UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Architectural Requirements Specification for the NavUP System

Jita, Hlengekile
u14077893

Moodley, Joshua
u14152152

Kazadi, Dieumerci
u16353383

Nell, Stephan
u15124861

Rayner, Peter
u14001757

Van Hattum, Jason
u15027458

Schuld, Bernhard
u10297902

March 11, 2017

Contents

1 Introduction **1**

2 Overall System Design **1**
2.1 Architecture 1
2.2 Deployment 2

3 Users **3**

4 Navigation **5**
4.1 Description 5
4.2 Requirements 5
4.3 Constraints 5
4.4 Design 5
4.5 Implementation Details 6

5 Data **9**
5.1 Description 9
5.2 Requirements 9
5.3 Constraints 9
5.4 Implementation Details 9

6 Surveillance **14**
6.1 Description 14
6.2 Requirements 14
6.3 Constraints 14
6.4 Design 14
6.5 Implementation Details 15

1 Introduction

This document will serve to outline the architectural requirements, constraints and design for the NavUP system. The general pattern that the system will be built around will also be discussed.

In addition, the subsystems will be descried along with their planned implementation, accompanied by relevant diagrams. With regards to the subsystems, focus will be put onto designing the subsystems to be modular and loosely coupled in such a way that the system can be deployed with a minimal set of subsystems, and each subsystem can be changed without affecting the others.

The modules that the document will outline are:

- Data.
- Users.
- Navigation.
- Surveillance.

2 Overall System Design

2.1 Architecture

The system will be designed using the **n-tier** architecture, where the system is divided into multiple layers, and information is passed from one layer to another. The direction of information transfer is described in further detail in our deployment diagram.

2.2 Deployment

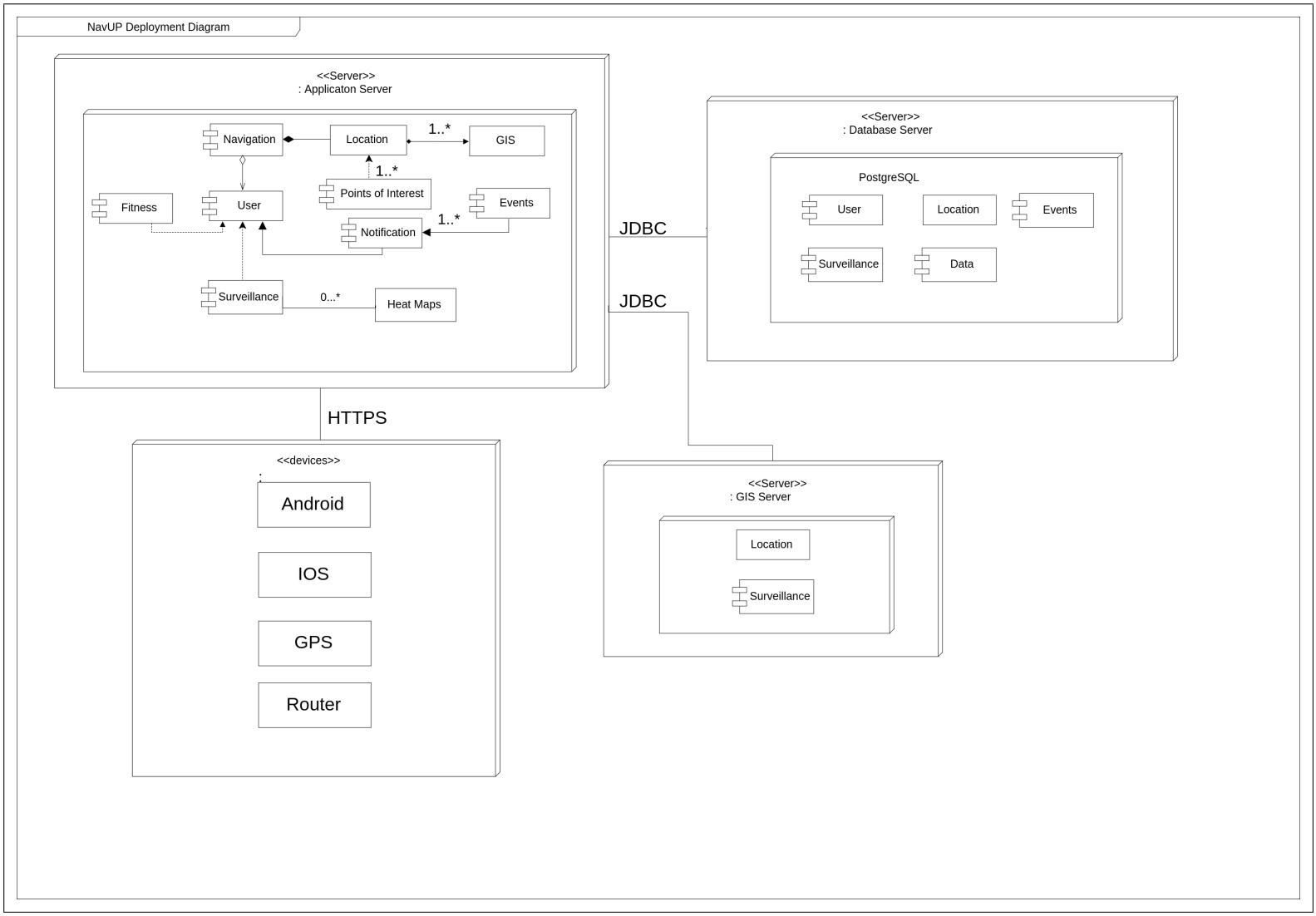

Figure 1: Deployment Diagram

3 Users

The Users module is responsible for the management of NavUP's users. There will be 3 categories of users of the system, Guest users, which will be able to access public services without having to be logged in. Registered users, which will in addition to the public services will be able to store profile information necessary for other activities provided by the system and Admin users which will have the additional ability to manage the users of the system.

In the figure below, is a use case diagram that shows how each user will be able to make use of the Users module and what functionality will be included in order to achieve the use cases.

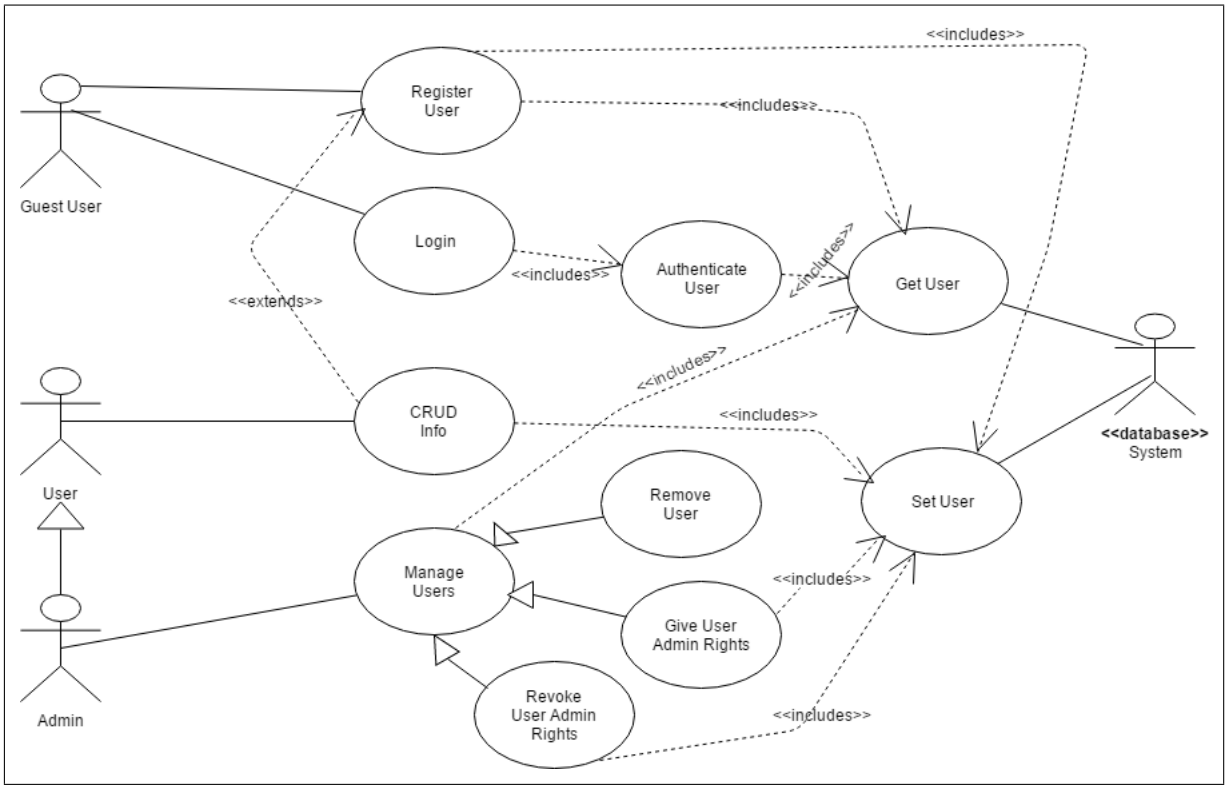

Figure 2: Use Case Diagram - Users Module

In the figure below is a class diagram that shows the classes of the Users subsystem.

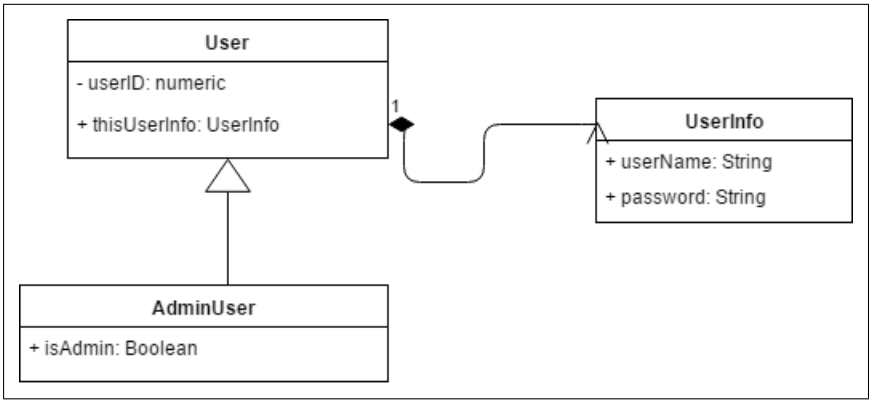

Figure 3: Class Diagram - Users Module

4 Navigation

4.1 Description

The purpose of the Navigation subsystem is to provide routes to users based on their destination and current location, as well as navigating the users along the route to their destination.
This subsystem will reside almost entirely in the application layer, as it is comprised mostly of decisions related to navigation.

4.2 Requirements

The Navigation system should be able to:

- Provide a navigable route between two locations to a user on request, taking into account a possible set of preferences.
- Navigate a user from waypoint to waypoint until they arrive at their destination.
- Detect when a user deviates from their given route and recalculate a new route accordingly.
- Persist routes to the database for later use.

4.3 Constraints

The system must operate under the following constraints:

- Any location data is received from the GIS subsystem, and is assumed to be correct.
- The destination is received from the user, and may be impossible to navigate to.
- The system has to work with the map as provided by GIS, which may have missing locations or locations that are incorrectly placed.

4.4 Design

The Navigation system is implemented as 10 classes organised in two well-known design patterns - the Strategy and Memento patterns.

Strategy The portion of the system which calculates routes is implemented in such a way that the algorithm can be changed easily by simple adding a class. This is done with the Strategy design pattern - each extra routing algorithm is simply added by adding a new class which inherits from `RouteAlgorithm`, and overrides the `calculate()` function. Decompiling each algorithm to a separate class will help with maintaining the code in the future and will reduce the overall complexity of the navigation module.

Memento One of the requirements of the system is to save the route. The system will accomplish this by saving the waypoints as they are reached. The Memento design pattern is used here - each waypoint is saved as a `WaypointMemento` object to the database, which ensures that the state is unchanged when the object is recovered at a later stage.

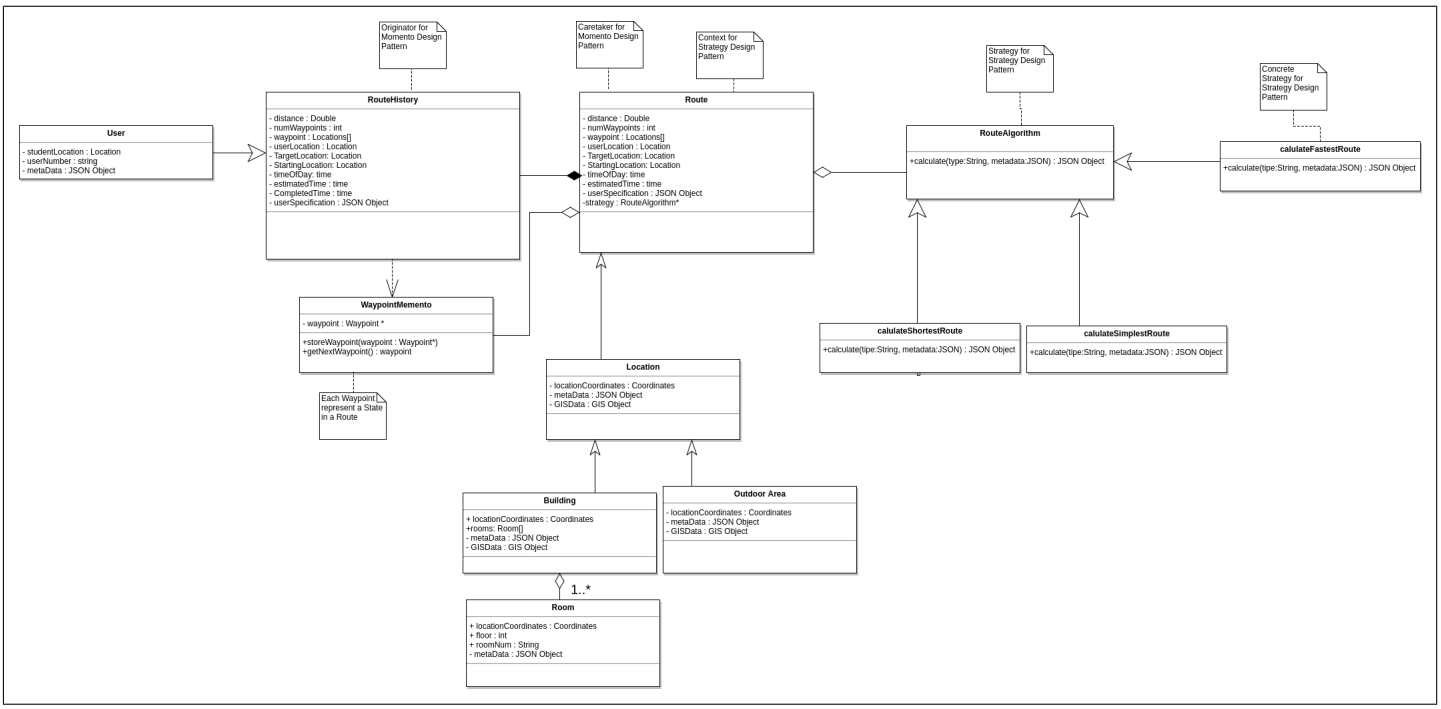

Figure 4: Class Diagram - Navigation Module

4.5 Implementation Details

The system will, for the most part, be composed of a graph. In the graph each node would represent a location, with the links between the nodes representing the paths from location to location. Navigating from one location-node to another would then simply be a matter of using some shortest-path algorithm to find the path. The algorithm would depend on the user's preferences such as finding the shortest path as opposed to the quickest path, and so on. These algorithms would be interchangeable due to the use of the Strategy pattern.
The nodes would be saved as *Location* objects, and the route calculated by a `RouteAlgorithm` object.

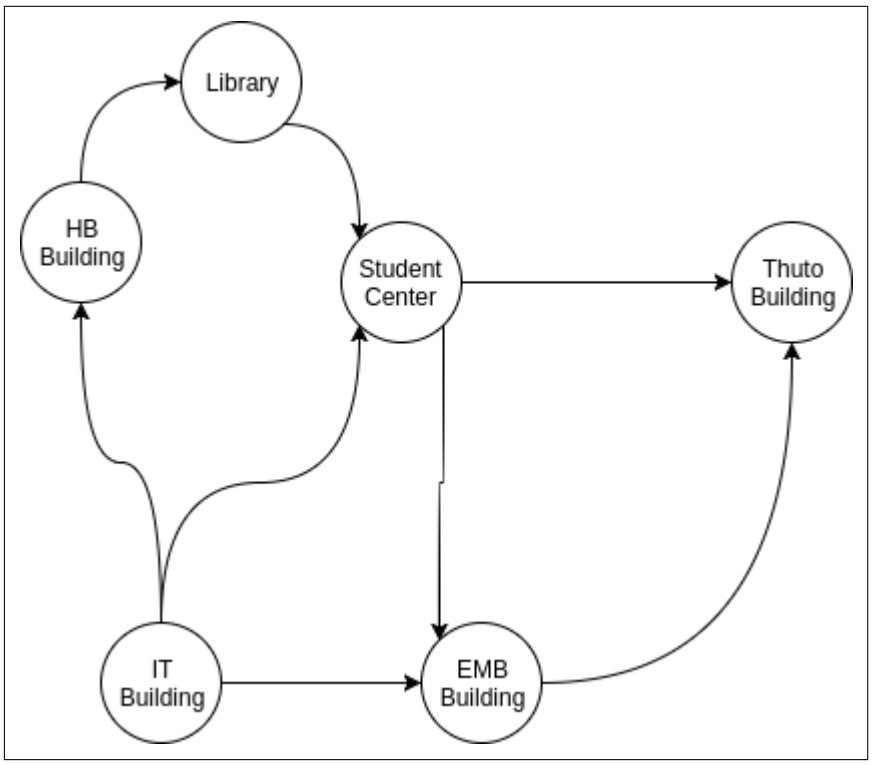

Figure 5: Example Navigation Graph - Navigation Module

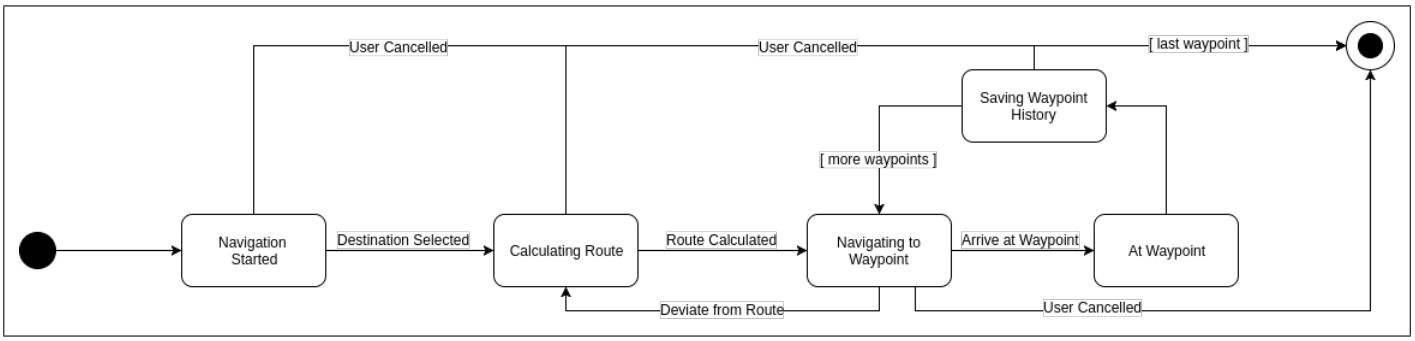

Figure 6: State Diagram - Navigation Module

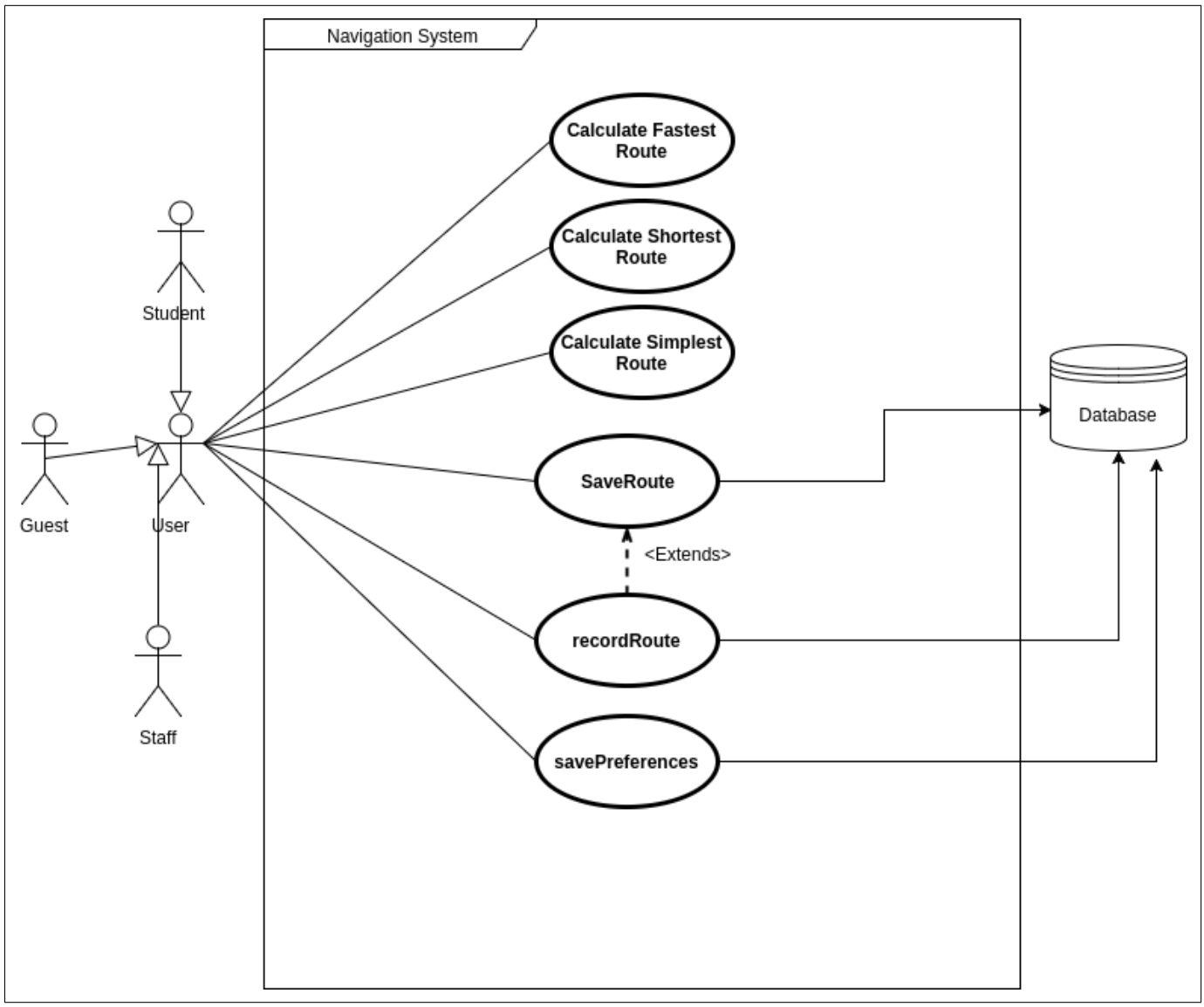

Figure 7: Use Case Diagram - Navigation Module

5 Data

5.1 Description

The data subsystem is responsible for handling and responding to requests for information, as well as persisting information to a database.

5.2 Requirements

- The database should be able to manage a very large amount of concurrent requests.
- Data requests should be performed reliably and consistently.
- Data should be duplicated (Such as in RAID 5) or backed up for redundancy and performance.

5.3 Constraints

- The system has to store data of similar types for consistency.
- The system has to make use of the existing UP server infrastructure.

5.4 Implementation Details

The Data subsystem provides the role of Broker in the entire system, in the sense that all requests for information are made through the data subsystem. This allows for low coupling. The actual data system will be implemented as an Iterator, as seen in the class diagram. This is to allow data responses to be accessed without exposing their underlying representation. This promotes component-based development, boosts productivity in data access, and reduces configuration management in the database.
The database used by the Data subsystem will make use of a SQL database in order to be consistent with the existing UP database infrastructure.

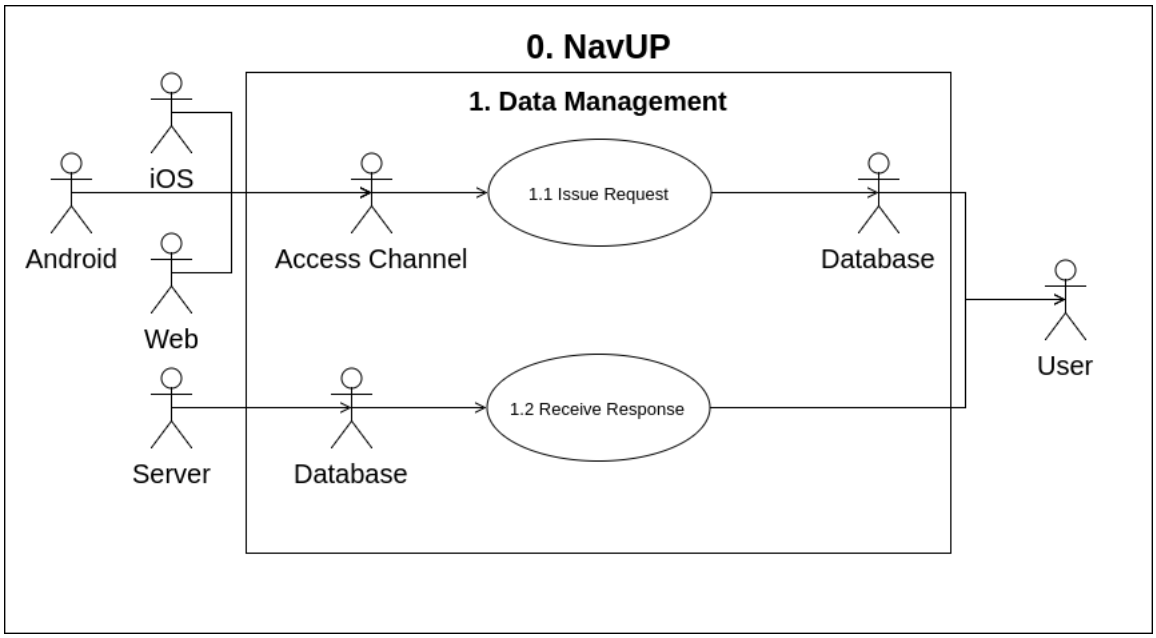

Figure 8: Use Case Diagram - Data Module

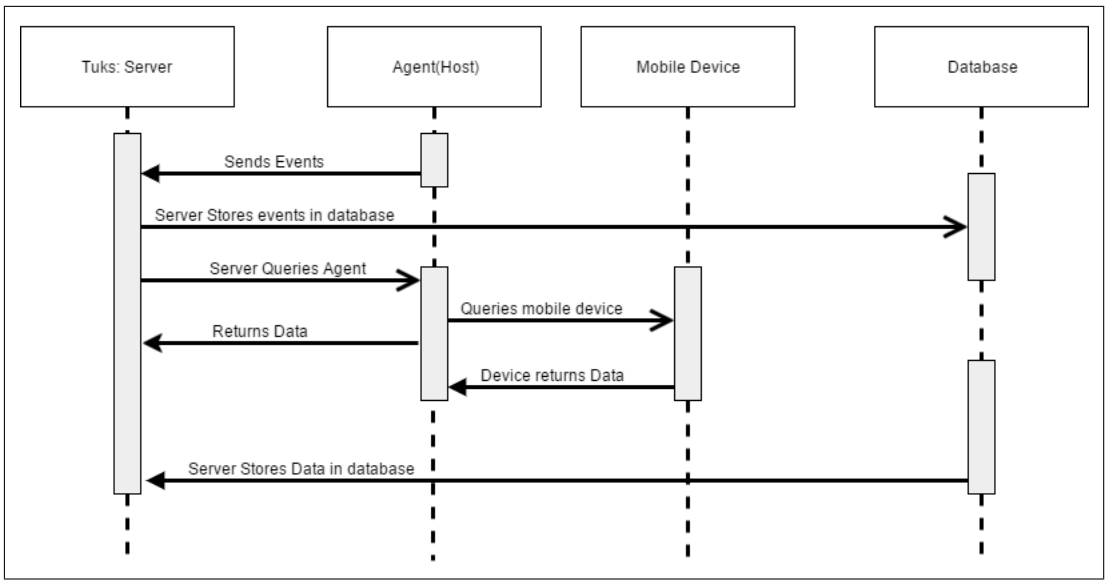

Figure 9: Sequence Diagram - Data Module

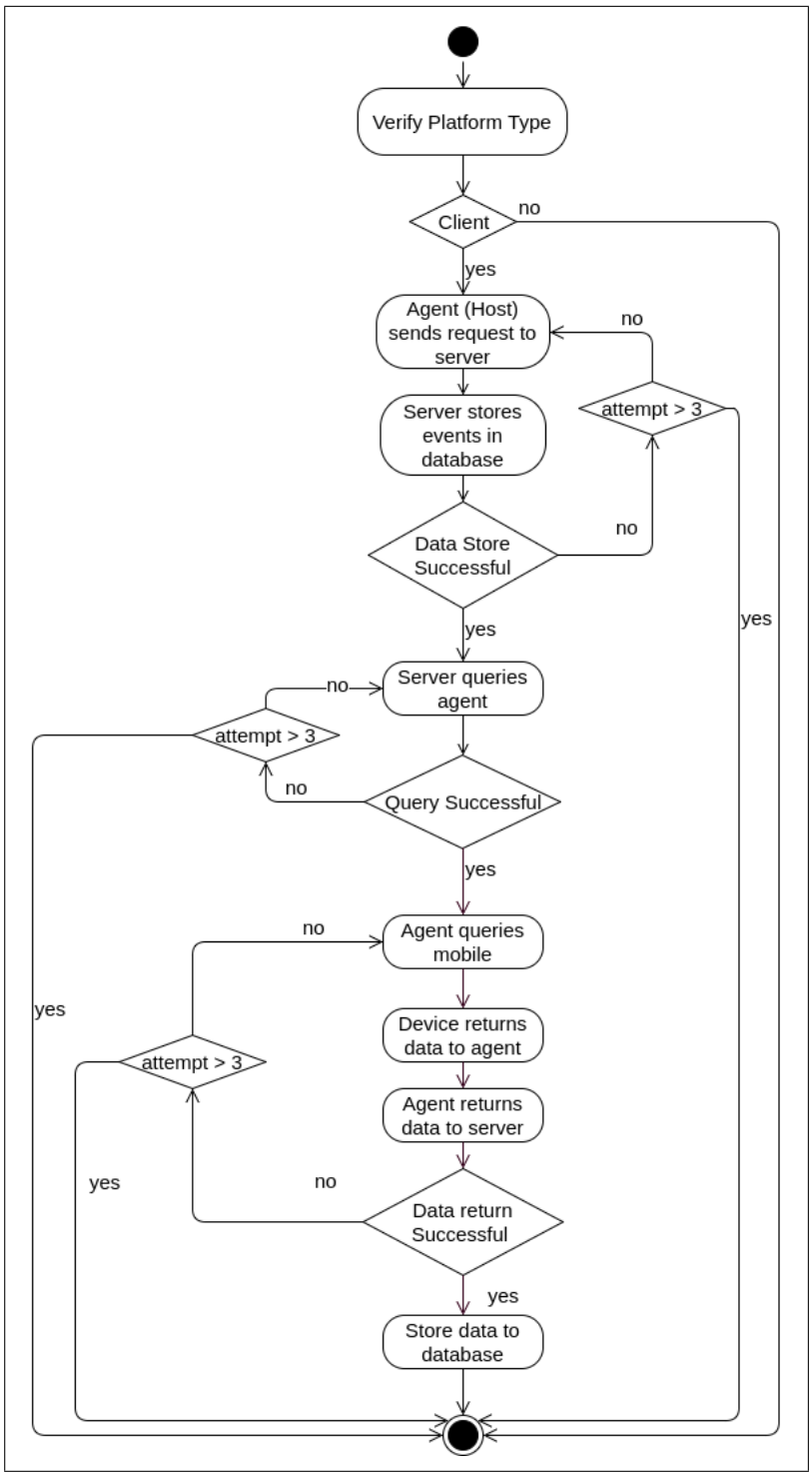

Figure 10: Activity Diagram - Data Module

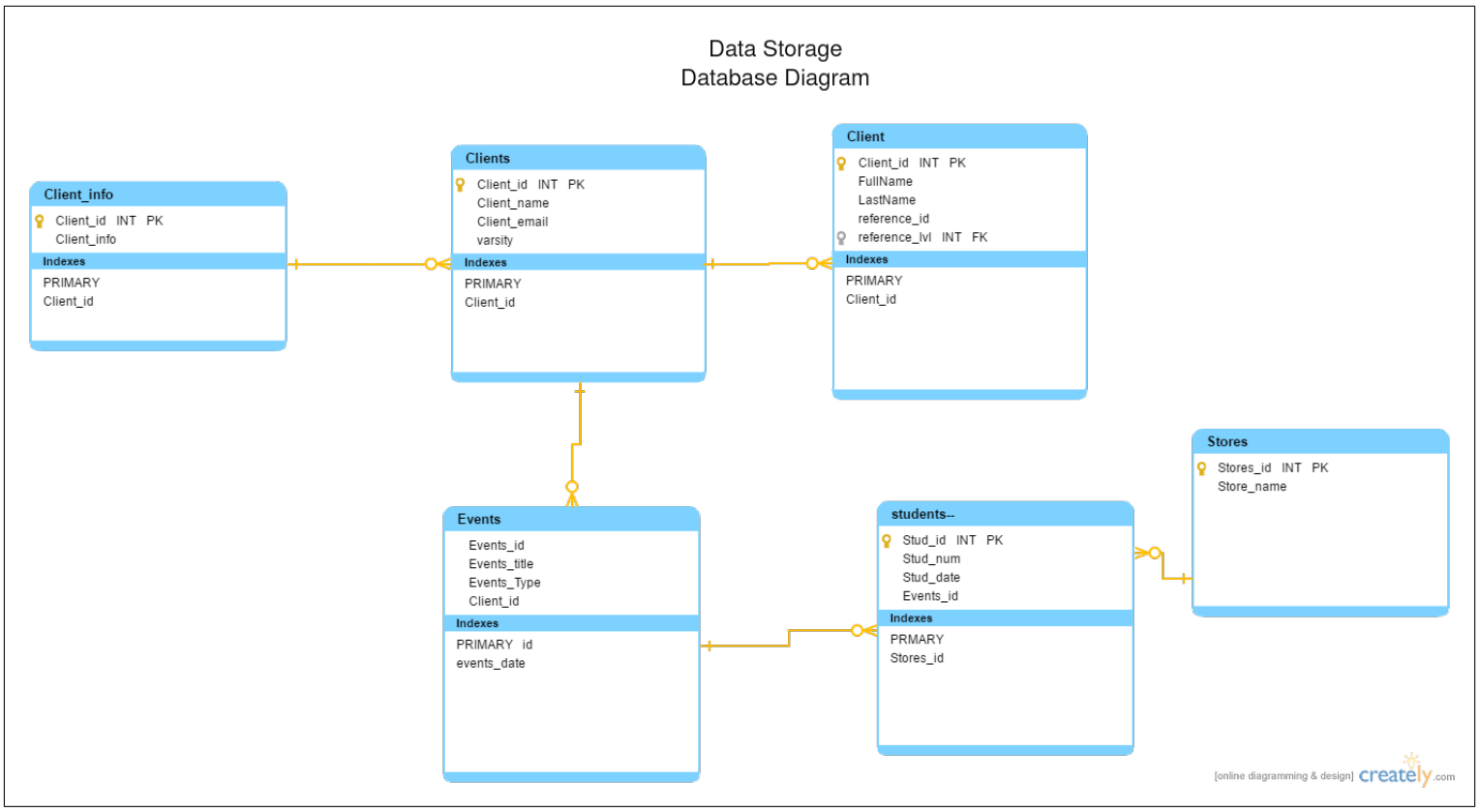

Figure 11: Database Diagram - Data Module

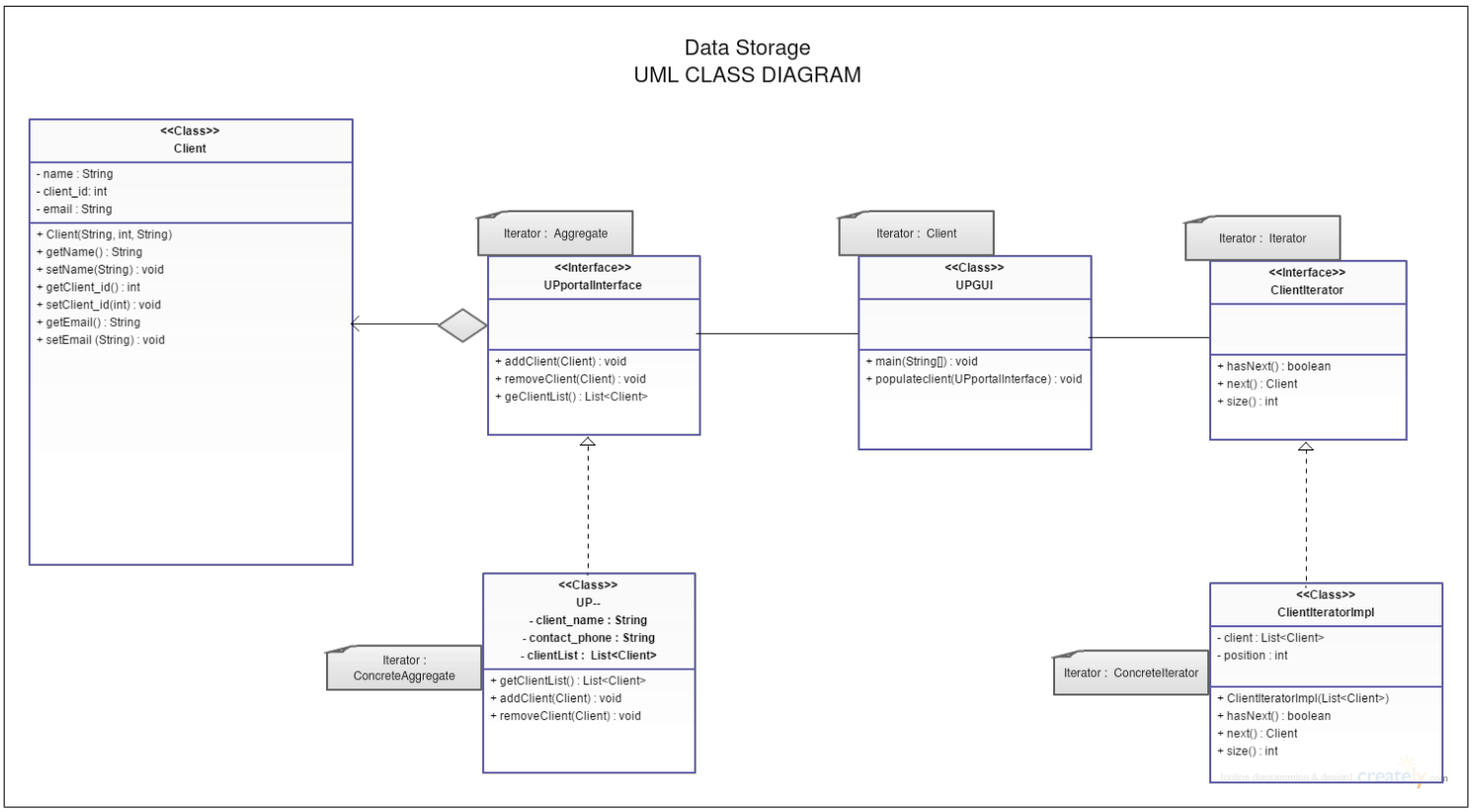

Figure 12: Class Diagram - Data Module

6 Surveillance

6.1 Description

The purpose of the surveillance subsystem is to be able to identify locations where there is pedestrian congestion on the walkways of the university. By being able to find how many people are connected to the UP routers, which cover a large portion of campus, we are able to more accurately identify these hot-spots and provide this data to the navigation team, in order for people to try avoid these areas. Thus spreading the amount of pedestrians to hopefully avoid highly congested areas.

6.2 Requirements

The Surveillance system should be able to:

- Count the number of users connected to a particular router.
- Provide a live time data feed of the number of users connected.
- Identify the location of each router.
- Add or update routers in the system.
- Provide loose coupling.
- Update the database with information of route congestion in a particular location.

6.3 Constraints

The Surveillance system should operate under the following constraints:

- The list of routers is supplied and assumed to be correct.
- Routers are able to connect and receive data from users.
- Usernames and passwords for all routers are provided by admins.

6.4 Design

The surveillance system is implemented using seven classes. We used the mediator design pattern for its implementation.

Mediator - The mediator design pattern allows for having two systems (Location and Database) that could then be updated and the same time.

This behaviour of this system is important as it allows other modules to also receive a live time feed of every router, but from the database, and not from the surveillance module itself. This means that should the surveillance module be taken away no other module would be directly affected and the system would be able to still work. This pattern has therefore allowed us to have low coupling with high cohesion.

6.5 Implementation Details

To implement the data structure we will use a data structure (linked list, array or vector) to store all the information of the routers and be able to dynamically add to that data structure, while being able to update both data structures on both the database and surveillance module. Way-points which inherit from the locations are used as the Navigation module has to exist for the program to perform its main purpose, and to avoid duplicating code I inherit from location rather than create a brand new class.

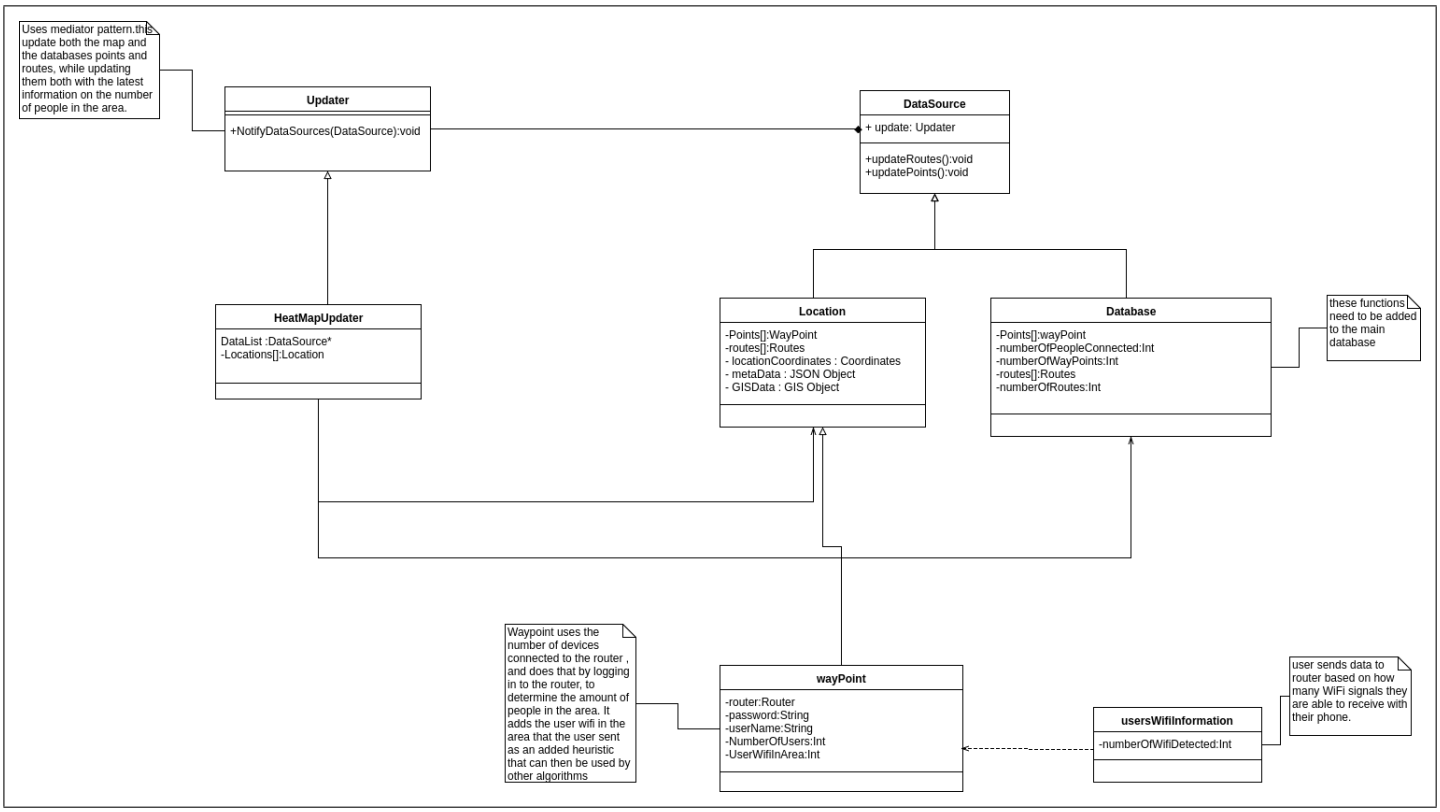

Figure 13: Class Diagram - Surveillance Module

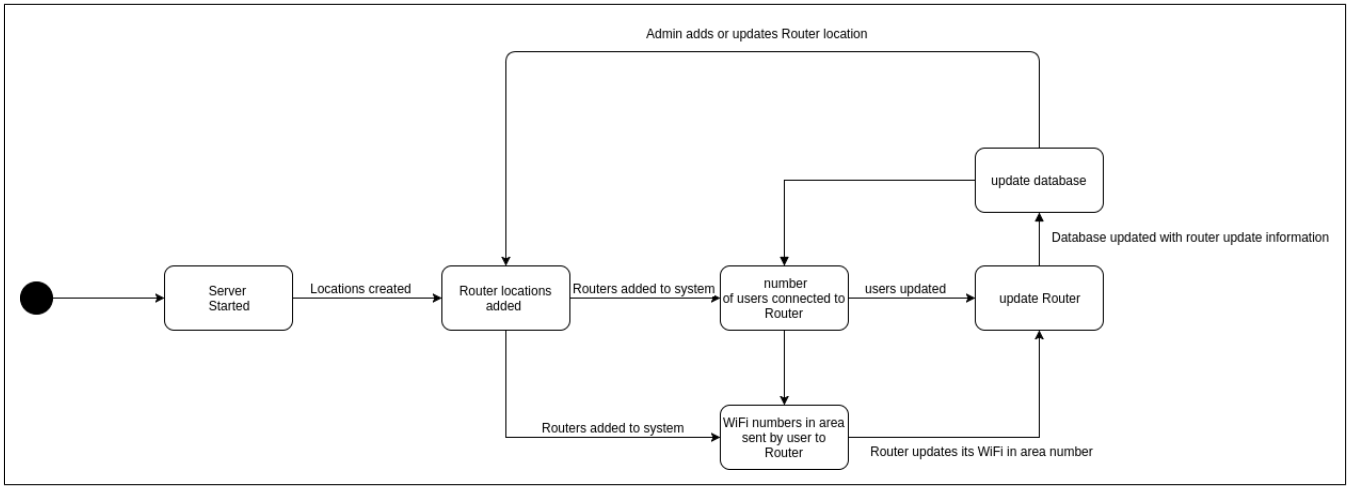

Figure 14: State Diagram - Surveillance Module

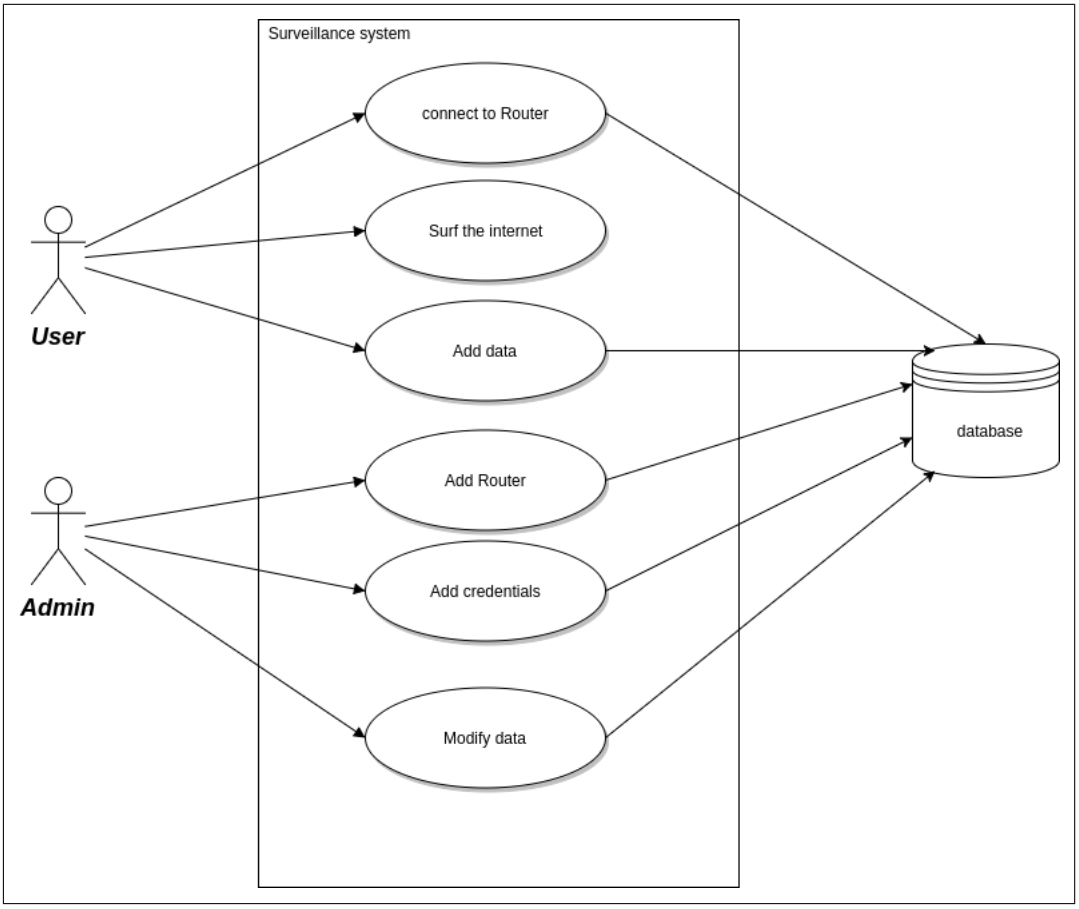

Figure 15: Use Case Diagram - Surveillance Module

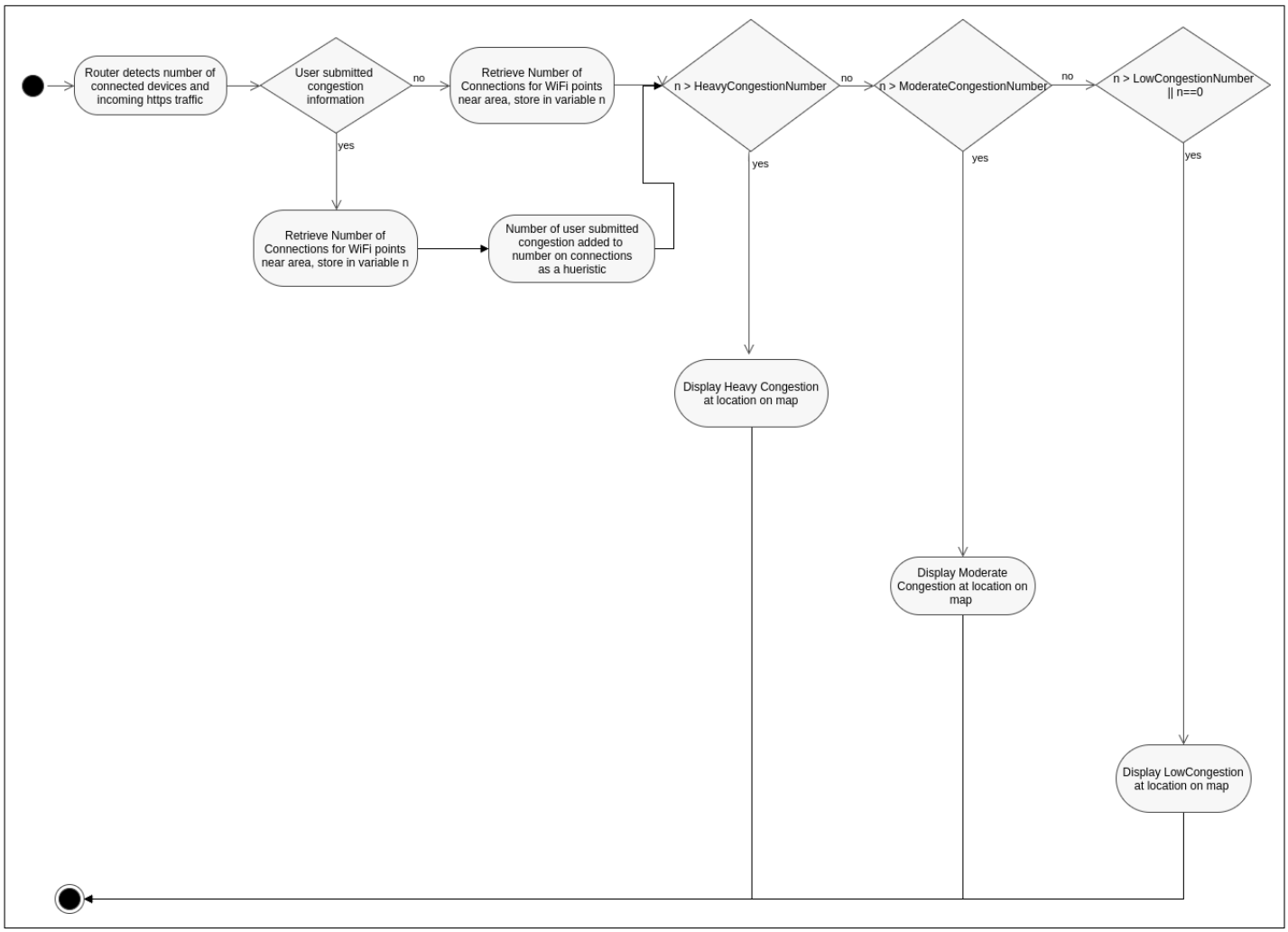

Figure 16: Activity Diagram - Surveillance Module