting the shortest path from a point on the grid to the user's GPS location. It first identifies the closest point on the grid to the corresponding GPS location type as explained in the first use case, stores the path GPSToGridPath and then calculates the shortest path from that point to the selected grid point using the shortestPath function.

- Room to Node Path:

This use case plots the shortest path from a selected room to a point on the grid. It first identifies the closest point on the grid that is at the same floor as the room and connects it to the selected room's door, and then calculates the shortest path from that point to the selected grid point using the shortestPath function.









- Node to Room Path:

This use case is the inverse of the previous one, plotting the shortest path from a point on the grid to a selected room. It first identifies the closest point on the grid that is at the same floor as the room and connects it to the selected room's door, and then calculates the shortest path from that point to the selected room using the shortestPath function.

Creating the path involves performing several actions for each use case. To aid users in understanding the path, markers, icons, and different colors are used in addition to the path itself. The "createPath" function is used to set different colors for each floor of the path. This function takes the shortest path that has been calculated and splits it into parts based on the floor, using an algorithm.

Once the paths have been split, they must be reconnected with the appropriate colors. To accomplish this, another algorithm has been created that matches the colors between the paths.

This ensures that the path is properly displayed to the user with each floor marked in its corresponding color. The algorithm that creates the new path is shown in Figure 33 and Figure 34.

Marker	Description
	Displays the entry point of the path.
	Displays the final point of the path.
	Displays the location of the destination.
	Displays the location of the GPS position.
	Displays the entrance of a building.
	Displays the use of an elevator to change floors.
	Displays the use of a staircase that lead to a lower floor.
	Displays the use of a staircase that lead to an upper floor.

**Figure 32:** The path symbols.

In each use case, markers are given appropriate coordinates to ensure they are displayed properly in the final path. However, setting these markers in a use case function is not always possible. To set the final variables of the path, such as markers, three functions are used: "PlotPath", "PlotPathWithClosestNode", and "DefaultPlotPath". These functions specialize in setting the last variables of the path.

An object called "iconNodeObject" is created in these functions, which contains the points that need an icon displayed at their location. Icons can represent doors, elevators, upstairs, and downstairs, as shown in Figure 32. To create the "iconNodeObject", the "filterIconNodes" function is called. This function takes the path filtered to an array of points as an argument. An algorithm is then run that iterates through the path array and filters if a point has a specified icon type, storing the index of the point in the specified icon type array. The algorithm is shown in the code snippets in Figures 36 and 37.

```

/* Filters the points Array the indicate the path to be plotted.*/
const filteredCoordinates = filterPath(pathToPlot, points);

let arrayIndex = 0; // Initializing the index of filtered coordinates array.
const pathsArray: Point[][] = []; // Initializing the paths array that will contain the all the paths of the plot.
let path: Point[] = []; // Initializing a temp path array.
let lastIndex: number = 0; // Initializing a lastIndex variable.

/* Creating the paths of each floor of the grid */
for (arrayIndex; arrayIndex < filteredCoordinates.length; arrayIndex++) {

  let currentPointsFloor = '';
  let nextPointsFloor = '';
  lastIndex = arrayIndex;

  /* If this point is not the last of the path. */
  if (filteredCoordinates[arrayIndex + 1] !== undefined) {

    /* Filtering the types of the current point and its next */
    /* If this point is a "road" point (not inside building point eg. 144) */
    if (filteredCoordinates[arrayIndex].id.split('.')[1] === undefined)
      currentPointsFloor = '0'; // Set its floor to be 0 floor.
    /* Else, if this point "floor" point (inside building point eg. 0.14.3) */
    else
      currentPointsFloor = filteredCoordinates[arrayIndex].id.split('.')[0]; // Set its floor to be the floor indicated.

    /* If the next point is a "road" point (not inside building point eg. 139) */
    if (filteredCoordinates[arrayIndex + 1].id.split('.')[1] === undefined)
      nextPointsFloor = '0'; // Set its floor to be 0 floor.
    /* Else, if this point "floor" point (inside building point eg. 0.14.4) */
    else
      nextPointsFloor = filteredCoordinates[arrayIndex + 1].id.split('.')[0]; // Set its floor to be the floor indicated.

    /* Push the current point in the path */
    path.push(filteredCoordinates[arrayIndex]);

    /* If the floor of the current point is different than the its next point, then,
    // it indicates a break point, they must be broken into 2 different paths.
    if (currentPointsFloor !== nextPointsFloor) {
      pathsArray.push(path); // push temp path in the path array.
      path = [];
    }
    /* If this point is the last of the path. */
  } else {
    path.push(filteredCoordinates[lastIndex]);
    pathsArray.push(path); // push temp path in the path array.
  }
}

```

**Figure 33:** Function createPath code snippet(a).

For each icon type array, the "iconCheck" function is run, which also implements an algorithm. This algorithm takes the index array of each icon type and accesses the corresponding point in the path array. The problem that the "iconCheck" function solves, can be easily explained with an example: if the starting point of the path is on the ground floor and the destination point is on the second floor, the filtered array finds that four points have an icon type. Two points have an icon type of 'down' and the others have 'up'. Without the use of the "iconCheck" function, all four points would display their corresponding icon in the map. This would confuse the user, as one icon type indicates the use of stairs to get to an upper floor and the other type indicates the exact opposite.

```

/* Connecting every path of the pathsArray, ultimately creating new path to act as connectors */
let pointIndex = 0;
const pathConnectors: [Point, Point][] = [];
let maxFloorOfConnectors: number[] = [];

for (pointIndex; pointIndex < pathsArray.length; pointIndex++) {

  const lastPointOfPath: Point = pathsArray[pointIndex].slice(-1)[0];

  let lastPointOfPath_FloorID = 0;

  /* If the last Point Of Path is a "road" point (not inside building point eg. 139) */
  if (lastPointOfPath.id.split('.')[1] !== undefined)
    lastPointOfPath_FloorID = Number(lastPointOfPath.id.split('.')[0]);

  /* If this point is not the last of the path. */
  if (pathsArray[pointIndex + 1] !== undefined) {
    const firstPointOfPath: Point = pathsArray[pointIndex + 1][0];

    let firstPointOfPath_FloorID = 0;
    if (firstPointOfPath.id.split('.')[1] !== undefined)
      firstPointOfPath_FloorID = Number(firstPointOfPath.id.split('.')[0]);

    pathConnectors.push([lastPointOfPath, firstPointOfPath]);

    /* If the last or first point has floor ID 00(is basement) then replace 00 with -1 */
    if (JSON.stringify(lastPointOfPath_FloorID) === JSON.stringify('00') || JSON.stringify(firstPointOfPath_FloorID) === JSON.stringify('00'))
      maxFloorOfConnectors.push(-1); // Push the -1 as the max.
    else
      maxFloorOfConnectors.push(Math.max(lastPointOfPath_FloorID, firstPointOfPath_FloorID)) // Calculate and Push the max of this 2.
  }
}

const pathObject: PathObject = { path: pathsArray, pathConnectors: pathConnectors, maxFloorOfConnectors: maxFloorOfConnectors };

return pathObject;

```

**Figure 34:** Function createPath code snippet(b).

The "iconCheck" function provides directional sense to the "filterIconNodes" function, so that only the icons that comply with the direction of the path will be rendered in the map. In the example given above these would be the points that have as icon type "up".

Initially, the algorithm checks if there is a point three points ahead on the path. If such a point exists, the algorithm proceeds to examine the icon type direction, which indicates whether the path leads to an upper or lower floor. Based on this information, the algorithm evaluates if the point's floor is greater than the user's current floor for icon types that signify an ascent to an upper level, or if the point's floor is smaller than the user's current floor for icon types that denote a descent to a lower level.

By utilizing these distinct functions and checks, the navigation feature is capable of plotting the most suitable path for a variety of situations, taking into account factors such as floor transitions and the user's current location. This enables the virtual guide platform to offer a seamless and

intuitive navigation experience for users, helping them to efficiently navigate complex multi-level environments with ease and confidence.

```

/* Filters the points Array the indicate the path to be plotted.*/
let filteredPathArray: Point[] = filterPath(pathToPlot, points);

const iconNodeObject = filterIconNodes(filteredPathArray);
setIconNodeObject(iconNodeObject);

/* If the filter is set and not null, THEN, */
if (typeof filteredPathArray[0] !== "undefined") {
  const coordinatesArray: LatLng[] = pathToCoordinates(filteredPathArray);

  // @ts-ignore
  const pathLastNode: LatLng = coordinatesArray.slice(-1).pop();

  // Sets and returns the data.
  /* If the GPS is set as the destination */
  if (GPSAsDestinationFlag) {
    setDestinationMarkerCoordinates(position);
    setDestinationNodeMarkerCoordinates(pathLastNode);
  } else if (roomAsDestinationFlag) { /* ELSE If, a ROOM is set as the destination */
    setDestinationMarkerCoordinates(position);
    setDestinationNodeMarkerCoordinates(position);
  } else { // ELSE,
    setDestinationMarkerCoordinates(pathLastNode);
    setDestinationNodeMarkerCoordinates(pathLastNode);
  }

  if (positionMarkerFlag)
    setPositionMarkerCoordinates(coordinatesArray[0]);

  setPathCoordinates(coordinatesArray);
}

```

**Figure 35:** Function PlotPathWithClosestNode code snippet.

```

const indexesStairsUp: number[] = [];
const indexesStairsDown: number[] = [];
const indexesElevator: number[] = [];
const indexesDoor: number[] = [];

filteredPathArray.forEach((point: Point, index: number) => {
  if (point.icon_type === 'up') {
    indexesStairsUp.push(index);
  }
  if (point.icon_type === 'down') {
    indexesStairsDown.push(index);
  }
  if (point.icon_type === 'elevator') {
    indexesElevator.push(index);
  }
  if (point.icon_type === 'door') {
    indexesDoor.push(index);
  }
})

const upstairsNodes = iconCheck(filteredPathArray, indexesStairsUp, "up");
const downstairsNodes = iconCheck(filteredPathArray, indexesStairsDown, "down");
const elevatorUpNodes = iconCheck(filteredPathArray, indexesElevator, "up");
const elevatorDownNodes = iconCheck(filteredPathArray, indexesElevator, "down");
const doorNodes = iconCheck(filteredPathArray, indexesDoor, "null");

return {
  doorNodes: doorNodes,
  upstairsNodes: upstairsNodes,
  downstairsNodes: downstairsNodes,
  elevatorUpNodes: elevatorUpNodes,
  elevatorDownNodes: elevatorDownNodes
}

```

**Figure 36:** Function filterIconNodes code snippet(a).



```

return indexArray.map((index: number) => {
    if (filteredPathArray[index + 3] !== undefined) {
        let indexID = filteredPathArray[index].id.split('.')[0];
        let indexIDUndefined = filteredPathArray[index].id.split('.')[1];

        let index3ID = filteredPathArray[index + 3].id.split('.')[0];
        let index3IDUndefined = filteredPathArray[index + 3].id.split('.')[1];

        if (indexIDUndefined === undefined)
            indexID = '0';
        if (index3IDUndefined === undefined)
            index3ID = '0';
        if (indexID === '00')
            indexID = '-1';
        if (index3ID === '00')
            index3ID = '-1';

        if (iconDirection === 'up') {
            if (indexID < index3ID) {
                return filteredPathArray[index];
            }
        } else if (iconDirection === 'down') {
            if (indexID > index3ID) {
                return filteredPathArray[index];
            }
        } else {
            return filteredPathArray[index];
        }
    } else {
        return undefined;
    }
}).filter((point: Point | undefined) => point !== undefined) as Point[];

```

**Figure 37:** Function filterIconNodes code snippet(b).

All the path variables established within the various functions mentioned previously play a crucial role in the overall functionality and efficiency of the virtual guide platform. These variables are accessible to several key components, including GridPolylines, NonGridPolylines, FloorRooms, HighlightedRooms, and Markers. Each of these components is designed to recognize when the geospatial data they rely on is not empty, and as a result, dynamically render the corresponding visual elements on the platform.

The GridPolylines component is responsible for rendering polylines on the map that are generated by the connection of points that are premade with QGIS, helping users to better understand the layout of the space and providing a clear visual reference for navigation. Similarly, the NonGridPolylines component is responsible for displaying non-grid polylines, which represent polylines that are made by the connection of dynamic points such as the connection of GPS position point to the closest point of the grid.

FloorRooms, on the other hand, focuses on rendering the rooms and spaces within a specific floor plan. By dynamically drawing these elements, users can easily visualize the locations and dimensions of rooms, aiding them in their navigation and understanding of the space. The

HighlightedRooms component is designed to emphasize specific rooms, either as a result of user input or based on certain criteria or filters. This allows users to quickly identify and focus on rooms that are relevant to their needs or preferences.

Finally, the Markers component is responsible for rendering various markers on the map, which represent the position of searched location. These markers can provide valuable context and information to users, enhancing their overall experience and interaction with the platform.

The interconnected nature of these components and their reliance on the path variables is essential in creating a dynamic and visually rich virtual guide platform. By ensuring that these components can access and render geospatial data effectively, the platform can provide users with an engaging, informative, and intuitive experience that caters to their needs and preferences.

## Chapter 6 - Results / Demo

In this chapter, we will present the final result of the application that has been designed and developed. The presentation will begin with the loading screen and follow the flow of screens that users will encounter when using the app. Each section will cover a specific screen and the elements that users must be familiar with in order to fully utilize the system's capabilities.

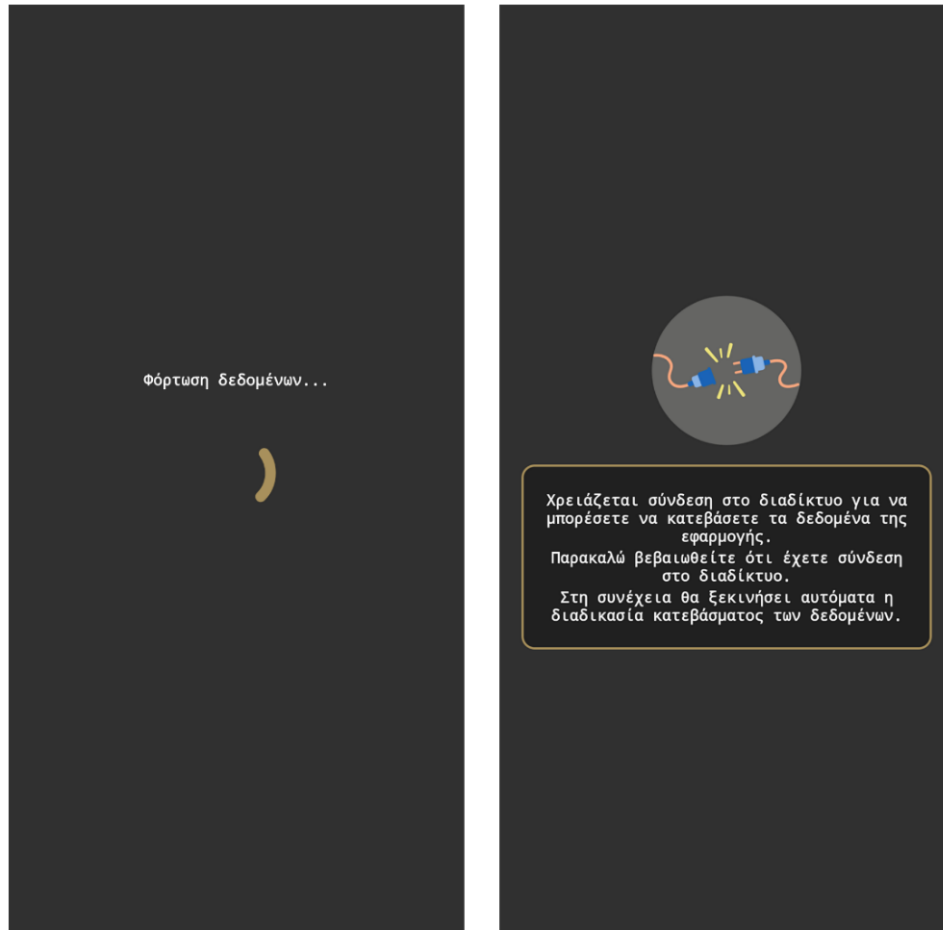
We will start with the loading screen, which provides users with visual feedback while the app initializes. Once the app is loaded, the user is presented with the home screen, which serves as the main hub for accessing all the app's features.

From there, we will discuss each screen in detail, including any input fields, buttons, or other interactive elements that are present. We will also cover any relevant functionality, such as search, and explain how users can use them to achieve their goals.

Throughout this chapter, we will provide visual aids, such as screenshots, to help users better understand the app's interface and workflow. By the end of this chapter, users should have a comprehensive understanding of how to use the application and take advantage of its features.

### 6.1 Loading Screen

The loading screen is the first screen that the user encounters when launching the application. Its primary purpose is to inform the user about crucial data required for the proper function of the application being downloaded. Furthermore, it ensures that these data are being downloaded. If the user is not connected to the internet, a modal will be displayed, informing the user of the lack of connectivity and encouraging them to connect to the internet. The user will be unable to access other screens of the application until the necessary data are retrieved.



**Figure 38:** Loading screen screenshots.

## 6.2 Home Screen

The home screen serves as the welcome page of the application, allowing users to navigate throughout the app. By pressing the central button on the screen, users can access the most important screen of the app, which is the map screen. A tab bar is displayed at the bottom of the screen, enabling users to switch between the main screens of the application, including the home screen, map screen, and settings screen. This tab bar is visible regardless of which main screen the user is currently on.

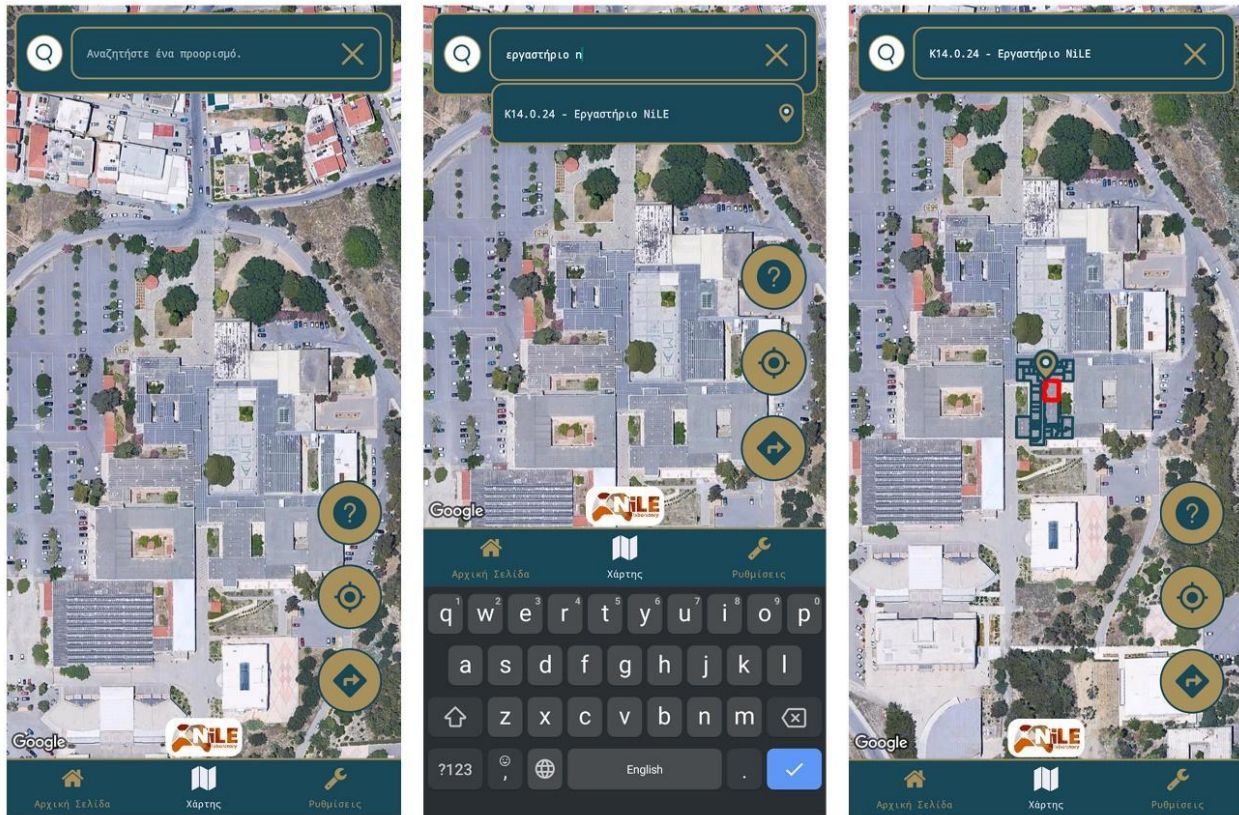


**Figure 39:** Home screen screenshot.

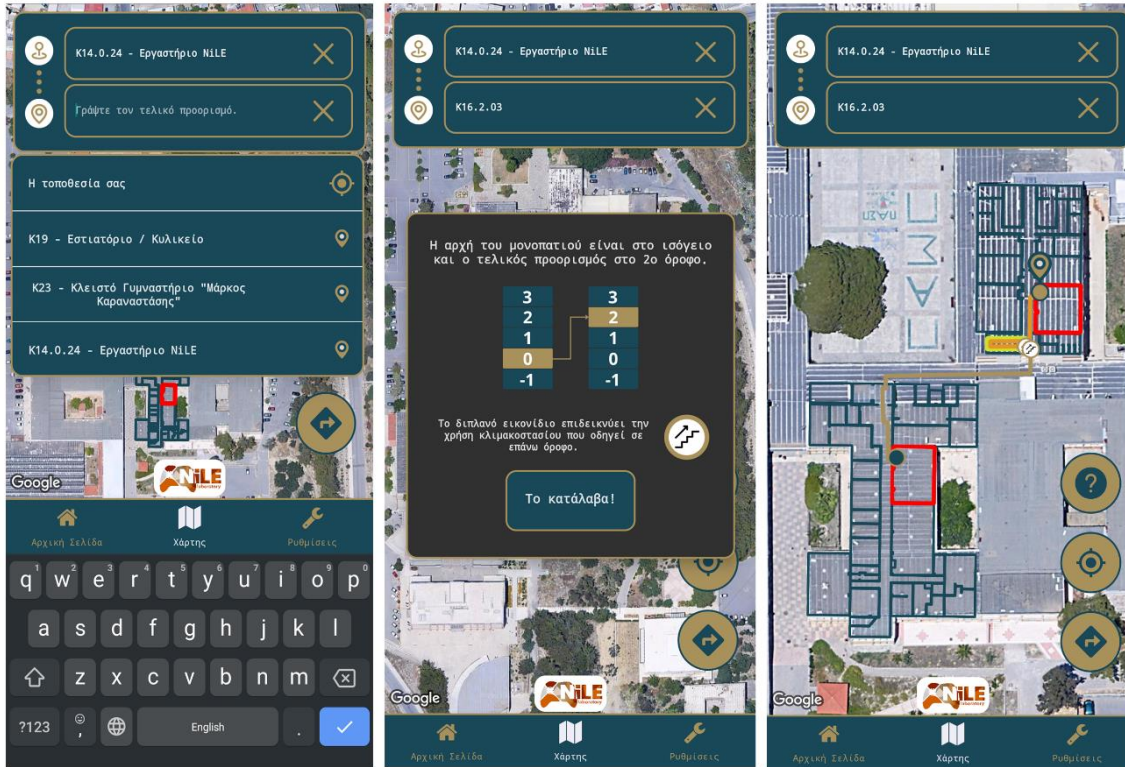
### 6.3 Map screen

The map screen is a central and indispensable feature of the application that provides users with an interactive map to search and navigate locations on the Hellenic Mediterranean University's Campus in Heraklion. It features an intuitive interface with easily accessible buttons that enable users to switch between search and navigation input fields, focus the map on their current location, and access the help screen. The flow of actions for searching, navigating, and accessing information is made clear through the accompanying figures. More specifically in Figure

41 in the middle screenshot the path info modal is visible. The path info modal contains verbose information about the path, helping the user to properly navigate.



**Figure 40:** Map screen iteration screenshots when user searches a destination.



**Figure 41:** Map screen iteration screenshots when user wants to navigate.

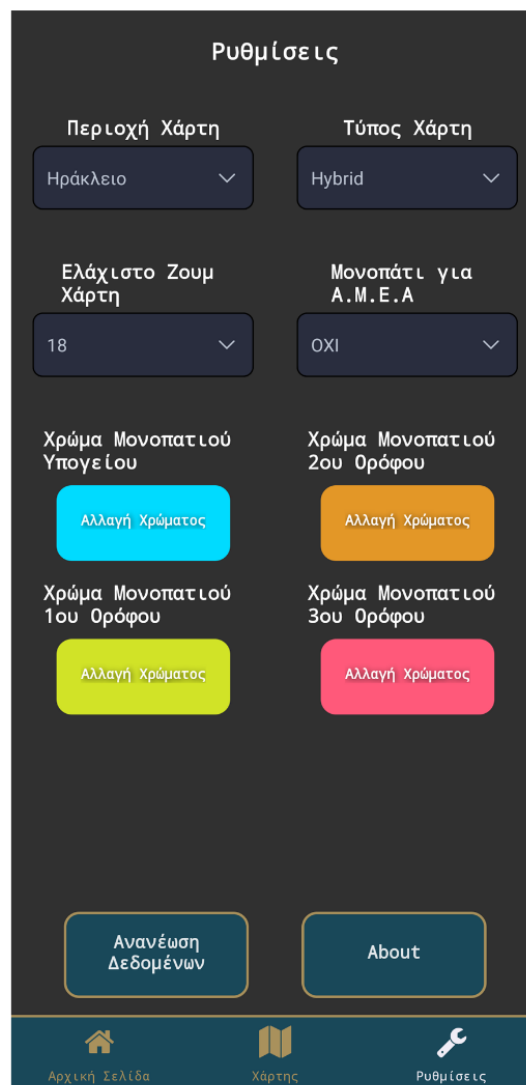


**Figure 42:** Map screen screenshot when user has pressed the marker of a searched location.

## 6.4 Settings screen

The settings screen is a crucial component of the application that enables users to customize the map's appearance and behavior to suit their preferences. It provides an easy-to-use interface that allows users to adjust settings such as the map's region, type, and minimum zoom level, which affect how the map is displayed on the screen. Moreover, the settings screen also offers users the ability to modify the color scheme used to depict paths and enable the platform to plot a path that is accessible for persons with disabilities.

The "update app's data" and "about" buttons provide additional functionality to the settings screen, allowing users to download the latest data for the application and access information about the app respectively.



**Figure 43:** Settings screen screenshot.



## 6.5 Information screen

The information screen is responsible for providing relevant information about a location, and it can be accessed by navigating through the "map" screen and pressing the marker as illustrated in Figure 40. The information screen comprises two distinct categories of data. The first category depicts the room reservations, which are presented in the form of a schedule for a given date. It is important to note that this category is only applicable for rooms that are reservable. The second category portrays the static data of a room, such as a small description of what this room is. In the event that a room does not have any data available, the screen will display a message indicating that no data is available. At the moment there are not available data from the Hellenic Mediterranean University's Administration, so every room that is not reservable would not have any data. It is noteworthy that the schedule displayed for reservable rooms can be modified by clicking on the "change date" button. This feature allows the user to access the reservation schedule for any given date.

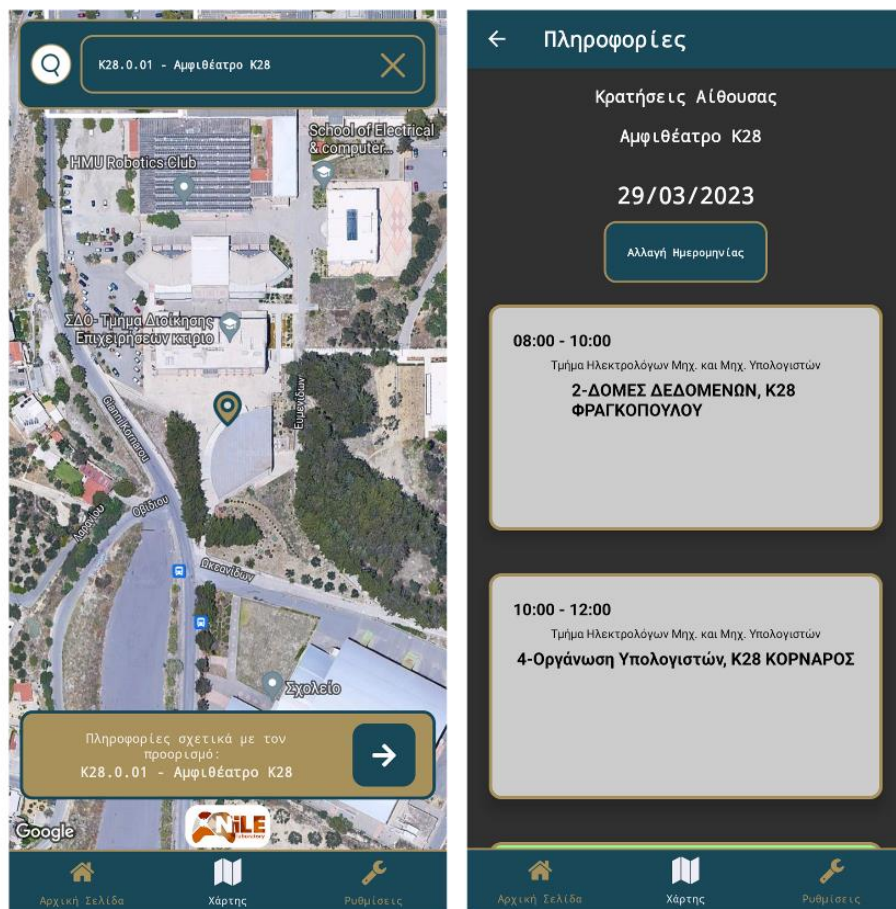
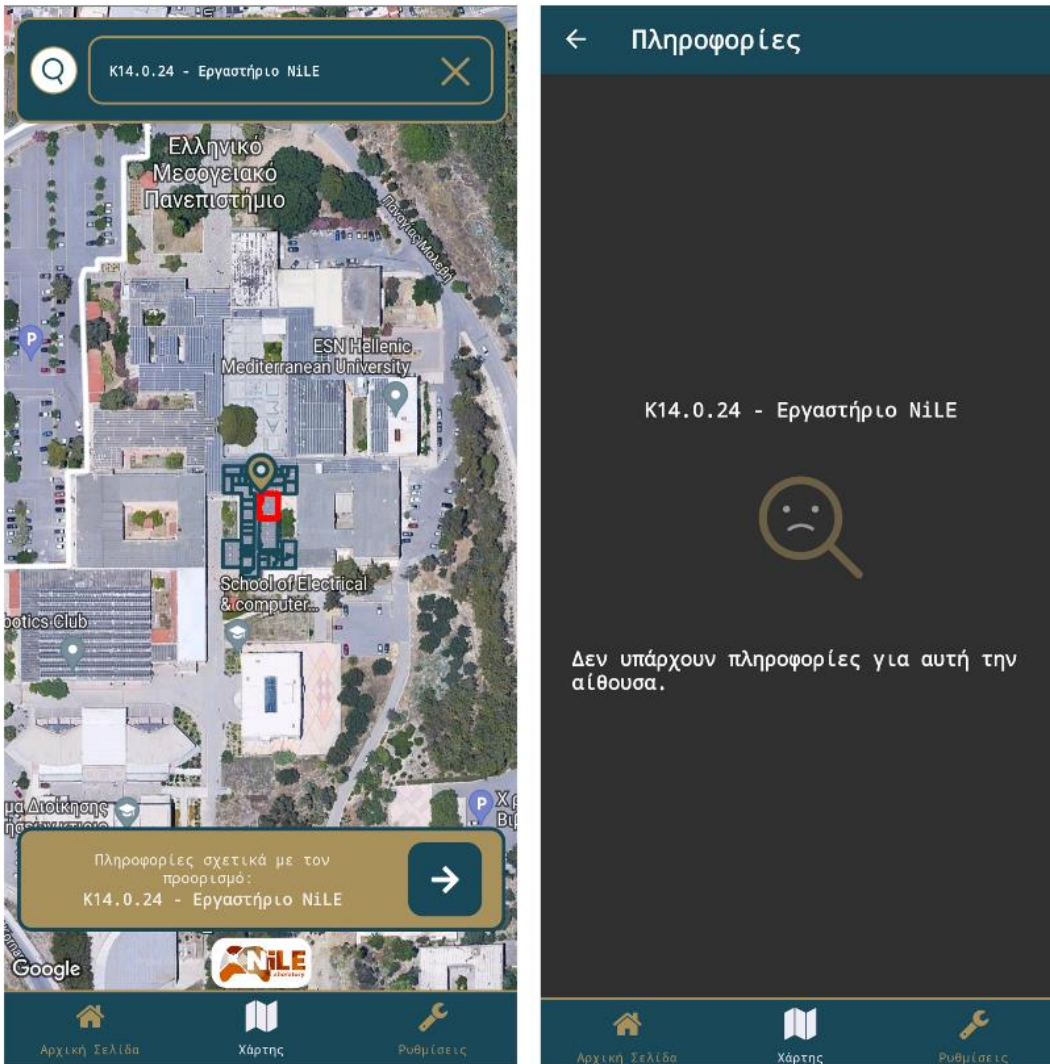


Figure 44: Information screen screenshot the room is reservable.



**Figure 45:** Information screen when the room is not reservable and has no data.

## 6.6 Help screen

The help screen of the application is responsible for providing the troubled user with vital information about the map and its functions. It consists of the map index, that explains several markings that will be present at the interactive map, such as the markers and the colors of the path sections. Additionally, it provides “tutorials” on how to use the map in its full capabilities.



Figure 46: Help screen screenshots.

## Chapter 7 - Conclusion and future work

In conclusion, this thesis has successfully demonstrated the development of a comprehensive virtual guide platform that addresses the navigation challenges faced by university students, faculty members, and visitors on campuses. By utilizing advanced technology and user-centered design principles, the platform provides a seamless navigation experience, enabling users to locate their destinations, access information, and appreciate the contextual aspects of their surroundings.

The platform's design emphasizes data scalability and ease of maintenance, ensuring its long-term adaptability and relevance in the face of evolving campus environments and user needs.

The resulting solution not only offers free development and publication but also empowers individuals to be informed about classes, effectively manage their time, and engage more deeply with the campus community. Ultimately, this innovative virtual guide platform has the potential to significantly enhance the overall campus experience for students, faculty members, and visitors alike, transforming the way individuals navigate and interact with complex university environments.

While this thesis has successfully developed a virtual guide platform for university campuses, there is still much potential for future work in this area. One possible area of future work could involve incorporating real-time position tracking and instructing the user with information on where to navigate to each step, making the navigation experience more intuitive and user-friendly.

Another potential area of future work could be to improve the platform's accessibility features for users with disabilities, such as by integrating assistive technologies like voice-guided navigation and text-to-speech capabilities. These enhancements would enable users with visual impairments to navigate the campus independently.

In addition, extra potential for future work could be to personalize the search experience for each student. By requiring students to sign in with their university credentials, the platform could prioritize locations and information relevant to their individual needs, such as displaying the classes they have in the current semester first.

Overall, these potential areas of future work could enhance the functionality, accessibility, and personalization of the virtual guide platform, providing an even more seamless and informative navigation experience for university students, faculty members, and visitors.

## References

- [1] H. Li and L. Zhijian, “The study and implementation of mobile GPS navigation system based on Google Maps,” in *2010 International Conference on Computer and Information Application*, 2010, pp. 87–90.
- [2] Google Maps, <https://google.com/maps>
- [3] F. Lu, H. Zhou, L. Guo, J. Chen, and L. Pei, “An ARCore-Based Augmented Reality Campus Navigation System,” *Applied Sciences*, vol. 11, no. 16, p. 7515, 2021.
- [4] E. D. Kaplan and C. Hegarty, *Understanding GPS/GNSS: principles and applications*. Artech house, 2017.
- [5] S.-C. Yeh, W.-H. Hsu, M.-Y. Su, C.-H. Chen, and K.-H. Liu, “A study on outdoor positioning technology using GPS and WiFi networks,” in *2009 International Conference on Networking, Sensing and Control*, 2009, pp. 597–601.
- [6] M. Yassin and E. Rachid, “A survey of positioning techniques and location based services in wireless networks,” in *2015 IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems (SPICES)*, 2015, pp. 1–5.
- [7] J. Kunhoth, A. Karkar, S. Al-Maadeed, and A. Al-Ali, “Indoor positioning and wayfinding systems: a survey,” *Human-centric Computing and Information Sciences*, vol. 10, no. 1, pp. 1–41, 2020.
- [8] V. Roy, “Smartphone Localization for Indoor Pedestrian Navigation,” Carnegie Mellon University Pittsburgh, PA, 2022.
- [9] J. Dong, M. Noreikis, Y. Xiao, and A. Ylä-Jääski, “ViNav: A vision-based indoor navigation system for smartphones,” *IEEE Trans Mob Comput*, vol. 18, no. 6, pp. 1461–1475, 2018.
- [10] H. Xia, J. Zuo, S. Liu, and Y. Qiao, “Indoor localization on smartphones using built-in sensors and map constraints,” *IEEE Trans Instrum Meas*, vol. 68, no. 4, pp. 1189–1198, 2018.
- [11] O. Asraf, F. Shama, and I. Klein, “PDRNet: A deep-learning pedestrian dead reckoning framework,” *IEEE Sens J*, vol. 22, no. 6, pp. 4932–4939, 2021.
- [12] X. Hou and J. Bergmann, “Pedestrian dead reckoning with wearable sensors: A systematic review,” *IEEE Sens J*, vol. 21, no. 1, pp. 143–152, 2020.
- [13] S. Shang and L. Wang, “Overview of WiFi fingerprinting-based indoor positioning,” *IET Communications*, vol. 16, no. 7, pp. 725–733, 2022.

- [14] R. Giuliano *et al.*, “Indoor localization system based on bluetooth low energy for museum applications,” *Electronics (Basel)*, vol. 9, no. 6, p. 1055, 2020.
- [15] A. K. Taşkan and H. Alemdar, “Obstruction-aware signal-loss-tolerant indoor positioning using bluetooth low energy,” *Sensors*, vol. 21, no. 3, p. 971, 2021.
- [16] D. H. Mai, H. D. Le, T. V. Pham, and A. T. Pham, “Design and performance evaluation of large-scale VLC-based indoor positioning systems under impact of receiver orientation,” *IEEE Access*, vol. 8, pp. 61891–61904, 2020.
- [17] R. Vleugels, B. Van Herbruggen, J. Fontaine, and E. De Poorter, “Ultra-wideband indoor positioning and IMU-based activity recognition for ice hockey analytics,” *Sensors*, vol. 21, no. 14, p. 4650, 2021.
- [18] A. Candra, M. A. Budiman, and R. I. Pohan, “Application of A-Star Algorithm on Pathfinding Game,” in *Journal of Physics: Conference Series*, 2021, p. 12047.
- [19] G. Tang, C. Tang, C. Claramunt, X. Hu, and P. Zhou, “Geometric A-star algorithm: An improved A-star algorithm for AGV path planning in a port environment,” *IEEE access*, vol. 9, pp. 59196–59210, 2021.
- [20] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.
- [21] D. Rachmawati and L. Gustin, “Analysis of Dijkstra’s algorithm and A\* algorithm in shortest path problem,” in *Journal of Physics: Conference Series*, 2020, p. 12061.
- [22] M. R. Wayahdi, S. H. N. Ginting, and D. Syahputra, “Greedy, A-Star, and Dijkstra’s algorithms in finding shortest path,” *International Journal of Advances in Data and Information Systems*, vol. 2, no. 1, pp. 45–52, 2021.
- [23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.
- [24] A. Stentz, “Optimal and efficient path planning for unknown and dynamic environments,” 1993.
- [25] A. E. Mirino and others, “Best routes selection using Dijkstra and Floyd-Warshall algorithm,” in *2017 11th International Conference on Information & Communication Technology and System (ICTS)*, 2017, pp. 155–158.
- [26] S. Hougardy, “The Floyd–Warshall algorithm on graphs with negative cycles,” *Inf Process Lett*, vol. 110, no. 8–9, pp. 279–281, 2010.

- [27] URWalking, <https://urwalking.ur.de/navi/>(accessed Apr. 3, 2023).
- [28] B. Ludwig, G. Donabauer, D. Ramsauer, and K. al Subari, “URWalking: Indoor Navigation for Research and Daily Use,” *KI-Künstliche Intelligenz*, pp. 1–8, 2023.
- [29] A. Satan, “Bluetooth-based indoor navigation mobile system,” in *2018 19th international carpathian control conference (ICCC)*, 2018, pp. 332–337.
- [30] J. Pearson, S. Robinson, and M. Jones, “BookMark: Appropriating existing infrastructure to facilitate scalable indoor navigation,” *Int J Hum Comput Stud*, vol. 103, pp. 22–34, 2017.
- [31] QGIS website, “QGIS” <https://www.qgis.org/en/site/> (accessed Mar. 22, 2023).
- [32] The Open Source Geospatial Foundation, “QGIS Desktop” <https://www.osgeo.org/projects/qgis/> (accessed Mar. 22, 2023).
- [33] PostgreSQL documentation, “PostgreSQL” <https://www.postgresql.org/files/documentation/pdf/15/postgresql-15-A4.pdf> (accessed Mar. 22, 2023).
- [34] PostGIS official website, “About PostGIS” <https://postgis.net> (accessed Mar. 22, 2023).
- [35] PostGIS documentation, “PostGIS” <https://postgis.net/stuff/postgis-3.3.pdf> (accessed Mar. 22, 2023).
- [36] Node.js official website, “NodeJS” <https://nodejs.org/en/about> (accessed Mar. 22, 2023).
- [37] Express.js official website, “Express.js” <https://expressjs.com> (accessed Mar. 22, 2023).
- [38] Ethan Brown, *Web Development with Node and Express 2nd edition*, “O’Reilly Media, Inc.,” 2020
- [39] Typescript official website, “What is typescript?” <https://www.typescriptlang.org> (accessed Mar. 22, 2023).
- [40] “W3 Schools – Typescript, [https://www.w3schools.com/typescript/typescript\\_intro.php](https://www.w3schools.com/typescript/typescript_intro.php) (accessed Mar. 22, 2023).
- [41] React Native official website, “React Native” <https://reactnative.dev/> (accessed Mar. 22, 2023).
- [42] Bonnie Eisenman *Learning React Native: Building Native Mobile Apps with JavaScript 2nd edition*, “O’Reilly Media, Inc.,” 2016
- [43] Gaurav Singhal, “Introduction to React Native Maps”, Oct. 18, 2021, <https://blog.logrocket.com/react-native-maps-introduction> (accessed Mar 22, 2023)
- [44] Dijkstra E.W.,”A note on two problems in connexion with graphs”, *Numerische mathematic*, 1(1), pp.269–271, 1959

- [45] Database.guide, “What is CRUD?” <https://database.guide/what-is-crud/>, (accessed Mar. 24, 2023)
- [46] Oracle, “Data Access Object” <https://www.oracle.com/java/technologies/data-access-object.html>, (accessed Mar. 24, 2023)
- [47] Baeldung, “Data Transfer Object” <https://www.baeldung.com/java-dto-pattern>, (accessed Mar. 24, 2023)