UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science

Functional and Architectural Requirements

Client: Gavin Potgieter

Team: CodeBlox

Tshepo Malesela (Bsc: Computer Science)

Lethabo Mogase (Bsc: Computer Science)

Lorenzo Spazzoli (Bsc: Computer Science)

Bilal Muhammad (BIS: Multimedia)

Dirk de Klerk (BIS: Multimedia)

May 27, 2016

Contents

1 Introduction

The purpose of this document is to provide a detailed overview of the functional and architectural requirements with regard to the DropOff Project that was assigned to team CodeBlox. The document will be under constant revision, and will change as the project progresses.

It will provide the development team with a reference that can be used to develope from, thus ensuring that all functional and architectural requirements are met.

The document will also serve as means of communication between the client and development team. Thus any additional requirements from the clients' side can be appended to the document, and has to be adhered to from the development side.

2 Project Objectives

The **primary objective** of the project is to create an automated service system that will provide customers with the ability to grant delivery personnel access to a drop safe and/or demarcated area of their home. This will ensure that deliveries to be made can be completed without the need of someone being physically present on the premises.

Initially this will consist of the generation of a One Time Pin (OTP), that will be sent to the delivery personnel, granting them single access to the demarcated area. Once the delivery has been made. The user needs to be able to ensure that the residence is secure.

If time allows it. The system will be expanded to use audio and video communication to validate the identity of the delivery person.

In addition to the functionality of the system, it needs to be as **cost effective** as possible. This will allow a larger audience to gain access to the system.

The **secondary objective** (if time permits) is to make the system scalable. This will enable services to expand to other areas of the home. Thus allowing users an affordable solution to home automation.

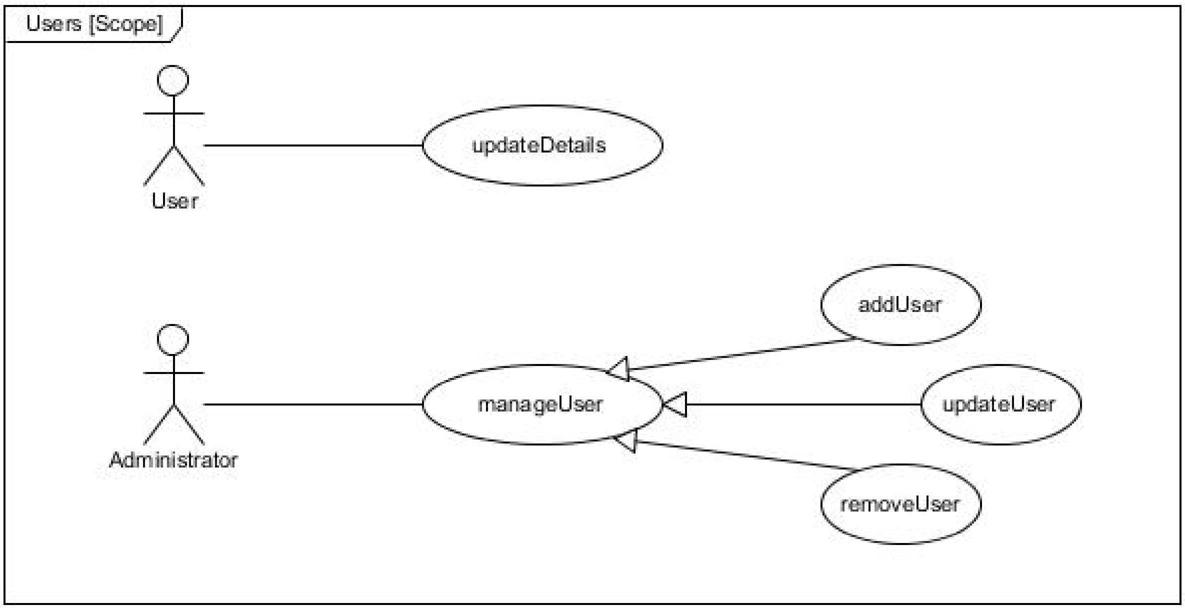
3 Functional Requirements

3.1 Users

The users module will be responsible for maintaining user information. Particularly it will specify what type of user it is. The module will also have the responsibility of storing demographic information about the users.

3.1.1 Scope

The scope of the Users module is shown below

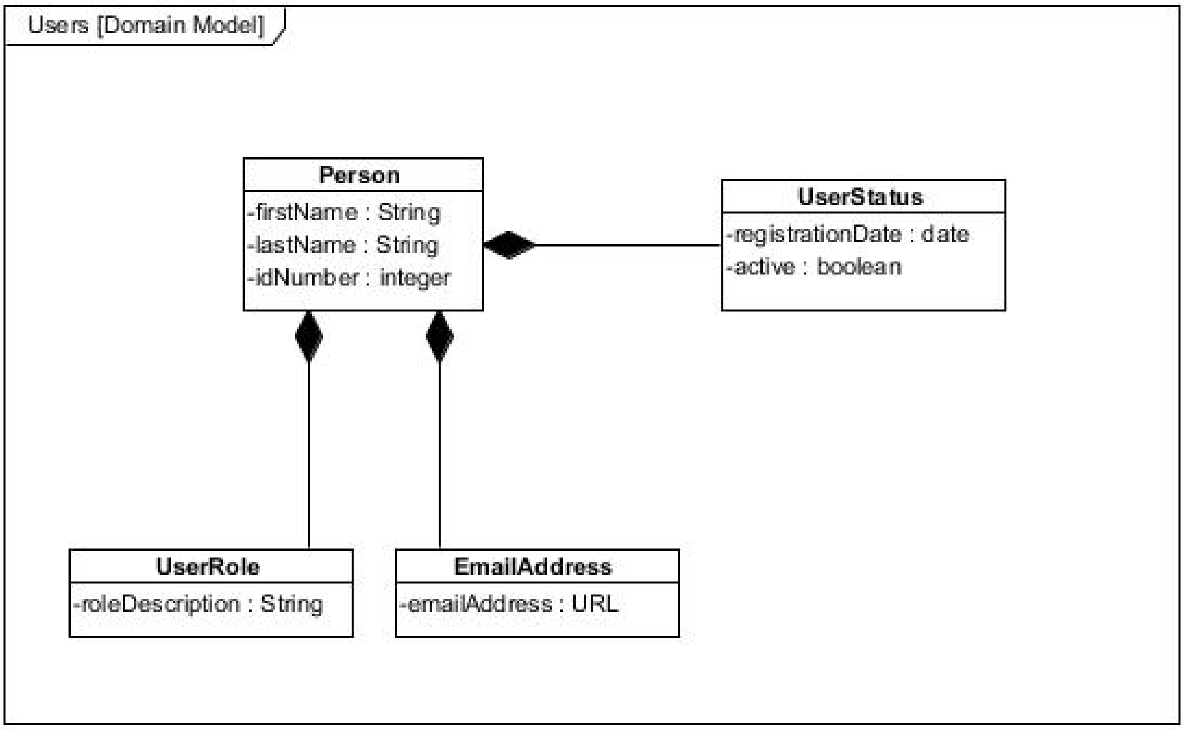


The scope of the Users module includes

- adding, removing, and modifying users on the system
- changing user information

3.1.2 Domain Model

The domain model for the Users module is shown below



Each user has a firstname and a lastname, as well as an id number. The person will also have an email address associated with them that will be used for login purposes. Each person will also be associated with a role that will indicate if it is an administrator or general user. Lastly a status will be designated to each user to determine if they are active or not.

3.1.3 Use Cases

- **addUser** - Allows an administrator to add a user to the system.

  preconditions:

  - user does not already exist on the system
  - creator possesses the role of administrator

  postconditions:

  - a user is created.
  - the user is added to the system.

- **removeUser** - Allows an administrator to remove a user to the system.

  preconditions:

  - the user exists on the system
  - remover possesses the role of administrator

  postconditions:

  - a user is removed.
  - the user is removed from the system.

- **updateUser** - Allows an administrator to modify a user on the system.

  preconditions:

  - the user exists on the system
  - modifier possesses the role of administrator

  postconditions:

  - a users information is modified.

- **updateDetails** - Allows a general user to update their information.

  preconditions:

  - the user exists on the system
  - the user has to correct credentials

  postconditions:

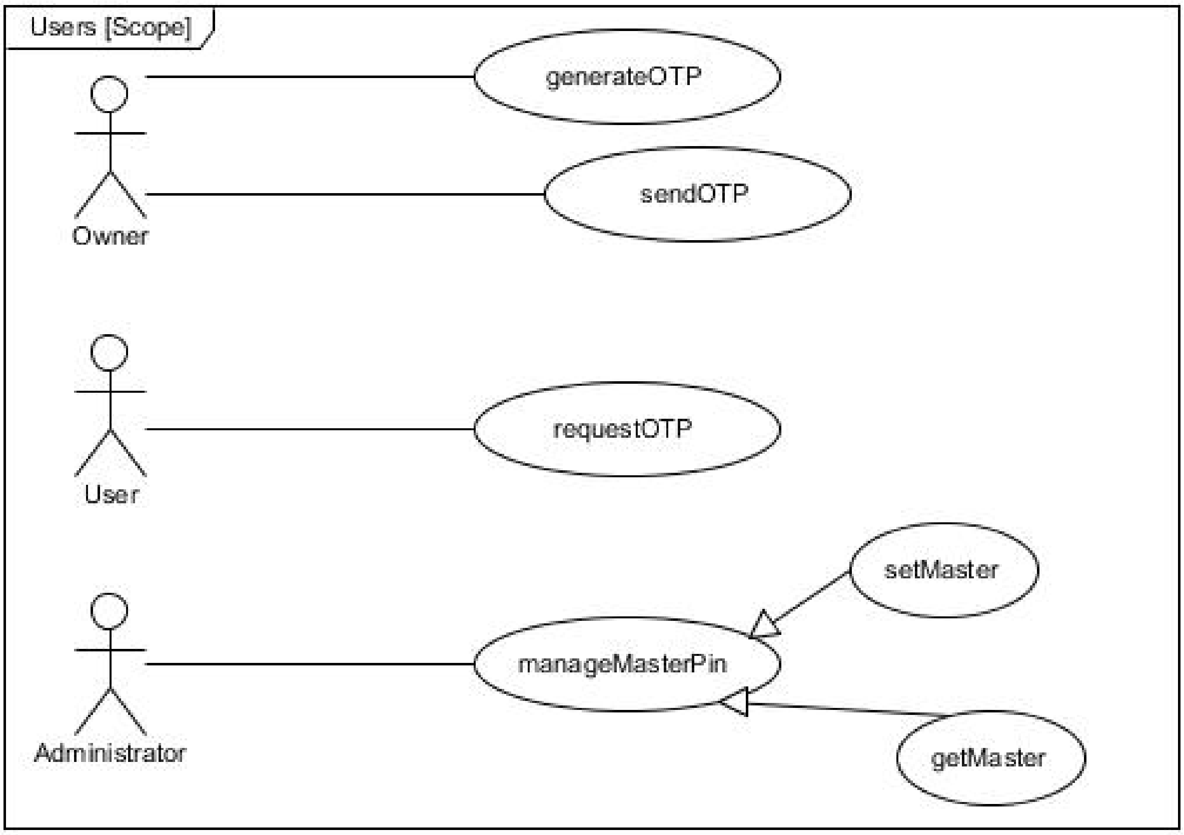
  - a users information is modified.

3.2 Controller

The controller module will be responsible for maintaining system status information as well as provide the various user with certain system functionalities depending on their authorization. Particularly it will specify the current status of the drop box/area and allow the user to change this status through specific functions. This module will also be responsible for generating the OTP.

3.2.1 Scope

The scope of the Controller module is shown below

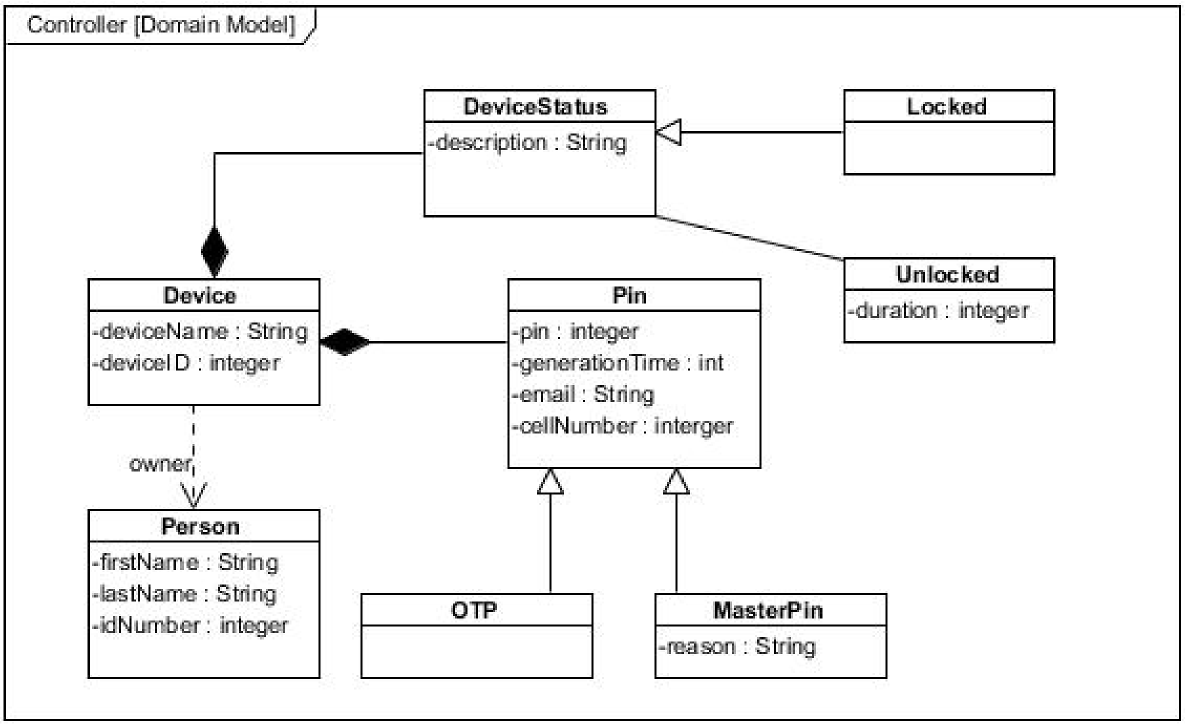


The scope of the Users module includes

- generating, and sending an OTP.
- setting and getting a master pin (To be used as a fail safe).
- Users i.e. delivery person may request a OTP to access designated area.

3.2.2 Domain Model

The domain model for the Controller module is shown below



The Device in this context will keep track of the status of a particular device connected to the system. Initially this device will refer to the drop safe or demarcated area within the house. Device will be a composite object consisting of a DeviceStatus and a Pin. The device will also be associated with a Person object which will be the owner of that particular device.

Person refers to the owner of the device as stated earlier. Only one owner will be associated with each device at a particular time. This will serve the purpose of authenticating the owner so that an OTP may generated and sent to the requesting user/delivery person.

DeviceStatus will indicate the current status of the device. This will be used as pre- and postconditions for the particular methods to be implemented. So far the device will possess to possible statuses: Locked or Unlocked. The **Unlocked** specialised class will have a textbfduration variable to set a time limit for when the system should lock up again after the demarcated area has been accessed.

Pin will serve as a template for both the OTP and MasterPin. It will provide users with a primary means of access to the system. The Pin class possesses **email** and

cellNumber variables that area the email and cell number of the delivery person to which a pin will be sent. Each Pin object will have a **generationTime** variable to indicate when a pin was generated. This will be used to set an expiry date for each pin as an extra precaution.

3.2.3 Use Cases

- **generateOTP** - Allows an Owner to generate an OTP once they receive a requestOTP request from a User/delivery person.

  preconditions:

  - The Owner received a requestOTP request from the delivery person.
  - The Owner has authorization on the system.

  postconditions:

  - An OTP is generated by the system.
  - The OTP is registered on the system.

- **sendOTP** - Allows the Owner to send an OTP to a specific delivery person.

  preconditions:

  - The OTP has been generated.
  - The delivery person has been authorised to receive the OTP.

  postconditions:

  - OTP is sent to the delivery person.

- **unlockDevice** - Allows the User/delivery person to unlock the demarcated area to make delivery.

  preconditions:

  - The User possesses a valid OTP that can be matched against the systems OTP.

  postconditions:

  - The System is unlocked and delivery can proceed.

- **requestOTP** - Allows the delivery person to request an OTP from the Owner of the residence.

  preconditions:

  - The system is locked.
  - A delivery has to be made by the delivery person.

  postconditions:

  - a requestOTP request is sent to the owner of the residence.

- **setMaster** - Allows the administrator to set a master pin for use as a fail safe.

  preconditions:

  - The user has administrative authorisation.

  postconditions:

  - a New master pin is generated and registered with the system.

- **getMaster** - Allows the administrator to get a master pin if it is required.

  preconditions:

  - The user has administrative authorisation.

  postconditions:

  - The master pin is returned to the administrator and may be used.

- **lockSystem** - The system is locked after the preconditions have been met.

  preconditions:

  - The duration has timed out.

  postconditions:

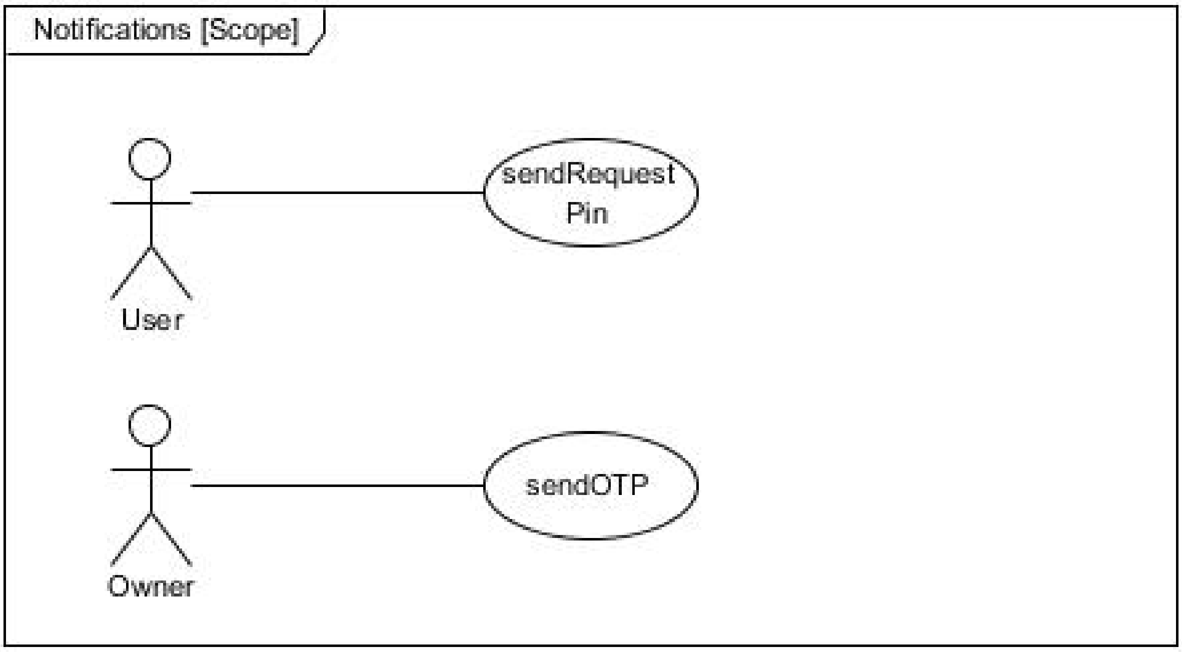
  - The system is locked and requires new OTP generation to be unlocked again.

3.3 Notifications

The notifications module will be responsible for sending messages to users, specifically to the Owner.

3.3.1 Scope

The scope of the Notifications module is shown below

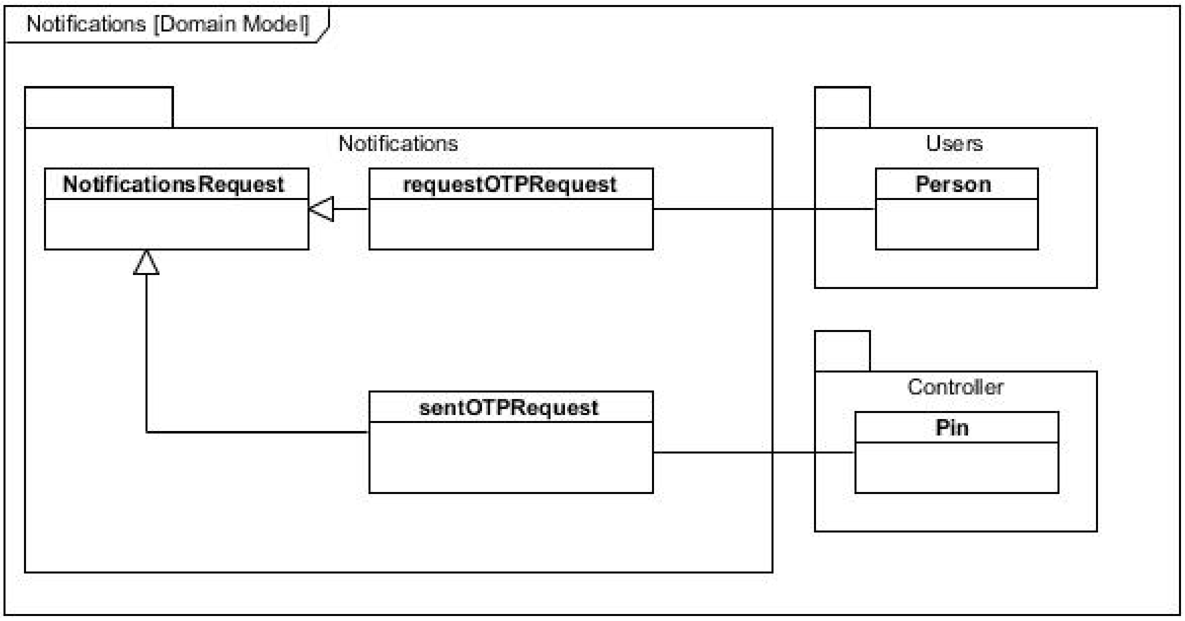


The scope of the Notifications module includes

- sending a request for an OTP.
- sending the OTP to the user/delivery person.

3.3.2 Domain Model

The scope of the Controller module is shown below



requestOTP used by the User/delivery person to request an OTP form the Owner. Message is sent to the Owner which then has to reply with an **sentOTP**.

sendOTP used by the Owner to sent an OTP to the requesting party. In this case it would be a delivery person.

3.3.3 Use Cases

- **sendRequestPin** - User sends a message to the Owner that informs them that a request for an OTP has been made.

  preconditions:

  – User requires access to the demarcated area.

  postconditions:

  – A pin request message is sent to the Owner.

- **sendOTP** - Allows the Owner to send a generated pin to the delivery person.

preconditions:

- The owner has received a pin request from the delivery person.
- The requester has authorisation.
- a Pin has been generated on the system.

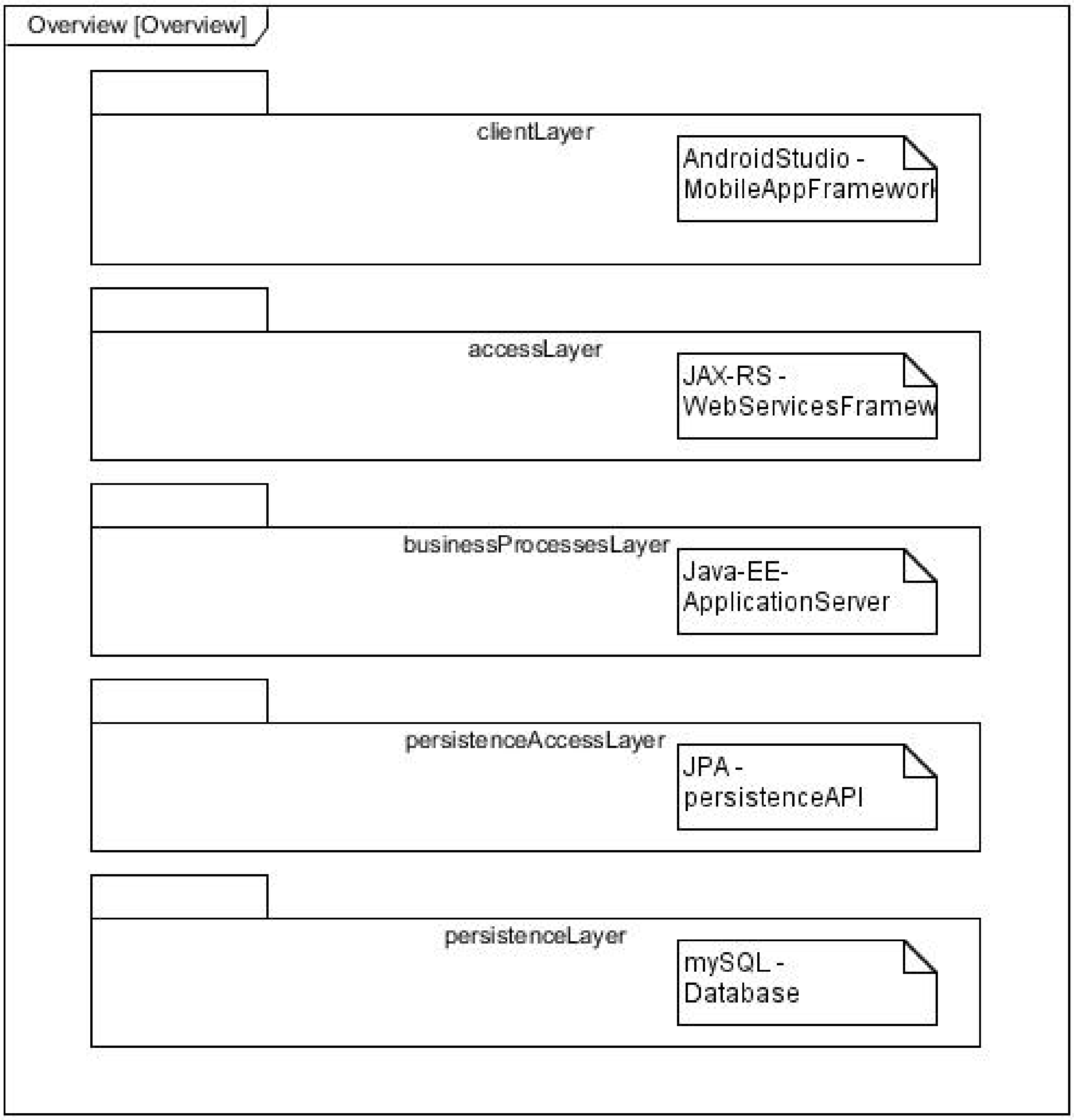
postconditions:

- The generated pin is sent to the requester.

4 Architectural Requirements/Design Specification

4.1 Software architecture overview

The diagram below provides a high-level overview of the software architecture as a whole. For this particular project we are making use of a layered software architecture, therefore the diagram shows how the respective components of the system will be layered.



4.1.1 clientLayer

The layer with which the client will be interacting. This will represent the interface with which the client will be interacting, be it an android or web interface.

4.1.2 accessLayer

The layer represents the Restful web services layer. It will be used to wrap business process as RESTful services so that multiple access channels may make use of these services.

4.1.3 businessProcessesLayer

This layer will host all the business logic within the application server. All methods will be executed server side, and all processes will be hosted on the JBoss server.

4.1.4 persistenceAccessLayer

This layer represents the Java Persistence API that is used to interact with the database. The client will never interact with the database directly but instead will make use of Entity managers.

4.1.5 persistenceLayer

This layer refers to the actual database technology that is used for persistence. In this case we will be using a mySQL relational database for persistence.

4.2 Access Channel Requirements

- An android application that allows the owner of the property to be notified when someone is outside the premises and to have a full video stream while the person is in the resident.
- The android application also allows the owner to remotely open and close the premises.
- The application will also let the owner generate a one-time pin to enable the person to enter the premises (This is the initial system and will be later used as the backup system).
- A JFrame touch screen intercom system that allows the delivery person who wants to enter the premises, to enter the pin that was generated by the owner and gain access to the resident.
- A web application to enable the owner of the property to view recorded camera footage and to configure the camera angles along with system settings.

4.3 Architectural Scope

- Video stream access of the premises from anywhere at any time i.e. constant camera footage of the premises. This will be done using SIP (Session Initiation Protocol).
- Voice will also be transmitted using SIP, when the delivery person interacts with the intercom system.
- An internal server that hosts all the devices in the household, allowing for additional devices to be connected at any time.
- An external could server to allow communication with the internal server and the android application the owner will be using.

4.4 Global Quality Requirements

4.4.1 Performance

- The system has to be stable and responsive enough to send a package with the auto generated pin. The system should also to stable enough to stream video to the android application used by the owner.

4.4.2 Security

- In terms of sending the OTP, end to end encryption with (check best hashing algorithm) will be used. For the initial system (which will be later used as the backup system) the OTP will be sent in a HTTP package.
- The OTP generated will be deleted once used, this ensures that a pin that was used will not work again if the delivery person attempts to use it at a later stage.
- In future the system will enforce confidentiality through encrypted communication of SIP (Session Initiation Protocol).
- Users of the system will will also need to be authenticated before they may make use of the various processes.
- Certain functionality will be restricted to the authorisation of a user, i.e. what role the user has within the system.

4.4.3 Reliability

- The property owner should be able to access the residence at any time using the android application
- The user should be able to grant access at any time using the application

- With regards to the backup system, the user should be able to generate a pin from anywhere and the pin should work by granting access to the delivery person.
- The response of the gate opening when the user opens it using the application should be reasonable.

4.4.4 Flexibility

- The user should be able to access the application from any of his/her devices.
- The application should be able to run smoothly on any android device that is Android 4.0 and above.

4.4.5 Maintainability

- Maintenance of this application will be handled by the developers, hence the use of an external server.
- The server will have to be well maintained because all the people who have this system in their homes will connect to the external server.
- Developers should be able to to add new functionality to the system, as well as change some if the existing functionality if they wish.

4.4.6 Monitorability

- While the delivery person is on the premises the owner of the property keeps track of all the activities on the resident.
- When there is no one on the resident, the cameras monitor all movement on the premises.

4.4.7 Usability

- The application should be simple enough for the user to use it without any training.
- The application should have basic instructions to ensure that the user is 100% sure of the action they are performing.

4.4.8 Cost

- The cost of the system should be kept to a minimum and made as affordable as possible.
- This system will mainly use Raspberry pis, keeping the development costs low.

4.4.9 Testability

- The individual components of the system needs to be testable through the use of mock objects and automated unit tests.
- Automated integration tests should be used to test the system as a whole.
- In particular the tests need to verify that the preconditions are met, and that post conditions hold true once a service has been provided.

4.5 Architectural Constraints

The client has some architectural specifications in place that will need to be adhered to, in order to successfully complete this project.

4.5.1 Use of Inexpensive Devices

The devices used to construct the physical system needs to be as inexpensive as possible, thus providing a larger audience with access to it. Devices such as the following will be included:

- Raspberry Pi 3
- IP Camera/Pi Camera
- Internet Enabled Switches
- WiFi Routers
- Android Smartphone (Users are assumed to posses one already).

4.5.2 Technology Stack

For monetary reasons, the client has requested that the entire system be developed using open source technologies. CodeBlox has therefore decided to settle on the Java Ecosystem, since we have already had exposure to such technologies. a Preliminary stack will include the following technologies:

- **Java EE** as the primary development platform
- **NetBeans IDE** as the primary development environment.
- **Android Studio** to the build the mobile application that will be used.
- **Apache Maven** will be used as the primary build tool
- **Mockito/TestNG** for unit testing
- **JAX-RS** for implementing RESTful webservices.

- **JBoss** for the primary application server (New version of glassfish seems to have some issues).
- **Linux** as the operating system
- **MySQL** as the primary persistence technology.

4.6 Architectural Design

This section specifies the architectural responsibilities of various architectural components.

4.6.1 Architectural Responsibilities

- The responsibility of providing an execution environment for business processes is assigned to an application server. i.e. **JBoss**
- The responsibility of persisting domain objects is assigned to the the database i.e. **MySQL**
- The responsibility of providing access to the database is assigned to the persistence API i.e. **Java Persistence API (JPA)**
- The responsibility of providing access channels to users is assigned to the web services framework i.e. **Jersey, which is an implementation of JAX-RS for RESTful Web Services**
- The responsibility of providing users access via mobile devices is assigned to the mobile application framework i.e. **Android Studio**

4.6.2 Infrastructure

For this particular project, the **layered architectural pattern** will be used as an infrastructure. The pattern limits access of components within one layer to components which are in the same layer or one level down. Thus it cannot access components in the layer above it. What makes it a very useful pattern is that different levels can be easily replaced without disrupting layers either above or below it.

4.7 Application Server

The application server refers to the architectural component within which business processes are deployed and executed. Put in other words, the application server will be hosting the business processes layer of the system.

4.7.1 Quality Requirements

- **Flexibility**
  - deploy different versions of the system with minimum downtime.
- **Reliability**
  - Requires that service requests are not partially executed. Thus the requested service will only be executed once postconditions are fulfilled. If a service could not be provided, a reason must be given.
- **Security**
  - The application server must support role-based authorisation. Thus only a user with a certain level of authorisation may execute certain processes. No direct access should be provided to the database.
- **Testability**
  - The application server should support out-of-container testing as well as unit testing.

4.7.2 Tactics as provided by Java EE

Java EE 7 supports all of the architectural tactics which are specified in the software architecture specific to the application server.

- **Flexibility Tactics**
  - Context and Dependency Injection (CDI) is fully supported by Java-EE by using the @Injection annotations.
  - Hot deployment is supported by the **JBoss** application server.
- **Reliability Tactics**
  - Java-EE supports container-managed transactions across multiple transaction-supporting resources. Transaction requirements can be specified by annotating services with TransactionAttribute.Required.
- **Security**
  - Java-EE supports role-based authorisation in the form of container managed declarative authorisation and programmatic authorisation. Methods are annotated with the required security roles.

- **Testability**
    - Java-EE supports dependency injection and Java-EE specific information is specified via annotations. All application code can thus be executed outside containers. Dependency injection can be used to inject mock objects allowing unit tests to be reused.

4.8 Database

For this particular project we will be utilising a relational database, namely **MySQL**. Support for the MySQL database is well documented and most developers are fairly familiar with it and its use.

4.9 Persistence API

The persistence API will provide indirect access to the MySQL database. For the particular project the Java Persistence API will be used.

JPA implements the following:

- **object-relational mapping**
- **query mapping**
- **object caching**
- **transaction support**
- **connection pooling**

For all query purposes, the Java Persistence Query Language (JPQL) will be used.

4.10 Web Services Framework

The web services framework refers to a wrapping layer that are used to wrap all business processes. This allows for the decoupling of user-interface technologies (in this case android app or web page) from the back end technologies.

4.10.1 Architecture Design

The project will make use of the Jersey implementation of JAX-RS of the JAVA-EE application server. Marshalling and demarshalling will be achieved by making use of the relevant annotations i.e. @produces and @consumes.