System Requirements Specification

Project: Electronic ID Verification

Team: Java the Hutts

2017

Nicolai van Niekerk
nicvaniek@gmail.com

Marno Hermann
marno@barnton-consulting.co.za

Stephan Nell
nellstephanj@gmail.com

Jan-Justin van Tonder
J.vanTonder@tuks.co.za

Andreas Nel
nel.andreas1@gmail.com

Contents

1 Introduction

This chapter aims to give a description, as well as an overview, of the content of this document. Additionally, it will include any terms, abbreviations, acronyms and references used throughout this document.

1.1 Purpose

The purpose of this document is to present the reader with a detailed description of the Electronic ID Verification system. It will delve into the purpose and features of the system, the various interfaces of the system, the capabilities of the system, as well as the constraints under which the system must operate. The content of this document is intended for both the various stakeholders and the developers of the Electronic ID Verification system.

1.2 Scope

The Electronic ID Verification is proposed as a standalone system that provides the core functionality of extracting client details from an image of some form of ID, and comparing existing client information with that which was extracted from the image of the ID.

1.3 Definitions, Acronyms, and Abbreviations

Term	**Definition**
OCR	Optical Character Recognition
ID	Identification Document
API	A set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service.
Linux	A Unix-like computer operating system assembled under the model of free and open-source software development and distribution
Python	Python is a widely used high-level programming language for general-purpose programming.

1.4 Overview

The remainder of this document will consist of two chapters.

The second chapter will address the overall description of the Electronic ID Verification system. In addition to this, the second chapter will also describe the context of the Electronic ID Verification system, its relations and the potential interfaces with other systems. This chapter will also provide a summary of the functions of the Electronic ID Verification system as well as consider the numerous user characteristics, constraints, assumptions and dependencies relevant to the system.

The third chapter serves the purpose of describing the software requirements of the Electronic ID Verification system. This chapter will address the External Interface Requirements, Functional Requirements, Performance Requirements, Design Constraints, Software System Attributes and any other requirements not previously explored.

2 Overall Description

This chapter aims to give an overview of the entire Electronic ID Verification system. The system will be contextualised in order to demonstrate the basic functionality of the system, as well as demonstrate how the system interacts with other systems. It will also describe the levels, or types, of users that will utilise the system and describe the functionality that is available to said user. At the end of this chapter, the constraints and assumptions for the system will be addressed.

2.1 Product Perspective

The system is designed in the form of a Web API, which hosted on an independent server.

The application focuses on two main criteria. First, the system is able to extract the following from the either an ID book or ID card:

- Name
- Surname
- South African ID Number
- Nationality
- Country of Birth
- Status
- Gender
- Date of Birth
- Face

The application then takes the information that was extracted and compares it to an existing profile followed by percentage match scores for each corresponding value which is returned

When comparing the face for a similarity percentage, the application deals with problems like an old photo or changed facial features, such as a beard, by applying facial landmarks to ensure that highest accuracy is ensured when a comparison is being performed.

The second feature is responsible for extracting the information from a given ID photo and returns the collection of the information that the system managed to extract.

2.2 User Characteristics

The intent of the Electronic ID Verification system is to function as a Web API, thus, the users will require some knowledge regarding restful programming. Electronic ID Verification is platform independent, therefore, users require no platform specific knowledge.

2.3 Constraints

Only a valid South African ID book or South African ID card can be used with the application.

People's facial features could change or the photo provided could be very old, or the individual in the photo could be standing skew and not providing a full frontal, facial image.

Low DPI, lighting, noisy backgrounds and resolution of images can drastically affect performance.

2.4 Assumptions and Dependencies

- It is assumed that enough resources will be provided to test multiple forms of ID, such as ID cards and ID books.
- It is assumed that a clear, well-lit and sufficiently high DPI image is to be provided to the system.
- It is assumed that all identification documentation follows the relevant, fixed format of documentation issued in South Africa.

3 Specific Requirements

This chapter addresses all the functional requirements of the Electronic ID Verification system. It gives a detailed description of the system and all its features.

3.1 External Interface Requirements

3.1.1 User-interfaces

The system is implemented as a Web API that is called over a network. As a result there are no (graphical) user-interface requirements.

3.1.2 Hardware Interfaces

No extra hardware interface is needed, since the application is designed as a Web API and is detached from any hardware requirements.

3.1.3 Software Interfaces

Since the system will be implemented as a Web API, it will interface with the existing systems of the client.

3.1.4 Communication Interfaces

Communication is an important aspect of an API to function correctly. The application must be able to pass the correct information to any system that makes use of the application functionality. It is also important that descriptive responses are provided in case of an error, or if the need arises to provide extra information.

3.2 Functional Requirements

This section describes the functional requirements of the system. The requirements are derived from the specific use cases that are modelled using use case diagrams. Non-trivial use cases are also further elaborated upon using Actor-System interaction.

1. **FR-01:** The system must be able to accept the relevant input data and a(n) ID card/ID book in a specified format.
2. **FR-02:** The system must be able to extract text from the provided image.
3. **FR-03:** The system must be able to extract the photo from the image.
4. **FR-04:** The system must be able to detect a face from the extracted image.
5. **FR-05:** The system must be able to align a face for better matching.
6. **FR-06:** The system must be able to compare the extracted text with provided data.
7. **FR-07:** The system must be able to compare two faces after image processing.
8. **FR-08:** The system must be able to give a percentage match on the validated data.
9. **FR-9:** The system must be able to perform extraction on a South African ID book.
10. **FR-10:** The system must be able to perform extraction on a South African ID card.
11. **FR-11:** The system must allow a user to specify a matching accuracy threshold.
12. **FR-12:** The system must allow a user to specify preferences of extraction

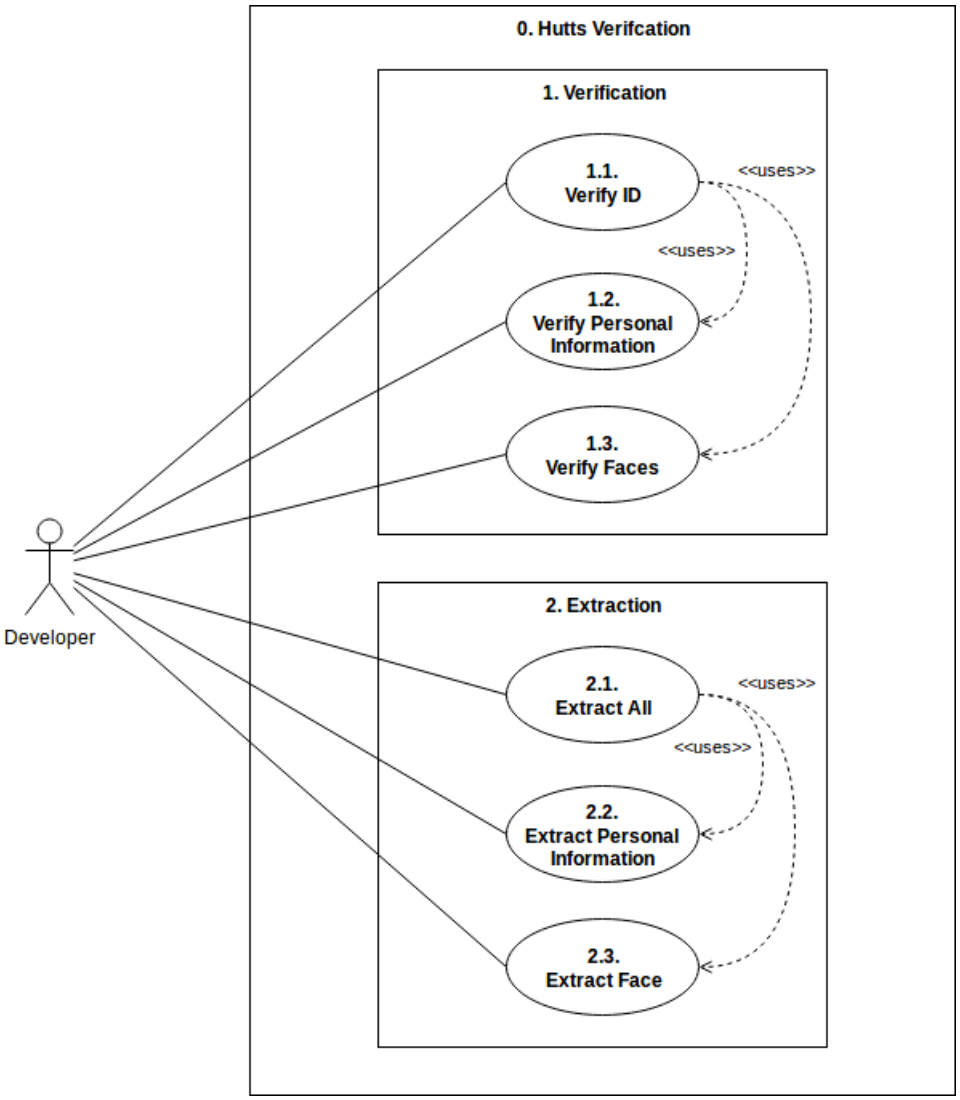

Figure 1: Use Case Diagram for Hutts-Verification

3.2.1 Use cases

1. Verification Subsystem
   (a) Verify ID
      i. **Description:** The developer must be able to send a picture of an ID and the system should verify the ID against a photo of the individual in question. The personal information of the individual should also be verified.
      ii. **Precondition:** Two photos with a high DPI and reasonable, proportionate lighting.
      iii. **Postcondition:** The system verifies and returns a percentage match.
   (b) Verify Personal Information
      i. **Description:** The developer must be able to send a picture of an ID and the system should verify the ID against provided personal information of the individual in question.
      ii. **Precondition:** Personal information of the person in question should be provided.
      iii. **Postcondition:** The system verifies and returns a percentage match.
   (c) Verify Faces

i. **Description:** The developer must be able to send a picture of an ID and the system should verify the ID against a photo of the individual in question.
ii. **Precondition:** Two photos with a high DPI and reasonable, proportionate lighting.
iii. **Postcondition:** The system verifies and returns a percentage match.

2. Extraction Subsystem

(a) Extract All

i. **Description:** The system should be able to extract the relevant personal information, as well as the face of the person in the given ID.
ii. **Precondition:** An ID with a high DPI and reasonable, proportionate lighting should be provided.
iii. **Postcondition:** The system returns the extracted text and the extracted face.

(b) Extract Personal Information

i. **Description:** The system should be able to extract the relevant personal information from a photo of the ID document.
ii. **Precondition:** An ID with a high DPI and reasonable, proportionate lighting should be provided.
iii. **Postcondition:** The system returns the extracted text.

(c) Extract Face

i. **Description:** The system should be able to detect and extract the face of the person in the given ID.
ii. **Precondition:** An ID with a high DPI and reasonable proportionate lighting should be provided.
iii. **Postcondition:** The system returns the extracted face.

Table 1: Traceability Matrix

		Functional Requirements											
Use Case	**Priority**	1	2	3	4	5	6	7	8	9	10	11	12
1.1	6	X											
1.2	4	X				X		X	X	X			X
1.3	5	X					X		X				X
2.1	3	X											
2.2	1	X	X								X	X	
2.3	2	X		X	X						X	X	
Requirement Priority		1	5	4	6	10	8	7	3	11	2	12	9

3.3 Performance Requirements

- A single extraction request should return in under 100 milliseconds.
- The system should be able to process 4 concurrent extraction requests in under 200 milliseconds.
- Any facial verification request should be handled in 5 seconds.
- Any internal errors and exceptions should be handle by the system itself.

3.4 Design Constraints

- The system will be implemented using Python 3.5 in order to ensure compatibility between the API and the Quant Solutions system.

3.5 Software System Attributes

This section describes all quality related requirements

3.5.1 Reliability

1. The system should not fail when given a clear, well-lit photo of the identification document.
2. The system should always give a percentage match above the specfied threshold if the faces are the same.
3. Error handling should be implemented and the application should be able to handle all errors in a graceful manner by making use of helpful and descriptive error messages.

3.5.2 Security

1. All incoming and outgoing data will be encrypted using the HTTPS protocol due to the sensitive nature of the data collected.

3.5.3 Availability

3.6 Other Requirements

- Documentation should be of such a standard that anyone can easily implement the application when needed.
- The system should indicate that images match only if the images match with an accuracy of at least 75%.

4 Technology Choices and Software Development tools

This section describes the technologies in use, and the arguments behind why we decided on these technologies. It was important to us to try and utilise technologies that are, free to use, efficient, relatively small in terms package size and requires as few as possible dependencies.

4.1 OpenCV-Python Version *3.2.0*

OpenCV (Open Source Computer Vision Library) is free for commercial use. OpenCV is a well-documented and supported Computer Vision library since it has been available and been developed on for over 15 years at this point. OpenCV is available in many programming language including Python, which was one of the requirements specified by the client. OpenCV now also supports multi-core hardware acceleration, which will increase efficiency drastically.

OpenCV will be used for real-time processing of images throughout the system, which will include image clean-up image extraction and image manipulation.

4.2 Dlib *19.6.0*

Dlib is an open source Machine learning toolkit and is also the only open source machine learning library that supports facial landmark optimisation and face likeness calculations out of the box.

Dlib allows us to make use of a high quality, pre-trained classifiers for facial likeness, which made use of over 3 million faces during training. This allows for an accuracy of 99.38 %, according to the author of Dlib, in

the *Labeled Faces in the Wild* bench mark, thus allowing us to meet most of our performance requirements in terms of the face likeness calculations.

Dlib also allows for face detection, as well as facial landmark detection, both of which is required to improve the accuracy and reliability of the face likeness algorithm by supporting the alignment and clean-up of detected faces.

4.3 Pytesseract *0.1.7*

Python-tesseract is an open source optical character recognition (OCR) tool for python. This allows for the reading and decoding of embedded text in images.

Tesseract allows for easy, automated scanning of cards. By making use of Tesseract we no longer need to:

1. Detect region of text.
2. Process region of text.
3. Use a trained classifier to read the extracted text.

If it is found later to be a necessity in future, Tesseract supports additional training of fonts, which will allow for greater reliability and accuracy when extracting text.

4.4 Travis CI

Travis CI, is a hosted continuous integration and delivery service for GitHub projects.

Travis CI allowed us to run each pull request in a clean VM build whilst running multiple Test in parallel. This ensured our system was always in a stable state due to the automated testing and installing of the project.It also allowed us to streamline code review process.

4.5 Codacy

Codacy is an automated code analysis/quality tool.

Codacy allowed us to continuously test and maintain our system, by checking for code complexity, duplication, security and unit test coverage of changes in every commit and pull request. Codacy allowed us to enforce our code quality standard as well practices that are considered the norm in the industry.

4.6 Snyk

Snyk supports testing and monitoring Python projects that have their dependencies managed by pip by helping fix and monitor known vulnerabilities in the dependencies in projects.

Hutts verification used several helpful open-source dependencies. However, not all these dependencies can ensure that they are up to date and secure. Snyk was used as a support tool to help prevent this problem by probing us the means to detect and fix any vulnerabilities that could affect the performance or security of our project.