
UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

# NavUP System

## Architectural Requirements Specifications and Design

Group name: Delphi

*Corne Els - u14148872*
*Eunice Naa Korkoi Hammond - u13222563*
*Kyle Erwin - u15015302*
*Lyle Nel - u29562695*
*Nikki Constancon - u15011713*

# Contents

# 1 Architecture Overview

The system architecture can be broken up into 4 high level component. The mobile interface, the web interface, the core system and agents that do work on data from the core system. Each of these high level component can be broken up into smaller subsystem. These subsystems are enumerated and their place in the system is explained with the diagram below.

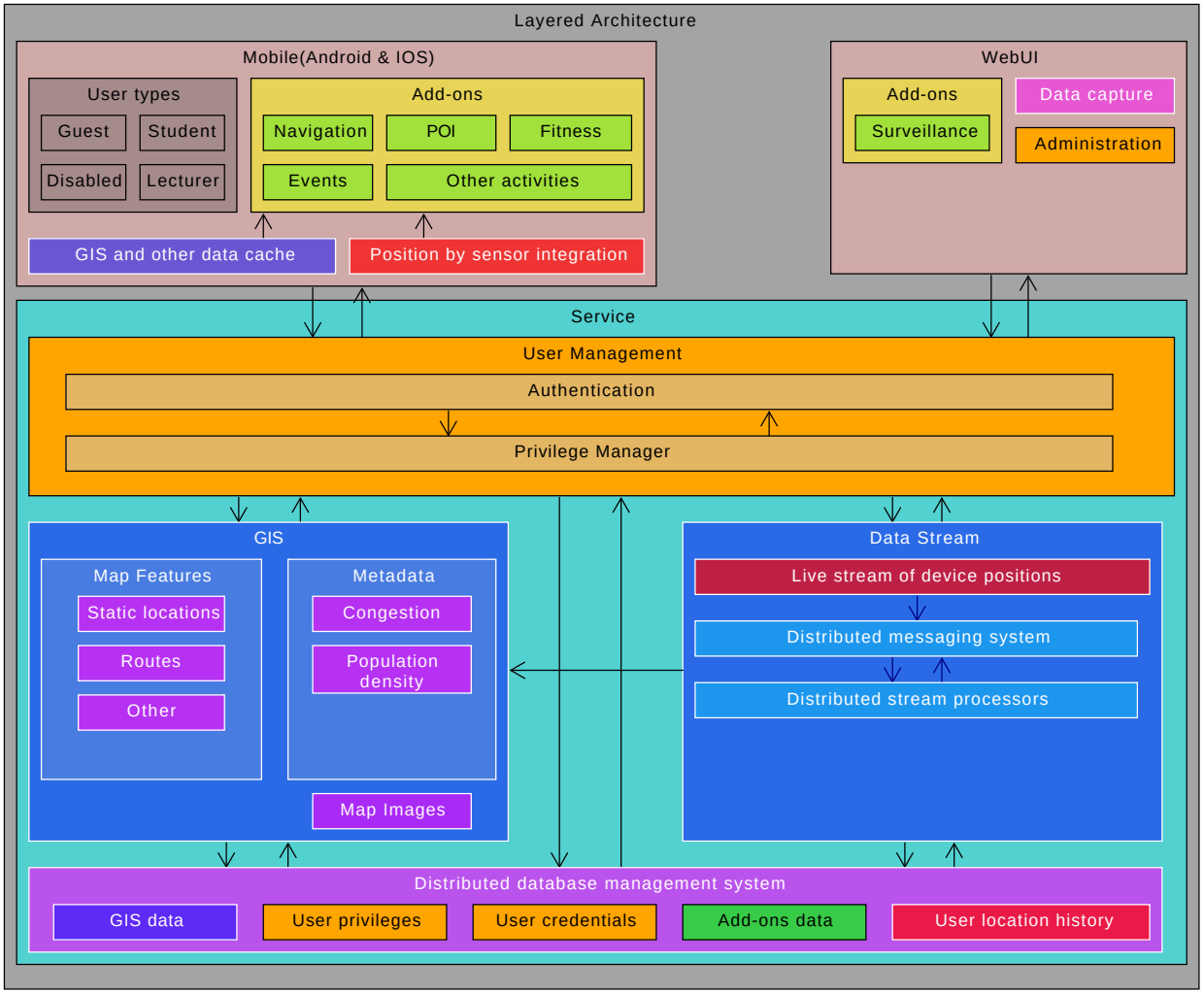


Figure 1: Architecture Overview

There is a lot going on in this diagram, so let us start with some of the conventions that we used to communicate how different subsystems relate to

each other. The Dark blue is everything relates to GIS data. The purple is all data that gets persisted in the database. The Yellow is to show that add-ons exist on more than one of the user interfaces, namely Mobile and WebUI. The green is everything relates to add-ons. The Red is everything that relates to the position information of mobile devices. The Orange is everything that relates to user management. You will notice that the WebUI has an Administration component that is Orange, which manages anything user related in the User Management subsystem.

Now that the colours have been described, let us give a cursory description of how some of these systems interact. First, let us turn our attention to the Orange sections. Notice how every external subsystem must go through the User Management subsystem, this is needed to prevent any unauthorised access to privileged information or features. For this to scale, user management can generate tokens for authorizing access to different subsystems within the server boundary. Examples of access control include, guests that have certain add-on functionalities deactivated since there is no persistent ID to identify them by. Furthermore a user logged in as a student may not modify map feature, however, the Data Capture user on the WebUI may modify map features assuming it has been authenticated and has been given the right privilege level. Also notice that the User Management submodule must consult the database to retrieve user credentials, which is why the arrows go both ways. The database is a distributed so that there is no bottleneck between any subsystem and the database.
Turning our attention to the Mobile add-ons, the reason why GIS data and position information flows into the add-ons section is that features such as Navigation requires information from both to function properly. The cache allows the app to attempt to do its work even though it is not connected.
The Data Stream and GIS subsystem is a bit harder to understand so let us go through that for the last part of our tour. The Data Stream takes live data stream of device positions and pushes it into a messaging system. Stream processors, which are registered in the messaging system, then generate metadata that can be used in the GIS subsystem. This metadata can be population densities and congestion data. Notice that the metadata is purple, therefore it gets commited to the database if necessary. Further analysis on the data stream may be done by the stream processors to generate other metadata such as user preference and interests for the add-ons. Both live and historical data may be needed for further analysis, which is why there are arrows from the Data Stream into the database.

# 2 External Interface Requirements

## 2.1 Mobile device

- Accelerometer
- Gyroscope
- GPS
- Wifi
- Touch screen
- Sound for disabled users

## 2.2 WebUI on PC

- Supported browser
- Network connection to the server
- Minimum low end machine capable of browsing
- Screen
- Keyboard
- Mouse

## 2.3 Things within the server boundary

- Fast network infrastructure linking at least as many machines as there are subsystems within the server boundary
- Multiple physical machines for scaling of the hadoop cluster when needed
- Outbound network connection to communicate with devices on campus

# 3 Performance Requirements

One of the major requirements for this system is how quickly the system should respond to requests, otherwise known as response time. Expanding further, it is simply maximising a feasible satisfactory time by which a system should or could respond, based on the kind of system, and more, in a user-computer/device setting.

When a user interacts with an application in general, a longer response from the system may cause the user to think that the system is down in one way or the other. In addition, response time degradations can cost so much depending on the time of the day the system is most used (e.g. during peak hours on UPs Main Campus, with respect to the NavUP System). Therefore, the system ought to adhere to protocols that would increase response time at any given, whether there are many individuals actively on the system vs vice versa. Users should not have to face the issue of observing system lags as such, which in essence would get them disinterested in using the application as a whole.

In addition to the above, the system should also engender throughput, which is the amount of data a system can handle per time or the workload at which the response time is to be met. With about 30000 people going in and out of UPs Main Campus every day, NavUPs system should be made to approximate how many users would be logged to their system per day and per time and be able to accommodate the traffic that will come through to the system. When the application launches, there will automatically be high traffic flooding into the system, as almost everyone will be a new user. Thus the amounts of data to be managed per time and per quantity should be in place to avoid any forms of system lagging, or high traffic levels to the point where new users cant register at all or other users cant even access their accounts, log in or even search for a location to navigate to, which would yield frustration for the user and thus discouraging them entirely, as earlier mentioned.

For smooth user experience, back-end technicalities should also not interfere with the user interface, and therefore items like the networks of the system should be kept at bay.

In essence, this system should be able to be as responsive as possible to the user with minimal to no lags, it needs to handle the capacity of the plus minus 30000 users that may continuously log on to the system. It should also be able to safely store information from this large user space (30000) to the system, and retrieve them in a timely manner (response time).

Relating to accounts, the system ought to allow concurrency in the creation of user accounts and the accessing of these profiles as well. Data usage should be kept to a minimum, allowing streams to be well managed.

Lastly, during search and navigation on the application, performance ought to be efficient (real-time search), with data updated and arriving as required.

# 4 Design Constraints

While designing NavUP the following needs to be minimized as much as possible:

**Memory**
NavUP will be designed in such a way as to require minimal local storage on user‘s phone while still keeping a cache that is sufficiently large for it to operate in offline mode. To accommodate lower end smartphones, the app should not use more than 400MB of ram at any point in time.

**Data Usage**
The Application will require little to non of the users data and will rather rely heavily on Wi-Fi around campus. However, the app will not congest the Wi-Fi on campus either.

**Battery Usage**
Battery usage for the application and web service will be minimized and ”eco-friendly” to all phones.

**Cost**
The system will require minimal costs, with most of the features self-made or freely available to all users.

**Complexity**
The user interface will be easy to user so that anybody can use it without difficulty. It will require little to no experience to enjoy all the features offered.

**Legal**
The NavUp system must comply with the POPI act in order to protect the private information of its users.

# 5 Deployment Diagram

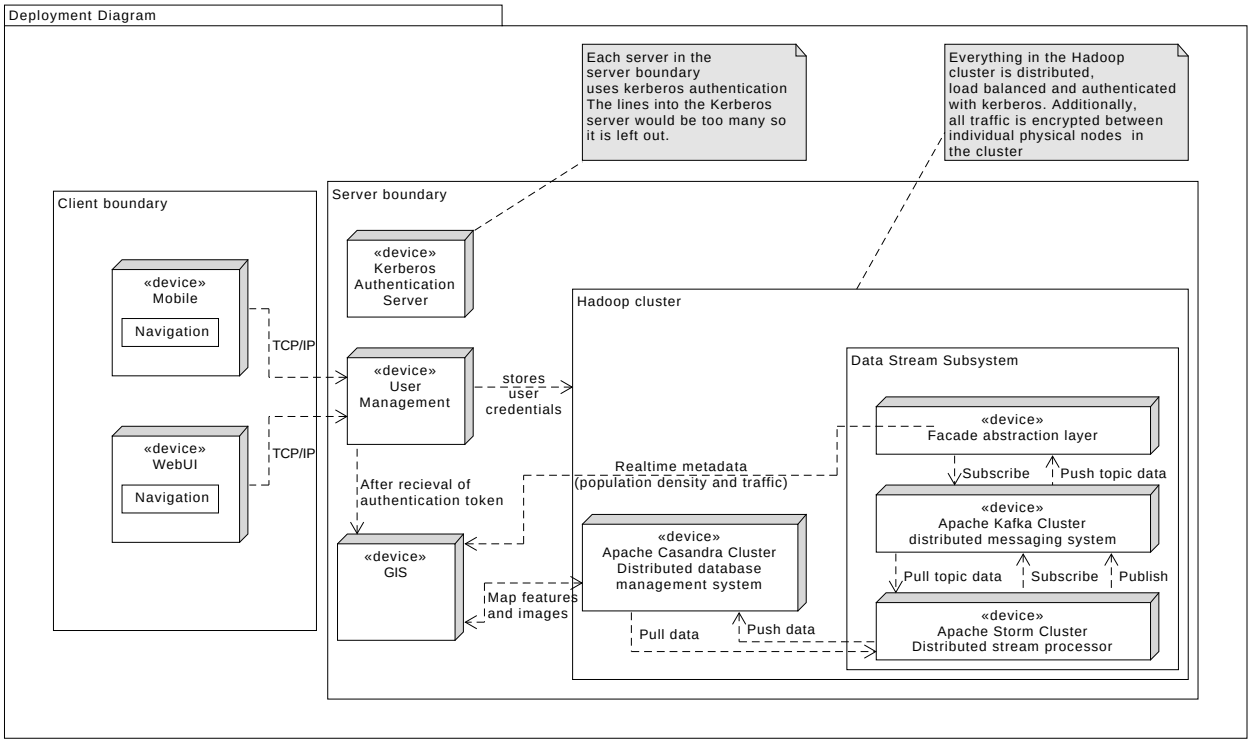


Figure 2: Deployment Diagram

# 6 Subsystem Architectural Requirements

## 6.1 Data Streaming

performance

### 6.1.1 Scope

From the provided requirements specification, the data streaming subsystem, in its most abstract form, is concerned with moving information between two or more entities. However, this subsystem is most fit to satisfy the additional requirement of aggregating and processing location data to generate metadata for user profiling, heatmaps and congestion of routes.

### 6.1.2 Technology choices

The data streaming subsystem can be broken up into 2 main systems, the publish-subscribe messaging system where all the data pours into, and the real-time stream processing system that analyses the data to generate useful meta-data. We will be using components from the Hadoop framework to build the data streaming subsystem. For the first system we will be using Apache Kafka which is a distributed messaging system. For the second system we will be using Apache Storm which is a distributed realtime stream computation system. Due to the distributed nature of these technology choices, they have some very attractive software attributes. See below for a full discussion.

### 6.1.3 Software Attributes

**Security**
Hadoop supports Kerberos network authentication so that when when the system scales to multiple clusters, individual machines cannot be impersonated to compromise the system. Additionally, encryption of network traffic between machines can also seamlessly be enabled to thwart a man in the middle attack as well as access control if that is needed.

**Portability**
The subsystem is highly portable, both in terms of operating systems supported as well as seamless failover due to its distributed nature.

**Performance**
It is a highly optimised system used by big companies such as Yahoo, Twitter and Pinterest to name a few. Because it is a distributed system, the performance needs can always be satisfied by deploying it on more machines. Moreover, the performance scales nearly linearly with the number of machines added to the cluster. Even a single Apache Storm node has a staggering benchmark perfomance of processing 1 million tuples per second.

**Integrity**
Since it is distributed, if a node fails catastrophically and there is a cluster of nodes, then the failover occurs seamlessly. Additionally, the system can recover from malformed messages and power failures. Due to the above reasons, data integrity as well as the operational integrity of subsystem is very good.

**Maintainability**
Due to the large number of big enterprise users, both Apache Kafa and Storm are well maintained with regular updates. This is unlikely to change any time soon, so the entire data streaming subsystem will be easy to maintain. In addition, this subsystem can be very forgiving towards poor maintenance or damage of the infrastructure hosting it because of its distributed and failover properties. Although of course one would want the infrastructure to be well maintained.

## 6.2 Apache Kafka

Apache Kafka is bonifide implementation of the publish subscribe messaging pattern. The distributed messaging system receives data either through the publisher API or the connector API, depending on the needs and implementation details. This data is published in appropriate topics and subscribers subscribe to the appropriate topic to fulfill its purpose. In this way Kafka fully satisfies the abstract requirements layed out in the provided requirements specification, namely, "the messages and their formats and contents must be independent of the actual carriage medium" and "The data module concerns itself primarily with the act of moving information between the two entities". Additionally, stream management and control can be done by a tool developed by Yahoo appropriately called Kafka manager and another built in tool called Kafka Control. In particular, UC10 to UC13 are all satisfied by the recommended technology choice.

As for the specific requirements pertaining to location data and other metadata. The messaging system will have a topic to which position information and the persistent ID of the mobile device will be published to. Furthermore, there will be a metadata topic for heatmaps, another one for congestion and another one for user profiling. It is important to note that since Kafka serves as a general purpose messaging system ,which other subsystem may make use of, there is no reason why future improvements on the system must be restricted to the above mentioned topics.

## 6.3 Apache Storm

Apache Storm is a distributed realtime stream processor. This system complements Kafka well for the extended requirements of generating metadata from the stream of location data. The system will link in through the stream processor API of Kafka, thus subscribing to the location topic and publishing in the different metadata topics after processing the location data. Even though

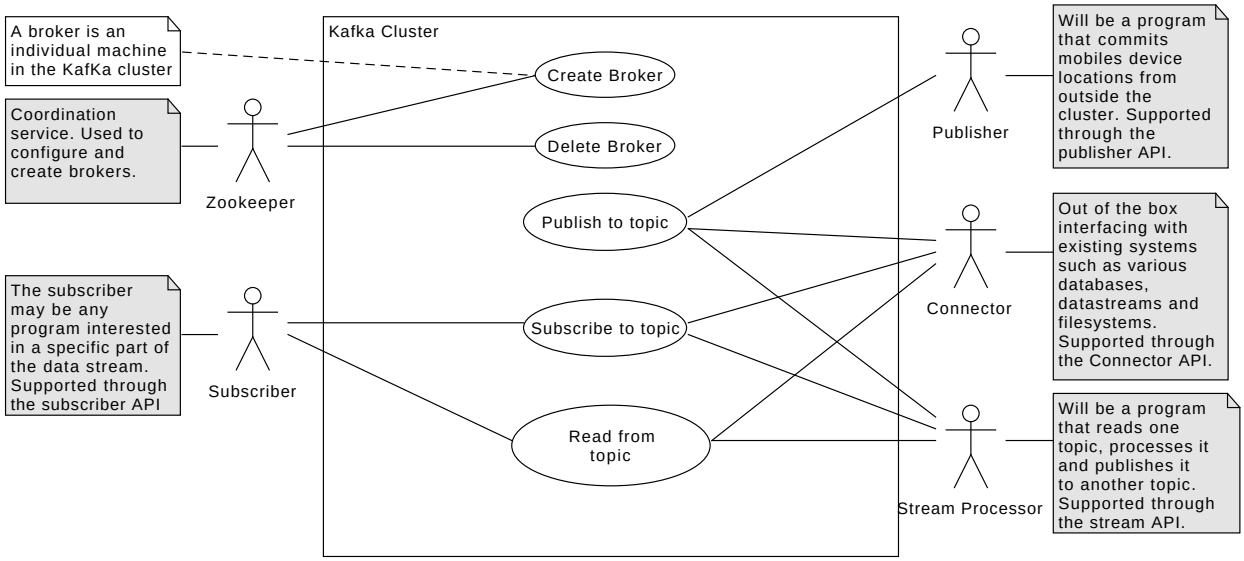


Figure 3: Kafka use case

processing of data streams is supported natively by Kafka, Apache Storm is a more robust and scalable solution supporting a graph processing model that distributes very well over many nodes if needed. Furthermore, this technology satisfies all the machine learning and analysis needs that might arise during the lifetime of the NavUp system no matter how process intensive it might get.

### 6.4 A simple API

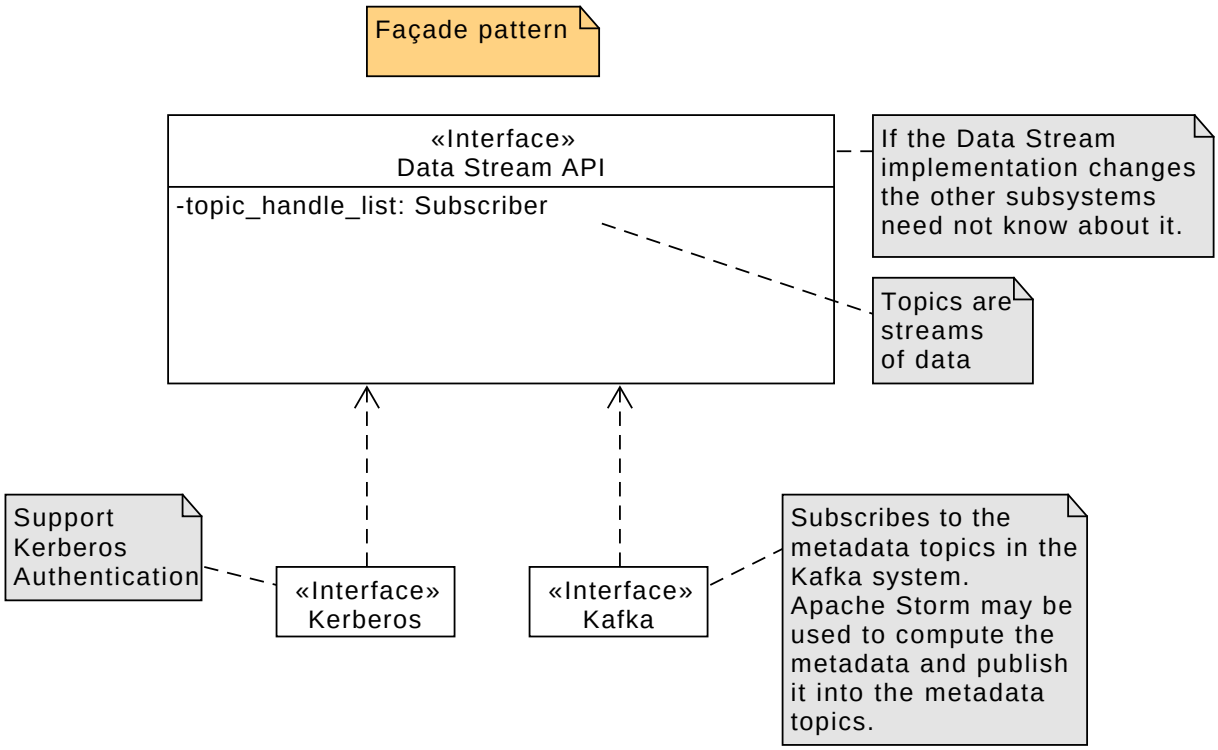


Figure 4: Facade pattern to abstract away from the internals of the data stream subsystem

## 6.5 GIS

### 6.5.1 Software Attributes

**Maintainability**
The GIS module should be easily understood and well documented in order to allow future developers the ability to add new functionality to the system or improve already implemented functionality.

**Scalability**
The GIS system should allow for n buildings and venues as new buildings or venues can be added in the future or the system could be deployed to other universities.

**Testability**
The GIS system should be easily testable to make sure locations are correctly placed and stored on the map. If the system does not test this well it would result in users being navigated to the wrong location.

**Security**
The GIS system should only allow authorised users to modify or create new locations on the map as a malicious user could place a location in a compromising area in order to for example steal the users phone.

### 6.5.2 Use Case Diagram

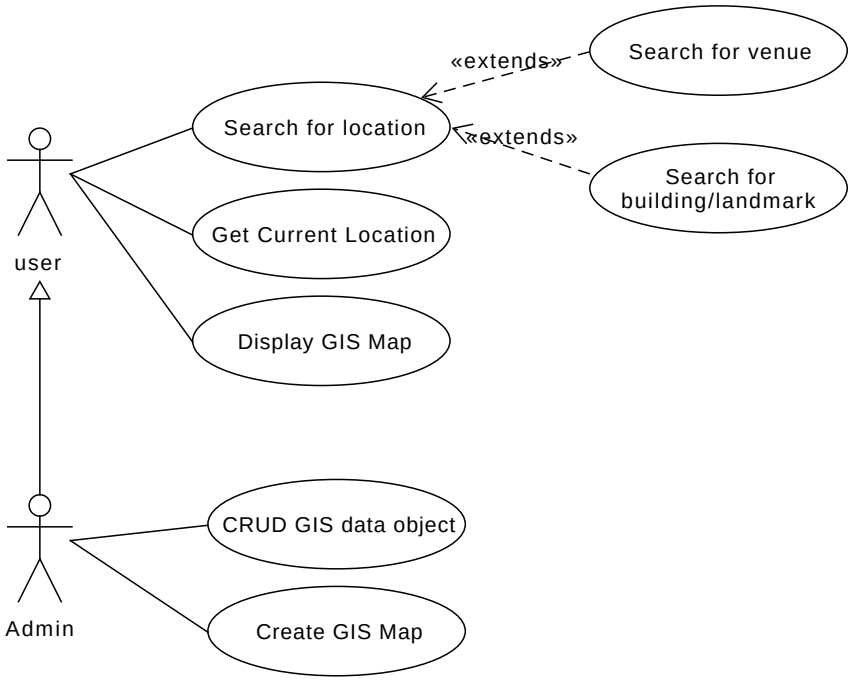


Figure 5: GIS Class Diagram

### 6.5.3 Class Diagram

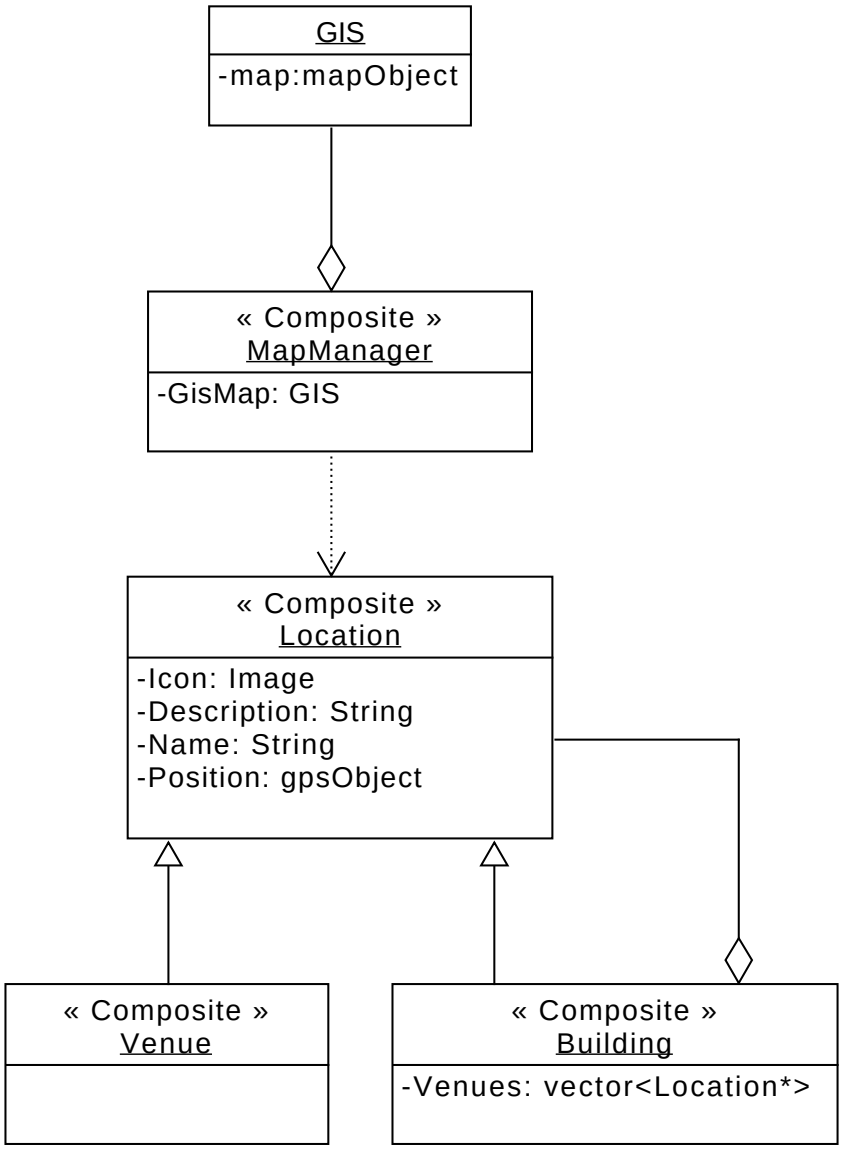


Figure 6: GIS Class Diagram

## 6.6 Users

### 6.6.1 Scope

As per the specification user management will be involved with the creation of users, specifically the creation of different types of users. The users will follow a hierarchy approach with each higher level having more privileges than the previous. The heir achy, from top to bottom, is as follows: Admin, User, Guest. Each type of user will fill a specific role and purpose. The admin being able to manage and create other users as well as the ability to update location information and set points of interest. The standard user will a student or employee using the app. Lastly the guest user, which will not require a profile, will be used by outsiders or those that are trying the app.

### 6.6.2 Software Attributes

**Security**
The user data needs to be secure in order to protect the users. By implementing a secure database in which user data is stored as well as a firewall to protect the server itself we can ensure that the system will remain secure. User passwords will be secured using the bcrypt hashing algorithm and the tuning parameter will be adjusted to balance both security and performance requirements. Due to some of the information stored in the user files being somewhat sensitive data (routes walked and common locations visited) it is crucial for us to keep these records secure and safe.

**Portability**
The system is inherently designed to be very portable. The user data can be entered regardless of the hardware device being used. This means that users can register on iPhone or Android. Admin users can create users if needed directly from the server. The software independence is also clear as again any device, regardless of thier operating system, can be used to access and manage the user data.

**Cohesion**
The system is already a very coherent system. The user data will have to be used across a lot of the different subsystems such as Navigation and Rewards and as such it needs to be designed to be portable. To do this the user subsystem will include a variety of functions to return the required data need by another subsystem.

### 6.6.3 Technology Choices

For the users subsystem we will need a server to host the relevant tables. This

server should have a database management software to allow for easy management. The server should be guarded by some form of firewall software to prevent any unwanted access. The recommended database management system will be Apache Casandra since it is distributed and can scale linearly as demand on the user subsystem increases.

### 6.6.4 Use Case Diagram

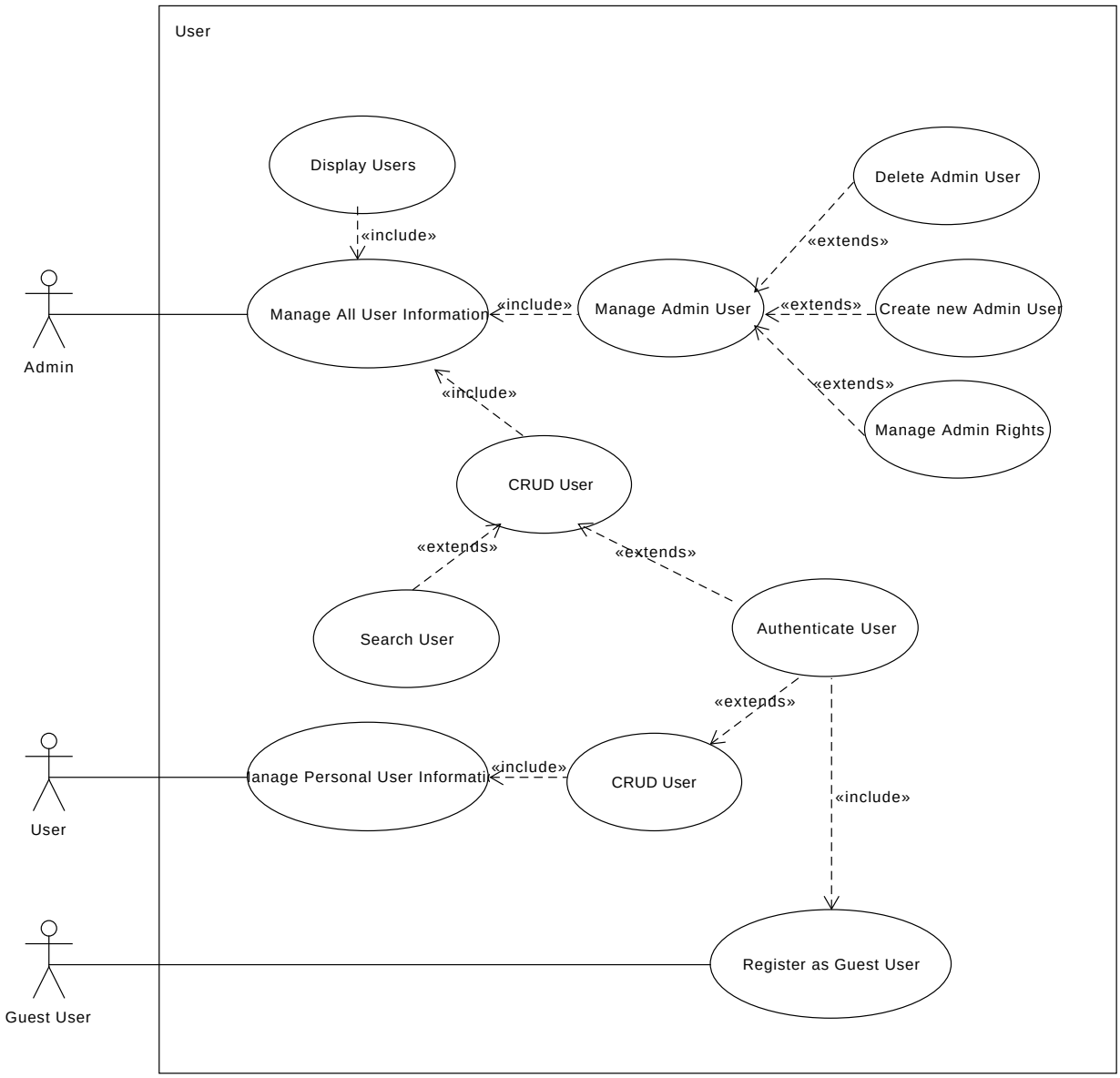


Figure 7: Users Use Case

### 6.6.5 Class Diagram

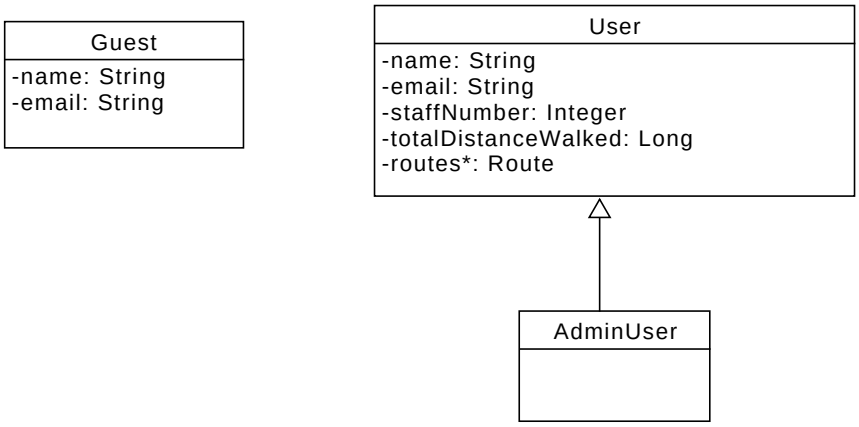


Figure 8: Users Class Diagram

### 6.6.6 Design Pattern Diagram

**Reasoning**
We chose the factory design pattern as it will allow us to stream line the process of creating users.

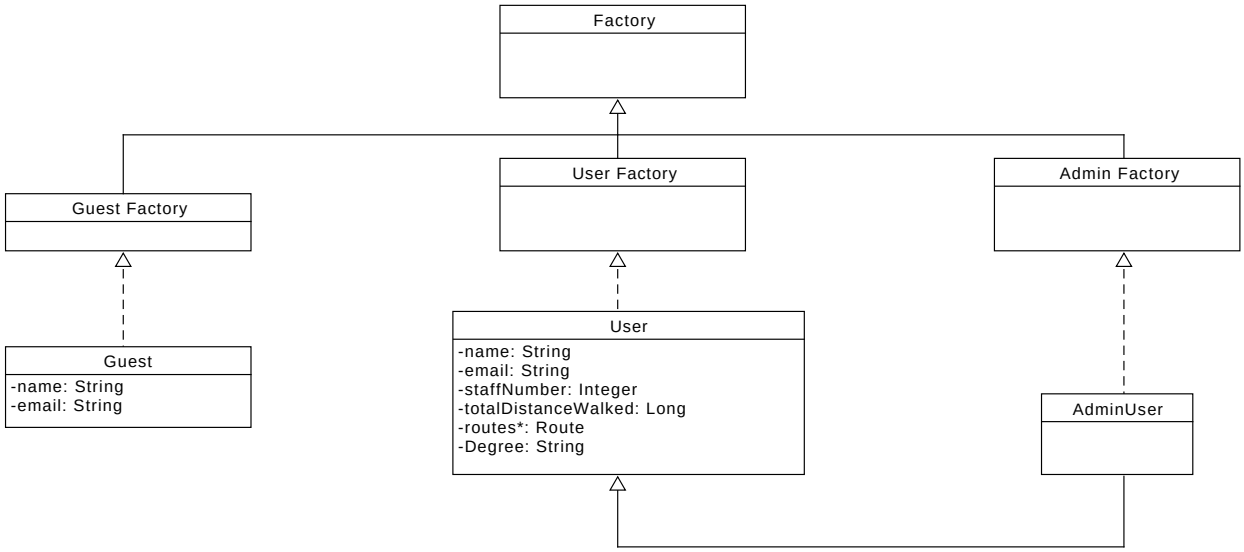


Figure 9: Users Class Diagram

## 6.7 Navigation

### 6.7.1 Scope

The navigation module of this system consists of providing services such as calculating routes for a user and directing one from a specific point to another, as a user would have specified to the systems input. The system is to be able to get, delete, save, modify and record routes, and effectively and efficiently steer a user to their destination, thus establishing the desired results.In addition to this, the system will endeavour to maintain reliability, robustness, high cohesion, low coupling, efficiency, low complexity, feasible maintainability and good portability.

### 6.7.2 Software Attributes

**Efficiency**
The navigation subsystem will give realtime directions and pathfinding with minimal expenditure of time and resources. Some of these resources include battery power and mobile data. The decision of what route to take should be calculated with little to no latency and should adhere to all the conditions set by the user. The system should also require minimal memory, be small in size and produce optimal performance whilst taking into consideration time and resource constraints.

**Portability**
The navigation system should be integrated in such a way as to ensure ease of portability between the web interface and the different devices that could host the mobile application.

**Maintainability**
It will allow for the evolution of the system by easily saving routes and if the route is popular enough, incorporating it into the default routes used. The system will be modular enough that separate sections can be modified and improved with minimal effect on the system as a whole. Ideally faulty or worn-out components can be repaired or replaced without having to replace still working parts, this will also allow for isolation of defects in the code and their subsequent correction.

**Cohesion**
This system will have high cohesion since all functions will be directed towards the core function of directing the user from one location to another. Each function will be strongly related and will work together as a functional unit.

**Reliability**
This system should always be able to find the shortest route as well as the

simplest route to get to the desired location. It will always find locations that are accessible and vice versa, always giving feedback. In addition, information that needs to be synced should be done frequently, so as to adequately and concisely direct to the chosen location based on accurate information.

**Robustness**
This navigation system will not be easily breakable as it will be error-proof to a feasible extent, to avoid unnecessary faults and not wholly be affected by single application failures. The system should be able to recover quickly from such failures, or at least be able to hold up, or return to a valid stage or state.

**Complexity**
This system should be as simple as possible, enabling users to easily find routes, select routes, save routes, and subsequently follow their chosen routes without being misled or lost in any way. After choosing desired routes, this system should allow users to easily modify routes. Thus it should endeavour to be flexible with users to a feasible point.

**Coupling**
Although this subsystem will be slightly dependent on some core systems such as GIS, this system should aim to incorporate loose coupling, being able to be as stand-alone as is necessary. That is, this system should be able to, for example, provide routes to the user with information pre-streamed to the system in case of any mishaps (i.e. offline functionality incorporated), or be able to give directions without a user having an account, with details to be relayed to and fro from the DBMS (i.e. Guest users).

### 6.7.3 Technology choices

The Anyplace API will be used as the naviation service for this sub-system. Anyplace is a good choice in API for a number of reasons: Anyplace is a free API hosted on GitHub and created by university students. It provides indoor and outdoor navigation with accuracy of 1.96 meters. This is perfect for NavUP since it uses Wi-Fi to navigate and provides floormaps, which can be used in the GIS module. It also has additional features such as Wi-Fi heatmaps which will supplement other modules. This API allows for fast accurate addition of new buildings and it supports point of interest(PIO) to PIO navigation (plus easy addition of POI‘s). On top of all this it also has support for crowdsourcing. This API contains many of the main features that NavUp intends to have, plus a number of extra features that will enhance NavUp‘s navigation abilities, making it a good technology choice for the navigation module.

### 6.7.4 Use Case Diagram

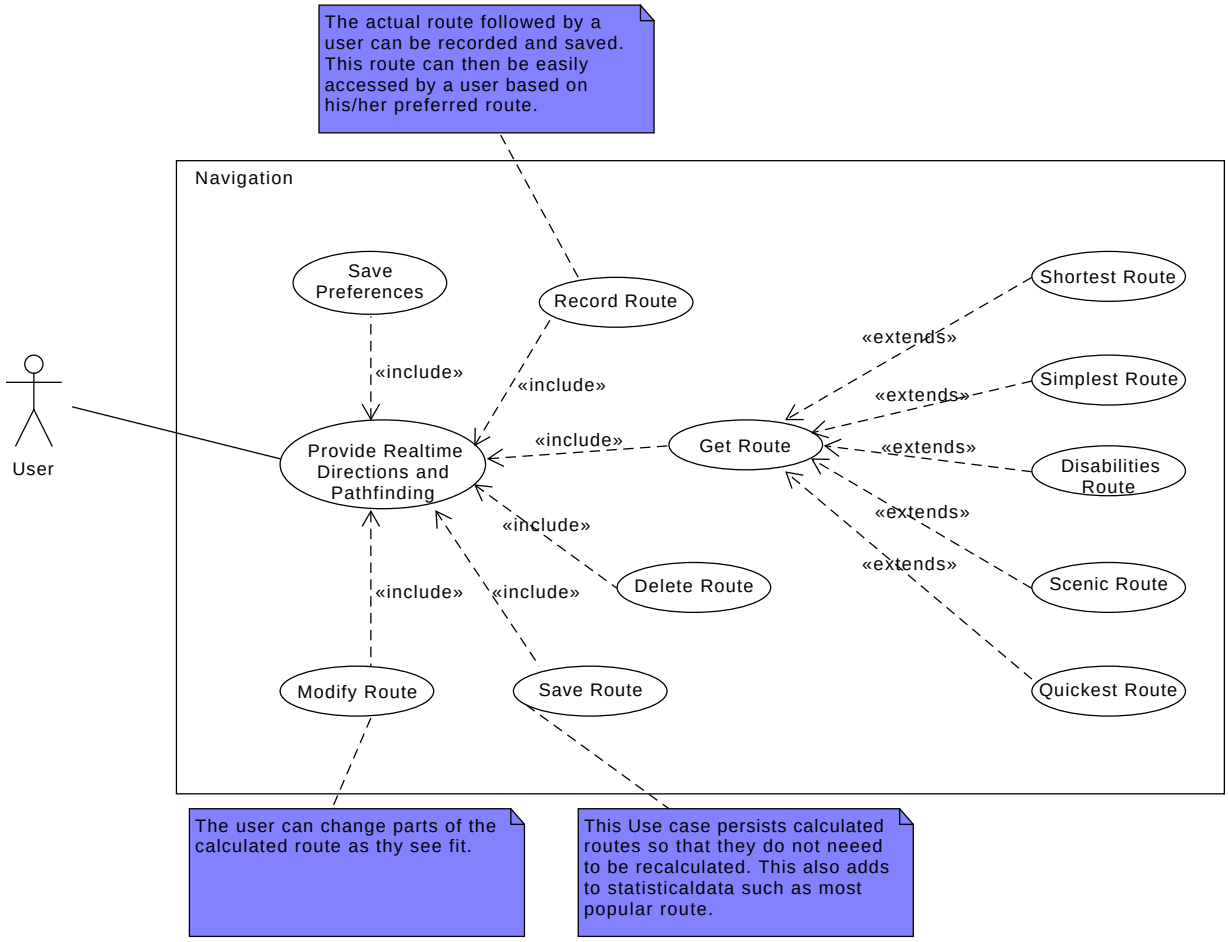


Figure 10: Navigation Use Case

### 6.7.5 Class Diagram

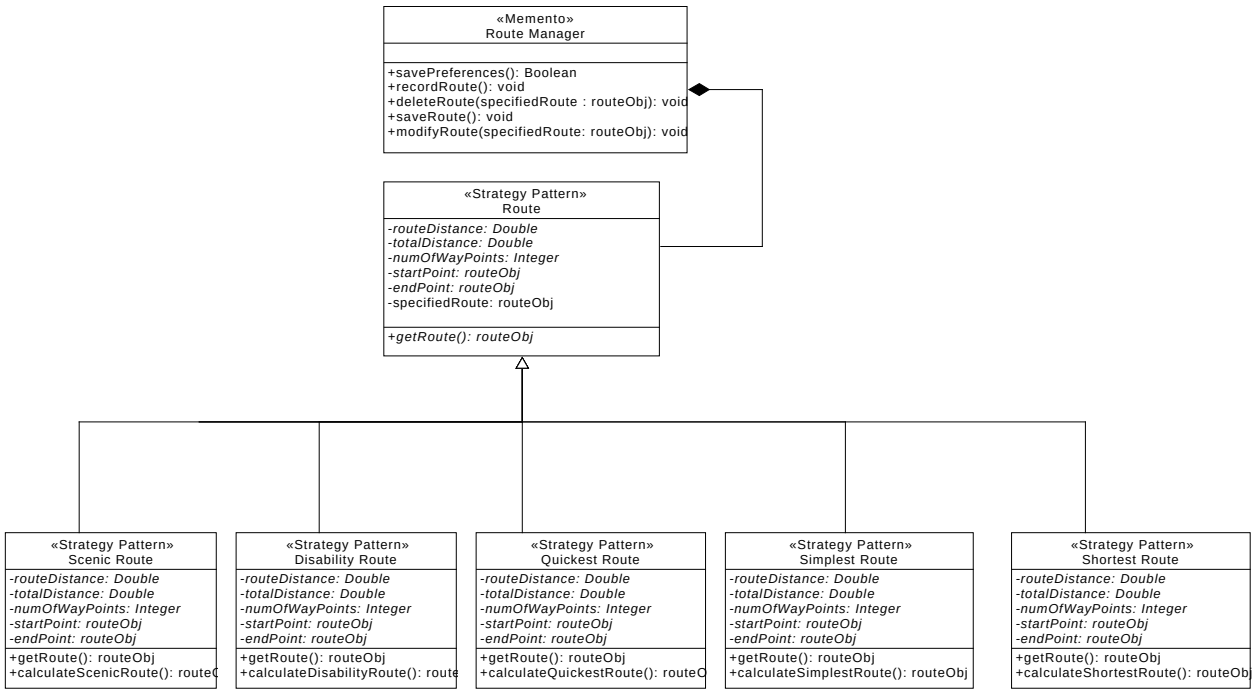


Figure 11: Navigation Class Diagram

### 6.7.6 Design Patterns

Both memento and strategy design patter will be used for this subsystem. Memento will persist the saved/recorded routes for later use and strategy will decide how the route will be calculated.