Face Recognition System
Design Document

Authors: Walid Balegh, Jakob Heyder, Sarpreet Singh Buttar, Henry Pap and Oscar Maris
Semester: VT 2017

Contents

1 Introduction

The FRS (face recognition system) design document is intended to be the groundwork of how the software should be built.

1.1 Purpose

The design documentation of the FRS is intended to create a mutual understanding of the software's functionality and contents. This document should contain all necessary information to start development of the face recognition system. This includes all known technologies, pitfalls and the architecture for the FRS as a whole.

1.2 Priorities

Security: As we are dealing with users private information it imperative that the system is both secure and has limited access for both users and administrators. The primary methods to achieve this will be: The use of HTTPS SSL/TLS and both admin and user authentication.

Reliability: When the user requests a PN from the server there needs to be a certain guarantee that that the PN returned matches the image provided.

Portability: Creating a application that uses the FRS should be both possible and simple for a variety of devices and there operating systems. It should also be reliable on a number of different operating systems that come with these devices.

Usability: The system should have clear defined methods and structure that will not be misinterpreted. The user application developed, should also be user friendly with the user interface having clear definitions.

1.3 Overview

This design document is for the face recognition software. The FRS purpose is to provide a PN (swedish Personal Number) using a comparison of a user provided image of a person's face. There should also be authentication to both users and administrators in place. Administrators should also have access to a list of the PNs from the database. Administrators are also able to add, edit and delete entries from the database. Additionally to the FRS a user application will also be developed to present the user functionality of the system. Said application should include the ability to authenticate the user and request PNs using images.

2 Major Design Issues

In this section all major design issues and decisions will be discussed. Rational and alternatives will be presented and the detail of evaluating the alternatives depends on the trade off the decision has.

2.1 Architectural Design

As desired from the customer the system will have a Client-Server structure. Where the in section **??** of the Requirements Elicitation defined UAM and AAM are the client modules and the URM and ARM are the server modules. Further we may reference URM or ARM as Server and UAM and AAM as Client. It is important to note that as defined in the requirements Req. Doc. **??** our developed product is only the Server with a clear API and

the Client application we talk about is the desired example client to show how integration and usage could be done. The architecture of the server is defined in detail in 4.

2.2 Languages & Frameworks

2.2.1 Server-side

As choice of programming language we choose the **Java programming language**. It is one of the most widely used ones on the server side. Especially the in **??** of the Requirements Elicitation defined REST functionality is supported by various frameworks with large communities. Rationals which led to the choice are listed below.

- Great flexibility & portability due to platform independence with the JVM.
- Scalable solution based on proven enterprise solutions
- High productivity because of existing frameworks and solutions
- Good support by a large community of developers

The Spring Framework will be used on the server side of the system. More detailed the Spring Web, Security, Data and Boot modules. This modules support the The Spring Framework allows fast, enterprise scale development of applications by providing features for security, RESTful applications and Data Management and thus is the perfect fit for the requirements defined in the Requirements Elicitation **??** and **??** such as authentication, encryption and REST functionality. The Rationals for using the different modules are listed more detailed below.

- Great Portability and Integration - it is supported by various cloud providers to make deployment and continuous development possible.
- Spring Boot provides an embedded application server which allows fast and easy setup of an application.
- Configurability - Spring is easily set up and gives good default solutions but also provides the possibility to configure the details to the application needs
- The Spring framework provides RESTful support which is asked for from the customer and fills the application needs.
- Spring Data provides a convenient way to implement CRUD functionality for accessing and modifying data. It supports various Database technologies such as JPA and generates boilerplate code at run time which reduces developing costs.
- Spring Security provides enterprise ready security features for authentication and encryption without much setup.

2.2.2 Client-side

The client application as defined in **??** of the Requirements Elicitation has to be working on mobile or web browsers. It is therefore suitable to make a web browser interface which is also responsive on mobile. Thus the application will be able to run on iOS, Android, Microsoft phone and WEB. Alternatively it could be developed with one of various Cross Platform Mobile development tools out there. Some of them are listed below.

- **Xamarin** - which is the most popular choice, a free trail is available and it use the language C#. This will make it more structured as C# is an *OO* language.
- **Phone Gap** - which is the most well known tool, it is open source meaning that it is free. It uses the common web languages to create hybrid apps i.e. HTML, CSS, JavaScript.
- **appcelerator** - lets developers use JavaScript to build their apps, provide mobile testing, it has a GUI to create design (which uses common HTML and CSS), and lastly it is free.

There is plenty more but since we only want to show the usage of the API by example as written in the Requirement Elicitation **??** we will not need a native application. Using common WEB languages to create a browser alternative fulfills the requirements. No requirements constrain the decision among various alternatives for the Web development. Thus this will be discussed in the next section 3

2.3 Communication Technology

2.3.1 Communication

The Communication will be over IP/TCP to have reliable transport and uses HTTPS on the application layer. This ensures general security by using SSL/TLS and ensures data integrity and privacy by authenticating the application. It provides the encryption and security defined in **??** of the Requirements Elicitation.

2.4 External Services

2.4.1 External Face Recognition API

One of the core features as defined in **??** of the Requirements Elicitation is to match a photo of a face with an existing one to a certain factor of equivalence/similarity. For this purpose as defined in the requirements an external service will be used. Following we compare some of the most known ones. Important aspects are cost factor, usability for the specific needs and security support. There are some External Face Recognition API out there such as

- *SkyBiometry:* It is a cloud-based face detection and recognition software which provides a high-precision biometric identification for over 20 years. In addition, it also provide API client libraries in various languages such as Java, C#, Pyton etc for giving a quick start to the developers. Regarding the usage limits, it has a free subscription which allows 100 methods calls hourly and 5000 monthly. It provide SSL support and its API uses REST interface which means all the API methods are called over the Internet using standard HTTP methods and responses are generated in XML or JSON.
- *Lambda Labs:* It permits developers to send an image link to their service for the identification. In addition, it also allow to create an album of photos, analyze and compare new images with existing ones. Regarding the usage limits, it has a no free subscription and minimum cost is $9/month. It does not provide SSL support and its API also uses REST interface and responses are generated in JSON.

- *OpenFace:* It is a open source web service which provide facial detection technologies. Its API uses REST interface and accept image from the developer and return a JSON response. Currently, it does not support SSL and can only detect up to 80 points on a given image.

We have found that *OpenFace* does not provide sufficient functionality as compared to others. Wheres *Lamba Labs* does provide needed functionality but with a cost of $9/month. In result, ***SkyBiometry*** is the free and suitable option for our project.

2.5 Application specific

Authentication as defined in **??** of the Requirements Elicitation will be done by providing a username and password for registered Users and Admins. The Registration will be exclusive over a non automated channel by contacting the Customer/Developers to verify a service. The in **??** the Requirements Elicitation mentioned credentials refer further to a user name and password.

2.6 Data format

The data will be formatted in standard JSON for communications between the server and the client. The format is human-readable and widely supported. It also supports the requirements of a RESTful application.(Req.Elic. **??**)

3 Technology choices

In this section we will discuss technology choices which are not constrained by the requirements and thus do not belong directly into the design space defined in 2

3.1 Client technology

In this section we will shortly discuss the technologies used for developing the WEB Client. *It is important to mention that this is not a client in the classical sense as that the browser is the communicating instance and the client only consists of the pages served from a web-server. So it is more only a user interface to access the API.*
We will use responsive design to style our app so it will be desktop, tablet and mobile friendly. For this we will use a CSS framework, all CSS frameworks comes also with a JavaScript framework for design purpose (animation etc.). Further a list of recent CSS frameworks:

- **Bootstrap** - the most common framework out there, easy to use and creates fast design. Great for dynamic designing thanks to its grid system. The major drawback is that it will look boring and old.
- **Material Framework** - Google's own framework, a google look alike framework.
- **Semantic UI** - a fresh framework that has grown quite popular in the last couple of years. It uses the JavaScript jQuery framework which is easy to use and has great AJAX calls which can be helpful.

This is only a design option and we will go with the Semantic UI because it has a suiting design, and for the use of jQuery.

To summarize The framework that will be in used for WEB development is Semantic UI for CSS and jQuery for JavaScript as it comes with Semantic UI.

3.2 Development Platforms

3.2.1 Platform Server

The Server will run on a cloud platform. This gives several advantages which are listed below. Especially easy setup and management are essential for this project during development and by using Java the components are platform independent which allows later changes during production.

- Easy to manage and setup - no System administrator needed
- Allows fast and continuous development and testing
- Cheap and scalable solution

For the development in the cloud, Heroku is among AWS, Microsoft Azure, Google application engine and others a common choice. It supports good conditions for development and support frameworks for features such as database deployment. This gives a convenient way to get the system fastly up and working. Listed are features it provides.

- Native support for Java and Spring Boot application deployment
- Addon support for rational Databases (e.g. MYSQL)
- Github integration for continuous development
- Free use for small scale applications (development)

3.2.2 Platform Client

The Client, more specific the in the to be developed UAM will be Web compatible. Since it is written in Javascript for mobile development it runs on iOs, Android and every system supporting modern web browsers.

3.3 Database

The database will be MYSQL a rational database. It is one of the most used rational databases and therefore provides sufficient features, support and scalability for the application. It is also compatible with the used Spring framework and the Heroku cloud platform. It validates data and ensures integrity.

4 Architecture (Component Diagram)

The subsystems of our system must be reusable and independent from each other, as that will make it easier to implement and test. The different subsystems or components are then combined by the application (which is our system). We define the different components in the component list, then we explain each of them briefly.

Component list

- authenticate
- admin
- user
- face
- database
- storage
- skyBiometry
- imgur
- MySQL

Component details Now we will discuss why we need the different components, how they will be used and what they will require.

1. authentication The authentication component is required for security and divides the system based on whether the connected client is an user or an admin **??**. The required interface for authentication is the database component, the provided interface is a login functionality that returns an integer 0 - no match, 1 - user and 2 - admin and a check which checks if current client is logged in (user or admin).

2. admin The admin component is a core component that will hold all the admin responsibilities which is management of client users (faces). The required interface is the face, storage and database component. The provided interface is the CRUD methods check **??** at line 3.3.4.

- CREATE : create a new user
- READ : get the user information by id or all.
- UPDATE : updates an existing user by id
- DELETE : removes an existing user by id

4. user The user component is also a core component that will be responsible of getting a Swedish social security number based on a image check **??** in the requirements elicitation. The required interface is the face, database and the provided interface is "*uploading a picture*".

4. face The face is the main component of this system (FRS), this needs to be as independent as possible in order of our system to be adaptive to changes (i.e. face-recognition algorithm may be replaced in the future for a better one). The required interface that is needed is skyBiometry (the face-recognition api) and the provided interface is CRUD methods **??**, which allows admins to manage client faces as well as letting user component get a PN based on a picture, see **??**.

5. database The database or actually database-access component is the component that will directly talk with the remote database. The required interface is a MYSQL-database since it is flexible and easy to use, but can easily be replaced. The provided is CRUD methods as well as a login method. This is a nessesary component check **??** in the requirements elicitation: "*Personal number database*" will also hold information of login-users.

6. storage The storage component is the component that will save the client-face picture to the storage, see **??**, the "*Image Database*" now we will use imgur.com api to store pictures as space is an issue but in future this can be changed into storing images directly in our server. The required interface is imgur.com-api and the provided interface is upload picture.

7. skyBiometry The face-recognition algorithm api, we will use skyBiometry as discussed earlier.

8. imgur The imgur or imgur.com-api will be used by our storage component.

9. MySQL MySQL is the database type we will use by the database-access component.

Architecture Pattern We already mentioned that the system will have client-server structure in **??**, but now we focus only on the server part as it is our only implementation part, clients then can connect and use our service the server provide. We want good reusability, good cohesion, low coupling, abstraction, security and portability all this because components should be easily changeable, independent so development can be as well and portable so the system can easily be changed to other servers (in the future). Our system is a layer type where the application (which will act as the glue code) will use authentication, admin and user component and these will use the others. See the component diagram to clearly see why it will be a layered system. The patterns that meets our required types i.e. cohesion, coupling, abstraction and portability is:

- Multi-layers
- Service-oriented

The Service-oriented pattern focus on the communication between the different web-services, in our case is with MySQL database, imgur.com api and skyBiometry. This will use standard communication HTTP which in the core basics is XML as the pattern says it should be (open standards). This is a very good pattern to follow when it comes to our remote components.

The Multi-layers pattern focus on how the communication should be, i.e. layer can only speak with the layer below it, in our case the different components can only speak with the required components and not the other way around (see component diagram to better understand). This is also a very good pattern for our system that focuses on the structure of the system.

Conclusion We will use a combined pattern, Multi-layers pattern (for developed component) and Service-oriented pattern (for communication with remote components).

4.0.1 Diagrams

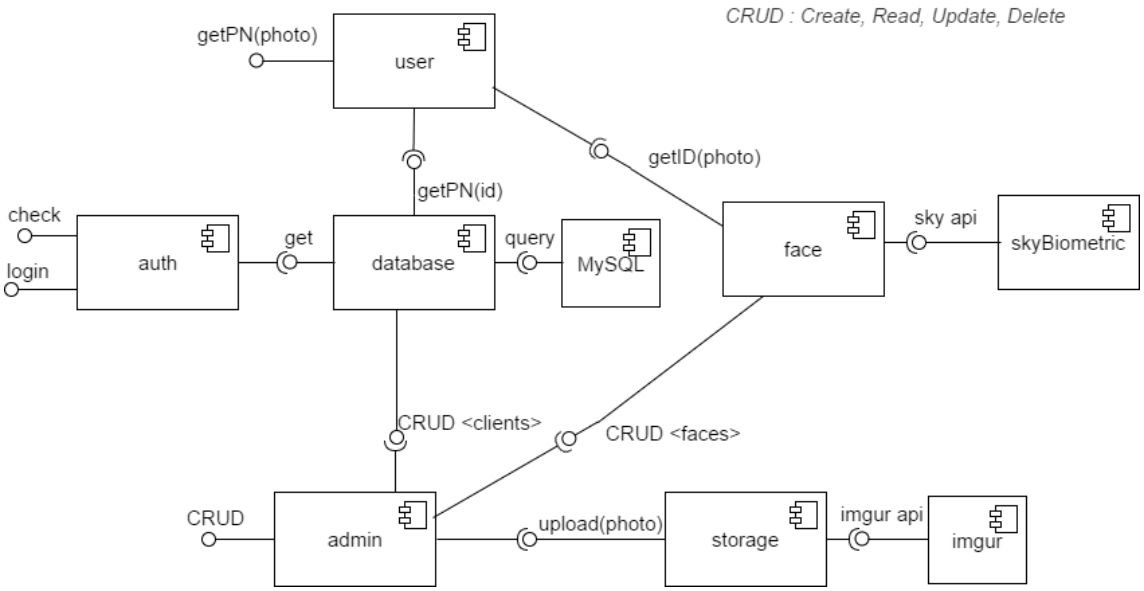

Figure 1: ComponentDiagram

Component Diagram

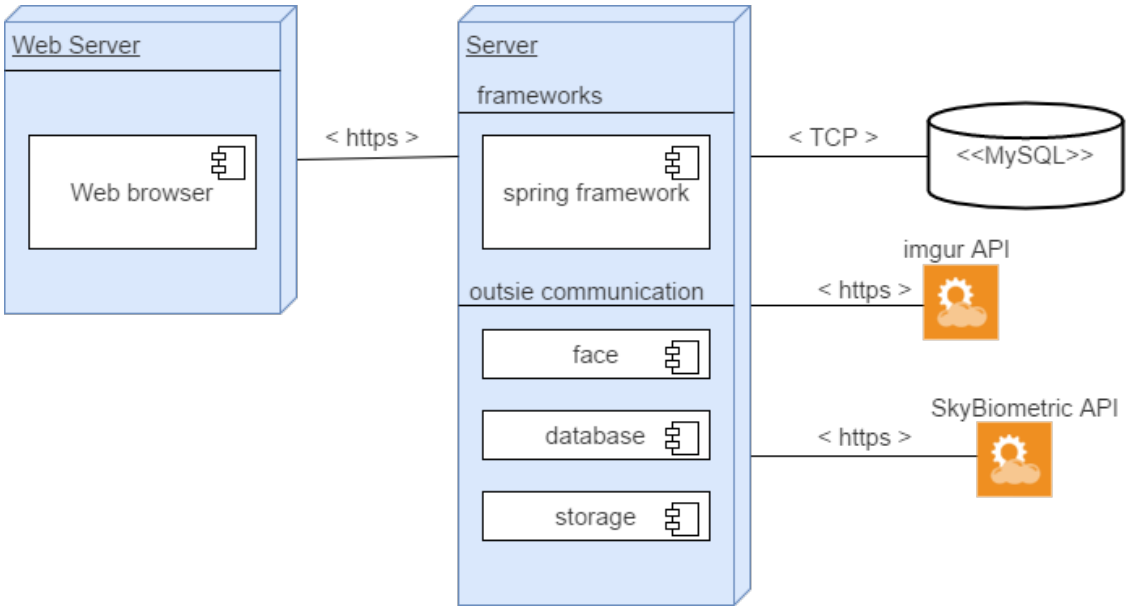

Figure 2: DeploymentDiagram

Deployment Diagram

5 Components - Static modeling (Class Diagrams)

6 Use cases - Behavioral modeling (Sequence Diagrams)

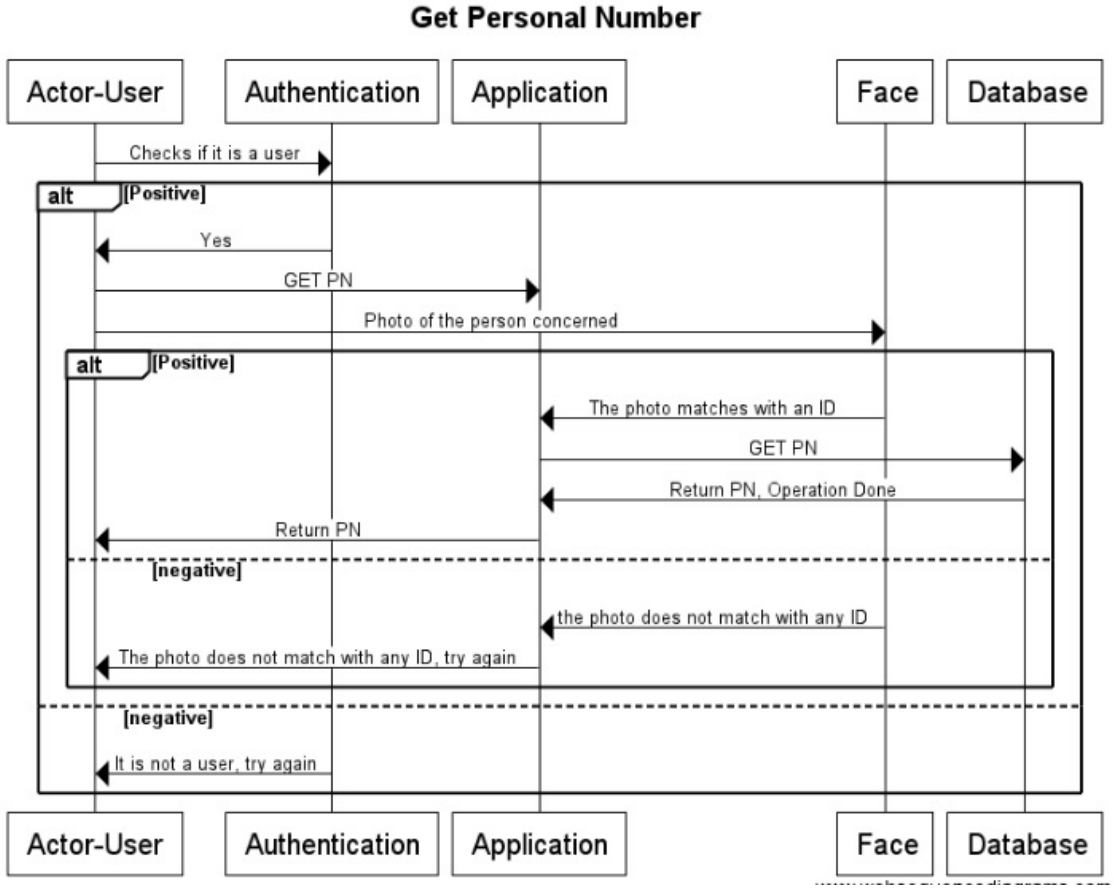

Figure 3: Sequence diagram referred to "5.2 UAM: GET Personal Number" in the requirement document.

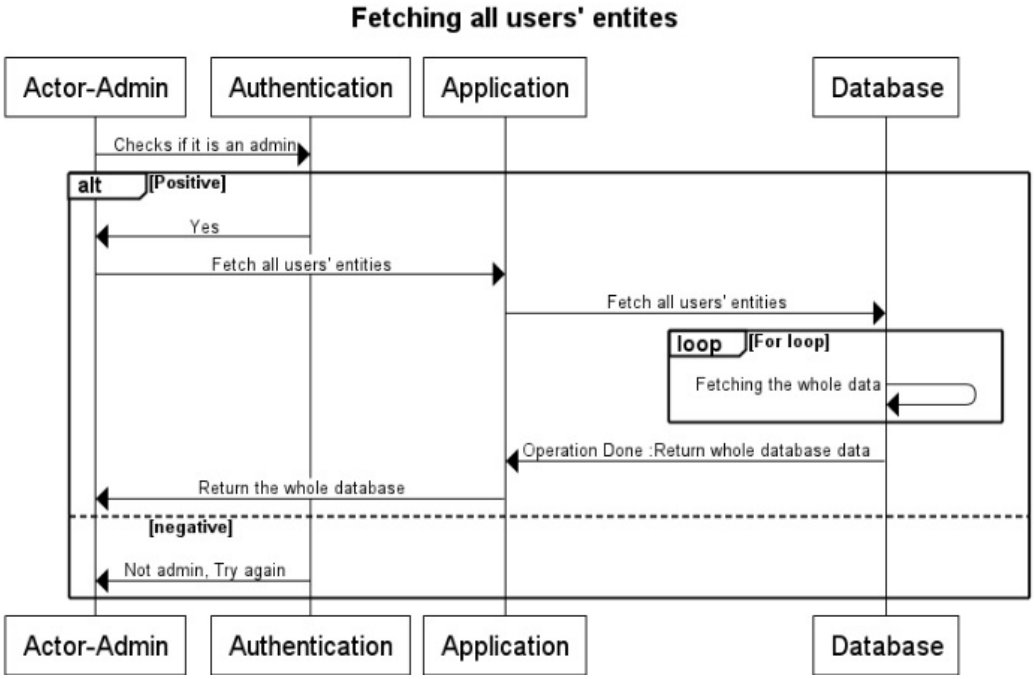

Figure 4: Sequence diagram referred to "5.3 ARM: Get List of User-Entities" in the requirement document.

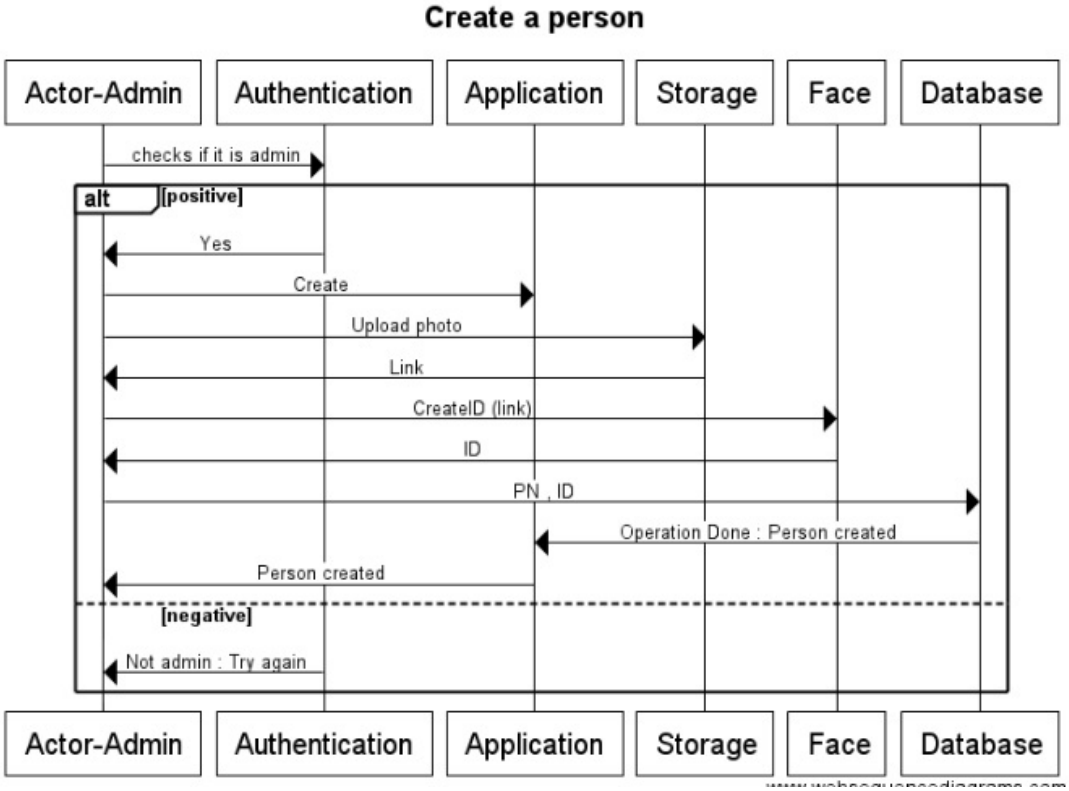

Figure 5: Sequence diagram referred to "5.4 ARM: Add an User-Entity" in the requirement document.

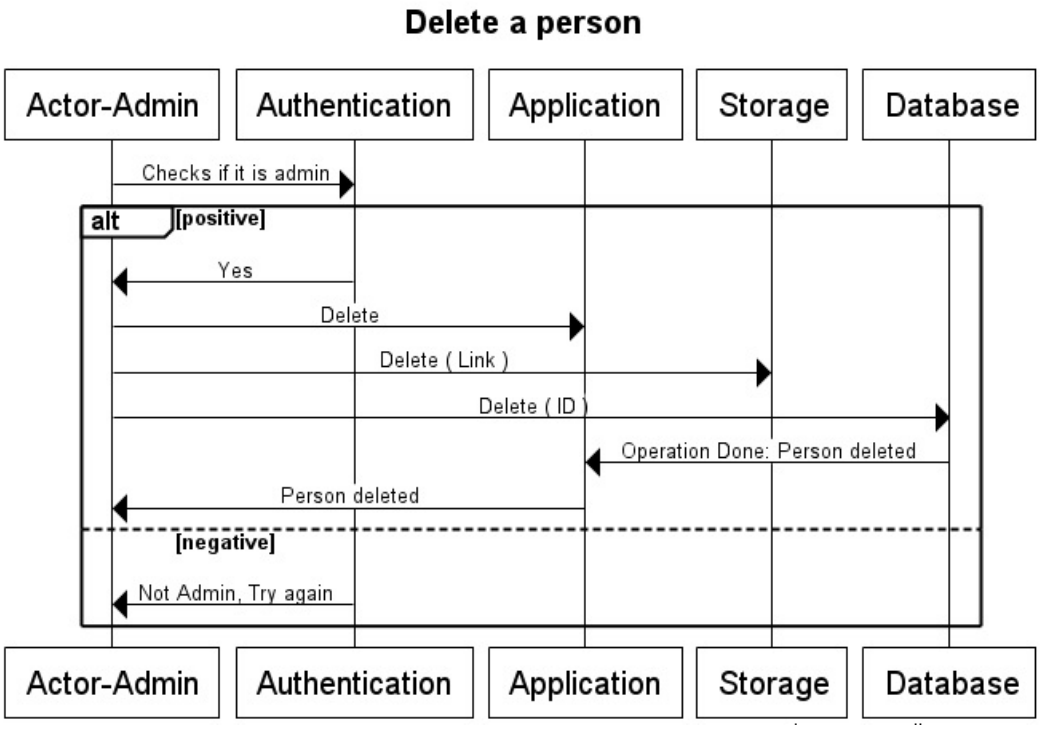

Figure 6: Sequence diagram referred to "5.5 ARM: Delete an User-Entity" in the requirement document.

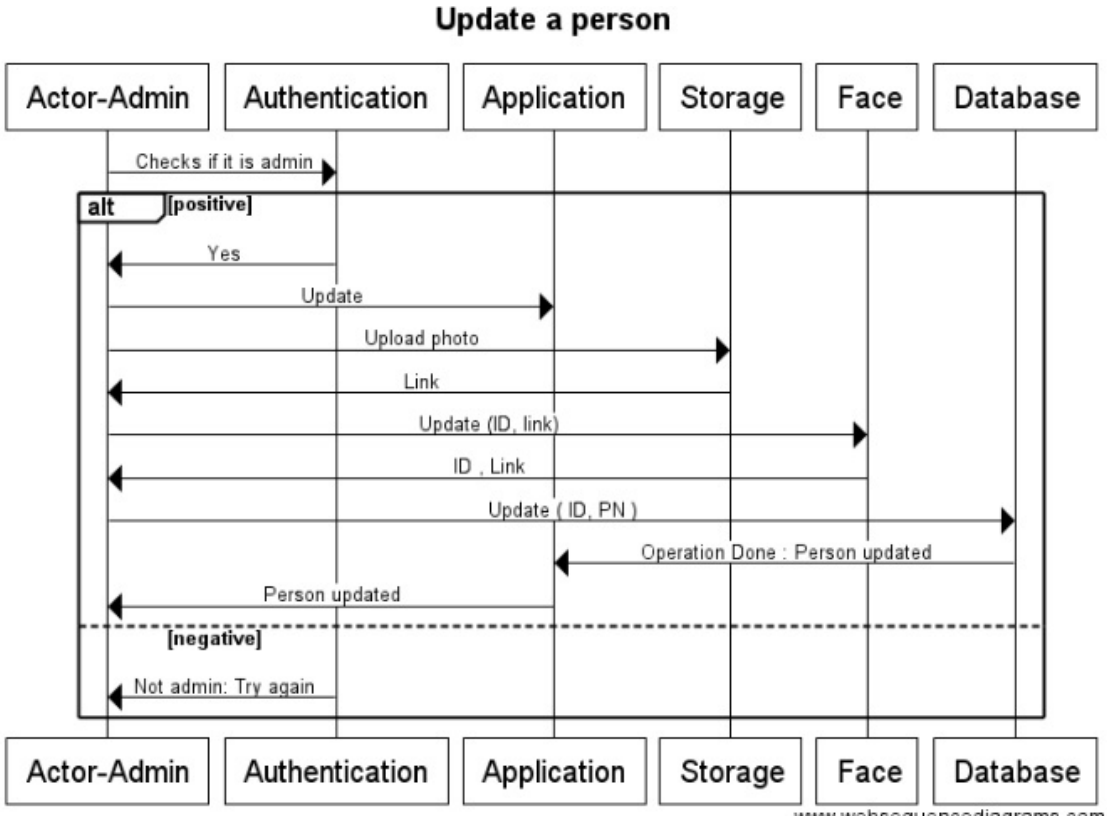

Figure 7: Sequence diagram referred to "5.6 ARM: Update User-Entity" in the requirement document.

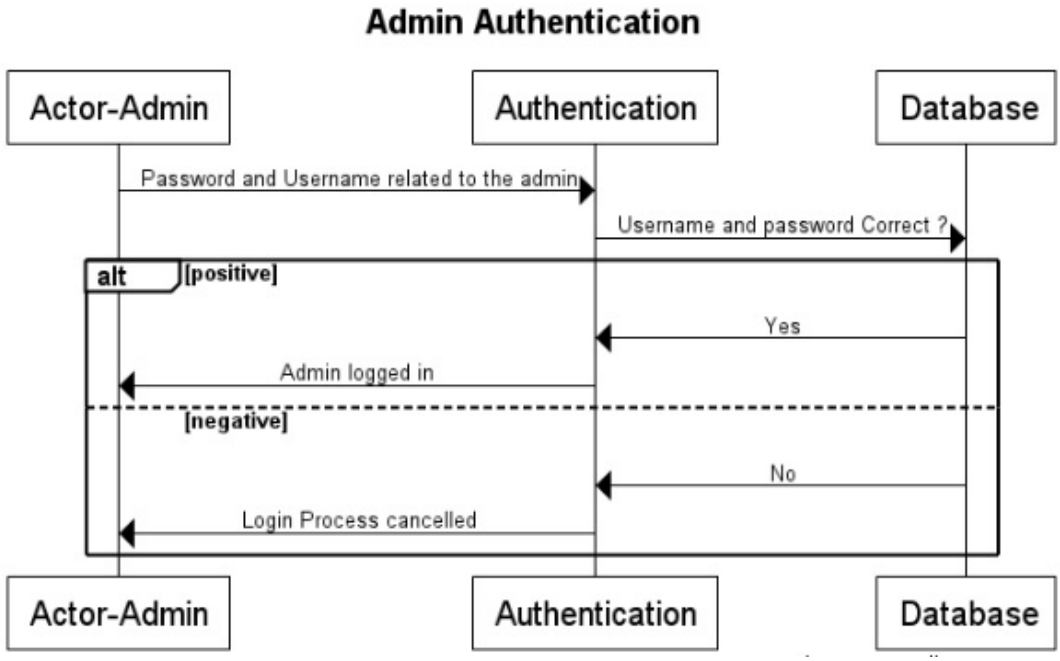

Figure 8: Sequence diagram referred to "5.7 ARM: Authenticate Admin" in the requirement document.

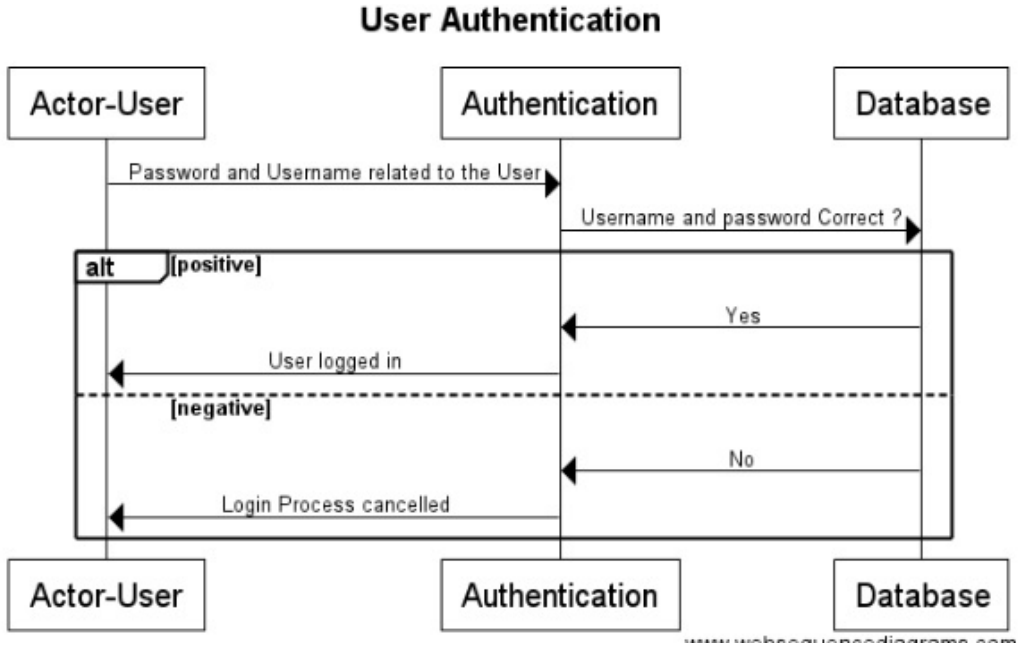

Figure 9: Sequence diagram referred to "5.8 URM: Authenticate User" in the requirement document.