UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Faculty of Engineering, Built Environment and Information Technology

EPI-USE® LABS
value through innovation

COS 301 - SOFTWARE ENGINEERING

HOME SECURITY SYSTEM (ARGUS)

SYSTEM REQUIREMENT SPECIFICATION

Initials and Surname	Student Number
B. Chu (Discontinued)	u18176951
B.A. Zietsman	u15228194
B.R. Fourie	u16024002
J.K. Lutz	u15323413
J.S. Stadler	u16008554
S. Vermeulen	u16078625

September 17, 2020

Contents

1 INTRODUCTION

1.1 PURPOSE OF THE SYSTEM

This document is used to outline the requirements of the 'Home Security System' which will be used by homeowners to better protect their homes through threat detection.

1.2 SCOPE OF THE SYSTEM

- Abilities and Constraints:
    1. The system will provide administrator functionality which allows an administrator to add or remove users that are authorized to view the system.
    2. The system will allow authorized users to view a live camera feed.
    3. All people and vehicles on the live feed will be classified as white-listed(known), grey-listed(unknown) or black-listed(threats).
    4. The system will keep a record and snapshot of all grey and blacklisted classifications.
    5. The system will send an email notification of every blacklisted classification.
    6. A web application will be used as the human interface for this system
- Goals and Benefits: The system will alert the owner to potentially unwanted people and vehicles on their property by using constant image classification on multiple video feeds to identify all people and vehicles and compare them to a white-list. The main benefit of this system will be a higher sense of security surrounding the owners' home.

1.3 VISION

Our vision is to create a secure system capable of instantly identifying threats on multiple video feeds around the home of a user while notifying the user of identified threats. This will be achieved within the 4 month time period we have been allocated.

1.4 USER CHARACTERISTICS

The user should be computer literate. The user should also be able to use a smartphone and know how to navigate a simple app. They should have internet connection to be able to access the web application, have their cameras connect to the Argus system and access it on their smartphone. The user must have at least one security camera. The system will be used for home security purposes and is therefore targeted at home owners.

- Home Owners:
    - Are the people looking to improve their security and control of who is on their property. This user's technical skills will vary from novices to well versed but the system will accommodate for this.
    - Are expected to be between the ages of 20-60
    - Are expected to use the full range of features included in our system.
    - Expected to be our primary user group.

- Business Owner:
  - Are the people looking to improve their security and control of who is on their property.
  - May not be technically skilled but will likely have a IT assistant who will know how to use the system
  - As many new customers will enter on a daily basis, this user may want to turn off any alerts while open and rather be alerted of suspicious behaviour once they have locked up for the day.

2 ARCHITECTURAL DESIGN

2.1 DEPLOYMENT MODEL

Description Shown in Figure 1, the deployment model is using a client-server pattern and has three components namely, Heroku Server, Raspberry Pi and a generic computational device. The **web application component** will run on a NodeJS run-time environment. The server will include several artifacts, including the database server that manages all user accounts. For Server.js will require certain supporting artifacts such as:

- server.js - Server framework tool
- package.json - Handles system dependencies and holds metadata for the system
- App.js - Our app we are making to control server requests and allow users to view their cameras and flagged threats.

The **generic computational component** with its corresponding operating system, will have a TypeScript Run-time environment for Angular running the following artifacts:

- webapp.html - HTML will be used to program the dashboard and to create an interface to interact with the system. CSS will be incorporated to make the web application as visually appealing as possible.
- webapp.js - JavaScript will be used to implement all the functions that will generate the functionality required by the dashboard web application.

Also in the **generic computational component** with its corresponding operating system, will have a JRE environment for SpringBoot running the following artifacts:

- SpringBootApplication.java - a Java class will be used to program the SpringBoot backend and to create queries to send to the Web application via a Http client.
- Liquibase - used to control the operation between the repository and the database.

Within the **Raspberry Pi component** acts as a camera controller, where a limited amounts of ports for available for camera connections. This Raspberry Pi device has a Raspbian operating system where the Python Run-time Environment will run on. The deep learning models will be run on tensorflow architechture. The following artifacts are used in the Python environment:

- monitor.py - the script used for management, communication to the server and use of the Neural Networks
- mtcnn - A deep learning model used for facial detection
- resnet50 - A deep learning model used for facial recognition
- reature.npy - A file used to store facial features for recognition.
- footage.avi - Footage captured of blacklisted detection will be stored on the device.

The **Heroku Server** device uses a TCP/IP and UDP connection between itself and the Raspberry PI as well as the Generic Computational Device. The Heroku server contains a Heroku VM where the Firebase database for user accounts and JavaScript exectution environment. The Javascript Environment contains the following:

- node.js

- server.js
- app.js
- package.json

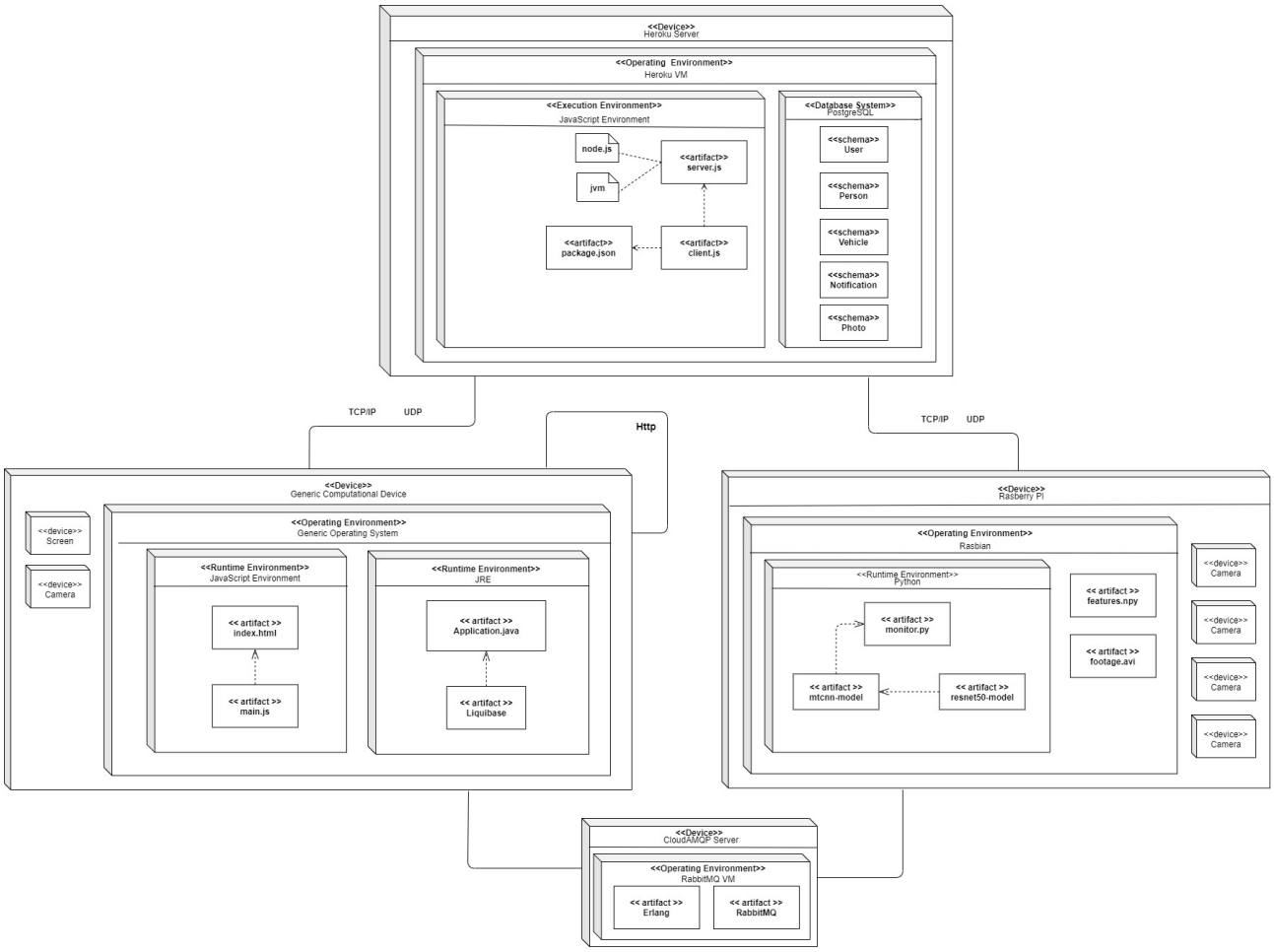

Figure 1: Deployment Model

2.2 ARCHITECTURAL DIAGRAMS

Description Shown in Figure 2, the system will work on a 3-tier system. The 1st tier, the client tier, contains the web dashboard in communication to the end-user. Within the client tier will be the view of the MVC system, where all of the user interaction will take place with the system. These interactions include an interface for all notification, settings and viewing of the entities within the Argus system. A service for camera interaction, an interface for creating profiles and an interface for user management is also on this tier. HTTP requests are made between the Client Engine and the Dispatch Servlet in the Logic tier.
Moving onto the Logic tier, the controller of the MVC is found. The controller will make all of the interactions possible between the user on the web dashboard as well as the system. Hence users will be able to log in or out of the system, upload photo's of people/objects and assigning user roles. Within the Logic tier, we have implemented a Pipe and Filter as well as a Broker pattern for communication between SpringBoot backend and the Raspberry Pi (also a controller to a different MVC model) that drives the whole camera feed of the system. The Repository on the REST API SpringBoot Application uses Liquibase to manage the PostgreSQL Database both locally and for Heroku.
The Data tier, will consist of only one PostgreSQL database that has different tables for user management, notification management and persons/vehicle management.
The logical flow of each component is as follows:
One separate MVC handles the camera interaction of the system. The camera is connected to the web dashboard where the user can handle all of the camera's interactions. If a user then wishes, the user can specify to save specific camera data, and that will be sent to the Camera feed database.
On the other MVC model, there is a web dashboard for the user to interact with the system. All of the user interactions are then part of the Controller of the MVC that talks to the web dashboard view. All user related data, adding profiles, user management etc., will then be stored onto the User profile database.

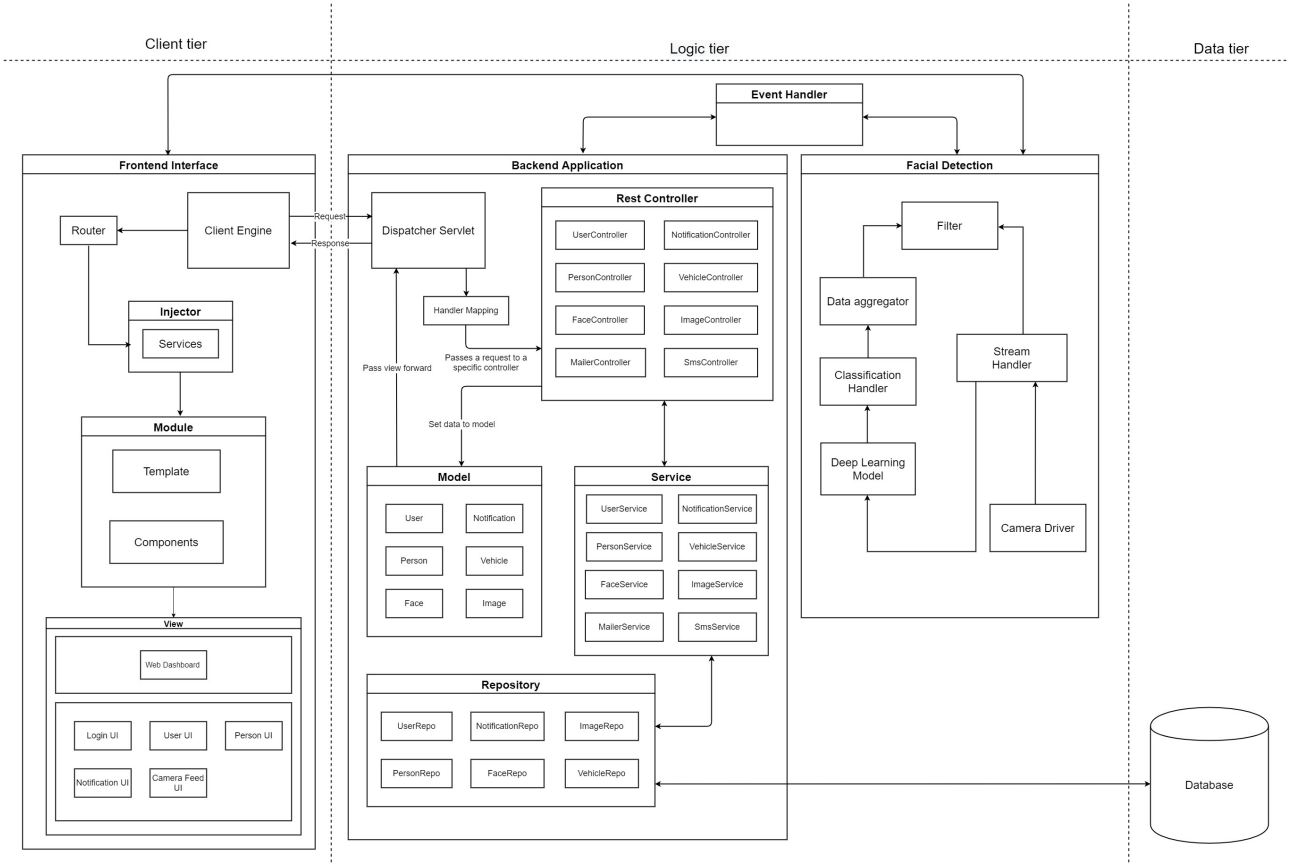

Figure 2: Architectural Diagram

3 NON-FUNCTIONAL REQUIREMENTS

3.1 QUALITY REQUIREMENTS

Maintainability

The application must be kept up-to-date with changing technologies and must be as modular as possible to remove complexity of replacing old programming with new. Updates must also take into consideration user grievances such as having a review page where users can comment on features they are unhappy with. This allows for a continuous cycle of deployment which is key in maintainability of software systems. In terms of the database, user records should always be kept up to date to ensure that accurate data is always shown on the user's dashboard.

Performance

Performance is a measure of how well the software responds to any circumstance the application may encounter and the speed at which it does so. The response time between components is a key metric in measuring performance. The main performance requirement for the system is the facial recognition and ability to alert the user of suspicious activity. In order to ensure top performance the application should be thoroughly tested under straining circumstances.

Reliability

It is important that the application does what is promised to the user: alert them of any suspicious behaviour, people or cars on their property. The software will need to accurately distinguish between suspicious and non-suspicious activity. This quality requirement refers to the ability of the system to work as expected and maintain its functionality over the course of its expected life time. Load and Regression Testing will be used routinely to test availability of the system under stress and a lack of bugs after updates, which is targeted at 95 percent. The system also needs to be reliable in terms of handling multiple users at a time. The dashboard, when hosted on a server, should be able to run efficiently and should not be able to crash. The database itself should also be able to handle multiple requests at a time as well as be able to store multiple user records without crashing. Seeing as Firebase is currently a realtime database, it can store multiple records efficiently as everything is script-based.

Security

A users personal information is important and should remain confidential. A secure connection should be guaranteed while the user is connected to website and their information must be stored encrypted on both the client and server side. This would include log in details. User passwords will have a strict password requirement (i.e at least: 1 capital letter, 1 special character, 1 number and 1 lower case letter). The website must be up to date and secure to prevent any hack to jeopardise the users information. From within the database, all sensitive user data will have to be encrypted. Unique ID's will also have to be issued to each user, in a non incremental way, but rather a randomized manner. This will ensure that user records are less likely to be accessible to malicious users. Script injection should also be catered for to prevent the system from crashing due to malicious activity. Role-based access control is also used as a user can be either Basic, Advanced or admin and depending on their role depends on the privileges granted to them by the system. This role-based display also ensures that users will not be able to upgrade their role or to edit information based on their role.

Scalability

This quality refers to the systems ability to expand and be scale its user base while maintaining all its other quality requirements in an effective and efficient manner. The system must be prepared for initially small numbers of users, we expect about 10 users initially to be using this product, but must be able to constantly scale up server capacity in order to guarantee little to no strain on the servers as the number of users increase. This system can be used, in a more extreme scenario, by up to 1000 people at a time assuming that they live in a complex. In a more general scenario the system will be used by more or less 2-3 people at a time which the system will easily be capable of handling.The system is divided into different independent components that can each be load tested individually to determine the system response time, throughput rates, utilization of resources and its breaking point. The aim is to have the system process the captured video or audio, determine if a threat is present via the neural network and alert the user if need be within a 10 second response time.

Usability

The user should be able to understand and learn how to use the system without requiring help from the developers. The systems most important features should be within reach of the main screen of the dashboard so that the user does not need to search around for functionality. The user must be able to navigate through menu items and find what they need as easily as possible. The interface of the system must be as simple as possible so as to not discourage users from using the system and so that the system is also visually enticing to use. The response time of the system must be as fast as possible to render all the notifications useful and to ensure that the system works as intended.

The design pattern used to lay out and implement the system include: Client-Server, 3-Tier , Model-View-Controller, Pipe and Filter, Broker, Interpreter and Micro-services.

The Client-Server design patters implemented with Heroku being the master and Argus application being the slave.

The 3-Tier design pattern was used across the entire system to divide the components into namely the Client tier, Logic tier and Data tier. The Client tier contains the Frontend interface which is implemented in Angular, the Logic tier contains contains the Backend application implemented in Springboot and the Facial detection component implemented in Python, and finally the Data tier contains the PostgreSQL database.

The Model-View-Controller (MVC) design pattern was used control the flow of data into objects on the database and the to update the viewing of these objects on the framework. The Angular frontend is able to interact with the database by displaying its contents, updating values on the database as well as adding new contents (users or persons) to the database. The CRUD operations caused changes to the database that needs to be immediately reflected on the Angular frontend.
The Pipe and filter design pattern was used to implement the events passed between the RabbitMQ event handler and the Spring backend, Python neural network and Angular frontend. This allows for smaller and independent processes or checks to be performed along the events path rather than having to do it all on sender end or the receiver end.

The Broker design pattern is implemented using RabbitMQ to coordinate the communication and distribution of events to the distributed components in the system. RabbitMQ handles the events passed between the Springboot backend, Angular frontend and Python neural network.

The Interpreter pattern is implemented by using Liquibase to allow the Springboot Backend to communicate and interact with the PostgreSQL database. Liquibase allows for CRUD operations to be performed on the database as well as acting as a middle man to handle the requests and make sure that no malicious activity can be performed.

The Micro-services design pattern was implemented using Docker. Docker runs an application in loosely isolated container environments, therefore decomposing the application into smaller components and running them individually, but simultaneously.

3.2 ARCHITECTURAL CONSTRAINTS

Hardware constraints

The maximum number of cameras will be determined by the internet speed at hand as well as the speed of the neural network and database connections. This is due to the response time of the system being set to a maximum of 10 seconds to ensure the validity and integrity of notifications and events from the system. The quality of the video feed is set to a maximum of 720p to allow a balance between quality and processing speed. Storage space should hold large amounts of data for video data. The storage spaces should provide support to keep 1.5GB per hour, where a day will be 36GB. A week of the footage must be saved, where some will be saved and the rest deleted. Some common video and audio codecs such as .mp4 and .mp3 must be supported. The system must be able to run on a Linux server(Ubuntu/OpenSUSE). The app should be usable on the latest Chrome and Firefox browsers and be responsive/mobile friendly.

3.3 TECHNOLOGICAL REQUIREMENTS

- Neural Network - Python, TensorFlow
- Web Application - HTML, CSS, JavaScript, TypeScript
- Database - PostgreSQL
- Event Handler(Message queues) - RabbitMQ/CloadAMQP

3.4 PROTOCOLS

The following protocol types will be used in the home security system:

- HTTP will be used to communicate with the database.
- SMTP protocols will be used for sending notifications via email to the user.
- TCP will be used for all communication between the server and the Raspberry Pi, except for the live-streaming of security footage.
- UDP will be used for live-streaming security footage on the webapp.

4 FUNCTIONAL REQUIREMENTS

- R1: Web dashboard for the security system
  - R1.1: The web dashboard will provide an interface to interact with the camera/s.
    - R1.1.1: The web dashboard will allow for enabling and disabling of camera live feeds.
  - R1.2: The web dashboard will provide functionality for notification management.
    - R1.2.1: The dashboard will allow for the enabling/disabling of notifications.
    - R1.2.2: The dashboard will allow the type of notification method to be changed sending to the user.
    - R1.2.3: The dashboard will allow notifications to be dismissed by the user.
  - R1.3: The dashboard will allow for the creation of profiles.
    - R1.3.1: The dashboard will allow creation of a person's profile in the system.
    - R1.3.2: The dashboard will allow to take a photo for a person's profile face
    - R1.3.3: The dashboard will allow to upload a photo for person's profile face
  - R1.4: The dashboard will allow for profile management.
    - R1.4.1: The dashboard will allow for a person's profile or number plate to be retrieved and viewed from the system.
    - R1.4.2: The dashboard will allow for a person's profile or number plate to be marked as Cleared(white-listed) on the system.
    - R1.4.3: The dashboard will display all people recognised by the system in the Unknown(grey-list).
    - R1.4.4: The dashboard will allow for a person's profile or number plate to be marked as Threat(black-listed) on the system.
    - R1.4.5: The dashboard will allow for a person's profile or number plate to be removed from or restored to the system.
  - R1.5: The dashboard will allow for user role management.
    - R1.5.1: The dashboard allows for admin roles
      - R1.5.1.1: Admins will be allowed to add and remove users to/from the database(either Admin, Advanced, Basic user) via the dashboard.
      - R1.5.1.2: Admin users will be allowed to add and remove people or vehicles to/from the database(either white-listed, black-listed) via the dashboard.
      - R1.5.1.3: Admin users will be allowed to add in cameras to the system(camera management allowed).
    - R1.5.2: The dashboard allows for advanced roles
      - R1.5.2.1: Advanced users will be allowed to add in users onto the database(either Advanced, Basic user) via the dashboard.
    - R1.5.3: The dashboard allows for basic roles.
      - R1.5.3.1: Basic users will be allowed to view camera footage from the camera/s.
      - R1.5.3.2: Basic users will be allowed to receive notifications from the system.
  - R1.6: The dashboard will allow for user management.
    - R1.6.1: The system will allow for a user to log on to the system.
    - R1.6.2: The system will allow for a user to logout of the system.
    - R1.6.3 The system will allow for a user to update their password.
    - R1.6.4 The system will allow for a user to recover their password.

- R2: Real-time monitoring of camera feed.
  - R2.1: The camera/s monitoring will provide imagery.
  - R2.2: The camera/s monitoring will take pictures of detected faces or number-plates.
  - R2.3: The camera/s monitoring will allows to either add or remove camera's.
- R3: Facial detection and classification (strangers vs. known people)
  - R3.1: The system will intelligent be able to identify people.
    - R3.1.1: The system will be able to identify high threat level people.
    - R3.1.2: The system will be able to identify unknown or known people.
  - R3.2: The system will be able to identify number-plates.
    - R3.2.1: The system will be able to identify high threat level vehicles.
    - R3.2.2: The system will be able to identify unknown or known vehicles.
- R4: Notification system
  - R4.1: The system will send notifications if an unknown person or vehicle was detected.
  - R4.2: The system will send notifications to the user via email if a threat level person was detected.

4.1 USER STORY

ADMIN USER

- As an **admin** I want to **switch on** the system to **use** all its functionality.
- As the **admin** I want to **switch off** the system to **stop** using its functionality.
- As the **admin** I want to **change** my password so that I can **login** to the system with a new password.
- As the **admin** I want to **recover** my password so that I can **login** to the system.
- As the **admin** I want to **add** an admin, advanced or basic **user** so that they are able to make use of the system functionalities.
- As the **admin** I want to **update** my **user** details in the system.
- As the **admin** I want to **remove** a user so that they no longer have system functionalities.
- As the **admin** I want to **change** the users role to admin, advanced or basic so they can have **different privileges** in the system functionalities.

ADVANCED USER

- As an **advanced user** I want to **switch on** the system to **use** some of its functionality.
- As the **advanced user** I want to **switch off** the system to **stop** using its functionality.
- As the **advanced user** I want to **change** my password so that I can **login** to the system with a new password.
- As the **advanced user** I want to **recover** my password so that I can **login** to the system.
- As the **advanced user** I want to **add** a **advanced or basic user** so that they are able to use some of the system functionalities.
- As the **advanced user** I want to **update** my **user** details in the system.
- As the **advanced user** I want to **remove** a user either with the same role or lower than min so that they no longer have system functionalities.

BASIC USER

- As a **basic user** I want to **switch on** the system to **use** some of its functionality.
- As a **basic user** I want to **switch off** the system to **stop** using its functionality.
- As a **basic user** I want to **change** my password so that I can **login** to the system with a new password.
- As the **basic user** I want to **recover** my password so that I can **login** to the system.

CAMERA

- As an **admin, advanced or basic user** I want to **switch on** the cameras to **view** the camera footage.
- As an **admin, advanced or basic user** I want to **switch off** the cameras to **stop** the camera footage.
- As an **admin user** I want to **add** cameras to **view** more camera footage.
- As an **admin user** I want to **remove** cameras to **view many or one** camera footage.

PERSON

- As an **admin or advanced user** I want to **create** a person's profile so that their details can be **recognised** by the system.
- As an **admin or advanced user** I want to **take a picture** of a person with the camera so that the system can **save** it to their profiles.
- As an **admin or advanced user** I want to **upload** an existing picture of a person so that the system can **save** it to their profiles.
- As an **admin or advanced user** I want to **update** a person's profile so that their details are **up to date** and correct.
- As an **admin or advanced user** I want to **change** a person's profile to be on the Cleared list (white-listed) so that the system can **recognize** friendly persons in the camera footage.
- As an **admin or advanced user** I want to **change** a person's profile to be on the Threat list(black-listed) so that the system can**recognize** a high profiled person from the camera footage.
- As an **admin or advanced user** I want to **view** a person's profile so that can analyse **analyse** it and make sure it is correct.
- As an **admin, advanced user** I want to **delete** a person's profile so that they are **removed** by the system.

VEHICLE

- As an **admin, advanced user** I want to **add** a vehicle's number-plate so that the system can **recognize** a known vehicle in the camera footage.
- As an **admin, advanced user** I want to **clear/white-list** a vehicle's number-plate from the unknown list so that the system can **recognize** known vehicle in the camera footage.
- As an **admin, advanced user** I want to mark as a **threat or black-list** a vehicle's number-plate so that the system can **recognize** threat vehicles in camera footage.
- As an **user** I want to **view** a vehicle's number-plate profile so that I can **analyse** it.
- As an **admin, advanced user** I want to **update** a vehicle's number-plate profile so that the details are **up to date** and correct.
- As an **admin or advanced user** I want to **delete** a vehicle's number-plate profile so that it is **removed** by the system.

NOTIFICATION

- As an **admin, advanced or basic user** I want to **receive notifications** of people/vehicles that are detected by the system near my home or around my area.
- As an **admin, advanced or basic user** I want to **cancel notifications** being sent of people/vehicles being detected by the system near my home or around my area.
- As an **admin, advanced or basic user** I want to **choose** what method of notification the system sends me so that I receive notifications that is **convenient to my communication** with the system.
- As the **admin user** I want to be able to **view** footage that was **captured** through out the day from the system.

4.2 USE CASES

- **U1: User Subsystem**
  - U1.1: Login
  - U1.2: Logout
  - U1.3: Change password
  - U1.4: Recover password
  - U1.5: Add user
  - U1.6: Update user details
  - U1.7: Remove user
  - U1.8: Assign user roles
- **U2: Camera Subsystem**
  - U2.1: Switch between camera feeds
  - U2.2: Enable/Disable camera feed
  - U2.3: Add camera
  - U2.4: Remove camera
  - U2.5: Record footage
- **U3: Event Subsystem**
  - U3.1: Enable/Disable notifications
  - U3.2: Change notification method
  - U3.3: Retrieve/View footage history
  - U3.4: Dismiss notification
  - U3.5: Send email alert
- **U4: Person Subsystem**
  - U4.1: Create person profile
    - U4.1.1: Take picture
    - U4.1.2: Upload picture
  - U4.2: Retrieve/View person profile
  - U4.3: Update person profile
    - U4.3.1: Change Vehicle details
    - U4.3.2: Cleared List profile
    - U4.3.3: Threat List profile
  - U4.4: Delete person profile
- **U5: Vehicle Subsystem**
  - U5.1: Create Vehicle profile
  - U5.2: Retrieve/View Vehicle profile
  - U5.3: Update Vehicle profile
    - U5.3.1: Change Vehicle details
    - U5.3.2: Cleared List profile
    - U5.3.3: Threat List profile
  - U5.4: Delete Vehicle profile

4.3 USE-CASE DIAGRAM

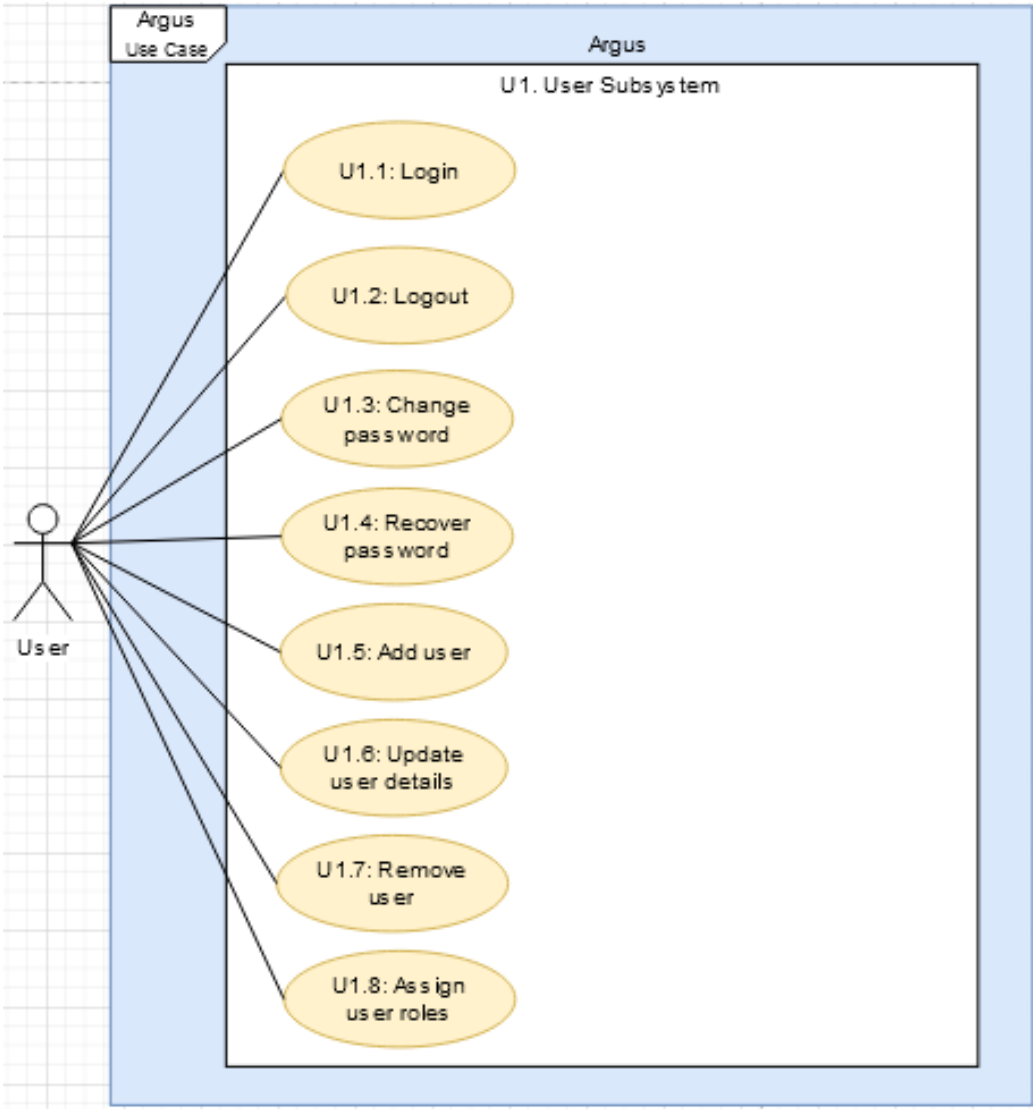

Figure 3: User Subsystem

Description Figure 3 shows the use cases for the user management subsystem. Depending on the role of the user he/she will have specific privileges. All users will be able to login(U1.1) and logout(U1.2) of the system. Once logged in he/she will be able to update user details(U1.6) as well as change their password(U1.3) however, if the user logs out and forget his/her password meaning they couldn't log back in to the system there's a option to recover the password(U1.4) which will the ask user to enter in his/her username and then proceed to send them an email with their password attached. If however, the the user is an Admin user he/she and wishes to add a family member of business colleague to the system they have the option of adding a user(U1.5) and assigning them a role(U1.8) of either as Admin, Advanced or Basic user, each having different privileges. Admin users have complete control of the system including removing a user(1.7) entirely from the system, whilst other users can only remove themselves. Advanced users only have access to adding certain users but won't be able to make system changes like adding/removing of cameras and finally, Basic users can only see alerts and footage but are basically read-only. Something to note is that Admin users cannot remove other admin users only themselves also they cannot change other admin users details or roles.

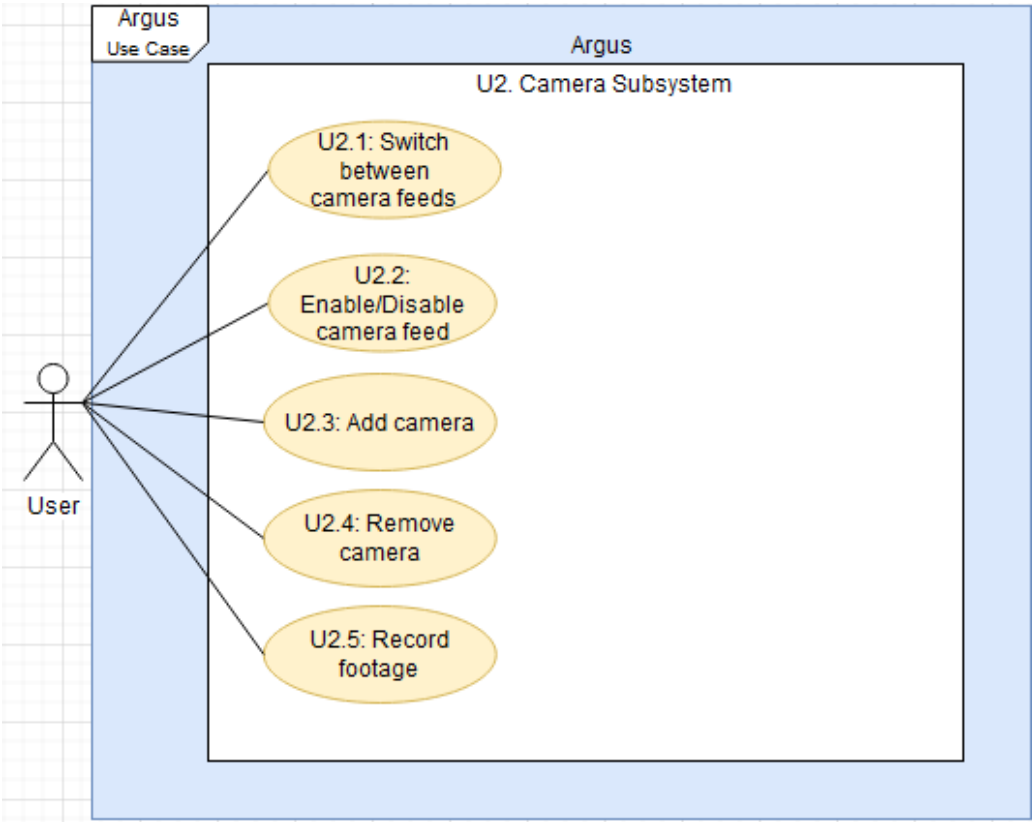

Figure 4: Camera Subsystem

Description Figure 4 shows the use cases for camera feed subsystem. An Admin/Advanced user can add(U2.3) camera or remove(U2.4) camera from the web dashboard. If multiple cameras are added the any user can switch between camera feeds(U2.1) by enlarging the one selected on the screen. He/she can also Enable/Disable(U2.2) any feed they wish and whilst a camera feed is enabled video footage will be recorded(U2.5).

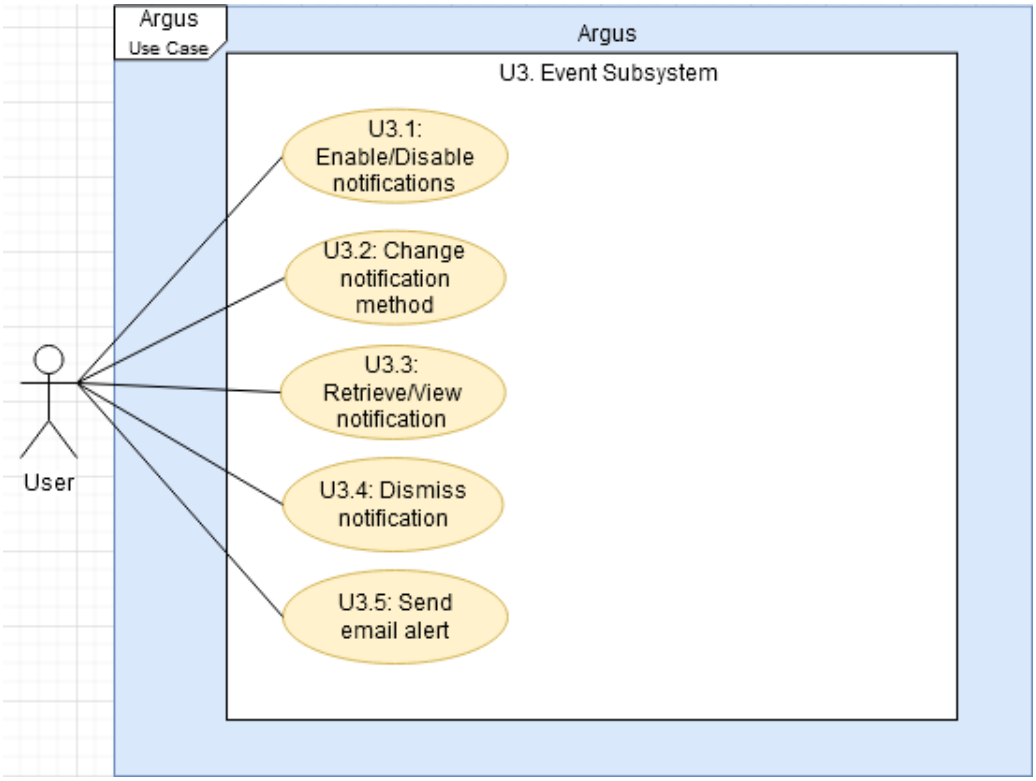

Figure 5: Event Subsystem

Description Figure 5 shows the use cases for the Event subsystem. The user will be able to receive notifications or not(U3.1). Change the type of notification method(U3.2) to either local or Email. Retrieve/view(U3.3) notification on the dashboard and dismiss(U3.4) if the notification is no longer are relevant.If an unknown person walks passed the property they should immediately be marked as Unknown(grey-listed) and the user should get a notification either via the dashboard or email(U3.5).

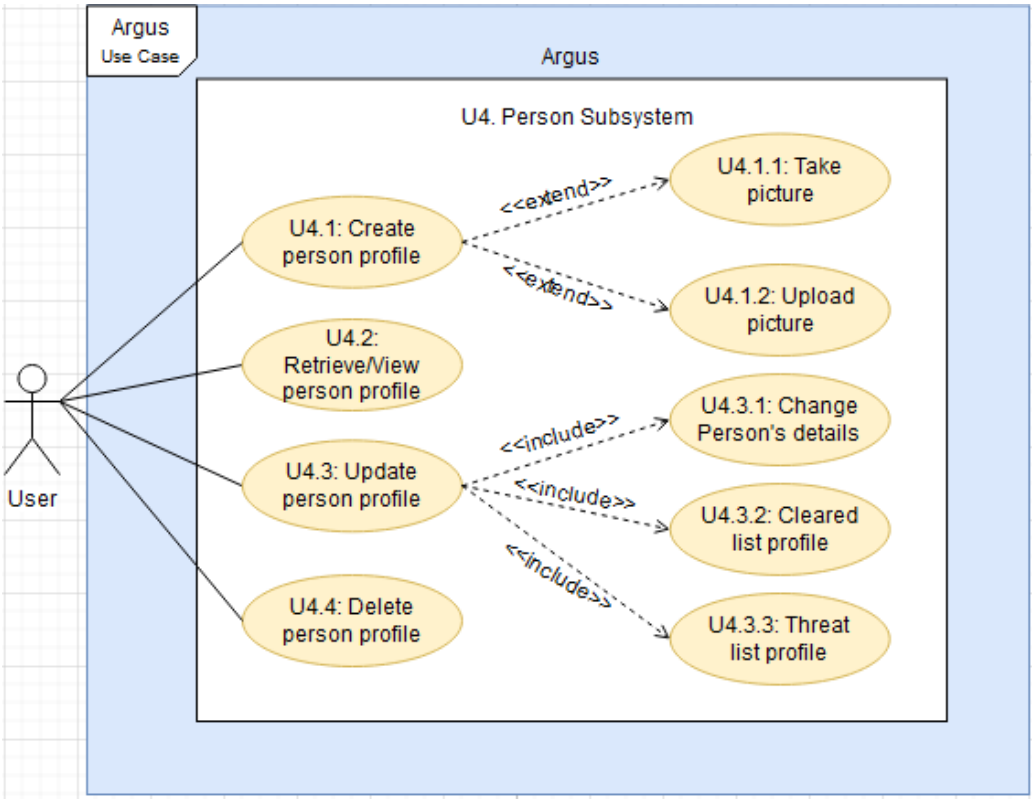

Figure 6: Person Subsystem

Description Figure 6 shows the use cases for the known/unknown person profile subsystem. An Admin/Advanced user should be able to create a person's profile(U4.1) by taking a photo(U4.1.1) of the person's face and uploading(U4.1.2) it to the system, the system should do this automatically for unknown people walking on the street using the camera feed. All users should be able to search for and retrieve(U4.2) a person's profile and in addition update(U4.3) their profile by marking them as either Cleared/white-listed)(U4.3.2) or Threat/black-listed(U4.3.3) or just simply changing their name(U4.3.1). Finally, if a user has the role of being Admin/Advanced he/she can delete(U4.4) a person profile.

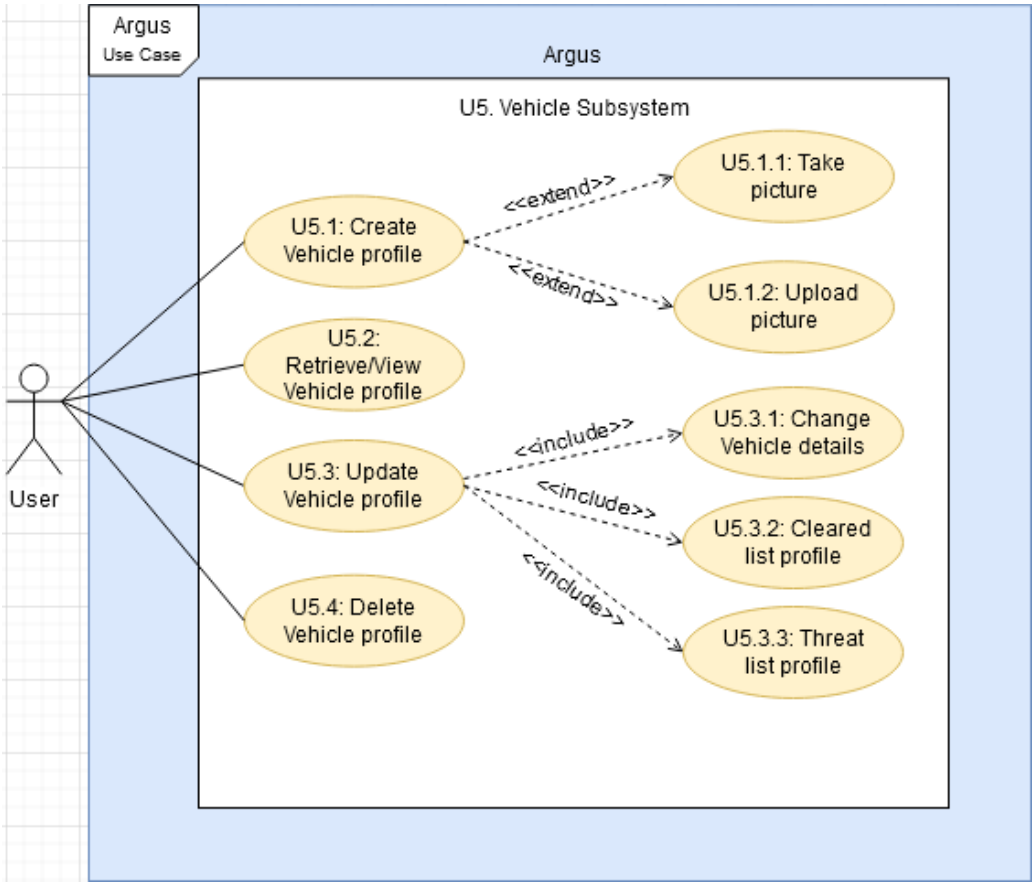

Figure 7: Vehicle Subsystem

Description Figure 7 shows the use cases for the known/unknown vehicle profile subsystem. An Admin/Advanced user should be able to create a number-plate profile(U5.1) by taking a photo(U5.1.1) of the vehicle's number-plate and uploading(U5.1.2) it to the system, the system should do this automatically for unknown vehicles driving on the street using the camera feed. All users should be able to search for and retrieve(U5.2) a vehicle's profile and in addition update(U5.3) the profile by marking it as either Cleared/white-listed(U5.3.2) or Threat/black-listed(U5.3.3) or simply just editing(U5.3.1) the vehicles number plate. Finally, if a user has the role of being Admin/Advanced he/she can delete(5.4) a vehicle profile.

4.4 SYSTEM TRACEABILITY MATRIX

Tracability Matrix

			Use cases																									
			User subsystem								Camera subsystem					Event subsytem					Person subsystem				Vehicle subsystem			
		Priority	U 1.1	U 1.2	U 1.3	U 1.4	U 1.5	U 1.6	U 1.7	U 1.8	U 2.1	U 2.2	U2.3	U 2.4	U 2.5	U 3.1	U 3.2	U 3.3	U 3.4	U 3.5	U 4.1	U 4.2	U 4.3	U 4.4	U 5.1	U 5.2	U 5.3	U 5.4
Function requirements	R 1.1	3									X	X																
	R 1.2	2														X	X		X									
	R 1.3	1																			X							
	R 1.4	4																							X	X	X	X
	R 1.5	2					X	X	X	X					X							X	X	X				
	R 1.6	1	X	X	X	X																						
	R 2.1	1									X							X										
	R 2.2	1													X													
	R 2.3	1									X		X	X														
	R 3.1	1																X										
	R 3.2	1																X										
	R 4.1	1																		X								
	R 4.2	1																		X								
		Priority	1	1	1	1	2	2	2	2	3	3	1	1	1	2	2	1	2	1	1	1	1	1	4	4	4	4

Figure 8: Traceability Matrix

Description Table 8 shows what use cases the user can interact with to satisfy the requirements that the system is capable of doing for the user. The priority indicates which use cases are most dependable on the functionality of the system requirements.

4.5 ACTOR-SYSTEM INTERACTION MODELS

The Actor-system interaction model describes the system's communication steps for functionality usage, so that a user participates as an actor that interacts with the system interface by understanding the steps process of all system functionality. The following are some use case narratives to explain some of the important use cases to the User, Person and Event subsystem.

USE CASE ID:	**1.5**	**USE CASE TYPE**	
USE CASE NAME:	**Add user**	Business Requirements:	☐
		System Design:	☐
PRIORITY:	Medium	System Analysis:	√
SOURCE:	Capstone – Home Security System		
PRIMARY ACTOR:	Admin, advanced user		
DESCRIPTION:	The user clicks the **Menu Button** in the **Web Dashboard Page** then chooses the **Users Button** where the **Users Page** opens. The user selects the **Add Users Button** and the system opens a **User form** for completion. The user fills the form in and clicks the **Done Button**. The user's profile is the saved as a basic user in the system.		
PRE-CONDITION:	User must be logged in.		
TRIGGER:	User wants to add a user to system to access certain system capabilities.		
TYPICAL COURSE OF EVENTS:	**Actor Action**	**System Response Manual Action Automated Action**	
	Step 1: User clicks on **Menu button** in **Web Dashboard Page** and then clicks on **Users button**.	**Step 2**: System displays **User Page**.	
	Step 3: User clicks the **Add user button**.	**Step 4**: System displays an empty **User form.**	
	Step 5: User fills in the **User form** and clicks on the **Done button.**	**Step 6**: System invokes **U3.1 Create Person** with the user details, for recognition in the system.	
		Step 7: System saves user profile to system.	
ALTERNATE COURSES:			
POST-CONDITION:	A new user is added to the system		

Figure 9: U1.5 narrative

USE CASE ID:	1.8	USE CASE TYPE Business Requirements: ☐ System Design: ☐ System Analysis: √
USE CASE NAME:	Assign user role	
PRIORITY:	High	
SOURCE:	Capstone – Home Security System	
PRIMARY ACTOR:	Admin	
DESCRIPTION:	The user clicks the **Menu Button** in the **Web Dashboard Page** then chooses the **Users Button** where the **Users Page** opens. The user selects the **Assign user role Button** and the system displays the users in the system. The user edits the **role attribute** of the user, from the option of admin, advance or basic user. The user clicks on the **Done button** and the system save the user with a new role.	
PRE-CONDITION:	A user must already exist in the system.	
TRIGGER:	User wants to change the user role to change their capabilities on the system.	
TYPICAL COURSE OF EVENTS:	**Actor Action**	**System Response** **Manual Action** **Automated Action**
	Step 1: User clicks on **Menu button** in **Web Dashboard Page** and then clicks on **Users button**.	**Step 2**: System displays **User Page**.
	Step 3: User clicks the **Assign user role button**.	**Step 4**: System displays the users in the system.
	Step 5: User chooses a user and changes the **role attribute** to **Advanced user.**	
	Step 7: User clicks **Done Button**	**Step 7**: System saves changes to user profile to system.
ALTERNATE COURSES:	**Alt-step 3 (1)**: User chooses a user and changes the **role attribute** to **Admin user.**	
	Alt-step 3 (2): User chooses a user and changes the **role attribute** to **Basic user.**	
POST-CONDITION:	Existing user's role has changed with certain capabilities.	

Figure 10: U1.8 narrative

USE CASE ID:	3.1	USE CASE TYPE
USE CASE NAME:	Create person profile	Business Requirements: ☐
		System Design: ☐
PRIORITY:	High	System Analysis: √
SOURCE:	Capstone – Home Security System	
PRIMARY ACTOR:	Admin, advanced user	
DESCRIPTION:	The user is on **Web Dashboard Page** and clicks on the **People button** in a side menu bar. The **People Page** opens, and the user clicks on the **Create Person Button** and fills in the user details that the system provides. The used takes a picture or uploads the picture of the person to the system. The user can also change the status of person (white-list and black-list). The user then clicks on the **Done Button** and the system saves the person profile.	
PRE-CONDITION:	The user must be logged in.	
TRIGGER:	The user wants to add a person to the system so that the system can recognise them in the camera footage with an event response as a threat or friend.	
TYPICAL COURSE OF EVENTS:	**Actor Action**	**System Response** **Manual Action** **Automated Action**
	Step 1: User clicks on **People Button** in the **Web Dashboard Page.**	**Step 2:** The system displays the **People Page**.
	Step 3: The user clicks on the **Create Person Button**.	**Step 4:** System provides a **Create Person form** for the user to fill in**.**
	Step 5: The user clicks on **Take picture button**.	**Step 6**: System invokes **U3.1.1 Take picture**
	Step 7: The user clicks on the **White-List Button**	**Step 8:** invokes **U3.3.1 White-list Profile**.
	Step 9: The user fills in the rest of the person's details and clicks on the **Done Button**.	**Step 10**: System saves the person profile to the system.
ALTERNATE COURSES:	**Alt-step 5-6**: The user clicks on **Upload picture button** and the system invokes **U3.1.2 Upload picture.**	
	Alt-step 7-8: The user clicks on the **White-list Button** and the system invokes **U3.3.2 Black-list Profile**	
POST-CONDITION:	A new person is added to the system to be recognised and detected in camera footage.	

Figure 11: U3.1 narrative

USE CASE ID:	3.3	USE CASE TYPE
USE CASE NAME:	Update person profile	Business Requirements: ☐
		System Design: ☐
PRIORITY:	High	System Analysis: √
SOURCE:	Capstone – Home Security System	
PRIMARY ACTOR:	Admin, advanced user	
DESCRIPTION:	The user is on **Web Dashboard Page** and clicks on the **People button** in a side menu bar. The **People Page** opens, and the user clicks on the **Update Person Button.** The system opens the details and allows the user to change the details. The user then clicks on the **Done Button** and the system saves the changes to the person's profile.	
PRE-CONDITION:	A person must already exist on the system to update.	
TRIGGER:	The user wants to change an existing person's details in the system.	
TYPICAL COURSE OF EVENTS:	**Actor Action**	**Actor Action**
	Step 1: User clicks on **People Button** in the **Web Dashboard Page** in the sidebar menu.	**Step 2:** The system displays the **People Page**.
	Step 3: The user clicks on the **Update Person Button**.	**Step 4:** System provides a **Update Person form** for the user to edit.
	Step 5: The user makes the changes to the person's details and clicks on the **Done Button**.	**Step 6**: System saves the changes to the person profile onto the system.
ALTERNATE COURSES:		
POST-CONDITION:	An existing person's profile is updated in the system.	

Figure 12: U3.3 narrative

USE CASE ID:	5.1	USE CASE TYPE
USE CASE NAME:	Enable/Disable notifications	Business Requirements: ☐
		System Design: ☐
PRIORITY:	High	System Analysis: √
SOURCE:	Capstone - Home Security System	
PRIMARY ACTOR:	Admin, Advanced user, Basic user	
DESCRIPTION:	The user navigates to a side bar menu in **Web Dashboard Page** and clicks on the **Notification toggle button.** The notification system then **toggles** between a disabled state to or from an enabled state.	
PRE-CONDITION:	At least one camera footage is monitored, and the notification method/s must be set for event alerts	
TRIGGER:	User wants to enable or disable system event alert notifications.	
TYPICAL COURSE OF EVENTS:	**Actor Action**	**System Response** **Manual Action** **Automated Action**
	Step 1: User presses the **enable toggle button** next to the **Notification accordion menu** in **Web Dashboard Page**.	**Step 2**: The system notification toggles from disabled to **enabled** notifications.
		Step 3: System **listens to events** in footage detection.
		Step 4: System alerts user by **email** triggered by footage detections.
	Step 5: User responds to notification on **Web Dashboard Page.**	**Step 6:** System responds.
ALTERNATE COURSES:	**Alt-step 2**: The system notification toggles from enabled to **disabled** notifications.	
	Alt-step 4: System alerts user by **SMS** triggered by footage detections.	
POST-CONDITION:	System event alerts notifications are toggled from enabled or disabled or vice versa.	

Figure 13: U5.1 narrative

4.6 DOMAIN MODEL

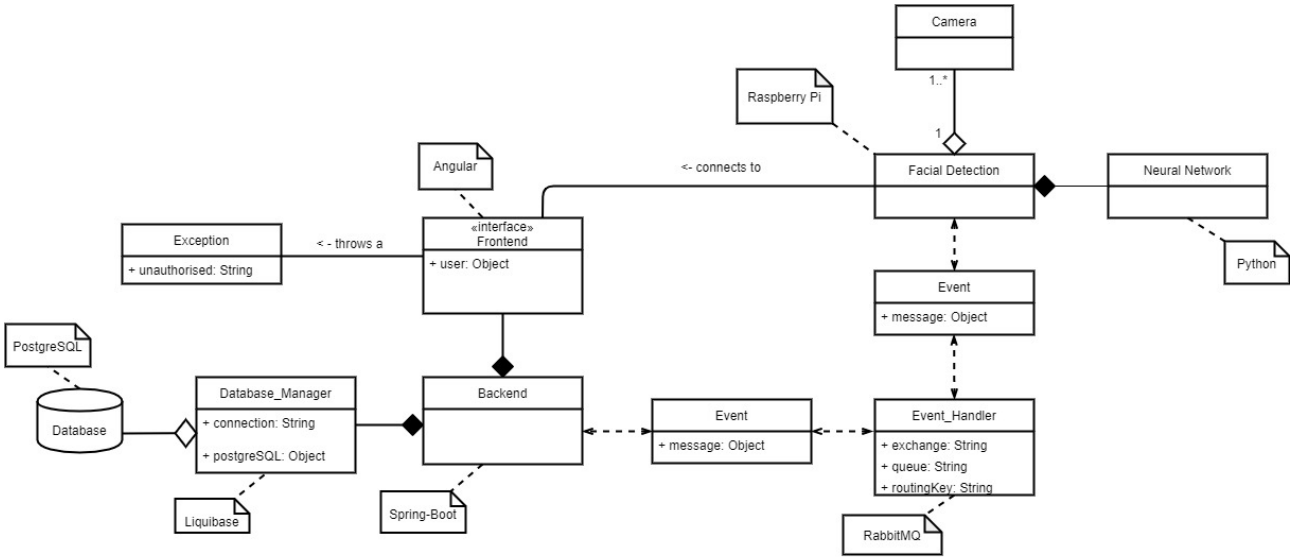

Figure 14: Domain Model

Description Shown in Figure 14, The PostgreSQL database is connected to the Database Manager which communicates CRUD operations through Liquibase. The Database Manager cannot exist without the Spring Boot Backend, which uses Hibernate to map CRUD operations and enforce persistence between models and repositories.Spring boot also contains configuration files security measures for Http requests. The Angular Frontend cannot exist without the Spring Boot Backend component and throws an Exception upon any errors encountered. The Event Handler component can take in a JSON object pushed from the the Backend's producer and matches the routing key with a queue along a direct exchange, after which that Object will either be sent to the Frontend or to the Raspberry Pi. The Raspberry Pi can also push a JSON object to the Backend when a new person is discovered or an alert needs to be made. The Raspberry Pi provides a running environment for the Python Neural Network and can have one or more camera attached to it. The Raspberry Pi connects to the Frontend to display the live camera feed and the facial detection from the Neural Network. The Neural Network cannot exist without the Raspberry Pi and does the facial detection and classification operations which causes an event to occur on the Raspberry Pi.

The Bridge design pattern was used to implement the Backend component, as the components making up the Backend implement the abstract concept of the Backend and the implementation can differ from the abstraction

The Decorator design pattern is used in the Person component, due to their being three different types of persons in the system. These different types are namely Whitelisted or Cleared people, Greylisted or Unknown people and finally Blacklisted or Threat people. The pattern allows varying and additional responsiabilities be attaheded or assigned to objects dynamically. Due to changes and features being done and assigned during runtime, the pattern is better than normal inheritance.

The Facade design pattern applies to the Frontend and Raspberry Pi components of the Argus system, as it is an simplified single interface to use the complicated and widespread backend interfaces. The Raspberry Pi is an interface to the neural network and the camera system, whereas the Frontend is an interface to the database, event handler, backend and Raspberry Pi.

The Proxy design pattern is used in the use of the Raspberry Pi to be a placeholder to the Neural Network and it also controlling the access the captured camera feed as well as the neural networks events.

The singleton design pattern was used in the implementation of the Database Manager, Email Handler and Neural Network. This is due to there only being one of each of these components and the assess and interaction to each of them are strictly controlled. The requests and events that are sent to the Event Handler and Database Manager are performed using the Chain of Responsibility and Iterator design patterns to ensure that the sender and receiver are not coupled, the requests and events are handled in a sequential manner and also that there can be error checking and security measures put in place.

Events are passed to different components in the system using the Command design pattern as the events are encapsulated in objects so that they can be queued, undone and prioritized.

The Mediator design pattern was used to separate the business logic from the implementation on the Backend's service layer.

The Memento design pattern was used on the Database to allow for undo-able operations, namely the "Restore" feature on the Frontend.

The Observer design pattern was used to implement the Frontend, Backend and Neural network. This is due to them being dependant on the Database and when an attribute is changed for a user or a person, a new user or person is added or removed, and an intruder is detected, the components dependant on the database need to be notified of the change.