Software Requirements Specification
COS-301
The Inevitables

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Drew Langley	11039753
Lyle Nel	29562695
Dawie Pritchard	13104340
Peter Rayner	14001757
Hendrik Jan van der Merwe	15101283

Clients: Morkel Theunissen and Maria Ramaahlo

23 May 2017

Contents

1 Architecture Requirements

1.1 Quality Requirements

To create a safe working system that will meet all the required specifications of the NavUP project, specific system requirements need to be looked at in depth. The following are the system requirements that need to be considered. Security, reliability, efficiency, maintainability and usability.

1.1.1 Security

Security is concerned with unauthorized access to specific software functions. An important security measure that needs to be satisfied by the NavUP system is the confidentiality of different user level usernames and passwords. Personal usernames will be set by each user, or their student/staff number will be used to log in. Each user will be required to set their own password. Passwords will be required to be more than 7 characters in length to decrease the likely hood of potential hackers hacking passwords successfully. In the case of a forgotten password, a users password may be reset via a link and an email with a One Time Password. If a user has entered the incorrect password more than 3 times, the account should be blocked and the user should be notified for a password reset. Passwords and any related personal information will be encrypted and safely stored in a 'Users' database.

Security will be put in place to allow only that user or other users whom the location is shared with to see the pinned location. As different levels of user exist each user will have a specific degree of authority. Admin users will be able to add and remove locations, venues and more permanent locations that will effect the map of the NavUP database. Admin users will have the power to remove lower priority users unwanted actions.

1.1.2 Reliability

Concerned with the level of risk and chance of application failure. To increase reliability, downtime needs to be reduced and prevented as well as application errors affecting users. Users will need to be navigated to specific locations for classes, practicals and crucial meetings. User authority needs to be kept in place, not allowing users access to prohibited functions allowing them to make unwanted changes to the NavUP system. The NavUP system will need the ability to recover from a failed state, bring back the system to full operation. Finally the NavUP application should withstand any environmental threats that have the potential to cause system failure.

1.1.3 Efficiency

The two primary sub characteristics of efficiency are broken down into, time behavior and resource behavior. The performance of the NavUP application will be a reflection of how efficient the resources have been used within the application. This will have an effect on the battery life of mobile devices as they are engineered to have a longer life with less processing being done. By making use of an N-Tier Architecture. We will develop a suite of API's that will be independent and usable software service

1.1.4 Maintainability

Since the use of an N-Tier architecture. The back end will be completelely loosely coupled from the front end, The front end will only be able to make calls exposed by the backend in the form of an HTTP request. Allowing for multiple changes on the database or logical code without affecting how the application will perform or run. This allows for easy maintainability of the system.

1.1.5 Usability

The usability is of utmost importance, as we will have a variety of users making use of the NavUP application. Users will range from students at the university of pretoria as well as guests attending campus for the first time. The system will also be usable for disabled students including visually impaired students.

1.2 Performance Requirements

These Requirements outline what NavUP should be able to maintain during the course of its lifetime in terms of the performance of the application.

1.2.1 Response Time

The system warrants that even if over 40 000 students are connected. The performance of the application will not fall under 10 seconds of waiting a response from the application on an Android device. This also includes the infrastructure of the application, as well as front - ends. The response time will be measured using filters within Java EE.

1.2.2 Workload

Since the system mostly transfers information in real time , it would need to be exposed by certain API's, Android devices do cater for such api's, The system would be able to run over 40 000 students at once since the actual transfer will happen on the users phone.

1.2.3 Scalability

The application will be capable of supporting at least 40 000 users each day. As long as the application is implemented in such a way as to support large amounts of users.

1.2.4 Platform Considerations

The application will make use of Android .

1.3 Software System Attributes

- **Security:**
  Users or guests are not allowed to alter fields in the database. These calls will only be available in the back end and will not be exposed to the front end.
- **Flexibility:**
  The architecture is designed in such a way that any layer may be altered without changing any other layer. Thus to adapt those layers becomes increasingly easy.
- **Maintainability:**
  By the use of a layered architecture the system will remain easy to maintain since we will be using common tested methods to make it easy to fix bugs, and improve the q system over time due to the loosely coupled nature of a layered architecture.
- **Availability:**
  If one layer fails, since a layered architecture is loosely coupled. The application will be available most of the time. Since a failure on one layer will not completely break the database or application in any way. This is because the database sits remotely and any change in back back end code would need a redeployment of the application and such deployment will only be made if the changes made are assured of no failure.
- **Efficiency:**
  The database will be accessed accordingly by setting up a JDBC connection and using a technology such as hibernate to do the actual persisting of information to the database. These hibernate calls will expose a RESTful endpoint where the front end will be able to make HTTP calls to retrieve the JSON objects.
- **Resilience:**
  The application will handle heavy loads due to the fast processing nature of a layered- architecture.
- **Modularity**
  The application is broken up into layers, each layer will be worked on seperately.

2 Architecture Constraints

Constraints are a result of design decisions that are fixed and may not be altered. Constraints may be intentional or unintentional often provided by stakeholders committed toward the project.

2.1 Design Constraints

- Minimal Dependency: The use of dependency injection to make sure modules are as loosely coupled as possible, In a layered architecture. Making sure layers are also loosely coupled.
- Well Defined Input: Input will exposed by an HTTP POST request that is exposed by the backend. Thus making sure that the front end will easily be able to retrieve information..
- Abstraction: Make sure changes later wont affect the application as a whole, by abstracting the implementation details.

2.2 Programming Language Constraints

- Web Based Development

  We will be making us of Ionic and Cordova, in which Ionic builds on top of Cardova. By using Cardova, it will take care of packaging the HTML application as a native application that is able to run on Android

  Ionic provides the 'missing peice' as it provides a set of front-end components that allows us to write a HTML application that looks like a native application. These front-end components include (HTML/CSS/JavaScript and AngularJS).

- Server Side Development

  User information, information regarding locations and venues will be stored within a database. To run the database we will make use of PostgreSQL and AJAX, where validation will be done in the backend via java EE

- Mobile Application Development

  The mobile application will be built using ionic and cordova.

2.3 Operating System Constraints

The NavUP application needs to run on Android at the very least. Building software that fails to satisfy the platform constraint means we have failed to deliver what is expected by the external stakeholder.

2.4 Network Constraints

The NavUP application will run on the Universities network allowing only users within the firewall to make use of the application.

3 External Interface Requirements

These requirements outline the external interfaces needed in terms of hardware, software and the GUI for the system.

3.1 User Interfaces

The NavUP application will consist of different user interfaces, depending on the user's level of authority. Each level of UI will be appropriate with regards to the ease of use. The 'Guest' users with the lowest level of authority with have a basic, easy to use view. Guest users will be new to using the NavUP application and assistance will be prompted to guide the user through all the available features. The application UI will be so that new users are able to use it as well as to satisfy experienced users.

Admin userswith higher levels of authority will have a variety of added features added to their UI. The ability to add events, venues and locations would be satisfied by a more detailed UI. These users will also be able to make permanent changes to the NavUP database as well as make changes to lower level user's information.

Visually the UI of all users will cater for people with disabilities with a easy to read and understand layout.

The system will make use of of Ionic and Cordova, which is a cross-platform user interface that will support Windows, iOS and Android. The reasoning behind these technologies is because you can develop natively and on web at the same time which allows for faster and more productive development.

The application should make use of a GUI that is user friendly, such that guests that come to the university will be able to navigate through the application with ease. The layout of the application is important and there should be a clear flow when using it The application will ensure that it's menu button is always visible no matter where the user might be in the application. This menu button will appear on every screen. The application will use a standardised font that is easily readable.

3.2 Hardware Interfaces

The NavUP application will be a an Android compatible application. The twor hardware interfaces that will be used will be a mobile phone and a tablet. The software will be written in the most efficient way as possible to put as little stress on the CPU as possible allowing for longer battery life. The application does not write directly to the users device but instead to the NavUP database which is located on a network server. Users will have to be within the network firewall of the University of Pretoria to make use of the NavUP application. The user's device transfers and receives data from the server using a set of secure network protocols ensuring information is not compromised.

3.3 Software Interfaces

As mentioned the application will run on Android . On the server side, the servers can either make use of Windows,Linux or UNIX, but it is required to have PostgeSQL installed and configured. The users module will have different interfaces for each section of a layered architecture. The presentation layer will make use of ionic and cordova. The business layer will use java EE's REST API. The persistence layer will use JPA and hibernate and the database layer will make use of JDBC and that database will be done via PostgreSQL. All of these interfaces make use of Java EE as to make it easier for dependency injection.

3.4 Communications Interfaces

Data and information between users and servers will communicate through the network of the university.

3.5 Platform Considerations

The application will make use of Android

4 Architectural Patterns or Styles

We will be implementing the N-Tier architectural pattern.

4.1 Benefits

- Deployability

  Redeployment will not be a big issue via the use of wildfly, since wildfly automatically redeploys the application if any backend logic changes. If the deployment is unsuccessful then wildfly will revert to the previous working deployment. this improves time management and contributes to the overall flexibility of the system.

- Changeability

  Due to the loosely coupled nature of an N-Tier architecture. Technologies may be changed without affecting the overall system.

- Ability to Utilize Different Technologies

  Java may be used for event processing business logic due to multithreading properties of JVM. This allows programmers to make use of new technologies on services without disrupting of causing system failure.

- System Resilience

  Since the system is loosely coupled if an issue occurs at the backend, it will only affect that specific call made from the front end and will not affect the system as a whole.

4.2 Concerns

- Complexity

5 Architectural Patterns for Subsystems

5.1 User Subsystem

For the User module we will be using a layered architecture, allowing the use of dependency injection allowing for pluggable application architectures

5.1.1 Benefits

- Seperation of Concerns

  Seperates the business logic from the presentation, thus allowing you to work on seperate layers independently, allowing non-repeatability

- Flexibility

  Coping with growth or traffic will be easier to handle.

- Each layer can evolve independently

  Since each layer is independent from each other, each layer will be able to change without affecting any of the other layers.

- proven and stable protocols and design

  One of the most commonly used architectures means that protocols and designs means the system will be stable.

5.1.2 Concerns

There are many problems that can occur but the main one would be how to design the application so that it supports maintainability, extensibility, security and scalability.

5.1.3 Design of the Architecture

- Class Diagram

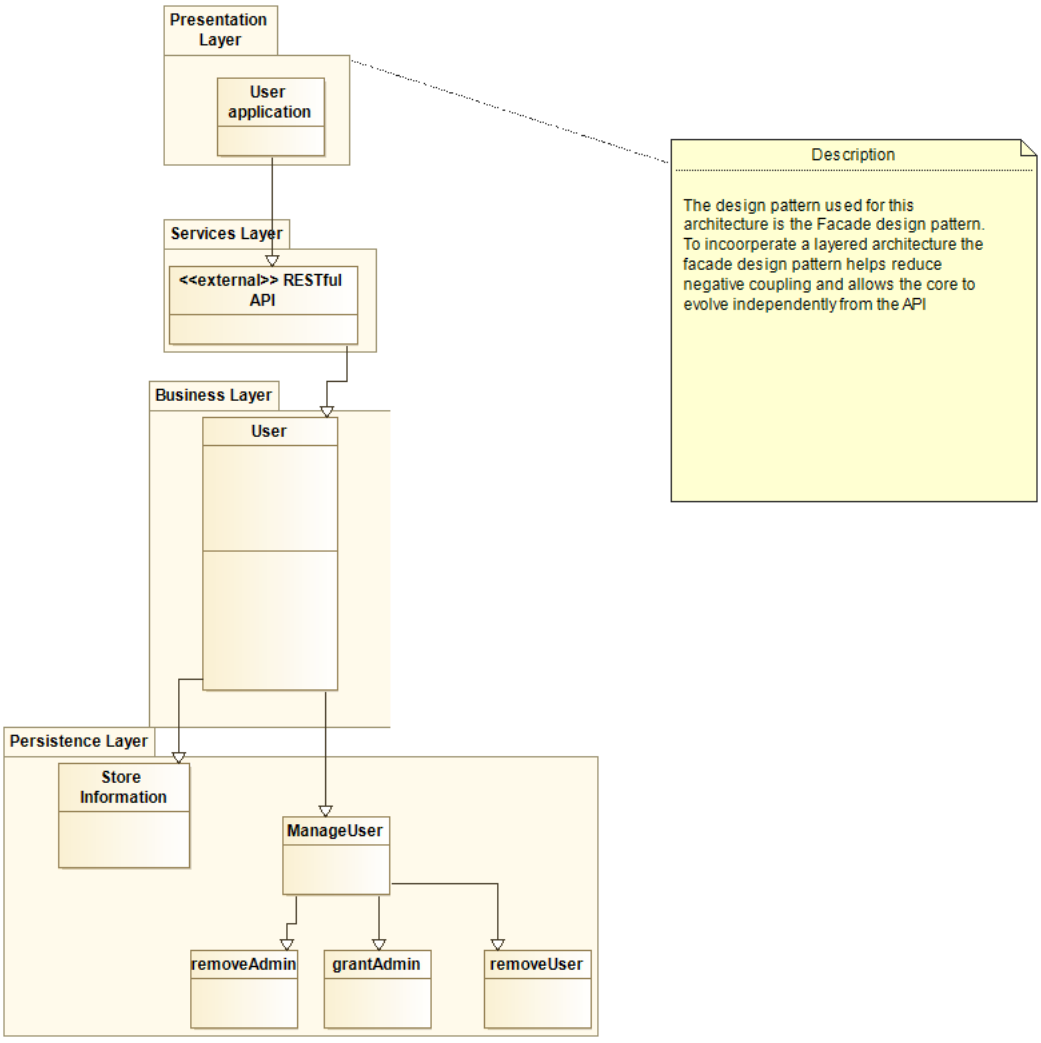

Figure 1: User Module Class Diagram

- Deployment Diagram

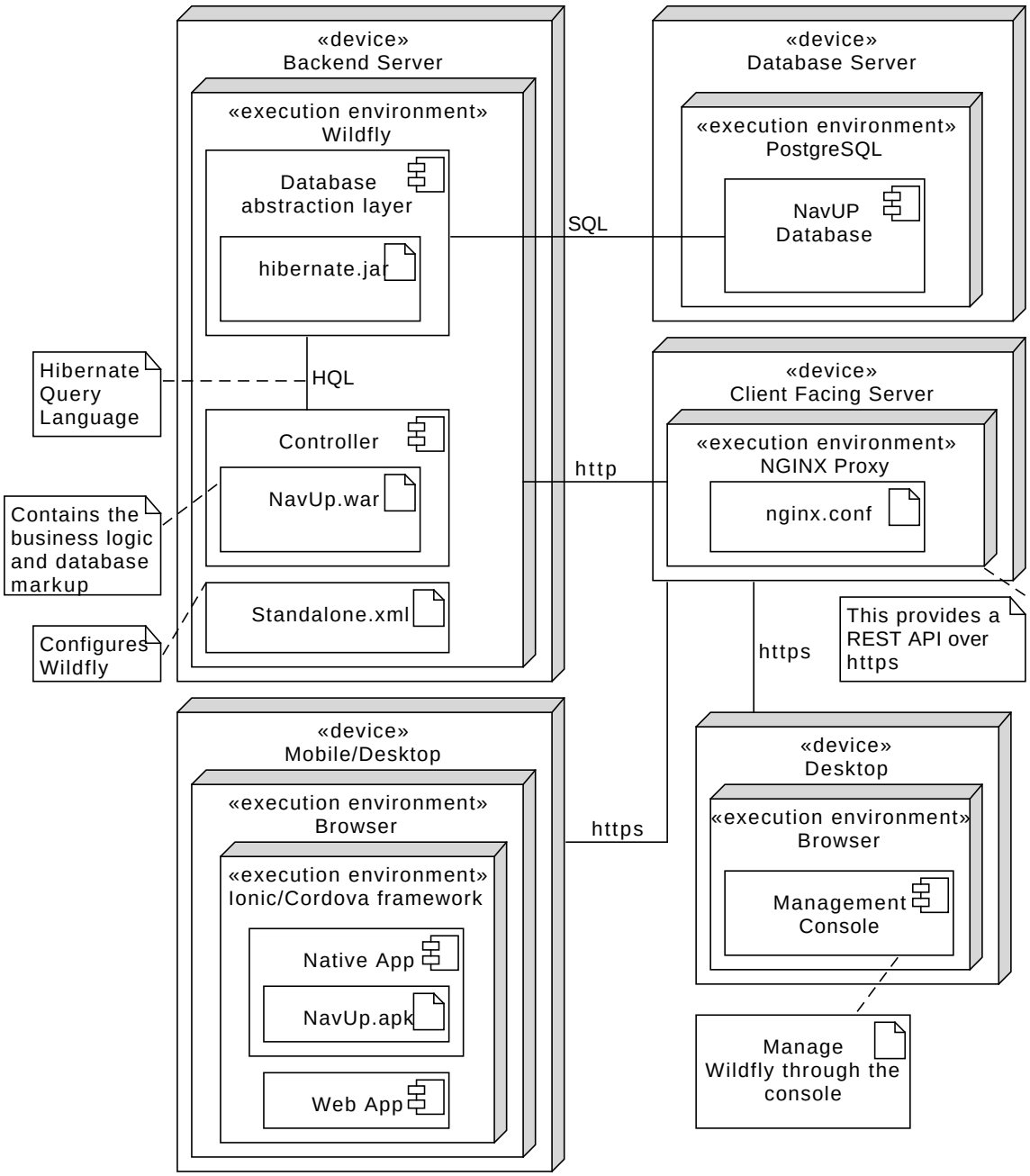

Figure 2: User Module Deployment Diagram

- Use Case Diagram

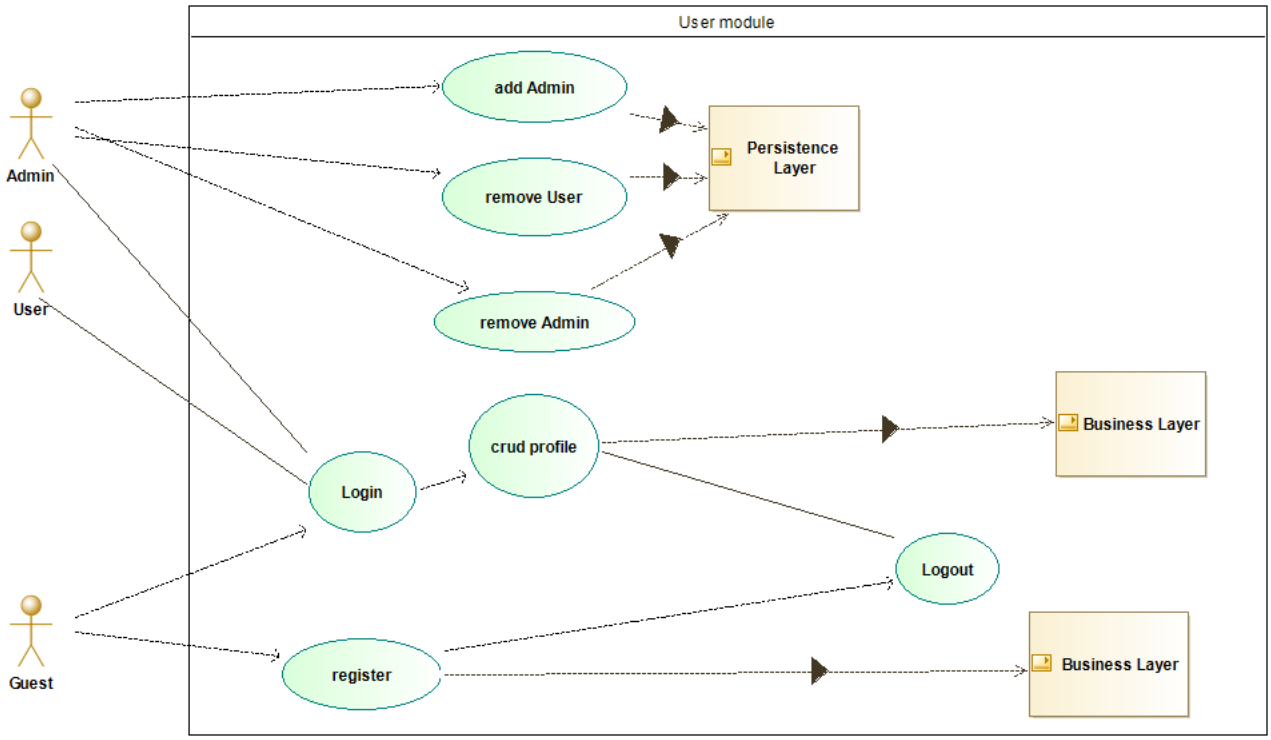

Figure 3: User Module Use Case Diagram

- Sequence Diagram

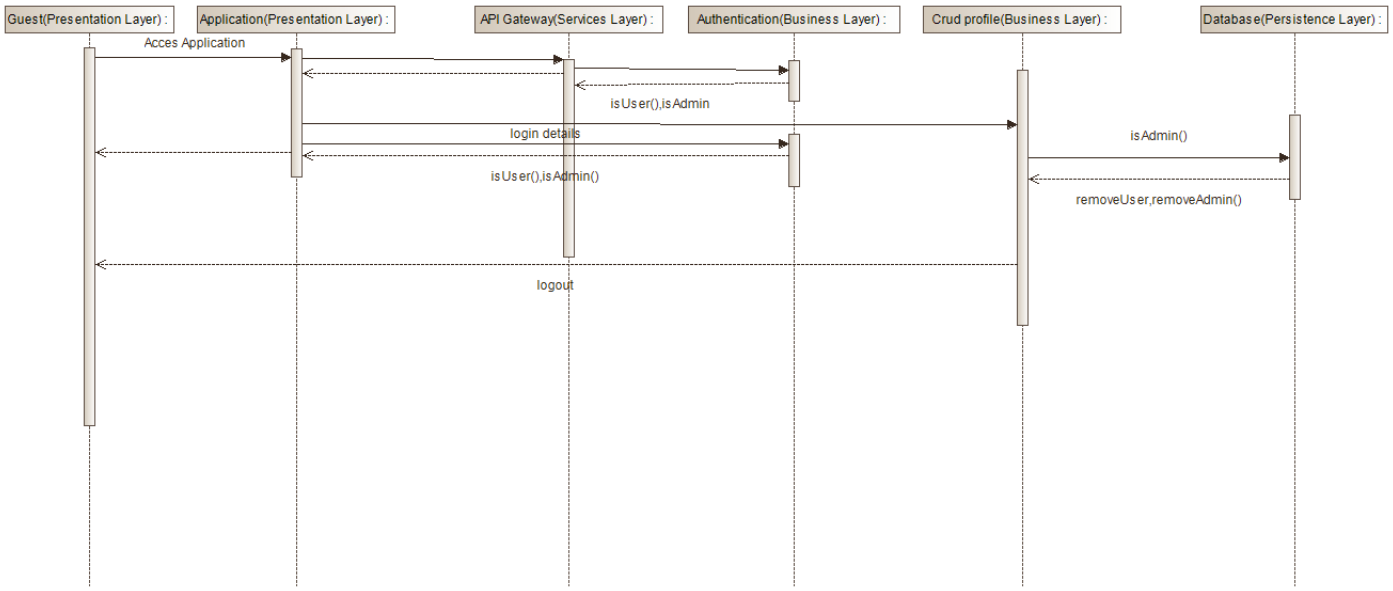

Figure 4: User Module Sequence Diagram

- Activity Diagram

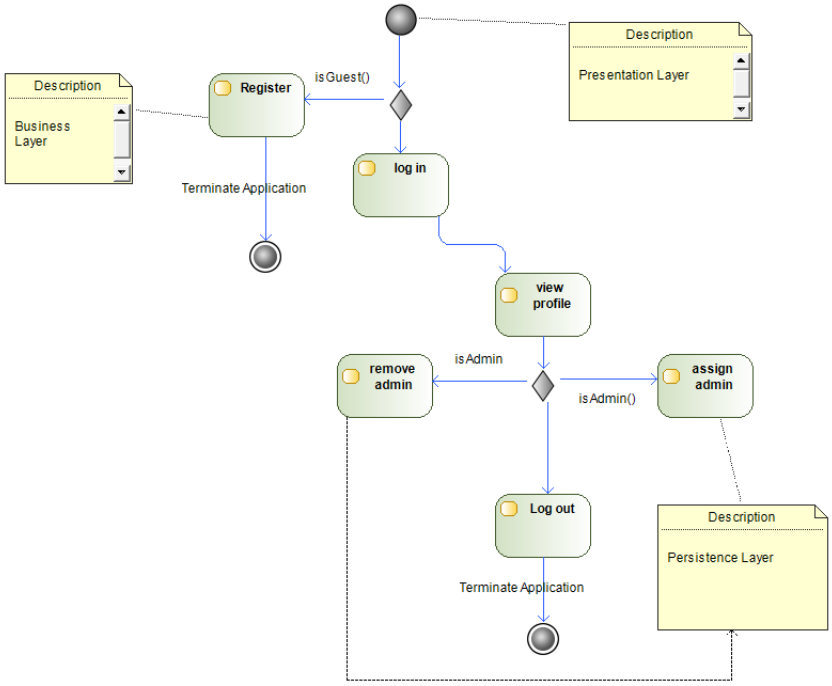

Figure 5: User Module Activity Diagram

- State Diagram

6 Technologies

6.1 Presentation Layer

For the Presentation layer we will use ionic and cordova. This is for native development which is essential in testing navUP

6.2 Business Layer

This is where we implement all the logical technologies that add the functionality to this module. We will use Java Enterprise Editions' REST API, this is for improved scalability since the client and server are loosely coupled.

6.3 Persistence Layer

For the Persistence layer we will use Java Persistence API (JPA), for accessing, managing and persisting data between data transfer objects. This allows us to focus more on the object model than the actual SQL since we will be using criteria queries ,this enables us to save time by generating/updating SQL queries and convert the results to our model classes. Java will be used in order to save locations and to get the current locations for Android devices.

6.4 Database Layer

The database layer will make use of POSTGRESQL. JDBC provides the connection to the database as well as hibernate, which provides a familiar Java interface which enables easier use of SQL via the use of criteria queries,

6.5 Service Layer

We will make use of the REST API for this module since its what we will be using for all our modules, this will allow us to easily communicate between modules as well as add additional functionality. We would also make use of ATOM for our maven built project.

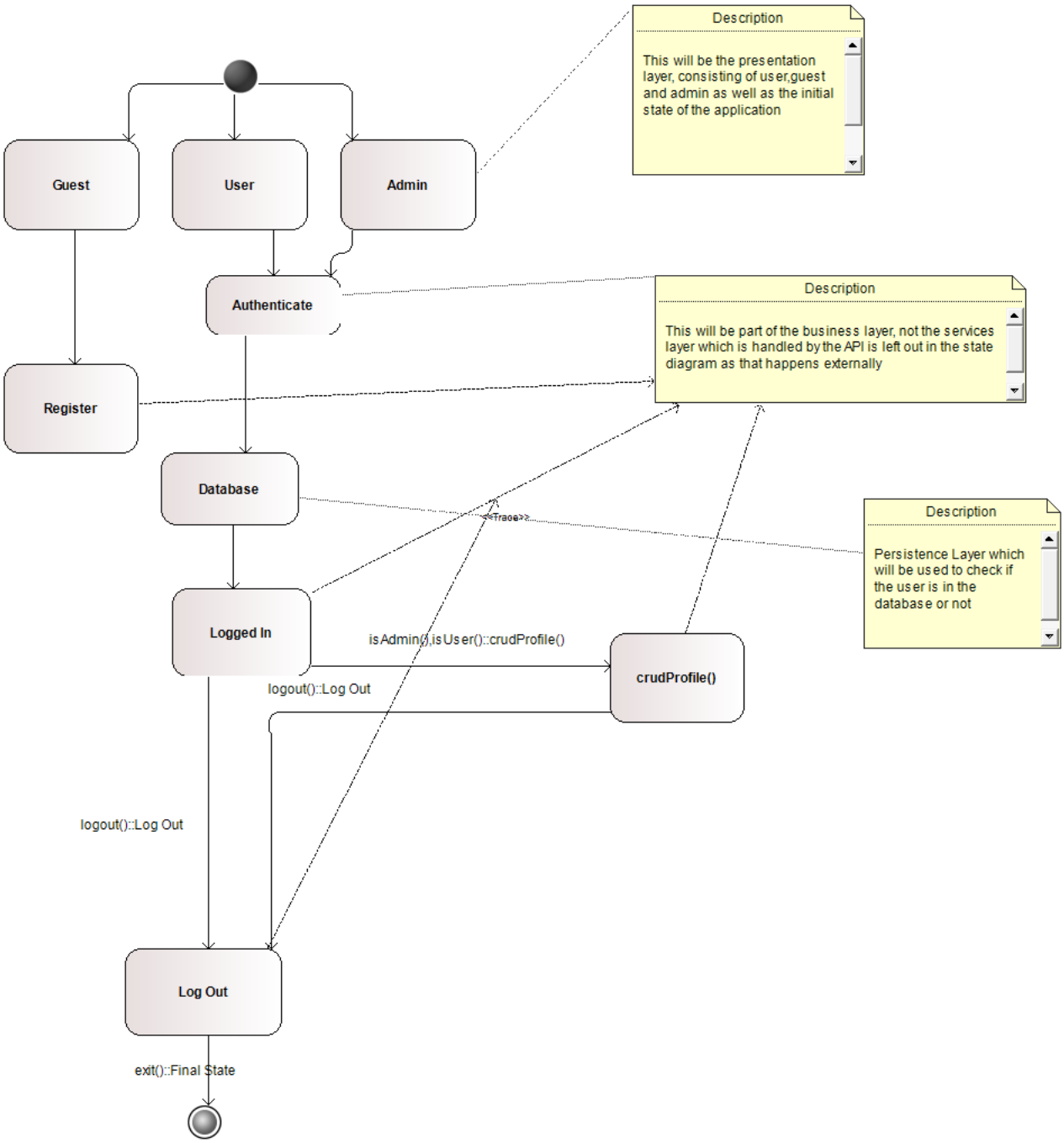

Figure 6: User Module State Diagram

Note: We are using apache maven which is a powerful project management tool this makes the use of dependencies far simpler since we will not be landing into any jar hell. It also simplifies the build process