# 

**ANDROLOGISTS**

# PROJECT: FACIAL RECOGNITION SYSTEM FOR COMMUNITIES

**ZANE BLOOM – 12236722**

**HEELIN MISTRY – 10299344**

**VERUSHKA MOODLEY – 29117454**

**GERARD LAMUSSE – 12206050**

**23 May 2014**

Contents

[Vision and Scope 3](#_Toc388612732)

[Project Background and Vision 3](#_Toc388612733)

[Scope and Limitations/Exclusions 4](#_Toc388612734)

[Scope 4](#_Toc388612735)

[Exclusions/Limitations 4](#_Toc388612736)

[Architectural Requirements 5](#_Toc388612737)

[Access and integration requirements 5](#_Toc388612738)

[Access channels 5](#_Toc388612739)

[Quality requirements for access and integration channels 5](#_Toc388612740)

[Architectural responsibilities 5](#_Toc388612741)

[Quality requirements 6](#_Toc388612742)

[Security 6](#_Toc388612743)

[Testability 6](#_Toc388612744)

[Usability 6](#_Toc388612745)

[Scalability 6](#_Toc388612746)

[Performance requirements 7](#_Toc388612747)

[Architectural Constraints 7](#_Toc388612748)

[Software Architecture 8](#_Toc388612749)

[Architectural Requirements 8](#_Toc388612750)

[Architectural scope 8](#_Toc388612751)

[Integration and access channel requirements 8](#_Toc388612752)

[Architectural Patterns/Styles 8](#_Toc388612753)

[Architectural Tactics/Strategies 8](#_Toc388612754)

[Reference architectures and frameworks 9](#_Toc388612755)

[Access and channels 9](#_Toc388612756)

[Access Channels 9](#_Toc388612757)

[Protocols 9](#_Toc388612758)

[Technologies 9](#_Toc388612759)

[Core Components Diagram 10](#_Toc388612759)

# Vision and Scope

## Project Background and Vision

Quant Solutions requires an automated facial recognition system that will be implemented in communities/neighbourhoods. The main purpose of this system is to identify those people that are regulars in the neighbourhood and those that aren’t. This system will also allow for story verification (if a person is accused of a crime in the neighbourhood, this system should place him/her at the scene).

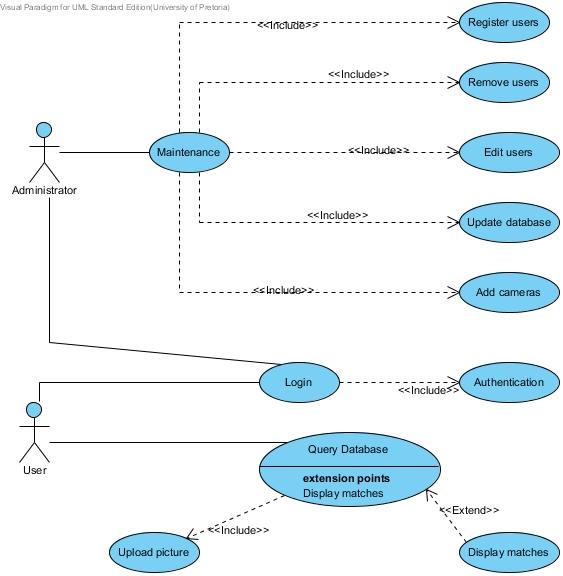
Quant Solutions already has 32 cameras placed around the neighbourhood in different street corners. An individual process will be started for each camera. This process will capture, process and persist images from the camera.

The system will also consist of a web application to query the backend system. The web application will be used to take a picture of the suspicious individual, and the picture will be compared to the data stored in database in the backend. The web application will then display possible matches and the user can scroll through these results and form a conclusion about the suspicious individual.

## 

## Scope and Limitations/Exclusions

### Scope



### 

### Exclusions/Limitations

* The system must detect frontal faces.
* The system is only required to work during the day (for lighting reasons).
* The system is not expected to be 100% accurate.

# Architectural Requirements

This section discusses the software architecture requirements that encompass the software infrastructure within which the application functionality is to be developed. The purpose of this infrastructure is to address the non-functional requirements. In particular, the architecture requirements specify

* The architectural responsibilities which need to be addressed
* The access and integration requirements for the system
* The quality requirements
* The architecture constraints specified by the client

## Access and integration requirements

This section discusses

1. The requirements for the different channels through which the system can be accessed by the people and the system
2. The integration channels which must be supported by this system.

### Access channels

The system will be accessible through the following channels:

1. The facial recognition will be made available through Restful web services.
2. A web-app frontend will act as the interface between the users and the web services.

### Quality requirements for access and integration channels

* All communication of sensitive data must be done securely using HTTPS. All other data can be communicated using HTTP.
* Connect to cameras via either HTTP or RTSP.
* FTP server.
* Postgre - message-based-protocol over TCP/IP
* Web App with web services

## Architectural responsibilities

The architectural responsibilities include the responsibilities of providing an infrastructure for

1. A web access channel
2. Hosting and providing the execution environment for the services of the system.

## Quality requirements

The quality requirements are the requirements around the quality attributes of the system and the services it provides. This includes performance, scalability, security, auditability, usability, and testability requirements.

### Security

1. A management portal is needed to ensure only verified members of neighbourhood watch and police are allowed access.
2. All system functionality should be only accessible to users who can be authenticated.
3. The system must make certain that any operation to be performed on any data is only allowed if that user may use the requested operation on the requested data object. Some services only require role based authorization for the service itself. However, the system must be able to constrain who is allowed to do what. For example, the administrator can add/remove users, but other users may not be allowed to add/remove users.

### Testability

All services offered by the system must be testable through unit tests which test

* That the service is provided if all pre-conditions are met (i.e. that no exception is raised except if one of the pre-conditions for the service is not met)
* That all post-conditions hold true once the services has been provided.

### Usability

1. 90% of the users should be able to use the system without prior training.

### Scalability

1. The system must be able to scale to 32 cameras, however multiple machines can be used to achieve this, thus the system needs to be scalable across multiple machines.
2. The deployed system must be able to operate effectively under the load of 20 concurrent users.
3. The software architecture should be such that it can, in future, be easily modified to scale by porting the system onto clustered and cloud-computing based architectures.

### Performance requirements

This system does not have particular stringent performance requirement, however, the data to be processed should not back up to a position where it will not be able to reach, due to new data to be processed incoming all the time.

1. Important to process video streams in real time.
2. All operations, not regarding the processing of images, should respond to less than 1 second.
3. Processing of images:
   1. Less than a second to detect a faces in an image.
   2. Less than a second to crop face from image, and store in a temp folder.
   3. Less than 10 seconds to recognise faces from temp folder:
      1. If face not recognised, store face dimension data in database.
      2. If face recognised, log data of location face was detected at.
   4. Less than a second to move face from temp folder to a permanent location.

## Architectural Constraints

1. The system services must be made available to smart devices.
2. The system must use the *PostgreSQL* database.
3. Web services must be published as Restful-based web services.
4. Must make use of C++ and OpenCV for image processing.
5. Must be executable on Linux.
6. Must offer web services to smartphones and desktops.
7. Must only open source libraries.
8. Can be deployable on raspberry clients.

# Software Architecture

## Architectural Requirements

### Architectural scope

Provisions:

* Web-app access channel
* Capture images from cameras
* Persist facial images and metadata of the image to database
* Facial Recognition (Comparing faces)
* Infrastructure for process execution
* User interfaces with login, maintenance and functionality to query databases.

### Integration and access channel requirements

The system must provide a web access channel, which will be mainly used by the administrator, as well as a mobile access channel, which will be used by the other users.

## Architectural Patterns/Styles

The system will be built using the Pipes and filters architectural pattern. Each filter will be a stage in the pipeline that does a portion of the processing of an image from a video feed. The system must be flexible in allowing

The Capturers (Pump/Producer):

* Stream capturing: capture images from a video stream.
* Directory capturing: capture images stored in directory.

The filters:

* Sampling: Sample the video stream for frames.
* Detection: Detect faces in images.
* Image Pre-processing: pre-process images to ensure more accurate results.

The Persister (Sink/Consumer):

* Local persistence: store the images locally on the same machine

## Architectural Tactics/Strategies

1. Reduce coupling by using pipes and filters and decrease dependencies between modules of the system.
2. Increase Cohesion by giving each module a single responsibility.

## Reference architectures and frameworks

The system will be built using the Qt application framework. This framework offers features such as Qt SQL for database persistence, qmake build tool for cross platform deployment, Qt Test for testing and Qt network for network functionality. These features offered by Qt are the reason for it being the chosen framework.

The system will also make use of the OpenCV framework. It offers many features such as facial detection and facial recognition that the system will make use of. This framework was recommended by the client.

## 

## Access and channels

### Access Channels

1. The facial recognition will be made available through Restful web services.
2. A web-app frontend will act as the interface between the users and the web services.

### Protocols

1. SOAP: The mobile application will use SOAP to communicate data with the web server.
2. HTTPS: The web front end must use the HTTPS protocol to communicate data. This ensures secure transmission.

## Technologies

* The C++ and Java Programming languages will be used.
* X86 servers will be used.
* Vivotek 8332 cameras will be used.
* Linux and Windows operating Systems will be used.

**Core Components Diagram**