NavUP
Architectural Requirements, Specifications and Design

Compiled By

Boikanyo Modiko - u15227678
Linda Potgieter - u14070091
Marthinus Richter - u15160671
Nkosenhle Ncube - u13247914
Quinton Swanepoel - u15245510
Ruan Klinkert - u14022282

201
TEAM Tlc

Contents

1 Introduction **3**
- 1.1 Deliverable . . . 3
- 1.2 Scope . . . 3
- 1.3 Definitions, Acronyms and Abbreviations . . . 3

2 Overall Architectural Design **4**
- 2.1 Overview . . . 4
  - 2.1.1 Performance Requirements . . . 4
  - 2.1.2 Design Constraints . . . 4
  - 2.1.3 External Interface Requirements . . . 5
  - 2.1.4 Software System Attributes . . . 5
- 2.2 Deployment Diagram . . . 7

3 User Management Module **7**
- 3.1 UML Class Diagram . . . 7
- 3.2 Use Case Diagram . . . 8
- 3.3 Architectural Design of Module . . . 8
- 3.4 Technologies Used . . . 9
- 3.5 Non-Trivial Implementation Tasks . . . 9

4 Notifications Module **10**
- 4.1 UML Class Diagram . . . 10
- 4.2 Use Case Diagram . . . 10
- 4.3 Notification Module Architectural Design . . . 12
  - 4.3.1 Classes . . . 12
  - 4.3.2 Design Patterns . . . 12
- 4.4 Technologies Used . . . 13
- 4.5 Non-Trivial Implementation Tasks . . . 13

5 Points of Interest Module **15**
- 5.1 UML Class Diagram . . . 15
- 5.2 Use Case Diagram . . . 15
- 5.3 Architectural Design of Module . . . 15
- 5.4 Technologies Used . . . 16
- 5.5 Non-Trivial Implementation Tasks . . . 16

6 Events Module **21**
- 6.1 UML Class Diagram . . . 21
- 6.2 Use Case Diagram . . . 22
- 6.3 Architectural Design of Module . . . 22
- 6.4 Technologies Used . . . 23
- 6.5 Non-Trivial Implementation Tasks . . . 23

1 Introduction

1.1 Deliverable

We are required to use the high level functional requirements of the NavUp system given in a separate document to identify the architectural design specifications that satisfy the identified functional requirements from our last assignment, focusing on the subsystems architectural designs.

1.2 Scope

The final objective is to create a NavUP mobile application as well as a web-based interface. The NavUP mobile app should be available on both android and ios platforms while the web-based interface will be used for administration and maintenance. The primary objective of NavUP is to allow students, visitors and staff to successfully navigate around campus in a efficient manner.

1.3 Definitions, Acronyms and Abbreviations

Term	**Definition**
App	Mobile application
CRUD	Create, Read, Update and Delete
GPS	Global Positioning System
TUCBW	This Use Case Begins With
TUCEW	This Use Case Ends With
UP	University of Pretoria

2 Overall Architectural Design

2.1 Overview

The development of the NavUP system takes a very modular approach, hence requiring effective communication between several modules all developed separetly. This involves the implementation of several design patterns and the linking of them with one another.

2.1.1 Performance Requirements

- The system will make use of campus wifi aswell network coverage.
- Users will be able to register on the system, but also be able to use the system as a guest.
- Admin users will have control of system tasks and user management.
- Admin users should be able to set up an event which users may attend.
- Notifications for events should be pushed through to the users when requested by the admin.
- The admin user/s should have easy access to the systems GIS module.
- Admin should be able to easily add points of interest to the system.
- The points of interest module should be used by the events module when selecting a destination for the event, and also by the navigation module for when providing directions.

2.1.2 Design Constraints

- The application should be within certain size contraints as to allow the application to install on even entry level smartphones without causing storage problems.
- The application must not be memory intensive causing entry level smartphones to become slow when running the application.
- The application requires access to network/wifi coverage aswell as location data for the device running it in order to provide accurate shortest and least congested routes.
- Logging into the system and loading relevant user specific data should take place within a reasonable amount of time. This requires accessing and retrieving database information in a efficient and effective manner.

2.1.3 External Interface Requirements

- User Interface - The android/ios version of the system should incorporate a user interface that is easy to navigate on touch screen devices of various sizes, from small entry smartphones to large tablets. Text should be clearly visible and buttons should not be tightly coupled together. The system notifactions should appear in the devices notifications centre/bar even when the application is not open. The web-based version should be easy to navigate and resemble the app as closely as possible.When logged into the system as an admin, the interface should allow for events to easiley be created and allow for effecient user management.

- Hardware Interface - The system will be designed in a manner whereby it can be ported to run on both all ios devices and all android devices. Also a web-based version should be able to open on all popular browsers.TCP/IP communication will be used to transfer data from the servers to devices running the NavUP application.

- Software Interface - The system will connect to a MySQL database in order to store user data, events, and other module specific related data. The system should be capable of running on the both Android and ios devices, aswell as have a web-based version. Google maps packages may also be imported to assist in navigation.

- Communication Interface - The system will require users to connect to a mobile network or wifi hotspot in order to provide accurate navigation routes. The system should also be able push email notifications through to a selected user or group of users. The system requires a reasonable constant data connection in order to provide live updates of routes and events.

2.1.4 Software System Attributes

- Conceptual Integrity - The system should be designed in such a way that the linkage between modules should be clearly evident in the UML diagrams. Variable/attribute names should be relative to their purpose.

- Maintainability - Modules/features should be able to be added to the system effectively and efficiently without the system having to undergo major changes.

- Reuseability - The system takes a modular approach, allowing the modules to be reused in other systems if necessary.

- Availability - The system should be availabe 24/7, unless server maintenance needs to be done, which should be done during hours when the system is used the least. This system should allow maintenance to be as quick as possible.

- Manageability - Admin users should easiley be able to perform system tasks, such as, removing users and creating events through logging into the system with their details.
- Performance - The servers must communicate fast and effectively with connected devices so that navigation is provided in real time and users can log in to their accounts within a matter of seconds.
- Reliability - The integrity of data should not be compromised during transfer. This will prevent innacurate navigation information being passed to the user.
- Scalability - The system must cater for large loads of data at any given point of time. It should be prepared for every student on campus to request navigation at the same time. even though this situation will most likely never occur.
- Security - System must ensure that user data is not accessed from outside sources and the servers running the system are not open to malicious attacks.
- Supportability - Error messages should provide a understandable description. A help section should be added to the app so that users may troubleshoot any problems they have with ease.
- Usability - The application must provide an easy to use interface that users of all levels of technological experience may use. The user experience will be improved on by the gamification aspect added by the fitness and treasure hunt modules.

2.2 Deployment Diagram

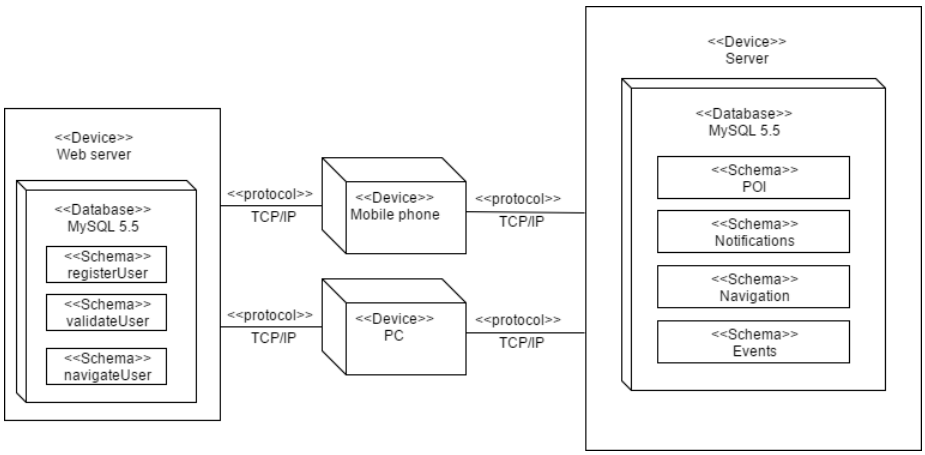

Figure 1: Deployment diagram

3 User Management Module

3.1 UML Class Diagram

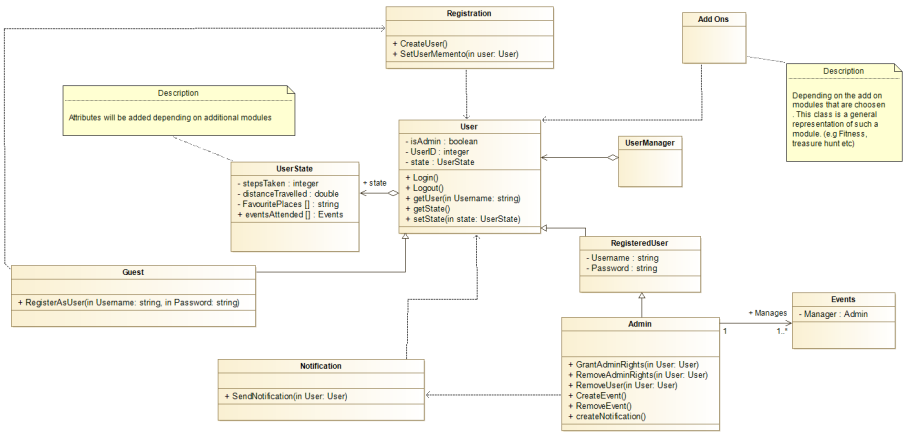

Figure 2: Class Diagram for User Management System

3.2 Use Case Diagram

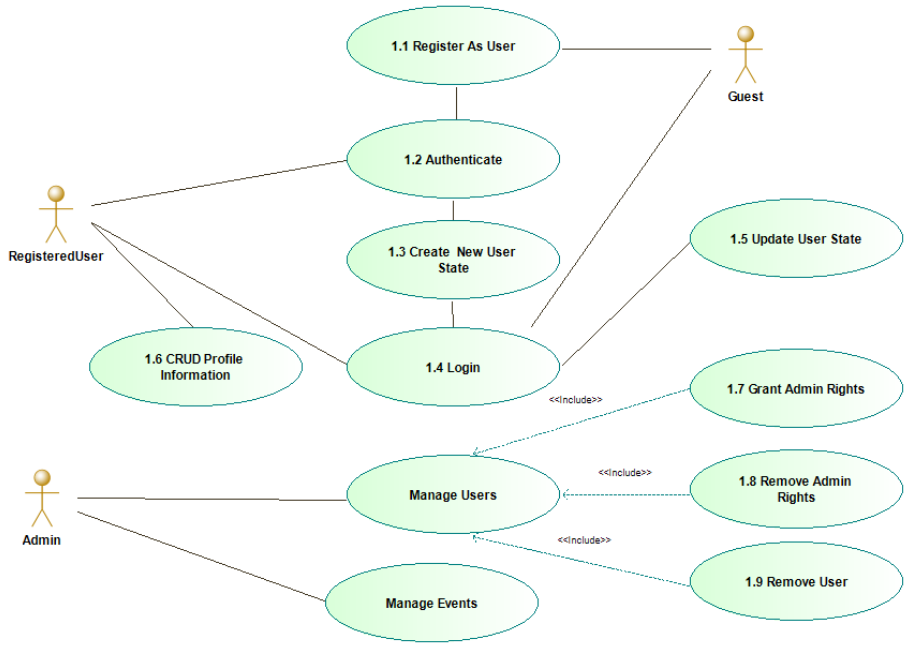

Figure 3: Use Case diagram for User Management System

3.3 Architectural Design of Module

The user management module will handle the login and registration of various users, namely Admin Users, Registered Users, and Guest Users. Guest users will still be able login into and make use of the NavUP system without registering but no data will saved on the system for them. The system stores all registered users' details including those from other modules.

The user management system makes use of two major design patterns, Template and Memento.The inheritance between user and the types of users demonstrates the template design pattern. The registration class acts as the originator to create objects of type user in the system;the user acts as the memento storing a state object for each class; the user manager is the caretaker for the memento. This shows the how the memento pattern is implememtened.The UserState class is used to define the objects that will hold state data that is unique to each user, such as, steps taken, places visited, events attended and other attributes that will be brought foward by extra modules that are added to the system.

This module connects to the notifications module, events modules and other add-on modules. Notifiactions are dependant on the user/s to which they are to be sent and hence there is a dependency present between these modules.The events module requires an admin user to instantiate and manage the event. One admin user may manage many events, but an event may only be managed by one admin user.

3.4 Technologies Used

A MySQL database will be used to manage basic user activities such as registration, authentication, and user privileges. The database will store information regarding all users and how each user should be able to interact with the system. This database should interlink with the ios, android, and web platforms.

The admin user should be able to perform basic CRUD operations such as adding and removing users, creating events, and updating details regarding points of interest.

Server scripting such as PHP will be required to facilitate interaction between the front end web interface and the database.

3.5 Non-Trivial Implementation Tasks

There are two types of users who are going to make use of the navUP system: Guest users and registered users. The "registered users" category includes administrative users, students, and also staff. All these users will ultimately have access to the system but be granted different privileges.

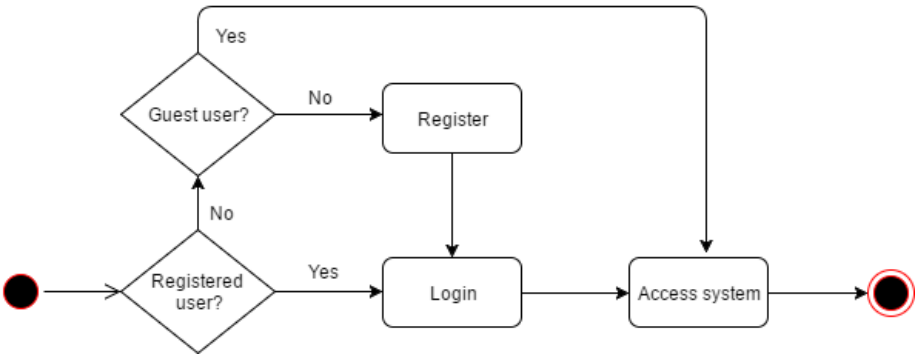

Figure 4: Non-trivial diagram

4 Notifications Module

4.1 UML Class Diagram

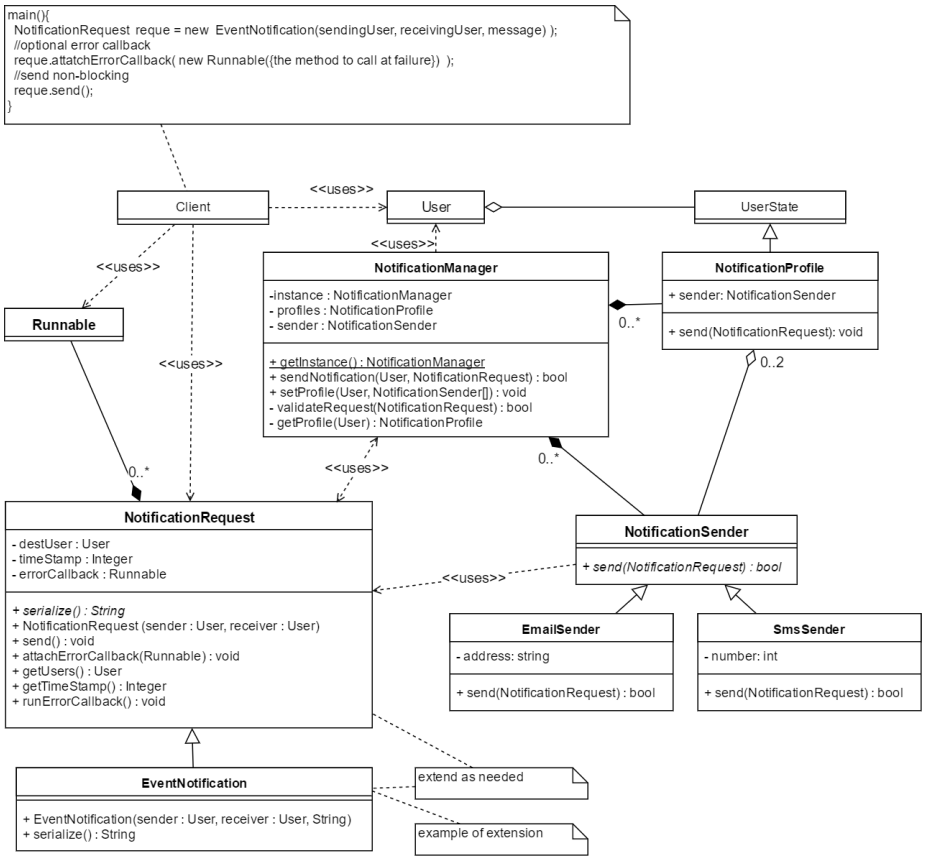

Figure 5: Class Diagram for Notifications Module

4.2 Use Case Diagram

Request and Receive Registered users will be able to receive notifications such as system updates and add on specific notifications. Admin users will be able to broadcast notifications, and add on systems will be able to notify users of their specific updates. To transmit a message, the request must first be validated. Validation includes request and system integrity checks, and validation of sending users. Only users that have admin rights can send messages. If an add on subsystem requires the service, it must first allocate a proxy admin user dedicated to said add on system.

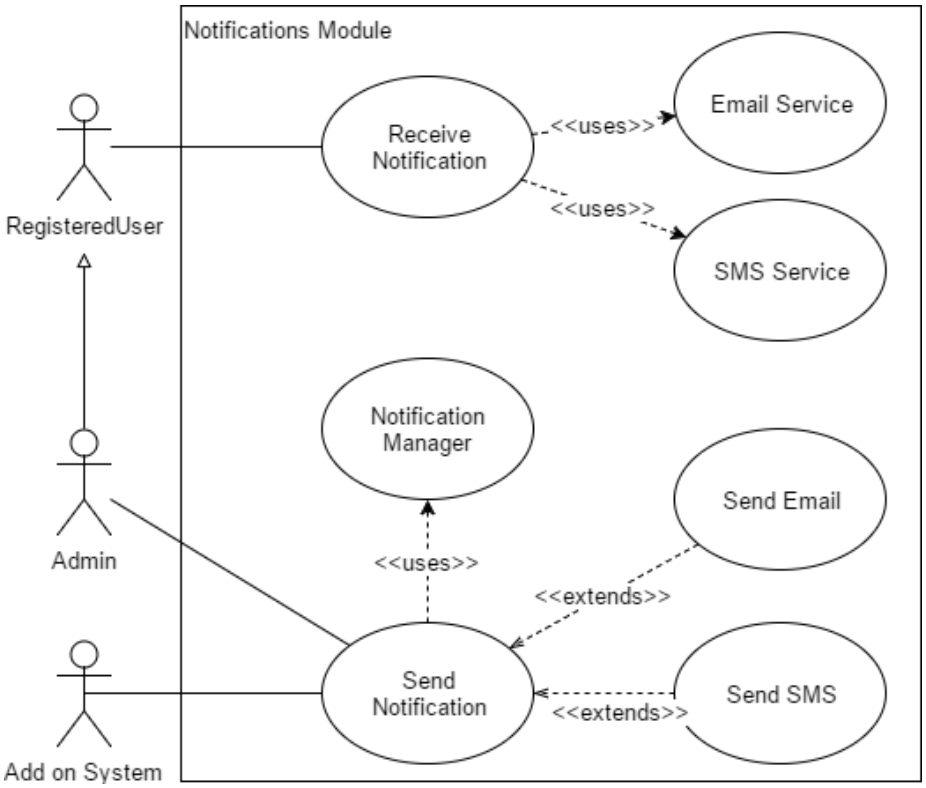

Figure 6: Notifications Request and Receive Use Case

Notification Profile Registered users will be able to chose their means of communication and customize their notification profile.

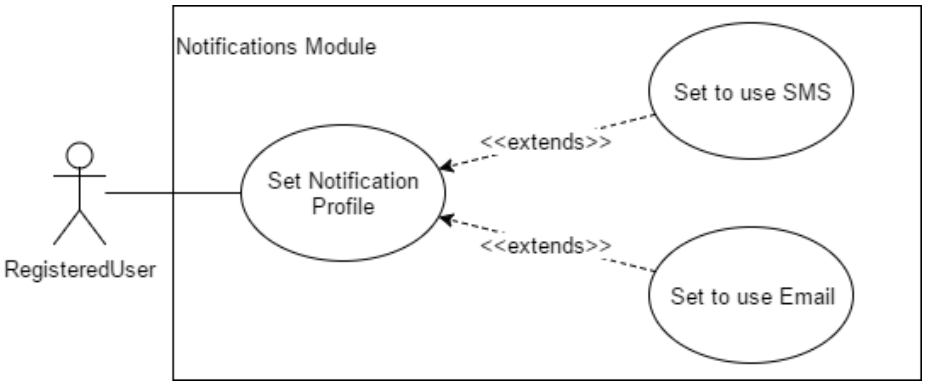

Figure 7: Notification Profile Use Case

4.3 Notification Module Architectural Design

The Notification Module is a core module that is responsible for delivering messages to system users via said users' preferred method of communication. Messages might include add on system updates that a user indicated they want to receive, or any other system related updates. Notifications will be sent via a medium external to the application, using ether SMS or email services. The Notification Module shall be implemented in such a way that it can handle messages concurrently, and clients requesting the service shall not be blocked until the request has executed.

The Notification Module will be comprised of a manager class i.e. NotificationManager, a request class i.e. NotificationRequest, a class related to the users' profile, i.e. NotificationProfile, and a class that will handle sending of the notification i.e. NotificationSender.

4.3.1 Classes

NotificationManager NotificationManager will maintain and store user profiles related to notifications, validate requests, and invoke the apropriate transmission service.

NotificationRequest NotificationRequest is a abstract class that will contain the message to transmit, the serialization thereof, and the target user. An optional runnable callback object can be attached that will be invoked in the event of failure of any kind. this will allow the operation of sending a request to be non-blocking, while providing a means of error tracing and correcting. NotificationRequest is the only class that acts as an interface of the Notifications Module.

NotificationProfile NotificationProfile will store system uses' preferred means of communication, i.e. SMS and/or email.

NotificationSender NotificationSender will implement the relevant technical requirements to use the underling service type, e.g. SMS service. A modular system is used that will simplify addition of additional transmission services. If such service is complex, a bridge pattern can be implemented with little difficulty.

4.3.2 Design Patterns

Singleton, Command, and Strategy design patterns must be used to implement the Notification Module.

Singleton The Singleton pattern will provide a well known entry point to access and make use of the service. This is well suited, as only one manager class for the Notification Module must be instantiated.

Command A modified version of the Command pattern must be implemented, where systems requesting to send a notification must implement their own concrete NotificationRequest class. The abstract method to be implemented is the serialization of the request object. This will decouple the systems that intend to send notifications from the Notification Module, and allow flexibility to implement add on specific notifications.

Strategy Two methods to transmit the notification must be implemented i.e. emails and SMS. The appropriate method (i.e. the method dictated by the user's profile) must be used to convey the notification. The Strategy pattern will allow the profile specific method to be used when transmitting the notification, while providing the implementation that is dependent on the underlying transmission system.

4.4 Technologies Used

Email Authenticated SMTP must be used to send messages to the email server that NavUp will use. This is to prevent the email server from relaying messages that are not properly validated. Internet Message Access Protocol (IMAP) will be used to store emails, as this is most prevalent system and thus support aught to be sufficient. POP3 might be an option as emails need not be stored for extended periods after the email has been delivered. TCP/IP will be used implicitly.

SMS A Direct-to-SMS gateway will be the most appropriate service to use, encoding all messages in plain ASCII text to promote device compatibility. The infrastructure such as GSM is determined by the service provider.

4.5 Non-Trivial Implementation Tasks

Notification Request Procedure The requesting client will have a narrow interface to interact with the Notifications Module. A concrete NotificationRequest class shall be instantiated and loaded with the appropriate parameters, such as the sending user, the receiving user, and the intended notification. An optional runnable callback object can be attached that will be invoked in the event of failure to deliver the notification. After the request setup, send is called, and the entire process is initiated. The receiving user's profile is acquired by the manager, and the appropriate external transmission services are invoked to send the serialized notification.

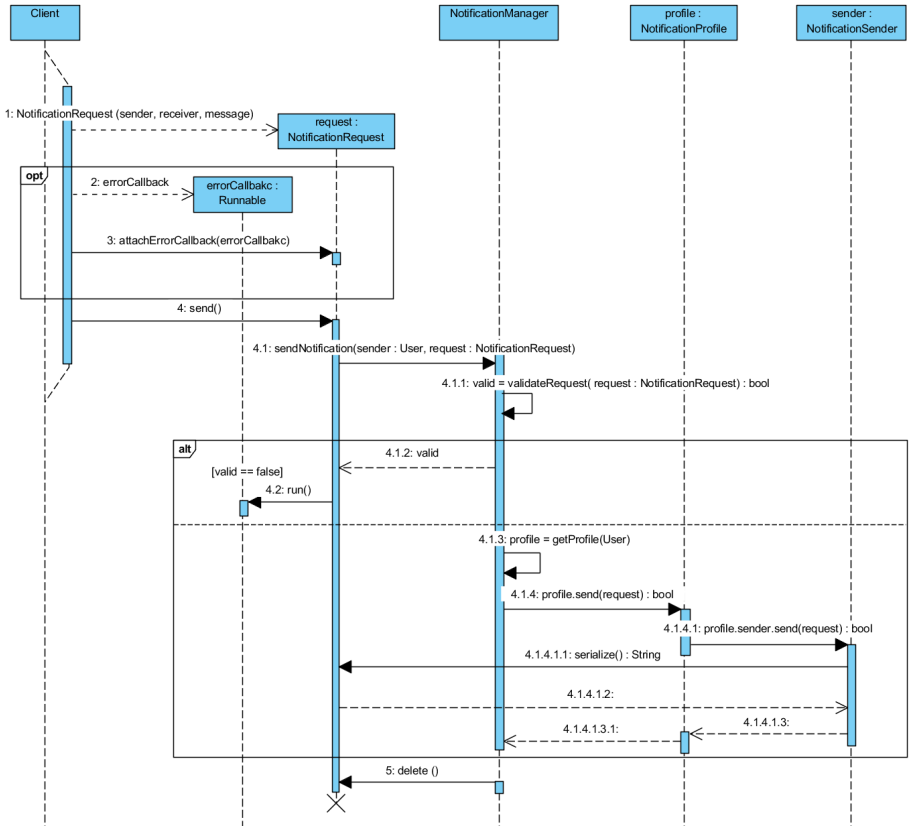

Figure 8: Notification Request Procedure

Notification Request There are multiple situations where a notification can fail to be delivered. Notice how errors are handled by a single error callback.

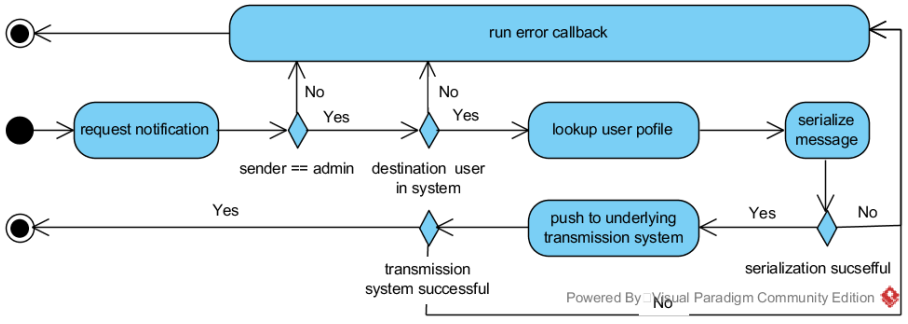

Figure 9: Activity Diagram for Notifications Module, Request Notification

5 Points of Interest Module

5.1 UML Class Diagram

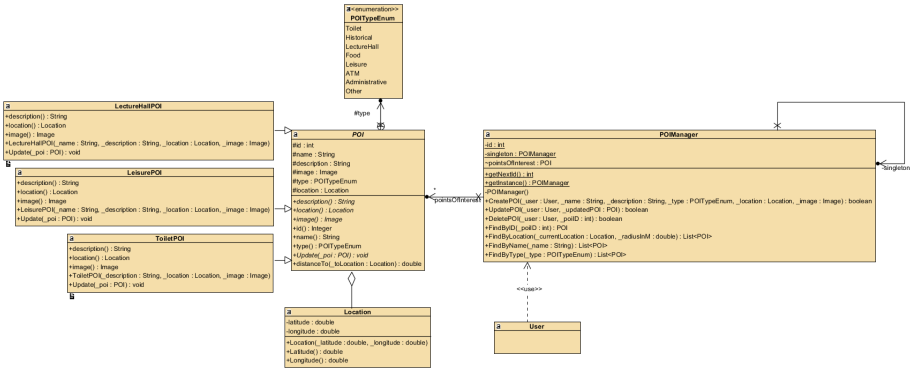

Figure 10: Class Diagram for the User management module

5.2 Use Case Diagram

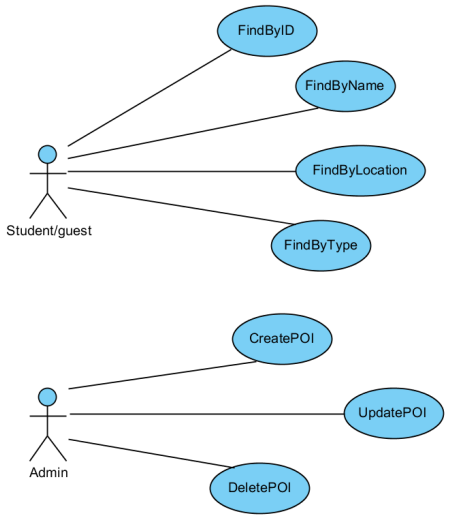

Figure 11: Use Case Diagram for the User management module

5.3 Architectural Design of Module

The Points of Interest module will be used to persist and deliver data regarding locations that a user might find of interest. It will interact with the Events

module, as well as the GIS module in order to retrieve and save it geographical location.

One can assume that different types of Points of Interest(POI) will have different fields, constraints and requirements, therefore the POI module will make use of the factory design pattern, in order to create objects without exposing the creation logic to the client and allow us to refer to the newly created object using a common interface.

The Points of Interest Manager class will make use of the Singleton design pattern to ensure that only one instance of the class exists, it will also act as the factory class to the points of interest class.

At the core a Point of Interest object will have a location, description, unique id, type, name and image. The concrete object will then be able to impose additional attributes and constraints, such as Gender for bathrooms for example.

Finally, the abstract point of interest class will be equipped with a stringify function to get a JSON representation of the object.

5.4 Technologies Used

It can be assumed that different types of Points of Interest will have different fields, constraints and requirements, therefore a traditional relational database will not be suitable for this type of data. Hence we will rather go for a document-oriented database like MongoDB, as the database server.

The Points of Interest CRUD functions will be exposed as Simple Object Access Protocol (SOAP) web services, using ASP.NET core and Entity framework carrying a JSON payload.

The logic of the module will be written in .NET.

5.5 Non-Trivial Implementation Tasks

The point of interest module will be called in two ways. Firstly, by administrative user who would be able to add, remove and update points of interest around campus. Secondly, students and guests will be able to use it, by searching for specific locations by name, type of location, specific areas etc. Lastly the points of interest classes SearchByLocation function will be called by the navigation module, as the user navigates campus.

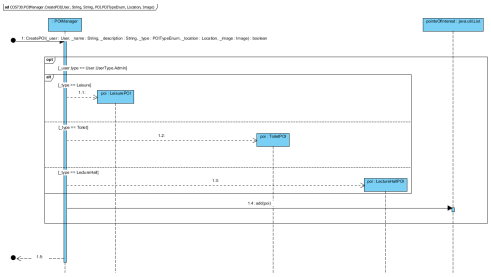

Figure 12: Sequence Diagram for the CreatePOI function

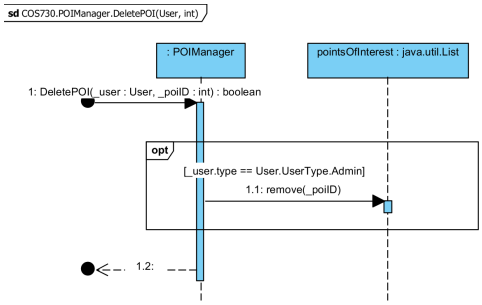

Figure 13: Sequence Diagram for the DeletePOI function

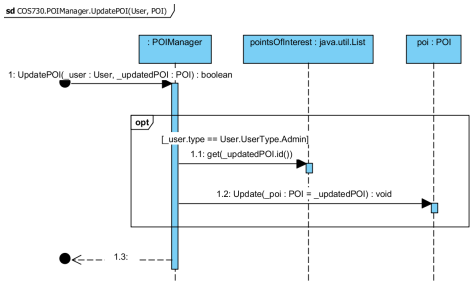

Figure 14: Sequence Diagram for the UpdatePOI function

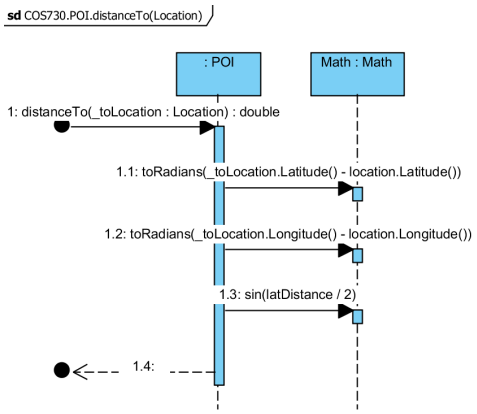

Figure 15: Sequence Diagram for the DistanceTo function

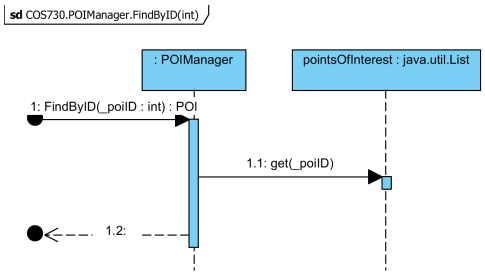

Figure 16: Sequence Diagram for the FindByID function

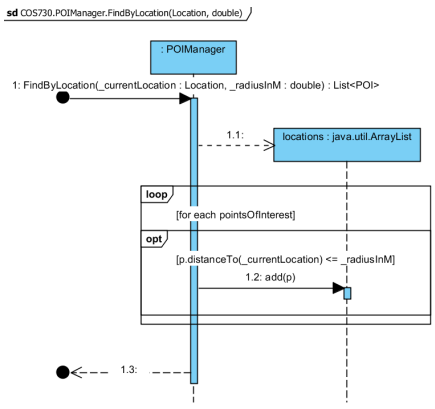

Figure 17: Sequence Diagram for the FindByLocation function

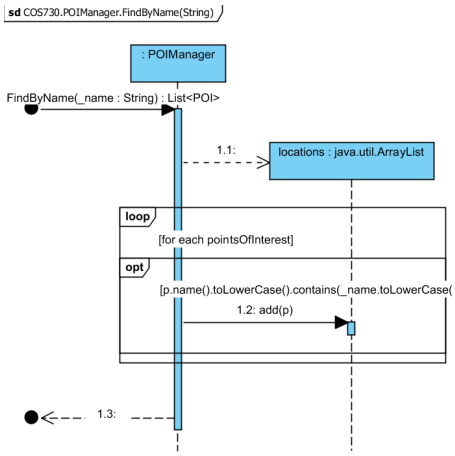

Figure 18: Sequence Diagram for the FindByName function

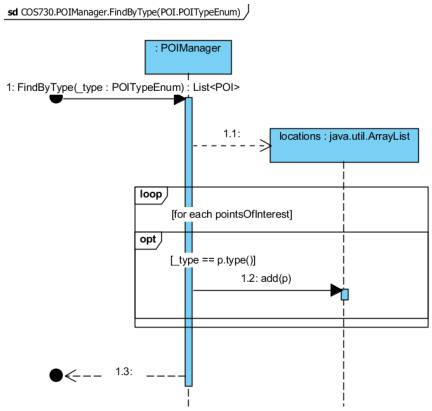

Figure 19: Sequence Diagram for the FindByType function

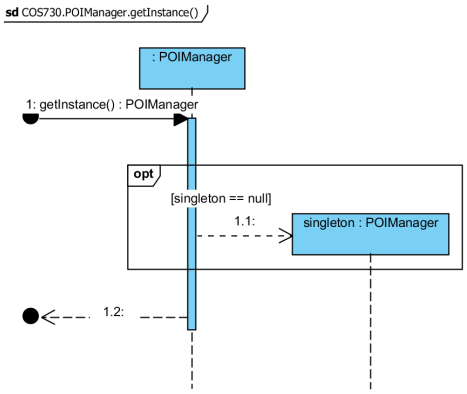

Figure 20: Sequence Diagram for the getInstance function

6 Events Module

6.1 UML Class Diagram

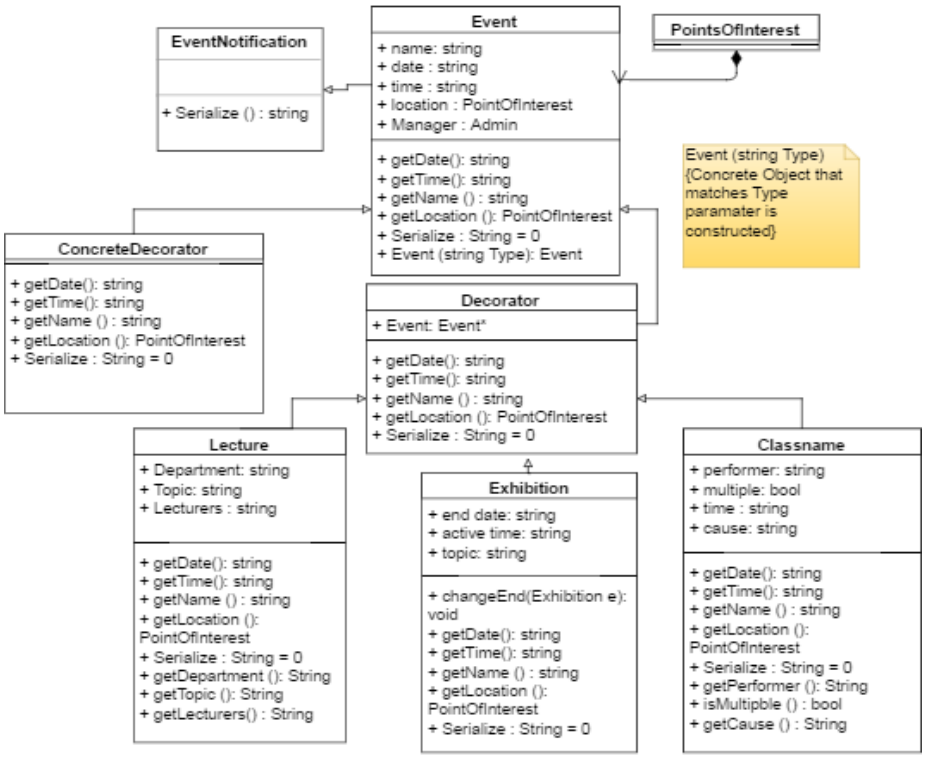

Figure 21: Use Case Diagram for the Events module

6.2 Use Case Diagram

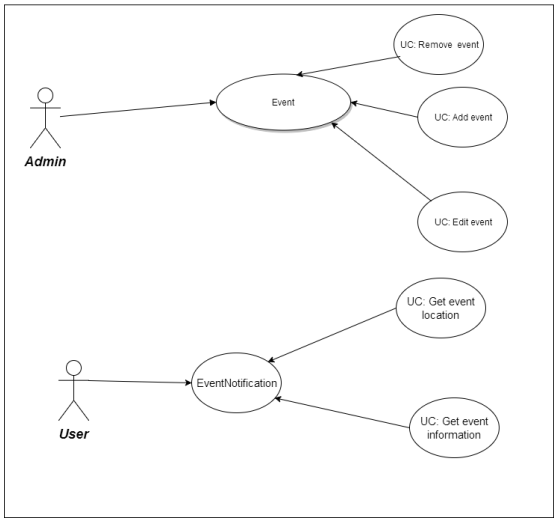

Figure 22: Use Case Diagram for the Events module

6.3 Architectural Design of Module

The events module will be able to inform different users about events on campus. The module will make use of the location services of the navigation system, thus when a user passes a particular point of interest which has public events, the user will receive a notification via sms or email about these events.

The NavUP system will make use of a variety of design patterns. In the events module there will be an events superclass which abstracts the core fuctionality and members that an event object will provide. This means it will be possible to specify different events based on the type, i.e. a special lecture, exhibition, performance etc. In order to add functionality to these different types of events, the decorator design pattern will be used. A concrete decorator will be define to implemnt core funtionality defined by the abstarct event class, and the decorator class will hold a reference to an event object which to which functionality can be added dynamically.

A single event object will have 4 different attributes namely a name for the event, a date, a time, and a location which is a Point of Intereste object. Different subcalsses will have addiotional attributes, such as a special lecture having a lecturers attribute, performances having performers information, and exhibitions having a topic description. The decorator pattern will allow us to add additional functionalities to each subclass based on the additional attributes which enhances specificity.

Each concrete event class will also be equiped with a Serialization function which will return the string containing information about the event which will be used in the notifaction that is sent to the user, which in the end is the goal of the module.

6.4 Technologies Used

For the CRUD operations, on the system, by the administrator, the Mongo database would be used. This is too ensure that events will appear instantly and be viewed fast enough.

The information entered by the administrator when giving other users administrative privileges will be stored on a MySQL database. This information should be easily accessible to the administrator.

6.5 Non-Trivial Implementation Tasks

The events module will be accessed when the user is using the NavUP system to navigate the UP campus. The system will continuously get the current location of the student and when a student passes a point of interest the system will check for events at the particular location. These events will then be compiled into a notification that the user will receive either as an SMS, email, or both. Once a notification has been sent the system will return to tracking the users location if the user is still using the system for navigation.

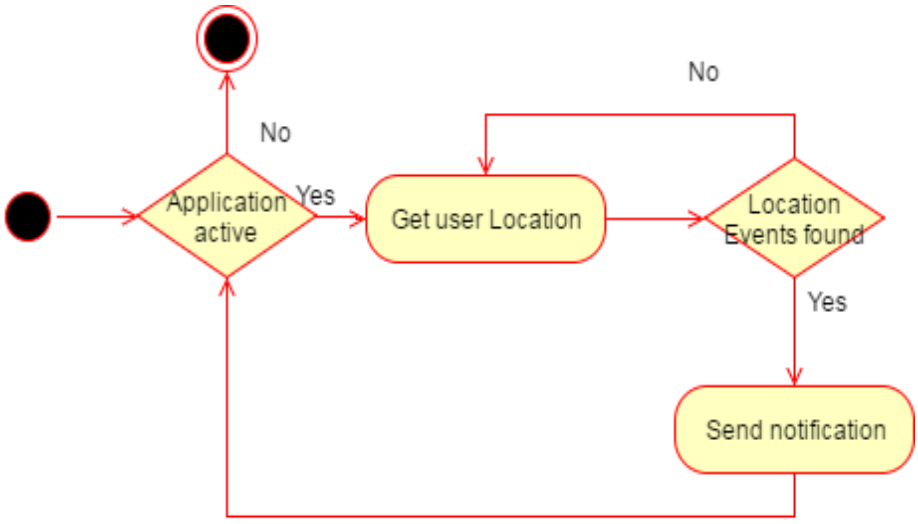

Figure 23: A