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)

Lorenzo Spazzoli (Bsc: Computer Science)

Bilal Muhammad (BIS: Multimedia)

Dirk de Klerk (BIS: Multimedia)

July 21, 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 PersonService

The PersonService module will be responsible for maintaining user information. The module will also have the responsibility of storing demographic information about the users, store information about the users' next of kin, and finally keep track of the users account status.

3.1.1 Scope

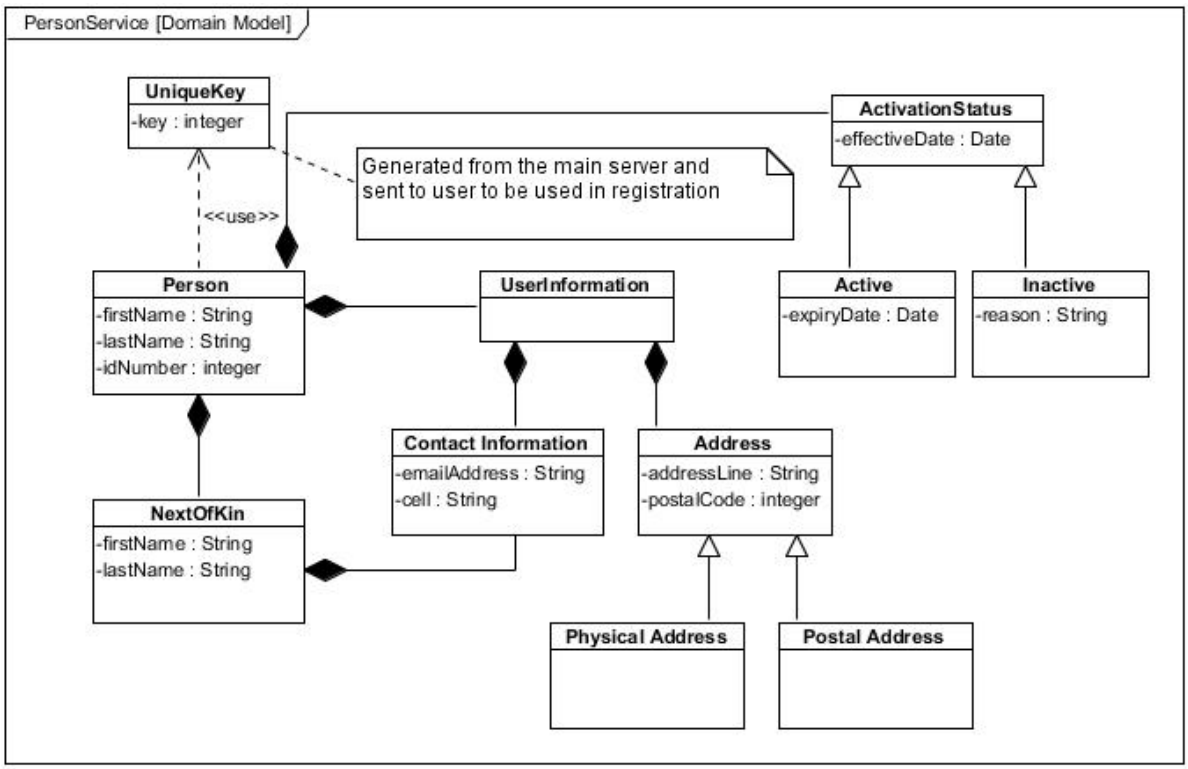
The scope of the Users module is shown below

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. Each person will also have associated with them UserInformation, which will include email, cellphone, as well as Address. A next of kin will also be available in case the user cannot be reached. Lastly the Person will have associated with it an ActivationStatus, that will determine if the user is still able to receive services.

3.1.3 Use Cases

- **loginUser** - Allows a user to log into the system.

  preconditions:

  - The user has already been registered with the system.
  - The user is still considered an active user.
  - The user has provided the correct login credentials.

  postconditions:

  - The user is logged into the system.
  - User gains access to all the services available to him/her.

- **logoutUser** - Allows a user to logout of the system.

  preconditions:

  - The user is already logged into the system.

  postconditions:

  - The user is logged out of the system.
  - The user no longer has access to the services provided by the system.

- **registerUser** - Allows a user to be registered on the system and provide them access to the various services.

  preconditions:

  - The user does not already exist on the system.
  - The user has been provided with a unique registration key

  postconditions:

  - The user has been added to the system and may continue.

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

  preconditions:

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

postconditions:

- The users' information is updated.

3.2 CameraService

The CameraService is responsible for providing the user with access to the live camera feed. The user needs to be able to access a feed, and also needs to be able to close a camera feed. This will ultimately allow them to verify the identity of the delivery person.

3.2.1 Scope

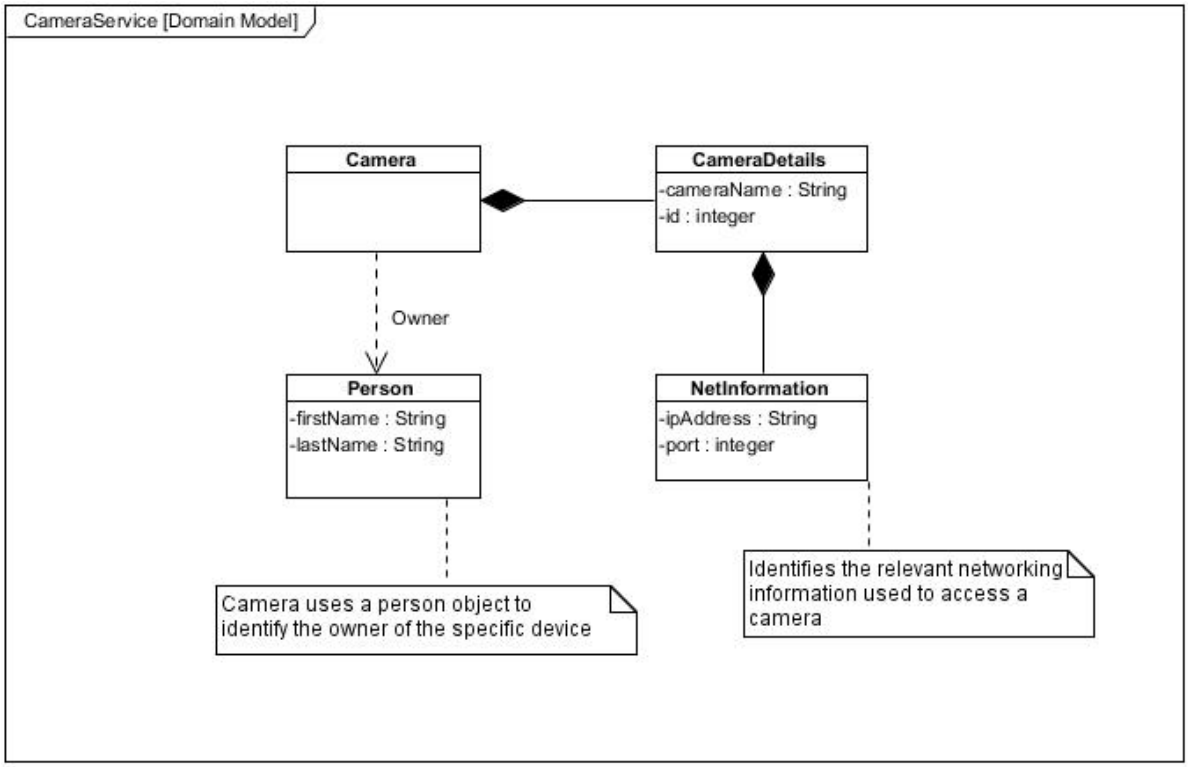
The scope of the CameraService module is shown below

The scope of the CameraService module includes

- Gaining access to the live camera feed.
- Close the camera feed as a means to save on bandwidth.

3.2.2 Domain Model

The domain model for the CameraService module is shown below



Each Camera will have associated with it CameraDetails that will provide information to how the camera may be accessed. Also the Camera Class will be associated with a Person class to identify them as the owner of the device.

3.2.3 Use Cases

- **getCameraFeed** - Enables the user to gain access to the live camera feed.

  preconditions:

  - The user is still active in the system.
  - The camera is available to be accessed.

  postconditions:

  - The user receives a live stream.

- **closeCameraFeed** - Allows a user to close a camera feed.

  preconditions:

  - The camera feed is currently streaming.

  postconditions:

  - The camera stops streaming a live feed.

3.3 PinService

The PinService Module is responsible for generating new pins as a backup, should the camera system fail. The module will also need to keep track of the status of the pin i.e. whether or not it has been used or not, as well as its life time. If a pin has been used or it has reached its life time maximum, a new pin has to be generated.

3.3.1 Scope

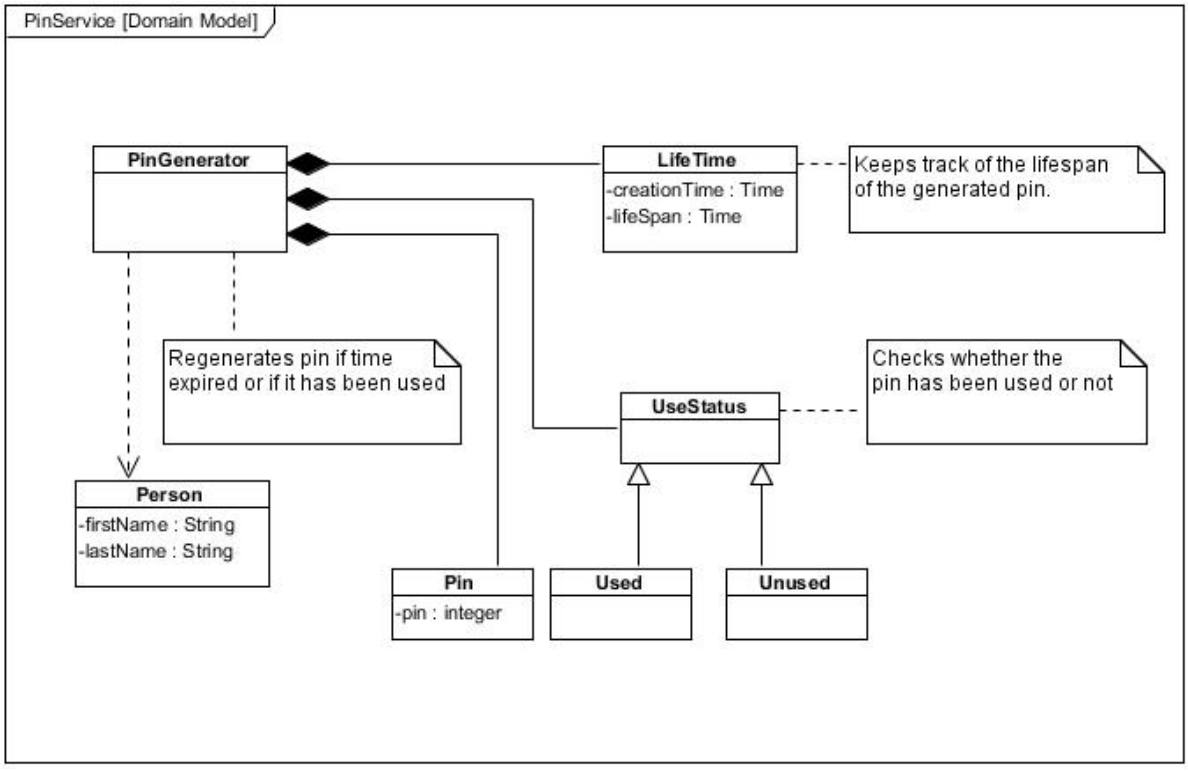
The scope of the PinService module is shown below

The scope of the PinService module includes

- Generating a new pin.
- Keeping track of a pins' validity through life time and status.

3.3.2 Domain Model

The domain model for the PinService module is shown below



The main class will be the PinGenerator. Associated with each pin generator will be a Pin, which will store the relevant pin that may be used as a fail safe. Also associated with the PinGenerator are the LifeTime and UseStatus classes, which are responsible for monitoring the Pins' validity. The PinGenerator is also associated with a Person so that each generated pin is sent to the correct person.

3.3.3 Use Cases

- **generatePin** - The System will generate a new pin to be used as a fail safe.

  preconditions:

  - A pin has expired.
  - A pin has already been used.

  postconditions:

  - A new pin is generated.

- **displayKeypad** - The keypad to enter the pin will be displayed.

  preconditions:

  - The system is making use of the failsafe.

  postconditions:

  - The keypad is displayed on screen.

- **updateServer** - The pin is updated server side for later use.

  preconditions:

  - A new pin has been generated.

  postconditions:

  - The new pin is reflected on the server.

- **getPin** - Allows the user to receive the generated pin.

  preconditions:

  - The user is logged into the system.
  - The pin is available.

  postconditions:

  - The pin is sent to the user.

3.4 LockService

The LockService Module will have the responsibility of controlling access to the dropbox. The lock mechanism can be comprised of any suitable technology for example a step motor or magnetic lock. The service should be able to unlock and lock the device, as well as keep track of the device status.

3.4.1 Scope

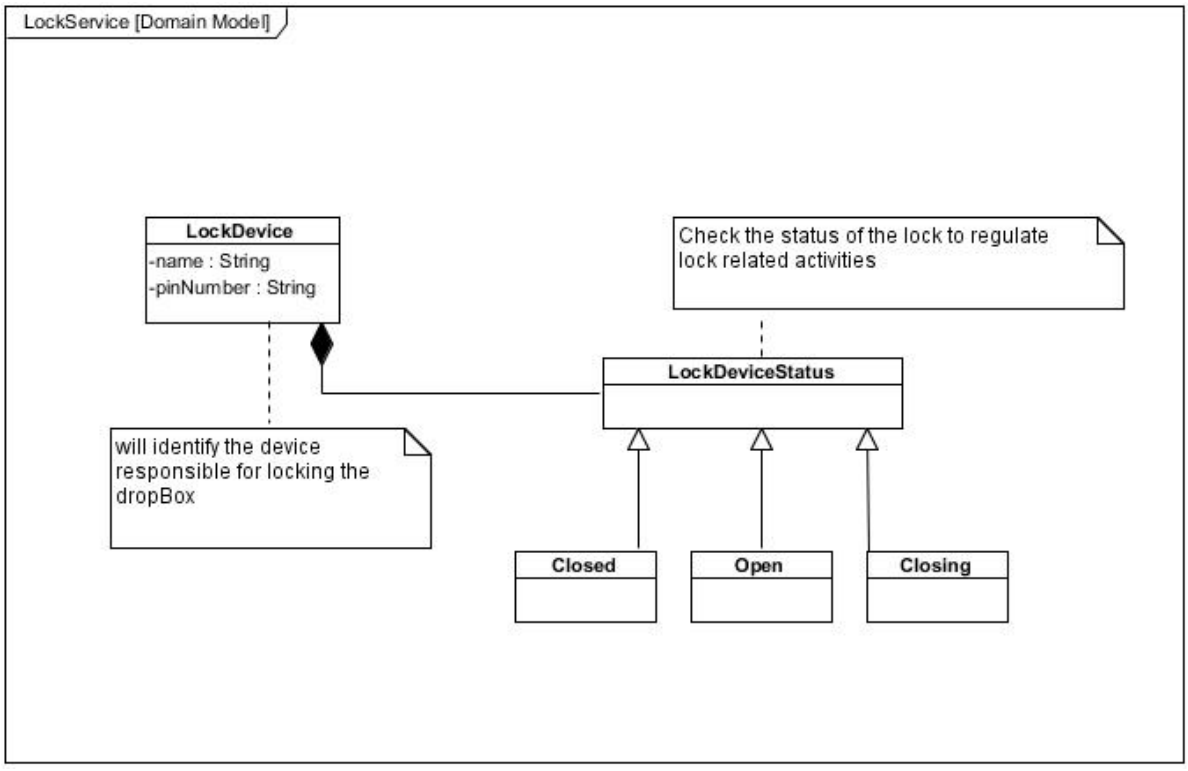
The scope of the LockService module is shown below

The scope of the LockService module includes

- Open up the dropOff box (via unlock or rotate of motor)
- Lock the dropOff box
- keep track of the devices' status.

3.4.2 Domain Model

The domain model for the LockService module is shown below



The LockDevice Class consists of a name and a pinNumber. The name is used to identify the device and pinNumber will be used to identify the pin used to interface with the raspberry pi. Associated with the LockDevice class will be a LockDeviceStatus, that will be used to keep track of the devices' current status i.e. Closed, Open, and Closing.

3.4.3 Use Cases

- **openDropOff** - Will allow the user to open up the box once identity has been verified.

  preconditions:

  - The box is closed.
  - The user is logged in.
  - The camera feed is accessible (System is functioning normally).

  postconditions:

  - The box is opened up.

- **lockDropOff** - This function will allow the dropOff box to be locked and secured again.

  preconditions:

  - The device is currently open/unlocked.

  postconditions:

  - The device is locked.

- **getDropOffStatus** - Provides the user/system with state information about the system.

  preconditions:

  - The system is in some state, either Open, Closed, Closing.

  postconditions:

  - The appropriate state is returned to the system/user.

3.5 NotificationService

The NotificationService Module is responsible for informing the user that a delivery person is at his/her residence. If the home device is unable to communicate with the server, a message has to be displayed locally informing the delivery person to contact the owner to obtain a pin.

3.5.1 Scope

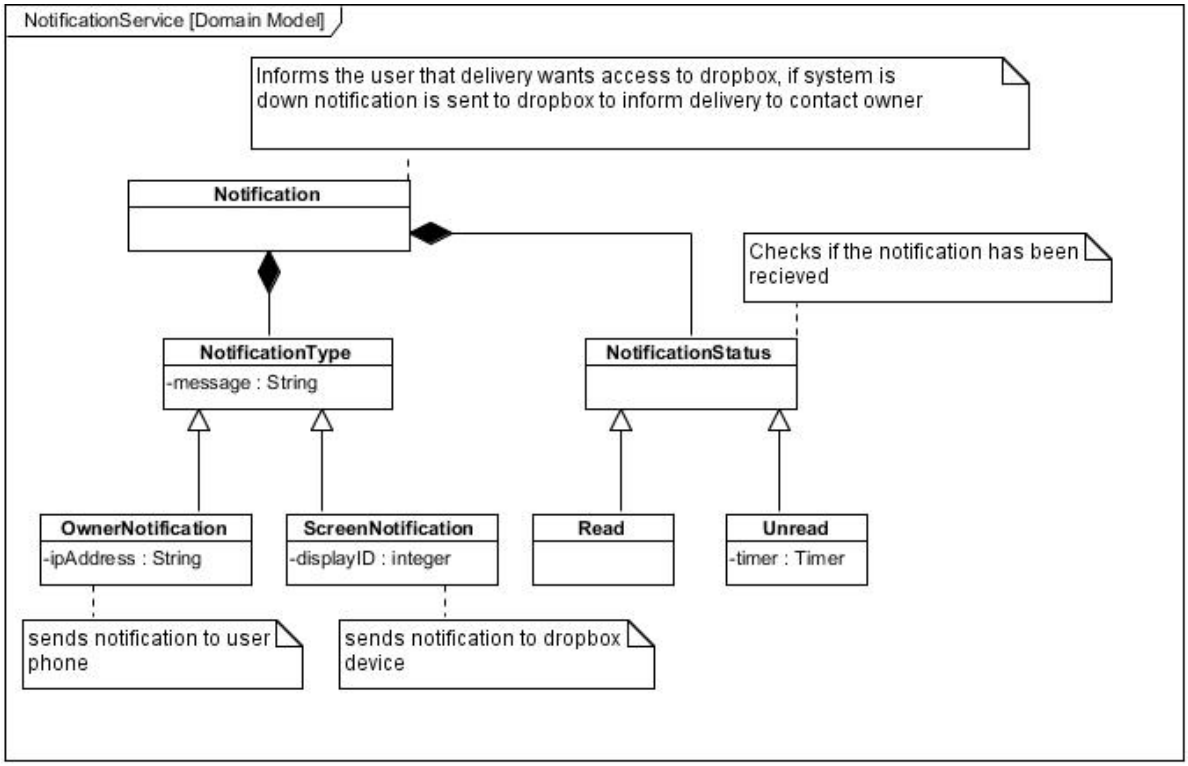
The scope of the NotificationService module is shown below

The scope of the NotificationService module includes

- Send a notification to the user to inform him/her of a potential delivery.
- Send a message to the local device to instruct delivery person.
- Set and get notification status.

3.5.2 Domain Model

The domain model for the NotificationService module is shown below



3.5.3 Use Cases

- **sendUserNotification** - Will send a notification to the user, informing him/her of a delivery.

  preconditions:

  - The alert button has been pressed.
  - The home device is able to communicate with the server.

  postconditions:

  - A notification is sent to the user.

- **sendLocalNotification** - Will generate a message that will be displayed on the local device.

  preconditions:

  - The alert button has been pressed.
  - The local device cannot communicate with the server.

  postconditions:

  - A message is displayed on the local device interface.

- **setNotificationStatus** - Allows the status of the sent notification to be set as Read, or Unread.

  preconditions:

  - A notification has been sent.

  postconditions:

  - The status of the notification is set to Unread. Once read it is set to red.

- **getNotificationStatus** - Provides the user/system with the current status of the notification.

  preconditions:

  - A notification status has been set.

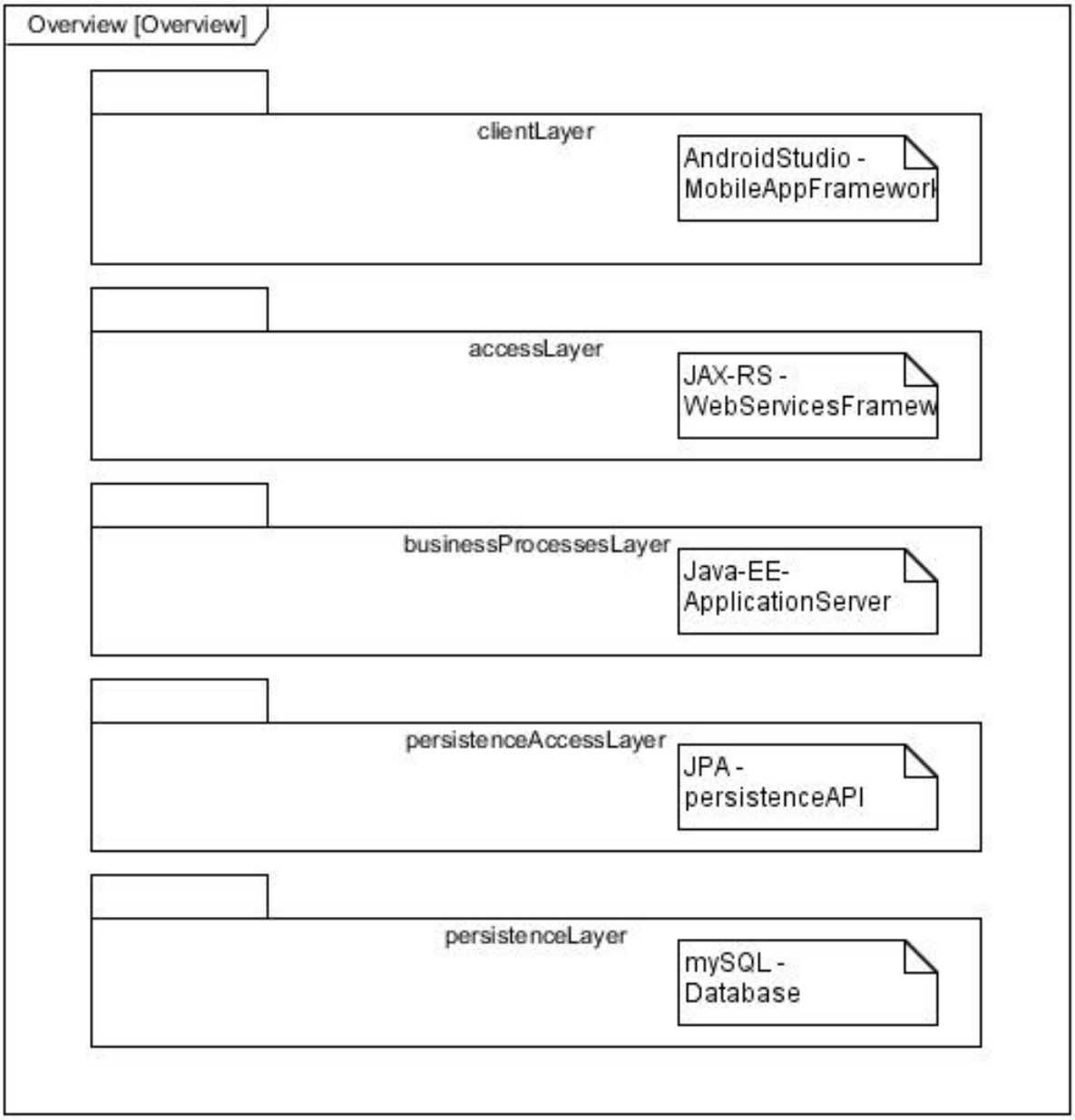
  postconditions:

– Returns the status of the notification.

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.