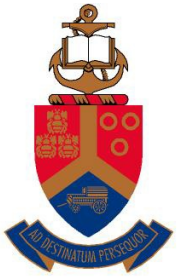

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

NavUP

Architectural Requirements Specifications and Design

10 March 2017

Team Scala

Merissa Joubert
15062440

Avinash Singh
14043778

Joshua Cilliers
14267196

Nicolai Van Niekerk
15025269

Gershom Maluleke
13229908

Cameron Trivella
14070970

Peter Soroush
13238958

GitHub https://github.com/RobOgirl/Team-Scala.git

Contents

1 Introduction

1.1 Overview

This document identifies the architectural design specifications that satisfy the functional requirements proposed in the System Requirements Specification. It addresses the needs of the various subsystems' non-functional requirements focusing on the quality attributes, architectural patterns as well as constraints and integration requirements of the NavUP system.

The following modules' designs are included in this document:

- Users
- GIS
- Notification
- Events

The document begins with an explanation on the chosen architectural pattern and how it will be used to modularize the NavUP system. It then outlines the architectural design of each module along with all relevant UML diagrams. Finally the chosen technologies are discussed as well as the deployment of the system.

1.2 Architectural Pattern

The NavUP system will be designed using the Three-tier architecture in which the user-interface, process logic and data storage are developed and maintained as independent modules. This will allow any of the three tiers to be upgraded or replaced without affecting any of the other layers.
The NavUP system will be divided into 3 tiers:

- Presentation tier
- Application tier
- Data tier

1.2.1 Presentation tier

This is the highest level of the NavUP application. It is the layer which users will interact with directly through the user-interface. The Presentation layer will reside on the mobile and web app and will display information pulled from the Application layer in a way that is simple and intuitive to the user of the application.

1.2.2 Application tier

This layer resides on the application server and controls the application's functionality through detailed processing. The results of this processing is passed back to the Presentation layer which will display it to the user.

1.2.3 Data tier

The Data layer includes all data persistence mechanisms and resides on the database server. The application server uses this layer to manage the stored data. It is crucial that there are no dependencies on the storage mechanisms to ensure that any updates or changes will not affect the application tier clients in any way.

2 Deployment Diagram

The diagram below depicts how the NavUP system will communicate and flow upon deployment. This can be seen as an overall view on the system in a deployment scenario.

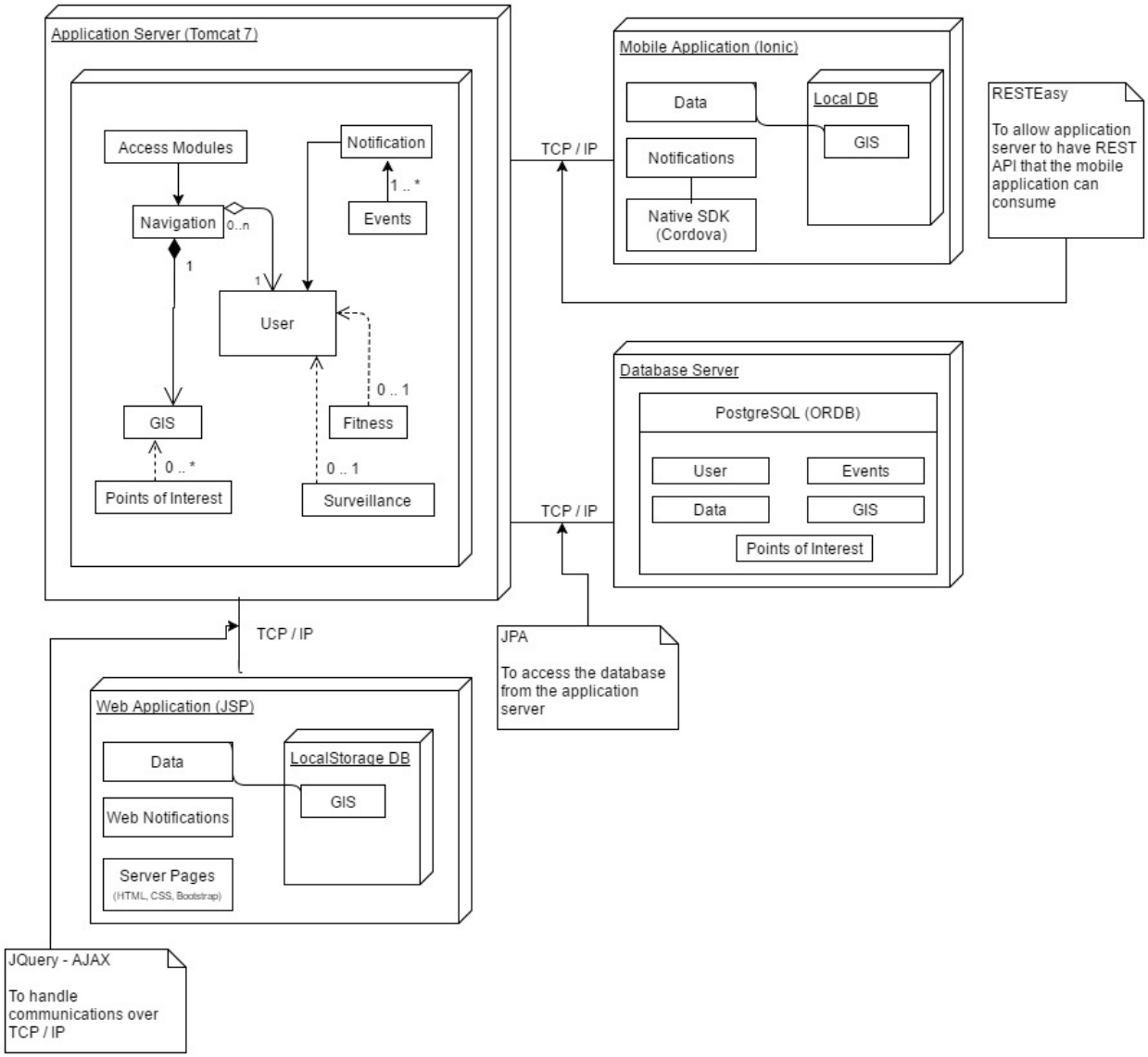

Figure 1: NavUP Deployment Diagram

3 User Module

3.1 Overview

The Users module is responsible for the management of NavUP's users. The system will consist of three types of users namely the admin,guest and normal users.The admin, who has more privileges than other users of the application, is mainly responsible for managing other registered users and managing information about venues and activities on campus.

3.2 External Interface Requirements

This section gives a detailed description of the system interfaces, hardware interfaces, software interfaces as well as communication interfaces.

3.2.1 System Interfaces

- The user module will interface with any subsystem or module that wishes to access the data or functions.

3.2.2 User Interfaces

- The basic function of the user interface is to enable users to interact with the system.All the functions necessary to use the NavUP application are performed through the user interface. The user interface allows the user to login or register if they are first time users.

3.2.3 Hardware Interfaces

- The will be a hardware interface with routers across campus in order to triangulate the position of the user on campus and this will require the hardware interface to interact with the Users module of the system.
- The hardware interface also includes devices that users use to access the NavUP application

3.2.4 Software Interfaces

The user module will communicate with the database in order to get the details of users for registration of updating user information, and also for saving locations etc. All the I/O interaction with the database.

3.2.5 Communication Interfaces

The User module would be the core communicator in this system since every other subsystem is directly/indirectly related to user, so communication can be done through local message passing with internal events or via the abstract user management class, another means of implementation would be to consume the RESTful API within the system however this is very inefficient for an local internal event.

3.3 Performance Requirements

User module should be able to perform under high CRUD operations,and have optimised methods to allow for faster execution and faster response times.

3.4 Design Constraints

- The speed at which NavUP can perform is constrained by the processing power and the memory that is available on the device on which it runs.
- The accuracy of the results produced by functions such as determining the user's location, which are done by the GIS module through the Users module are constrained by the strength of the WIFI connection on campus.
- This system's ability to give updates and events to users through the Users module is constrained by the external management and maintenance which is performed by the administrator of the system.

3.5 Software System Attributes

3.5.1 Availability

The user module should always be active since the majority of the application deals with the user module so having no down-time will be beneficial and required for the system to prevent crashes,

3.5.2 Reliability

The user module should make use of queuing for CRUD database operations, which most databases have and should be reliable since the database should be ACID compliant.

3.5.3 Security

Data within the user module should not have external classes accessing the information without being authenticated, user passwords and usernames should not be able to be accessed outside the scope of this module. Passwords should be encrypted using a strong encryption algorithm such as SHA-512 and should also support end-to-end encryption to guarantee more safer data transfer. The REST API should not be overexposed and have the necessary security implementations such as public/private key encryption, and should only accept authentic connections and prevent against DDOS.

3.5.4 Auditability

All CRUD operations within the user module should be logged for security and data integrity purposes.
All login attempts should also then be logged and notify an administrator if too many attempts to login was unsuccessful to ensure system security, since this could be proof for an audit trail.

3.6 UML

3.6.1 Class Diagram

The class diagram of the user sub-system makes use of the template method design pattern so that if need be one can easily construct different types of users with minimal code modifications.

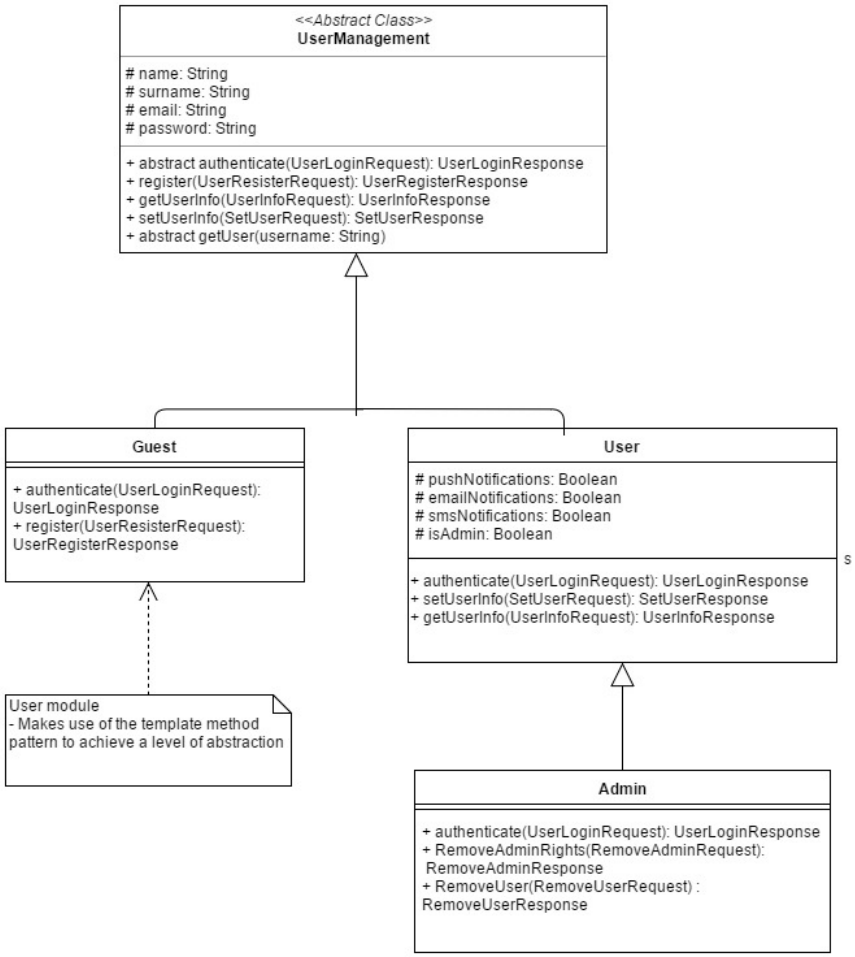

Figure 2: User Class Diagram

3.6.2 Activity Diagram

This diagram models work-flow activities of what users can do in the user module.

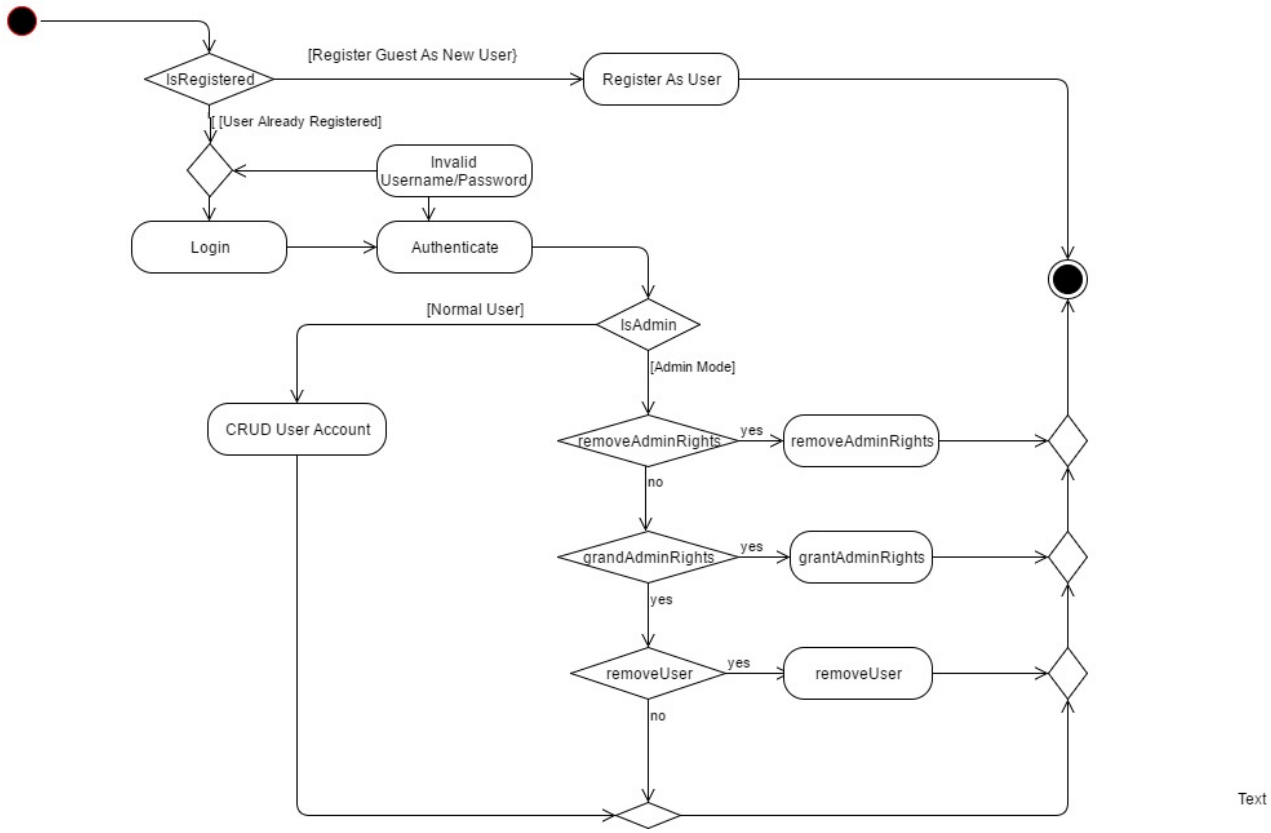

Figure 3: User Activity Diagram

3.6.3 Sequence Diagram

This diagram models time-ordered interaction behaviour between a user logging in and the interaction with the user subsystem.

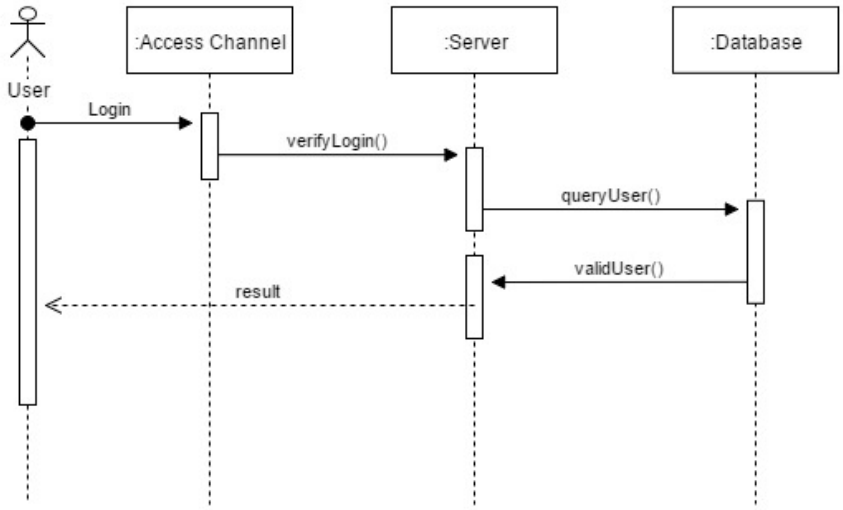

Figure 4: User Login

3.6.4 State Diagram

This diagram models state dependant behaviour of user login and user functionality.

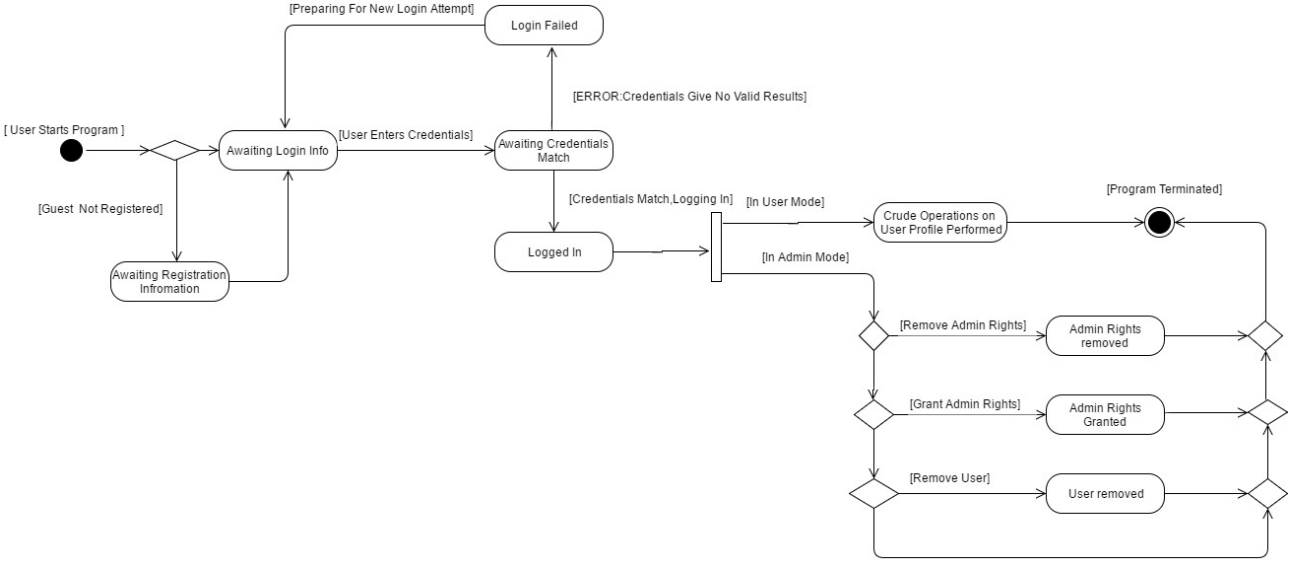

Figure 5: User Login State Diagram

3.6.5 Use Case Diagram

This is a refined version of the use case diagram given in the Requirements Specification. It shows the functions of the subsystem as well as relationships of external entities or actors. This refined version also includes a detailed flow of use cases.

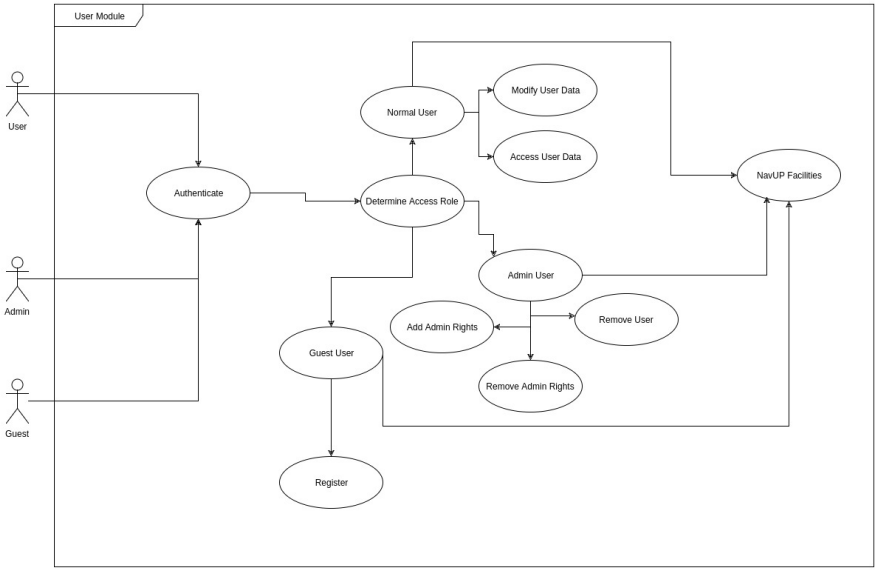

Figure 6: User Login with core functionality

4 GIS Module

4.1 Overview

The GIS module provides services to gather, maintain, persist and provide information related to the world serviced by the system. It is about the creation and maintenance of a GIS Map of the campus by using WiFi signal strengths and other available sources of GIS information. In addition it provides services to search for locations such as landmarks, buildings as well as venues such as offices, lecture halls and labs.

4.2 External Interface Requirements

This section gives a detailed description of the System Interfaces, hardware interfaces, software interfaces, communication interfaces and user interfaces.

4.2.1 System Interfaces

- The GIS module will interface with all subsystems or modules that work with location based data.
- The GIS module will interface with the Navigation module in order to send and receive navigation information. The navigation module will send requests to the GIS module, these requests will include location information. The GIS module will use the location information to build map objects and the mapping unit in the GIS subsystem will plot the map objects on a map which is sent back to the Navigation module for use by the user.
- The GIS module will interface with the Points of Interest module to create special gis objects like heritage sites and historic landmarks in the mapping subsystem of the GIS module

4.2.2 User Interfaces

The GIS subsystem will not directly interface with the user. All GIS related content will be shown to the user via the navigation subsystem.

4.2.3 Hardware Interfaces

The GIS module receives all its information from the navigation subsystem, thus it does not have any specific hardware interfaces.

4.2.4 Software Interfaces

- The GIS module will interface with the database in order to persist gis objects and retrieve location information needed to create gis objects.
- The GIS module will use position tracking gps software such as **Position Logic** to track user movements and map the areas around them.

4.2.5 Communication Interfaces

The GIS subsystem does not communicate directly with the user, and it does not send out any notifications. Thus the GIS module has no communication interfaces.

4.3 Performance Requirements

The GIS module must be able to create a working area map that changes with the movements of the user. The map must be accurate within an area around the user to an extend that the user can view both large areas like buildings and smaller areas like offices. The generated map must be able to update or refresh at a rate high enough for it to be used effectively while walking on campus.

4.4 Design Constraints

The map produced by the GIS module should:

- be uncomplicated and convenient to use.
- be straightforward and useful.
- support viewing on all devices, taking into consideration different sized displays.
- be colorized and visually appealing.
- integrate easily with the navigation subsystem.
- be user friendly to all target users, especially disabled users.
- updated/refreshed efficiently so as to not annoy the user with lag or slow response times.

4.5 Software System Attributes

4.5.1 Reliability

The GIS module's primary purpose is to act as a database or a repository for all the map points in and around campus. For this reason it needs to meet the typical database standards of reliability, availability, scalability, and recoverability (RASR). The GIS module database should make be ACID compliant so that the typical CRUD operations of most databases can be performed as reliably as possible.

4.5.2 Availability

The GIS module should always be active as the majority of services the app provides are all dependant on the GIS module so to maintain the app's functionality it should have zero downtime.

4.5.3 Scalability

The GIS module needs to be able to scale to handle a large amount of interactions so that it can still provide prompt feedback to users and not be slowed down or crash because of the large influx of user interaction.

4.5.4 Recoverability

The GIS module needs to be able to back up and recover the maps stored in it's database as well as all the map points stored on it so that system failures do not have catastrophic consequences.

4.6 UML

4.6.1 Class Diagram

The GIS class diagram makes use of several design patterns. It makes use of Decorator for quick assembly of different types of objects that the map will use; it makes use of Strategy for different methods to search through the database; it makes use of Memento to back up and restore the map database; it makes use of Builder to separate the construction of complex map objects and their representation.

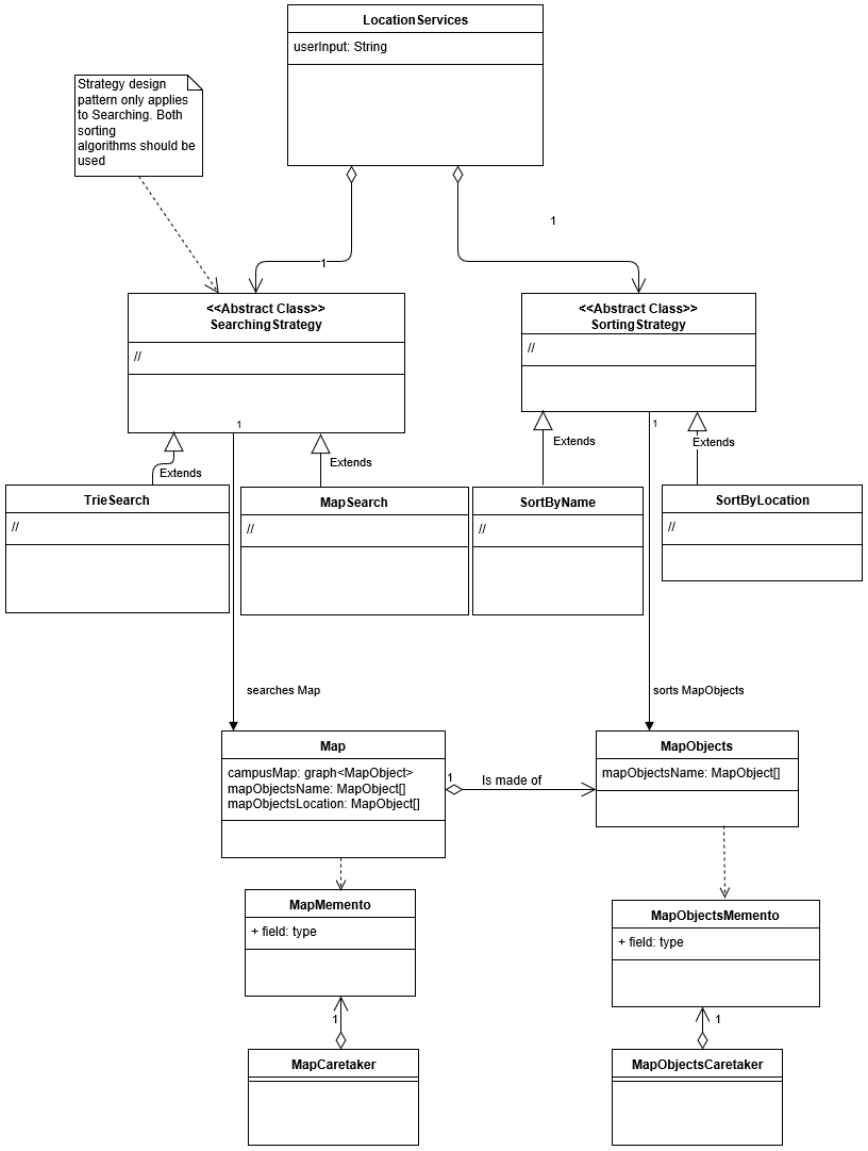

Figure 7: GIS Class Diagram

4.6.2 Activity Diagram

This diagram models work-flow activities of what users do when interacting with the GIS module.

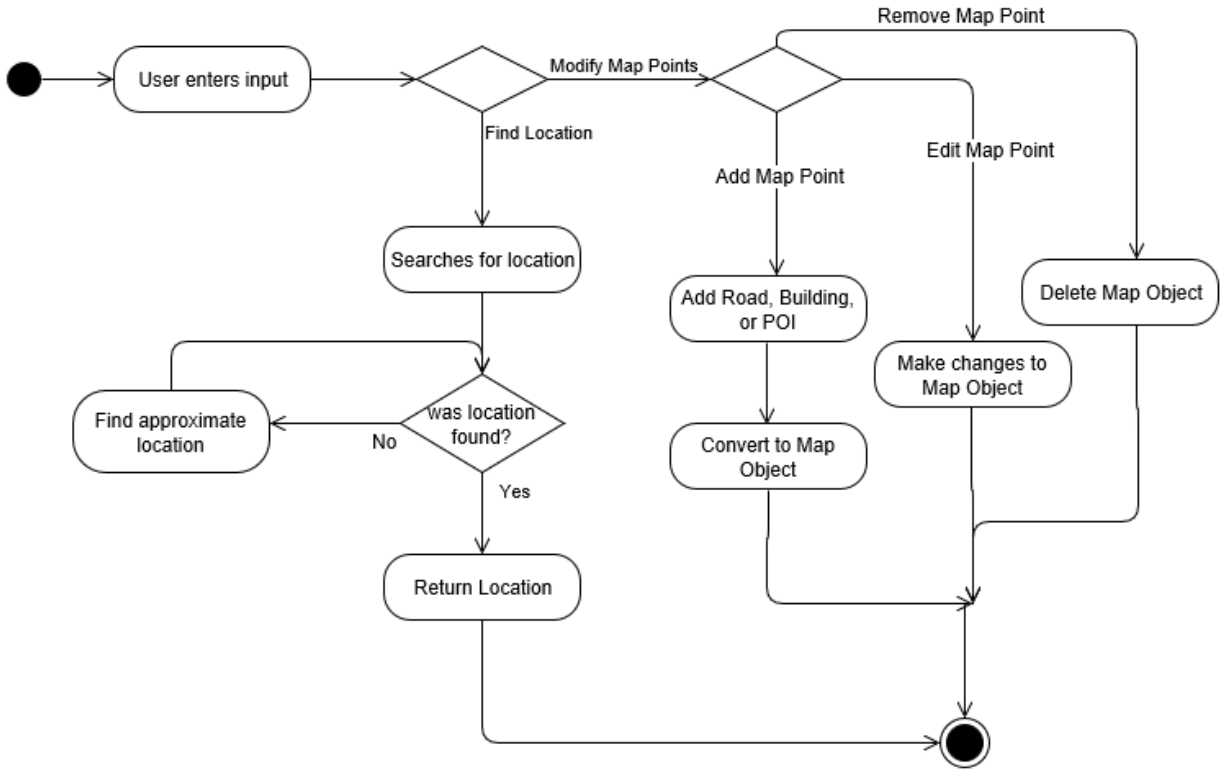

Figure 8: GIS Activity Diagram

4.6.3 Sequence Diagram

This diagram models a user's interaction with the GIS module over time.

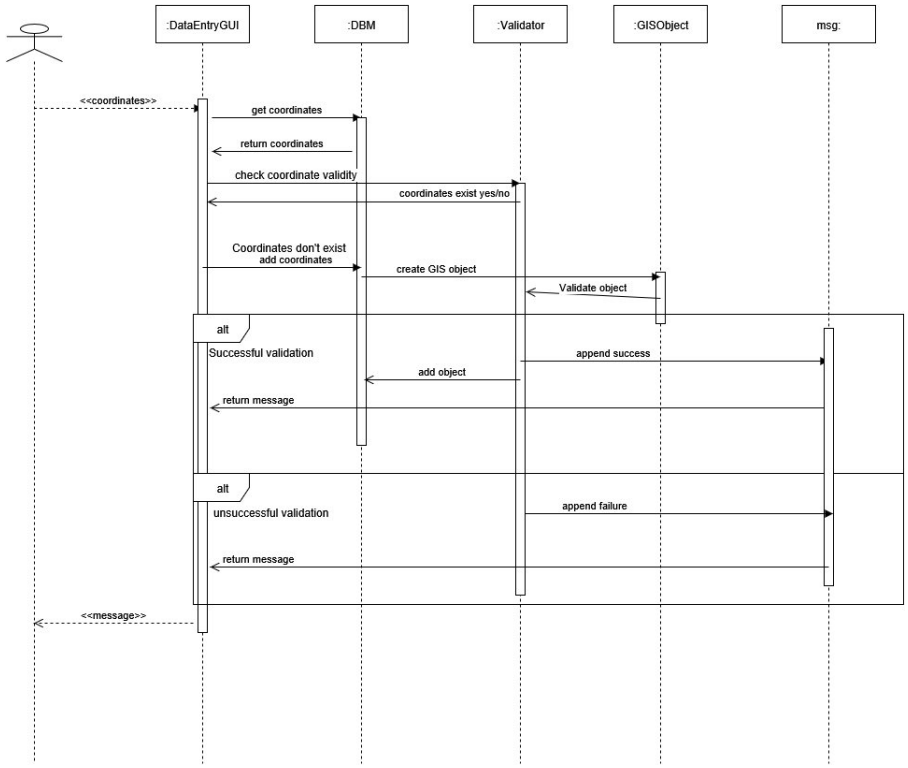

Figure 9: GIS Sequence Diagram

4.6.4 State Diagram

This diagram models state dependant behaviour of user interaction with the location-searching subsystem and the map-editing subsystem.

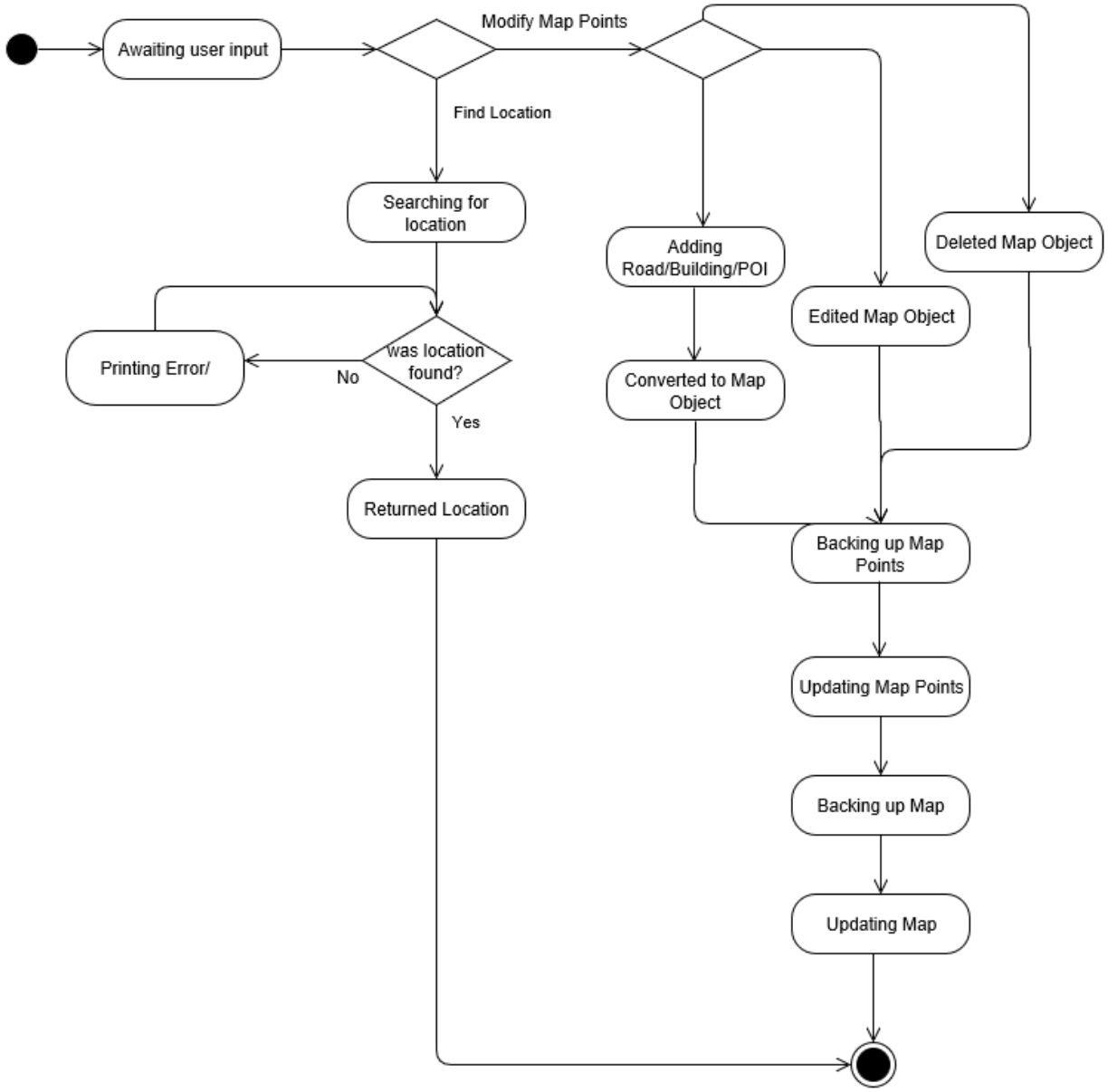

Figure 10: GIS State Diagram

4.6.5 Use Case Diagram

This is a use case diagram showcasing the interactions that users have with the GIS module

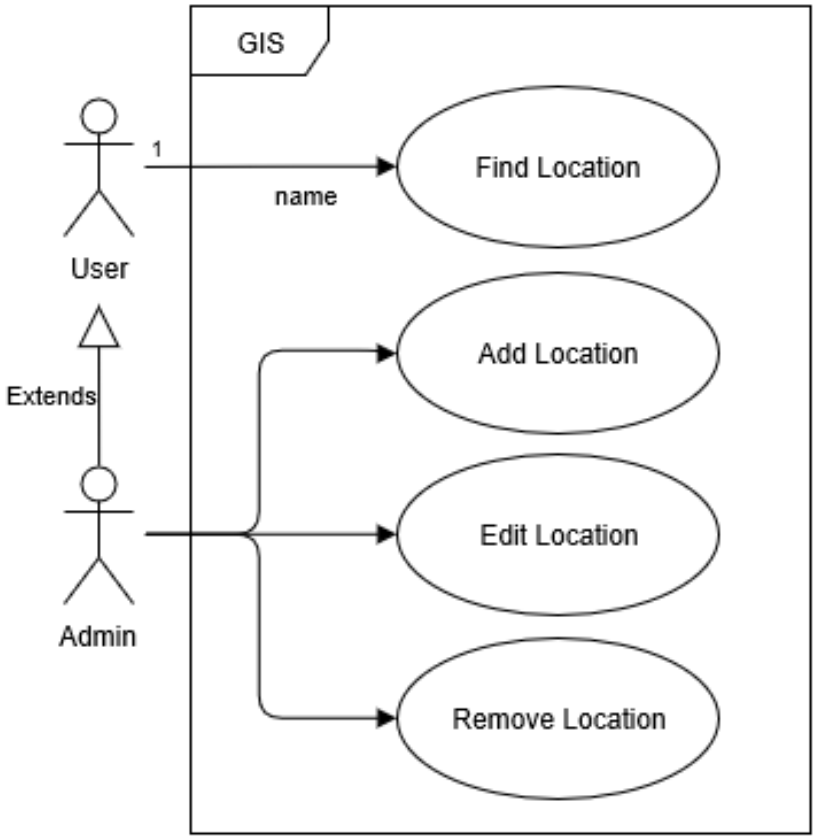

Figure 11: GIS Use Case Diagram

5 Event Module

5.1 Overview

The Event module is responsible for the Event Management for example lectures, Programs, tests and events.

5.2 External Interface Requirements

This section gives a detailed description of the system interfaces, hardware interfaces, software interfaces as well as communication interfaces.

5.2.1 System Interfaces

- The user module will interface with any subsystem or module that wants to access Events.
- The user module will interface with Notification to Notify User about Events

5.2.2 User Interfaces

- User Interface is in Calander Form which Shows all the events in it.

5.2.3 Hardware Interfaces

The Event Module does not have any explicit hardware interface

5.2.4 Software Interfaces

The event module will communicate with the database in order to get the details of events, and also for saving and editing new Events etc. All the I/O interaction with the database.

5.2.5 Communication Interfaces

- The Event module will need to comunicate with Notification to notify Users.
- needs to comunicate with Users as only registered user can see Events.

5.3 Performance Requirements

Event module should be able to perform under high CRUD operations,and have optimised methods to allow for faster execution and faster response times.

5.4 Design Constraints

- The speed at which NavUP can perform is constrained by the processing power and the memory that is available on the device on which it runs.
- This system's ability to give updates and events to users through the Users module is constrained by the external management and maintenance which is performed by the administrator of the system.

5.5 Software System Attributes

5.5.1 Availability

The Event module should always be active since the it has Time Countdown and also there is posibility od Changing and updating all the time.

5.5.2 Reliability

The Event module should make use of queuing for CRUD database operations, which most databases have and should be reliable since the database should be ACID compliant.

5.5.3 Security

Data within the Event module should not have external classes accessing the information without being authenticated, only admins can modify Events and add them or remove it, so No Guest or Normal User must be able to access to editing functionalities, also not registered User cannot see Events.

5.5.4 Auditability

As Admin,Normal and Guest Users have different Authorities to View and Modify Events, so user must be Logged on the Back-end Server before Getting to Events section

5.6 UML

5.6.1 Class Diagram

The class diagram of the Events sub-system makes use of the template method design pattern so that if need be one can easily construct different types of Event with minimal code modifications.

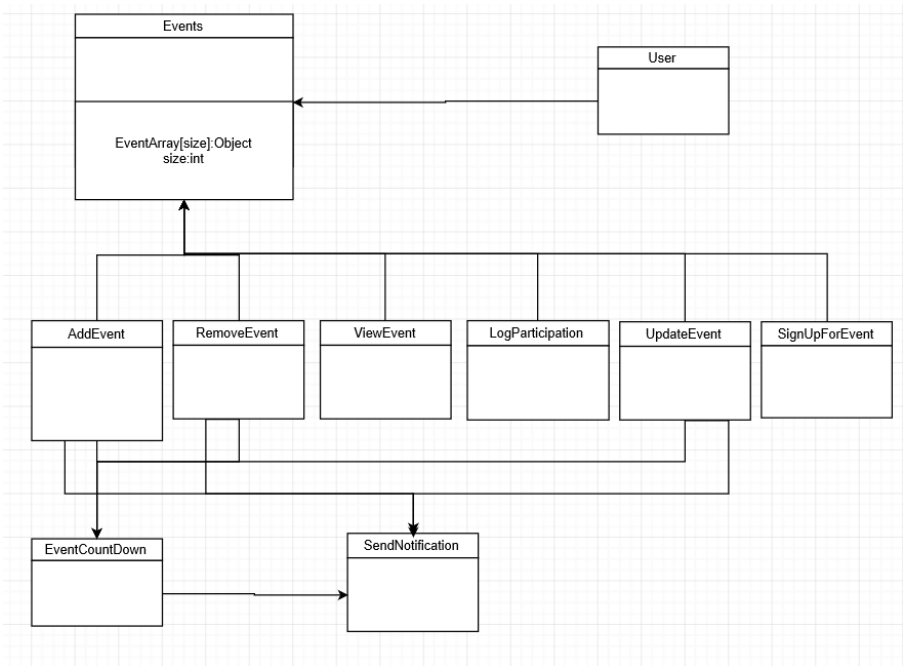

Figure 12: Event Class Diagram

5.6.2 Activity Diagram

This diagram models work-flow activities of what users can do in the Event module.

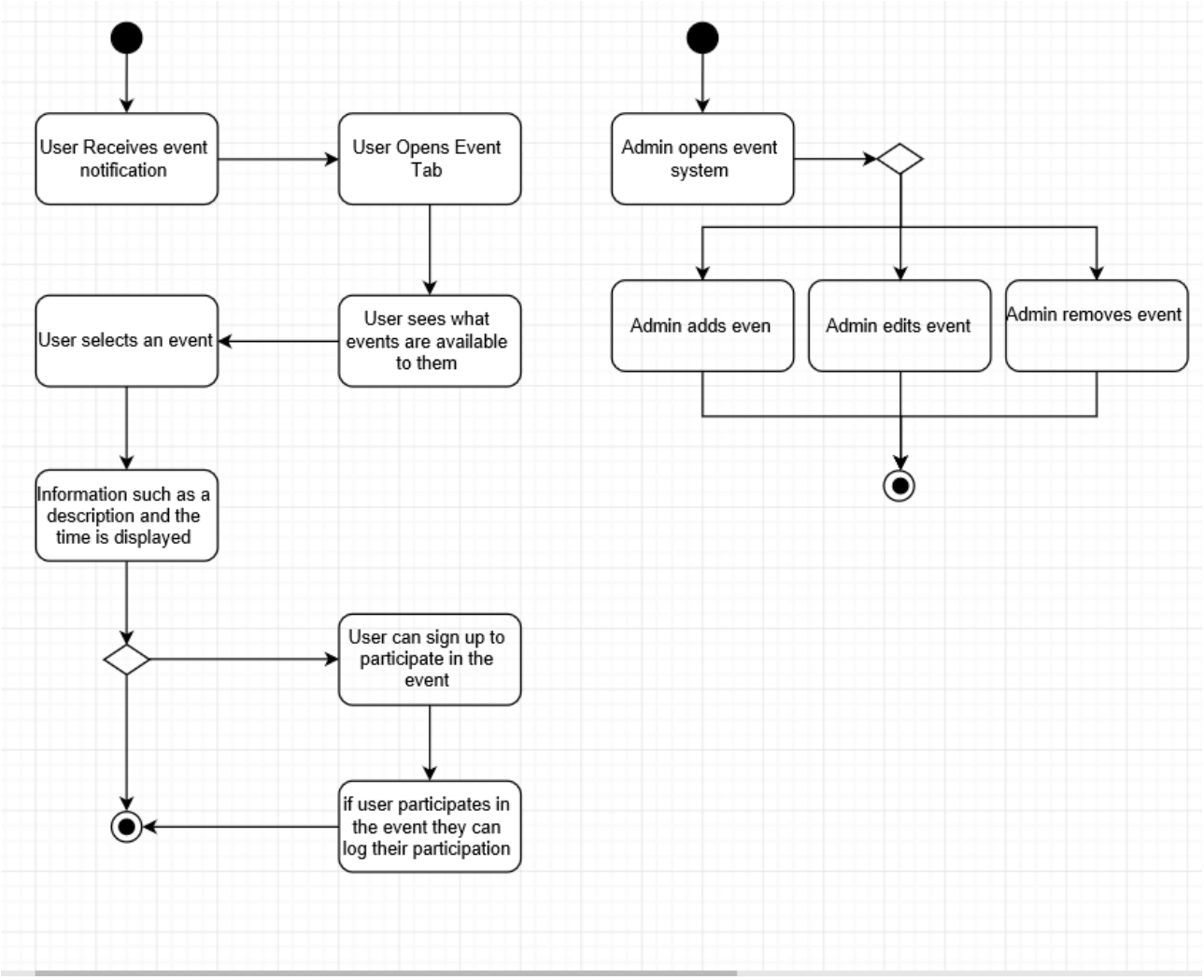

Figure 13: Events Activity Diagram

5.6.3 Sequence Diagram

This diagram models time-ordered interaction behaviour between a user logging in and the interaction with the Event subsystem.

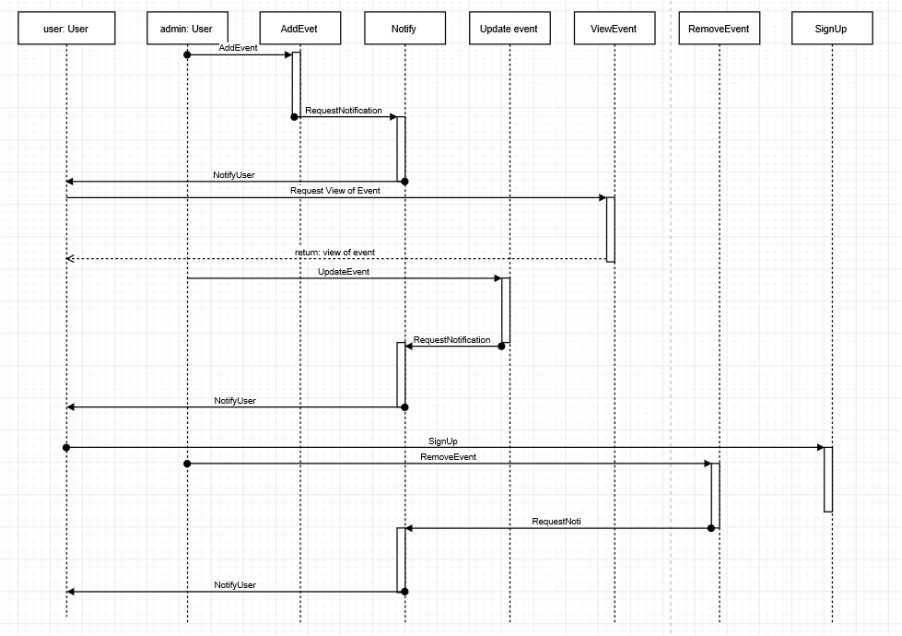

Figure 14: Events Sequence Diagram

5.6.4 State Diagram

This diagram models state dependant behaviour of a user with Event Modules.

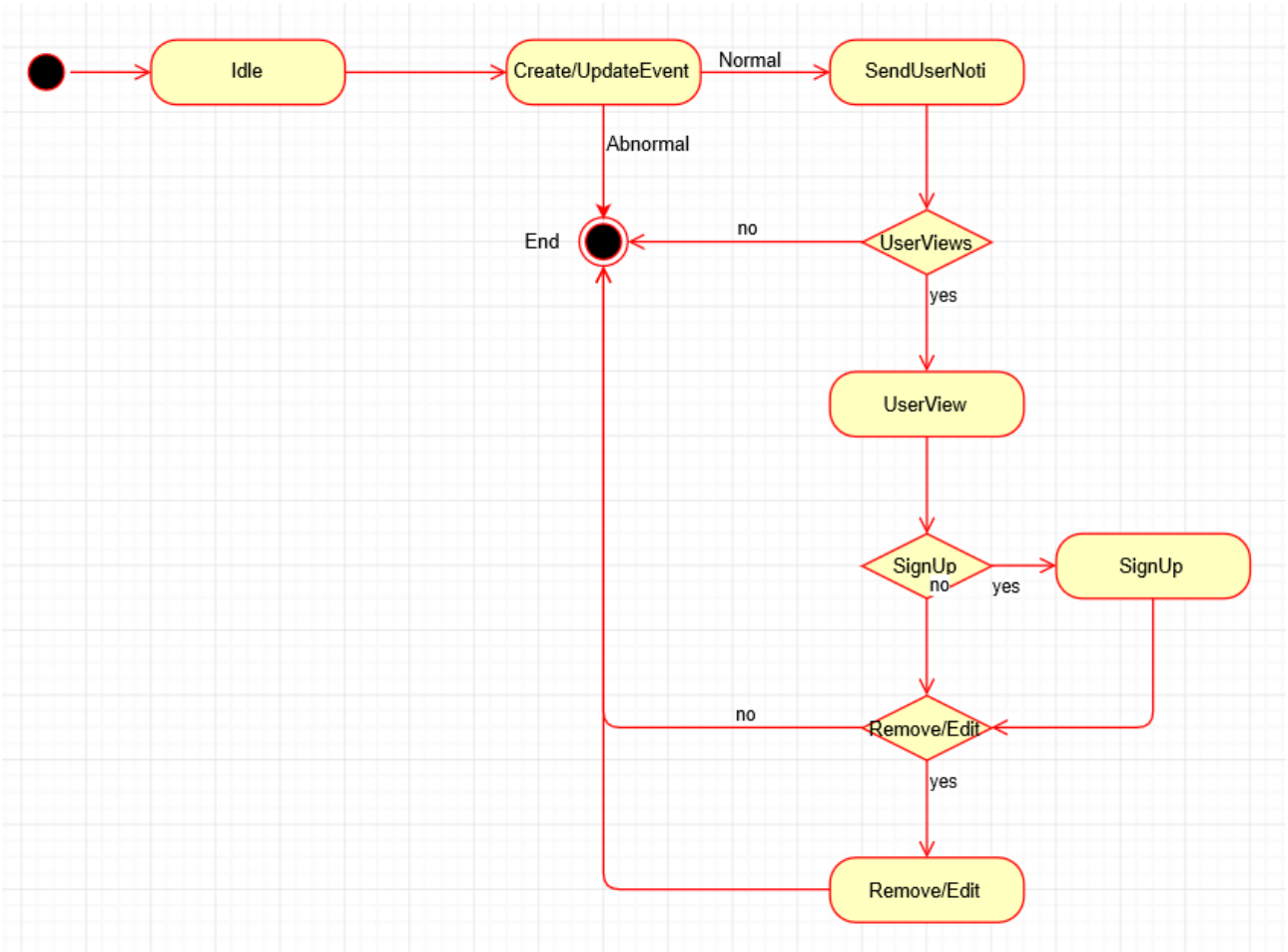

Figure 15: Events State Diagram

5.6.5 Use Case Diagram

Use case diagram which shows the functions of the subsystem as well as relationships of external entities or actors. This refined version also includes a detailed flow of use cases.

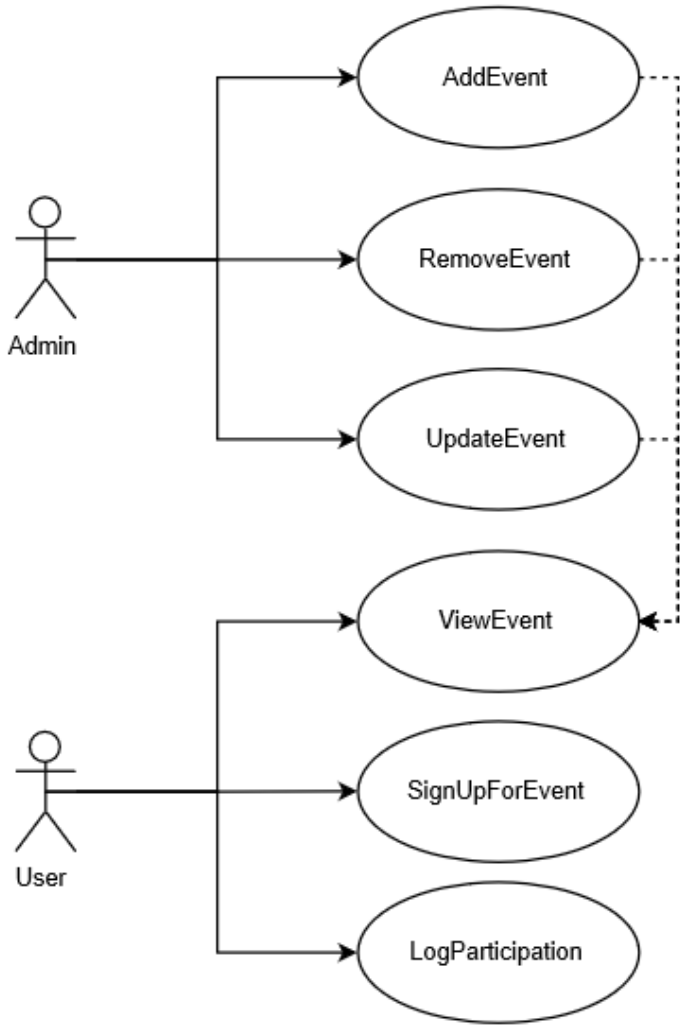

Figure 16: Events core functionality

6 Notification Module

6.1 Overview

The notifications module provides notifications to system users regarding particular system updates that a user would like to be notified about through some medium external to the application.

6.2 External Interface Requirements

This section gives a detailed description of the system interfaces, hardware interfaces, software interfaces as well as communication interfaces.

6.2.1 System Interfaces

- The notification module will interface with any subsystem or module that wishes to send out notifications to users. The subsystem requesting for users to be notified is modelled as the Notifying Module. The Notifying module will interface with the Notification Module through the setState() function to send a notification request.
- The notification module will interface with the User module in order to sufficiently dispatch notifications. Each instance of the Notifier class will maintain a list of attached users and will send notifications through the notify() function.

6.2.2 User Interfaces

- The system will dispatch notifications in the form of an **SMS** to the appropriate users. The SMS will contain the content of the notification and the interface will depend on the user's device.
- The system will dispatch notifications in the form of an **email** to the appropriate users. The email will have an appropriate subject and will contain the content of the notification. The interface will depend on the mail application used by the user.
- The system will dispatch notifications in the form of **Push Notifications**. push notifications will contain a title, content and a timestamp. The interface will depend on the user's operating system.

6.2.3 Hardware Interfaces

The Notification Module does not have any explicit hardware interfaces

6.2.4 Software Interfaces

- The notification module will communicate with the database in order to get the details of users to be notified.

6.2.5 Communication Interfaces

- The notification module will need to communicate with an Email API when sending out emails to users
- The notification module needs to interface with the SMS Manager API to send SMS's to users.
- The notification module will interact with the Operating system push notification service (OSPNS) to allow users to receive push notifications.

6.3 Performance Requirements

Notifications should be able to perform and guarantee message delivery under high volumes, allow real time notifications to be received and have good network access for this purpose.

6.4 Design Constraints

The notifications must be:

- Visually appealing and easy to read, have a gamification point of view
- Support viewing on different sizes of device screens
- Support colour for all devices
- Being able to integrate easily with notification providers

6.5 Software System Attributes

6.5.1 Reliability

The system should make use of a queue to send notifications, so that no messages can get lost on the way and must guarantee that the notification must be delivered. Notifications should be authentic and be able to verify that delivery of the notification.

6.5.2 Security

The sending of the notifications must be secure using proper SMTP setup with SSL encryption for emails, and a authentic SMS provider validated by the WASPA, and a secure means of push notifications using IONIC App that allows secure remote push notifications to all user devices that contain the app.

6.5.3 Auditability

All notifications that have been sent by any means, by email, SMS, or even push notifications should be logged on the back-end server.

6.5.4 Availability

The notifications module should be available throughout the entire execution of the system since it needs to provide real-time notifications to the user, of events etc. nearby. This becomes vital for the module to be optimised and have good communications network to account for the minimal possible network delay to make meet the real-time expectations.

6.6 UML

The Notification Subsystem is designed using 2 design patterns:

1. **Strategy -** This pattern encapsulates the send() function within the Notification class, making it interchangeable. This allows the send() function to vary based on whether the notification is an email notification, SMS notification or a push notification.
2. **Observer -** This pattern defines a one-to-many relationship between the Notifier and User classes so that when the Notifier's state changes, all users are notified automatically.

6.6.1 Class Diagram

This diagram models the structural elements, their composition and classification.

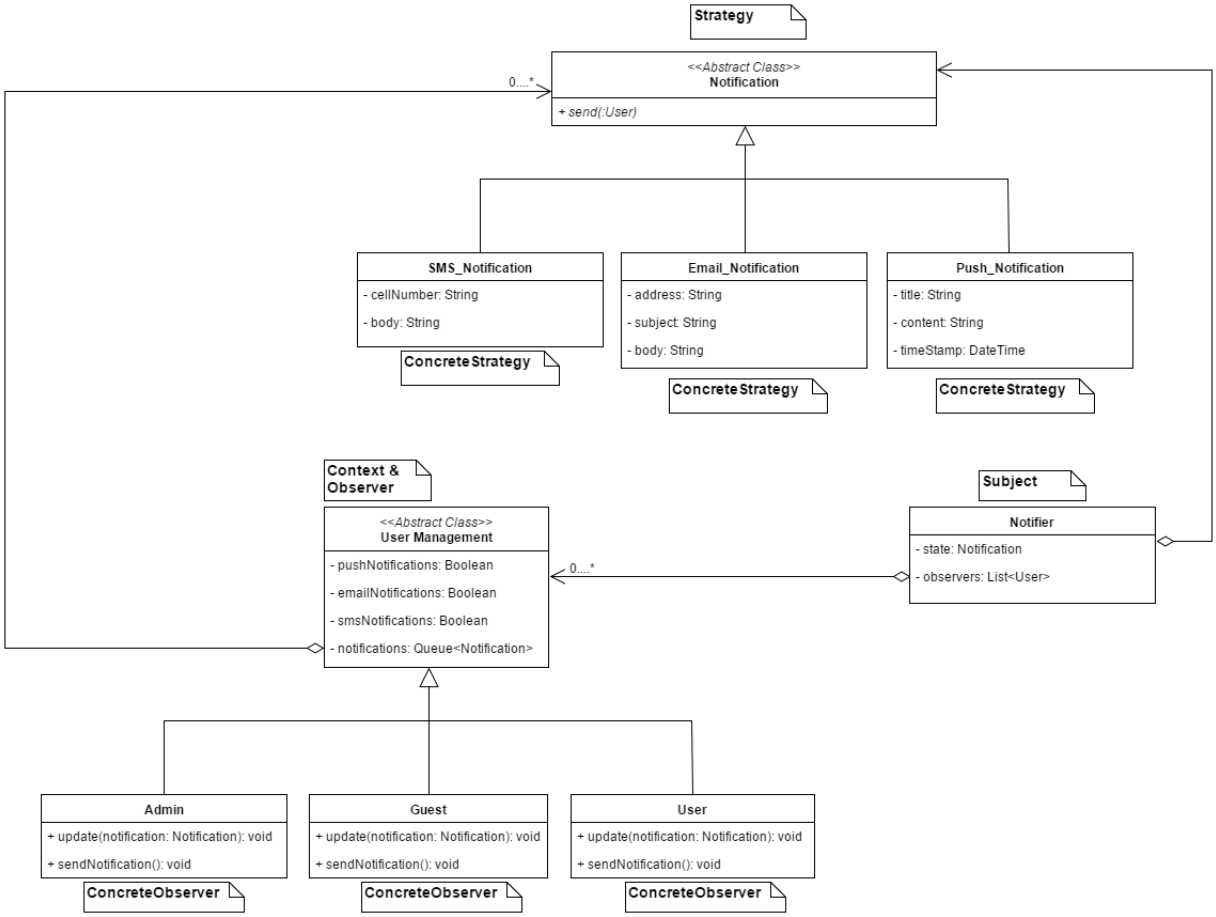

Figure 17: Notification Class Diagram

6.6.2 Activity Diagram

This diagram models work-flow activities and exhibits sequencing, exclusion, synchronization and concurrency.

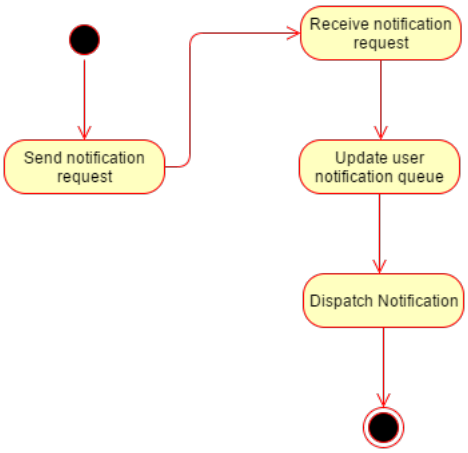

Figure 18: Notification Activity Diagram

6.6.3 Sequence Diagram

This diagram models time-ordered interaction behaviour between the objects within this subsystem.

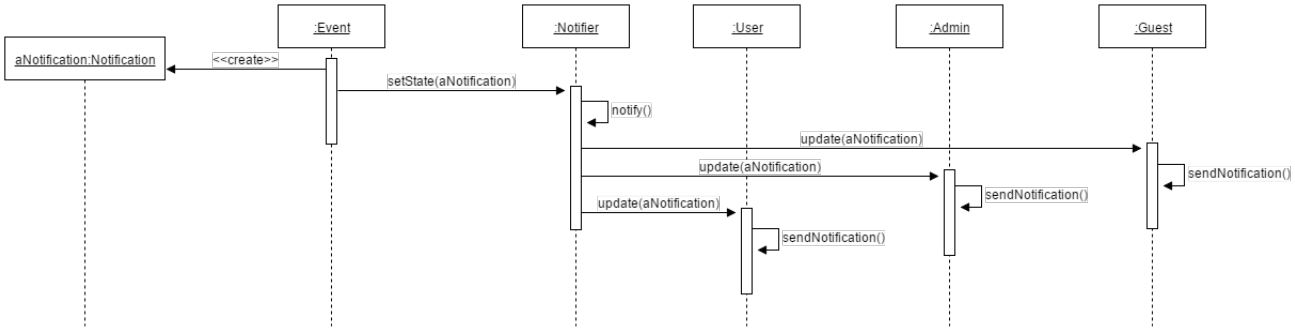

Figure 19: Notification Sequence

6.6.4 State Diagram

This diagram models state dependant behaviour of an object.

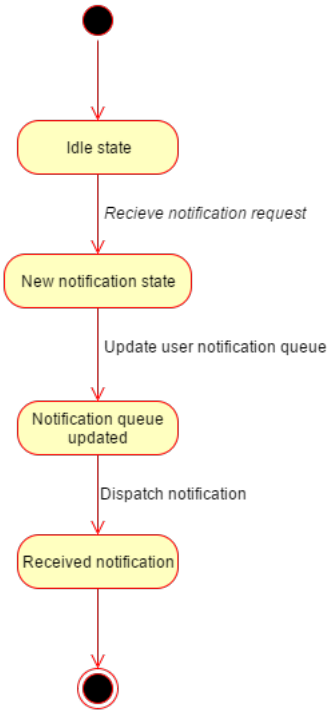

Figure 20: Notification State Diagram

6.6.5 Use Case Diagram

This is a refined version of the use case diagram given in the Requirements Specification. It shows the functions of the subsystem as well as relationships of external entities or actors.

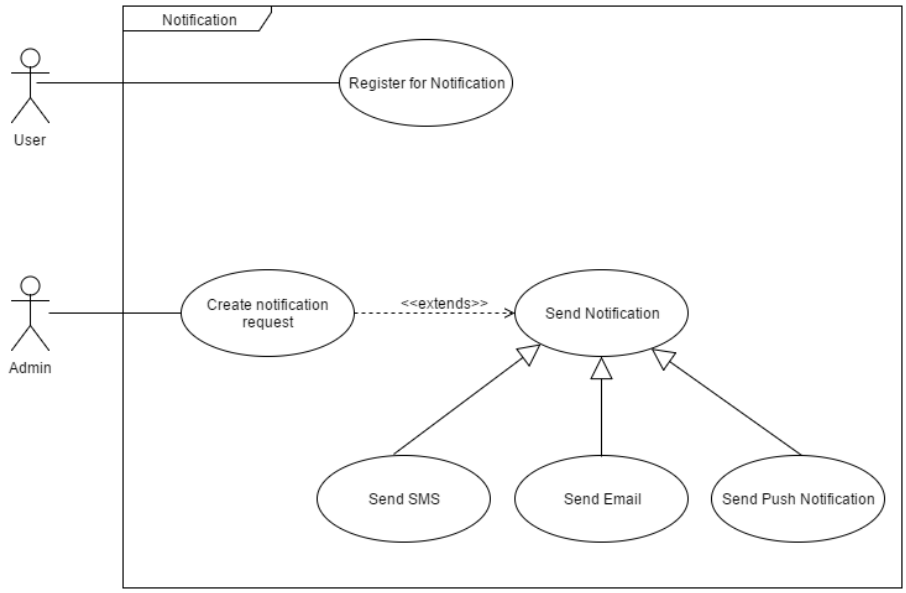

Figure 21: Notification core functionality

7 Technologies

The technologies to be used for the system will be discussed below:

- TOMCAT - it is a free Apache application server where one can deploy and run JSP and Servlets and provides good logging and management facilities since every app gets wrapped into a war file and is mostly used for web applications as its limited to functionality.
- WILDFLY - similar to tomcat server but this is a full fledged JavaEE application server for the back-end system.
- JAVAEE - will be used develop the business logic of NavUP which forms part of the back-end. JavaEE is a good choice for NavUP because this platform uses "containers" to simplify development. JavaEE containers provide for the separation of business logic from resource and life-cycle management, which means that developers can focus on writing business logic, rather than writing enterprise infrastructure.
- TCP/IP - the reason why we chose this over UDP since we want to know if the user has received the data we sent which will help making sure that the user has the correct map all the time and since it's end-to-end we can also have encryption for safety and privacy issues.
- JAX-RS REST - Using the RESTEasy implementation of JAX-RS since it is portable and can run on any Servlet container, and integrates nicely with JBOSS and provides client framework to make writing HTTP clients easy.
- ACTIVEMQ - is a message broker which deals with synchronization and messaging queue facilities which will be necessary for notifications module, it provides a lot of features such as REST API to provide technology agnostic and language neutral web based API to messaging, supports Ajax, fast and supports pluggable protocols such as TCP and SSL in specific that we will use.
- JPA - The Java Persistence API (JPA) is a specification for object-relational mapping in Java. The main advantage JPA will have on NavUP will be its database independence. JPA It provides a database independent abstraction on top of SQL. As long as you're not using any native queries, you don't have to worry about database portability.
- POSTGRESQL / POSTGIS - The entire code of the system should be decoupled from which concrete database is used. For NavUP PostgreSQL will be used. It is a mature, efficient and reliable relational database implementation which is available across platforms and for which there is a large and very competent support community
- JUnit - JUnit will used to perform unit tests on separate modules of NavUP. Junit has been chosen due to its simplicity when testing Java applications. JUnit can be used separately or integrated with build tools like Maven and Ant and third party extensions.
- JSP - Java Server Pages (JSP) is a technology that helps software developers create dynamically generated web pages based on HTML, XML, or other document types. The main benefit that JSP will have on NavUP is that JSP pages are compiled dynamically into Servlets when requested, so page authors can easily make updates to presentation code.
- HTML5 & BOOTSTRAP - The main advantage of using bootstrap on NavUP will be its responsiveness.Creating mobile ready websites is a breeze with Bootstrap thanks to the fluid grid layout that dynamically adjusts to the proper screen resolution. There is virtually no work that needs to be done to achieve proper responsiveness. HTML5 is a markup language used for structuring and presenting content on the internet. One of the coolest things about HTML5 is the new local storage feature which is better than cookies because it allows for storage across multiple windows and also has better security and performance and data will persist even after the browser is closed.
- JQUERY - works off JavaScript providing easier methods and powerful functionality to developers with ease, it provides easier methods such as Ajax which makes it better for communications, and event handling for triggering functionality when certain conditions are met.
- CSS - CSS(Cascading Style Sheet) will be used for styling the web front end of NavUP. It was chosen because of its Lightweight code and because it's easier to maintain and update.

- MAVEN - Maven is a powerful project management tool that is based on POM(Project Object Model).It is used for projects build,dependency and documentation. Maven will be suitable for NavUP because maven projects do not need to store third-party binary libraries in source control, which reduces undue stress on developer checkouts and builds which results in better dependency management.
- POSITION LOGIC - is a GPS tracking platform that provides accurate information about the users location which we can incorporate to make the user experience and accurateness of the map more significant.
- IONIC v2 - is a open source hybrid mobile application framework which deals with web technologies such as HTML, Angular, JavaScript and Typescript to make a nice looking application with minimal effort, they have their own custom styling to make it easier to build native looking applications with ease. It is also powerful for data processing since v2 works with typescript which is a OOP language derived from JavaScript, this typescript gets transpiled into JavaScript at run time. It unlocks native APIs and features by wrapping Cordova plug-ins into typescript especially good for accessing wireless information and also GPS.