POLITECNICO
MILANO 1863

SafeStreets - ITPD

Politecnico di Milano - A.Y. 2019/20120

Andrea Aspesi - andrea.aspesi@mail.polimi.it
Elia Battiston - elia.battiston@mail.polimi.it
Alessandro Carabelli - alessandro2.carabelli@mail.polimi.it

Contents

Part I
Introduction

1 Purpose

This is the Integration Test Plan Document of the SafeStreets project.
The objective of this document is to provide a detailed analysis of the system that has been implemented based on the RASD and on the DD documents.
The description of the architecture of the system, the components it is made of and their interactions, are completely described with the purpose of guiding a developer in the comprehension of the implementation of the project.
The addressees of this document are developers looking for a documentation of the development and the testing.
Addressees of this document are also users of the platform.

2 Scope

Scope of this document is to describe the implementation of a prototype of the SafeStreets system.

2.1 Goal

Goal of the prototype is to maintain all the core functionalities described in the RASD and in the DD.
Simplifications of the Backend have been made in order to decrease the time-to-market while maintaining the end-user functionalities.

2.2 Changes in deployment

In order to develop the prototype of the SafeStreets system, some changes in the backend has been made while keeping its user functionalities.
AWS has been completely replaced with an hosting on the Altervista platform.

- Amazon RDS has been replaced with Altervista's default DB;
- Amazon S3 has been replaced with Altervista's storage;
- Amazon EC2 (and its components: the Apache Web Server and the PHP Engine) has been replaced with Altervista's dedicated functionalities;
- Amazon Elastic Load hasn't been replaced;
- Amazon Firewall Manager hasn't been replaced.

3 Definitions, Acronyms, Abbreviations

- *DBMS:* DataBase Management System
- *RDBMS:* Relational DBMS

4 Revision history

- *Version 1.0:*
  Initial release

5 Reference documents

- ▷ SafeStreets specification document: “SafeStreets Mandatory Project Assignment”
- ▷ Implementation assignment document: “Implementation and Testing project assignment”
- ▷ SafeStreets RASD
- ▷ SafeStreets DD

6 Document structure

1. Brief description of this document, its nomenclature and its purpose.
2. Summary of which functions of the system were actually developed in for this prototype.
3. Description of the development tools and frameworks used to implement the system.
4. Structure of the prototype source code in the repository.
5. Information on how testing was performed, which test cases were considered and the tools used to test the prototype.
6. Reporting of the time spent by each team member while redacting the document.

Part II
Implemented requirements & functions

In this part it is exposed which requirements and functions have been developed in the project implementation. For a better clarity, the order of presentation follows the one used in the RASD document.

1 Implemented functions

All the basic functions have been implemented, as required.
In addition to the basic functions our group has decided to develop the *Automatic check generation* advanced function discarding the Data analysis and notification one.

1.1 Signup

Users can register providing personal informations and a valid email, choosing a username and a picture of a personal ID. Before granting access, an administrator has to revise the sign-up request.
The request is made through the dedicated form in the mobile app.
Users can also be directly created by an administrator, for example in case of municipality request.

1.2 Report creation

Every enabled user can send a report, choosing the related option in the mobile application.
The application then asks for infraction type (drop-down menu), plate number, notes about the infraction and at least one picture. The position is GPS-retrieved clicking on the related button.
Once filled the form, the user can send the report clicking on the "Send report" button.

1.3 Past reports list

As for report creation, users can access the functionality in the mobile application menu.
In there the list of past sent reports is provided, with the possibility to access detailed informations about the report.

1.4 Street Safety

As last function of the mobile application, a map with information about reports, accidents and emitted traffic tickets is available.
Data about accidents and emitted tickets are also retrieved through a dedicated endpoint provided by the municipality.

1.5 Automatic reports sending to police officers

At every report submission the backend automatically associates data about the user who created the report with the report itself and sends the whole data to the municipality endpoint for ticket generation by the police officers.

1.6 User handling

Administrators can access the web backend for user management and eventual suspension/rehabilitation, in addition to new user validation.

2 External Interface

2.1 User interfaces

User interfaces implementation follows the specifications of the RASD mockups and constraints, as planned during the analysis phase.

2.1.1 Mobile application

The mobile application is deployed for Android and iOS platforms, with a total of six pages.
User managing such as password change and logout are collected in the Account Options page (slightly differentiating from the mockups, which designed the options directly in the lateral menu).
Login, Signup, Report and Past Reports are implemented exactly as the mockups, with some minor graphic improvements.
Street Safety page is implemented over a dynamic map as in the related mockup, but due to Xamarin map NuGet limitation it hasn't been possible to highlight the streets with related data in addition to the pointer.
The Report Details page has been added, with more details about the past made report than the reports list.

2.1.2 Web backend

Web backend implementation follows the mockups provided in the RASD document, just moving the menu bar from the left to the top of the page.
All the required sections have been developed; moreover, a *User acceptance* section has been added to highlight users waiting for identification and enabling.
It has also been added a page for direct creation of a user and a page with single report details (as for the application).

2.2 Hardware & Software interfaces

As reported in the RASD document, the project doesn't include the development of such interfaces as they are not needed.

3 Functional requirements

All references to requirement codes (marked between square brackets) starting with G (ex. [G1]) refers to requirements of section 2.5 of the RASD document.

3.1 Visitor

As expected, visitors are allowed to download the app through the related app stores and sign-up.

Requirements traceability The function satisfies requirements [G1] and [G2].

3.2 User

The user is enabled to use all the expected functions (Login, Send report, See past reports, See Street Safety) through the mobile app. During login phase, the backend verifies the user acceptance and eventual suspension before allowing access. Every active and validated user can login and use the functionalities of the application regardless of the role.

3.2.1 Requirements traceability

The function satisfies requirements [G3], [G4], [G5] and [G6].

3.3 Authority

The authority, as mentioned above, can access and use all the functionalities of a normal user. In addition, the web backend access is granted for whole reports data access.
The system is also designed to automatically send new report information to the municipality through the municipality provided API with the aim of traffic tickets generation.

3.3.1 Requirements traceability

The function satisfies requirements [G6], [G7] and [G8].

3.4 Administrator

Administrator web backend access grants the possibility to vision the user list, to eventually change a user role or suspend a user, and to filter and accept new user requests.

3.4.1 Requirements traceability

The function satisfies requirements [G10], [G11], and [G12].

Part III
Adopted development frameworks

To develop a client-server application a variety of frameworks can be adopted to ensure a more reliable, updatable and efficient outcome.
SafeStreets can be divided in three main components, each developed using dedicated technologies:

- ▷ Mobile Application (Android & iOS)
- ▷ Web Application
- ▷ Backend

Mobile Application The mobile application has been developed using the cross platform framework "Xamarin Forms".
The code is written in a shared C# codebase from which both Android and iOS native applications can be built. The UI is written in XAML, a language created for this purpose that, thanks to the newest "Hot Reload" feature, can show changes in the UI while the app is running in debug mode, consisting in a much faster process of interface development.
Xamarin has been chosen in a competition where multiple frameworks could fit the objective. Its strengths are:

- ▷ Thanks to a shared codebase it makes implementation much faster than making two different applications (one per platform);
- ▷ It maintains the same experience for both the OSs;
- ▷ It can be compiled for even more OSs (Windows & MacOS) giving more possibilities for the future;
- ▷ It is widely used, common problems and procedures are well documented.

On the other end, other choices had other strengths:

- ▷ Native: (PRO) faster, (PRO) more documentation, (CONS) time required;
- ▷ React Native: (PRO) faster, (CONS) js, (CONS) updates that break the code.

We also calculated that, for SafeStreets, making a single application that can run on both platforms, will decrease the time spent on the mobile application side up to 35%.
In the end, the use of C#, the great documentation and the development time requested where the points that make us choose Xamarin over its competitors

Web Application The web application is written in PHP, HTML, CSS, JS, using the jQuery library and the SemanticUI development framework for its frontend.
While HTML, CSS, JS are forced choices in web development, the adoption of PHP, jQuery and SemanticUI has been chosen for the following reasons:

- ▷ PHP: PHP has been chosen as the backend script language because it's widely used, well known by the developers, provides great documentation and it's multithreaded;
- ▷ jQuery: plain JS is nowadays easy to use but for Ajax requests nothing is better than jQuery and its methods;
- ▷ SemanticUI: the simplicity, cleanliness and speed of SemanticUI made it the right choice against plain HTML/CSS/JS (required more time and design skills) and CMSs like WordPress or Joomla lack of speed and personalization to the core.
  Its most useful feature is the ability to express complex UI layouts just by appending natural language-like attributes to HTML entities.
  More information at SemanticUI's website`https://semantic-ui.com/`.

The WebApp like the APIs are served through a secured TLS connection (HTTPS) to ensure privacy and security of the communication channel.

Backend The backend is on a LAMP web service stack.

- ▷ PHP: as for the Web App. The version 7.4 has been chosen because it's the actual available version that will be supported longer;
- ▷ Apache Web Server: it has been chosen over nginx because it handles every request as a different thread;
- ▷ Maria DB: the adoption of a Relational Database has been chosen based on the type of data processed by SafeStreets. The SQL language is also useful to handle the statistical data needed by the Safety&Suggestions engine. MariaDB has been chosen because it is open source, widely used and implements the SQL standard language
- ▷ Linux has been chosen over Windows. Some of its PROs are: it is free, open source, highly customizable and faster.

Part IV
Source code structure

The source code, placed in the Implementation folder of the repository, is split in different folders which reflect the component the code belongs to.

1 Database

This folder contains the files which define SafeStreet's database structure (`safestreets.sql`) and data used in test cases (`test.sql`).

2 API

This folder contains all the code which implements SafeStreets APIs.
In its root, `config.php` can be used to set the database settings to be used. For more information, see the Installation instructions part.

2.1 Modules folder

This folder contains PHP files which define the classes used to manage access to database data in a standardized way across APIs.

2.2 Actual APIs folders

The `accounts`, `mobile` and `web` folders contain PHP files which will be accessible through the APIs endpoint and implement the endpoint functionality.

2.3 Tests folder

This folder contains PHP files implementing every API test with the support of PHPUnit. For more information see the Testing part.

3 MunicipalityStub

This folder contains the single PHP file used to simulate the endpoint of the municipality APIs, used for testing purposes.
For more information see the Municipality Stub section.

4 WebClient

This folder contains all the code used to implement the SafeStreets Web client backend.
In the root of the folder, the PHP files which define the layout of every page of the client can be found.

4.1 Components folder

This folder contains PHP files implementing recurrent layout or logic elements to be included by several client pages.
For example, `header.php` defines the content of the `header` element for all the pages, which includes CSS and JS files.

4.2 Images folder

This folder contains images used in the web client pages

4.3 Semantic folder

This folder contains the CSS and JS libraries of the SemanticUI framework, used to define the pages structures. For more information see the Adopted development frameworks part.

4.4 Src folder

This folder contains PHP files implementing the execution of actions from the pages of the web client. These pages will try to execute the requested actions by contacting the APIs and redirect the request accordingly.

5 MobileApp

Part V
Testing

As outlined in the DD document of the project, implementation, integration and testing of the system's components has been executed incrementally.

1. After the components of the Data Tier were completed (Report database, User accounts database and Accidents&Tickets database), development of the Business Tier (APIs) could start.
   During the development of APIs both integration tests with the Database and unit tests to check its correctness have been written and executed.

2. Once the Business Tier development reached sufficient completion, the implementation of the Implementation Tier components (Mobile application and Web client) could start.
   During the development of both presentation components, integration tests with the APIs have been executed.

3. When the development of the Presentation tier reached sufficient completion, the execution of system testing could start.

In the following sections, more details on testing methodologies and considered cases are described.

1 API unit testing

Unit tests have been written to assure every aspect of the API, which represents the most part of the system's logic, behaves as expected.
The main focus of these tests is to reasonably assure that:

- ▷ Users only have access to information available to their authority level
- ▷ Provided information is complete and correct
- ▷ The format in which responses are provided is consistent with the one described in the endpoints documentation

Test cases of this section also cover **Integration testing** between the APIs and the database, using data provided in the `test.sql` database.

1.1 Tools

The API unit testing has been performed using *PHPUnit* (`https://phpunit.de/`), one of the most well known PHP testing frameworks.
Tests have been configured to be executed on every new commit in the repository using a *GitHub Actions Workflow* (`https://github.com/features/actions`), GitHub's CI/CD platform. The automatic and continuous execution of tests helps assuring that changes in the codebase don't affect previously working features and prevents non-working code from being deployed, leaving the live system always updated but working correctly at the same time.

1.2 Modules test cases

1.2.1 Accounts

1.

Name	Fiscal code
Components	Accounts Module, Accounts Database
Input	Correct username
Output	Fiscal code of the user
Description	The module provides the fiscal code of the user with the provided username

2.

Name	Wrong fiscal code
Components	Accounts Module, Accounts Database
Input	Non existent username
Output	`NULL`
Description	The module tries to acquire user data from the database and signals none was found

3.

Name	Is officer
Components	Accounts Module, Accounts Database
Input	Correct username of an officer
Output	Confirmation of the user's officer status (`true`)
Description	The module confirms that the user has a role equal or more privileged than "officer"

4.

Name	Is admin
Components	Accounts Module, Accounts Database
Input	Correct username of an administrator
Output	Confirmation of the user's administrator status (`true`)
Description	The module confirms that the user has a role equal or more privileged than "administrator"

5.

Name	Admin is officer
Components	Accounts Module, Accounts Database
Input	Correct username of an administrator
Output	Confirmation of the user's officer status (`true`)
Description	The module confirms that the user has a role equal or more privileged than "officer"

6.

Name	Is not officer
Components	Accounts Module, Accounts Database
Input	Correct username of a regular user
Output	Denial of the user's officer status (`false`)
Description	The module denies that the user has a role equal or more privileged than "officer"

7.

Name	Is not admin
Components	Accounts Module, Accounts Database
Input	Correct username of an officer
Output	Denial of the user's administrator status (`false`)
Description	The module denies that the user has a role equal or more privileged than "administrator"

8.

Name	Wrong officer
Components	Accounts Module, Accounts Database
Input	Non existent username
Output	Denial of the user's officer status (`false`)
Description	The module tries to acquire user data from the database and, as none was found, denies the "officer" status

9.

Name	Wrong admin
Components	Accounts Module, Accounts Database
Input	Non existent username
Output	Denial of the user's administrator status (`false`)
Description	The module tries to acquire user data from the database and, as none was found, denies the "officer" status

10.

Name	User data retrieval
Components	Accounts Module, Accounts Database
Input	-
Output	List of data containing all the users in the system
Description	The module provides a list of all the users of the system from the database

11.

Name	Single user data retrieval
Components	Accounts Module, Accounts Database
Input	User fiscal code
Output	Detailed data about the requested user
Description	The module provides detailed data of the requested user from the database

12.

Name	User role changing
Components	Accounts Module, Accounts Database
Input	Username, Code of the role to assign
Output	The user's role is correctly changed
Description	The module contacts the database to update the role of the requested user

13.

Name	User acceptance
Components	Accounts Module, Accounts Database
Input	Username of the user to accept, Username of the accepting administrator
Output	The user's acceptance is correctly recorded
Description	The module contacts the database to update the accepting admin and acceptance timestamp of the requested user

14.

Name	User suspension
Components	Accounts Module, Accounts Database
Input	Username of the user to suspend
Output	The user's suspension is correctly recorded
Description	The module contacts the database to update the "suspended" flag and suspension timestamp of the requested user

15.

Name	Correct signup
Components	Accounts Module, Accounts Database
Input	Username of the user to suspend
Output	The user's suspension is correctly recorded
Description	The module contacts the database to update the "suspended" flag and suspension timestamp of the requested user

1.2.2 Reports

1.

Name	All reports
Components	Reports Module, Reports Database
Input	-
Output	List of all the reports in the system
Description	The module provides a detailed list of all the reports of the system from the database

2.

Name	User reports
Components	Reports Module, Reports Database
Input	Username
Output	List of all the reports in the system created by the user
Description	The module provides a detailed list of all the reports of the system created by the requesting user

3.

Name	Report detail
Components	Reports Module, Reports Database
Input	ID of the report
Output	Details of the report
Description	The module provides all data of the requested report

4.

Name	Nonexistent report detail
Components	Reports Module, Reports Database
Input	ID of a nonexistent report
Output	`NULL`
Description	The module tries to retrieve data about the requested record, but none is found

5.

Name	User report detail
Components	Reports Module, Reports Database
Input	Username of the requesting user, ID of a report from the user
Output	Details of the report
Description	The module provides all data of the requested report

6.

Name	Wrong user report detail
Components	Reports Module, Reports Database
Input	Username of the requesting user, ID of a report from another user
Output	`NULL`
Description	The module rejects the request of a report which was not created by the requester

7.

Name	User nonexistent report detail
Components	Reports Module, Reports Database
Input	Username of the requesting user, ID of a nonexistent report
Output	`NULL`
Description	The module tries to retrieve data about the requested record, but none is found

1.3 Accounts API test cases

1.

Name	Correct login
Components	Accounts API, Accounts Module, Accounts Database
Input	Existent username, Correct password
Output	Correctly formatted response with result code 200
Description	Login is correctly performed and the response has the expected format

2.

Name	Wrong password
Components	Accounts API, Accounts Module, Accounts Database
Input	Existent username, Wrong password
Output	Correctly formatted response with result code 401
Description	Login fails due to wrong credentials and the response has the expected format

3.

Name	Nonexistent user
Components	Accounts API, Accounts Module, Accounts Database
Input	Nonexistent username, Password
Output	Correctly formatted response with result code 401
Description	Login fails due to nonexistent username and the response has the expected format

4.

Name	Nonexistent user
Components	Accounts API, Accounts Module, Accounts Database
Input	Nonexistent username, Password
Output	Correctly formatted response with result code 401
Description	Login fails due to nonexistent username and the response has the expected format

5.

Name	Correct user data
Components	Accounts API, Accounts Module, Accounts Database
Input	Existent username, Correct password
Output	Correctly formatted response with result code 200
Description	The response has the expected format and the user's data is correct

6.

Name	Change password
Components	Accounts API, Accounts Module, Accounts Database
Input	Existent username, Correct password, New password
Output	Correctly formatted response with result code 200
Description	The response has the expected format and the user's password is changed on the database, so that he can correctly log in with the new one

7.

Name	Suspended user
Components	Accounts API, Accounts Module, Accounts Database
Input	Suspended user's username, Correct password
Output	Correctly formatted response with result code 402
Description	The response has the expected format and it signals there are missing parameters

8.

Name	Missing parameters
Components	Accounts API, Accounts Module, Accounts Database
Input	-
Output	Correctly formatted response with result code 404
Description	The response has the expected format and it signals there are missing parameters

9.

Name	Correct signup
Components	Accounts API, Accounts Module, Accounts Database
Input	Username, Password, First name, Last name, Email address, Fiscal code, Encoded document photo
Output	Correctly formatted response with result code 200
Description	The response has the expected format and it signals the creation of the user is successful, the document photo is correctly stored

10.

Name	Not authorized login
Components	Accounts API, Accounts Module, Accounts Database
Input	Not yet accepted user's username, Correct password
Output	Correctly formatted response with result code 403
Description	The response has the expected format and it signals the user can't log in because it hasn't been accepted yet

1.4 Mobile API test cases

1.4.1 Reports

1.

Name	Mobile wrong credentials
Components	Mobile API, Reports Module, Reports Database
Input	Nonexistent username, Correct password
Output	Correctly formatted response with result code 401
Description	The response has the expected format and it signals the username used to authenticate the request does not exist

2.

Name	Mobile reports
Components	Mobile API, Reports Module, Reports Database
Input	Existent username, Correct password
Output	Correctly formatted response with result code 200
Description	The response has the expected format and its content is a detailed list of all the reports created by the requesting user

3.

Name	Mobile report detail
Components	Mobile API, Reports Module, Reports Database
Input	Existent username, Correct password, Report ID of a report created by user
Output	Correctly formatted response with result code 200
Description	The response has the expected format and its content is a complete description of the report

4.

Name	Mobile not own report detail
Components	Mobile API, Reports Module, Reports Database
Input	Existent username, Correct password, Report ID of a report not created by user
Output	Correctly formatted response with result code 400
Description	The response has the expected format and it signals the user can't access data about the requested record

5.

Name	Mobile nonexistent report detail
Components	Mobile API, Reports Module, Reports Database
Input	Existent username, Correct password, Report ID of a report which does not exist
Output	Correctly formatted response with result code 400
Description	The response has the expected format and it signals that the requested report does not exist

6.

Name	Mobile no reports
Components	Mobile API, Reports Module, Reports Database
Input	Existent username, Correct password
Output	Correctly formatted response with result code 201
Description	The response has the expected format and it signals the user has no past reports

7.

Name	Mobile create report
Components	Mobile API, Reports Module, Reports Database
Input	Existent username, Correct password, License plate number, Violation type, Latitude, Longitude, Encoded pictures
Output	Correctly formatted response with result code 200
Description	The response has the expected format and the new report is correctly created, received pictures are correctly created

1.5 Web API test cases

1.5.1 Accounts

1.

Name	Web users wrong login
Components	Web API, Accounts Module, Accounts Database
Input	Existent username, Wrong password
Output	Correctly formatted response with result code 401
Description	The response has the expected format and it signals the password is not the correct one for the specified user

2.

Name	Web users unauthorized
Components	Web API, Accounts Module, Accounts Database
Input	Regular user's username, Correct password
Output	Correctly formatted response with result code 403
Description	The response has the expected format and it signals the user doesn't have sufficient permissions to access the system's users list

3.

Name	Web users data
Components	Web API, Accounts Module, Accounts Database
Input	Administrator's username, Correct password
Output	Correctly formatted response with result code 200
Description	The response has the expected format and its content is a detailed list of all the system's users

4.

Name	Web user detail
Components	Web API, Accounts Module, Accounts Database
Input	Administrator's username, Correct password, Requested user's fiscal code
Output	Correctly formatted response with result code 200
Description	The response has the expected format and its content is a complete description of the requested user

5.

Name	Web suspension wrong login
Components	Web API, Accounts Module, Accounts Database
Input	Existent username, Wrong password
Output	Correctly formatted response with result code 401
Description	The response has the expected format and it signals the password is not the correct one for the specified user

6.

Name	Web suspension missing parameters
Components	Web API, Accounts Module, Accounts Database
Input	Administrator's username, Correct password
Output	Correctly formatted response with result code 404
Description	The response has the expected format and it signals there are missing parameters

7.

Name	Web suspension unauthorized
Components	Web API, Accounts Module, Accounts Database
Input	Officerr's username, Correct password, Username of the user to suspend
Output	Correctly formatted response with result code 403
Description	The response has the expected format and it signals the user doesn't have sufficient permissions to suspend another user

8.

Name	Web suspension
Components	Web API, Accounts Module, Accounts Database
Input	Administrator's username, Correct password, Username of the user to suspend
Output	Correctly formatted response with result code 200
Description	The response has the expected format and the user is successfully suspended

9.

Name	Web acceptance wrong login
Components	Web API, Accounts Module, Accounts Database
Input	Existent username, Wrong password
Output	Correctly formatted response with result code 401
Description	The response has the expected format and it signals the password is not the correct one for the specified user

10.

Name	Web acceptance missing parameters
Components	Web API, Accounts Module, Accounts Database
Input	Administrator's username, Correct password
Output	Correctly formatted response with result code 404
Description	The response has the expected format and it signals there are missing parameters

11.

Name	Web acceptance unauthorized
Components	Web API, Accounts Module, Accounts Database
Input	Officerr's username, Correct password, Username of the user to accept
Output	Correctly formatted response with result code 403
Description	The response has the expected format and it signals the user doesn't have sufficient permissions to accept another user

12.

Name	Web acceptance
Components	Web API, Accounts Module, Accounts Database
Input	Administrator's username, Correct password, Username of the user to accept
Output	Correctly formatted response with result code 200
Description	The response has the expected format and the user is successfully accepted

13.

Name	Web role change wrong login
Components	Web API, Accounts Module, Accounts Database
Input	Existent username, Wrong password
Output	Correctly formatted response with result code 401
Description	The response has the expected format and it signals the password is not the correct one for the specified user

14.

Name	Web role change missing parameters
Components	Web API, Accounts Module, Accounts Database
Input	Administrator's username, Correct password, New role level
Output	Correctly formatted response with result code 404
Description	The response has the expected format and it signals there are missing parameters

15.

Name	Web role change unauthorized
Components	Web API, Accounts Module, Accounts Database
Input	Officerr's username, Correct password, Username of the user to change, New role level
Output	Correctly formatted response with result code 403
Description	The response has the expected format and it signals the user doesn't have sufficient permissions to change the role of another user

16.

Name	Web role change
Components	Web API, Accounts Module, Accounts Database
Input	Administrator's username, Correct password, Username of the user to change, New role level
Output	Correctly formatted response with result code 200
Description	The response has the expected format and the user's role is successfully changed

1.5.2 Reports

1.

Name	Web wrong credentials
Components	Web API, Reports Module, Reports Database
Input	Nonexistent username, Correct password
Output	Correctly formatted response with result code 401
Description	The response has the expected format and it signals the specified user does not exist

2.

Name	Web role change unauthorized
Components	Web API, Accounts Module, Accounts Database
Input	Regular user's username, Correct password
Output	Correctly formatted response with result code 403
Description	The response has the expected format and it signals the user doesn't have sufficient permissions to access data of all the reports in the system

3.

Name	Web reports
Components	Web API, Reports Module, Reports Database
Input	Existent username, Correct password
Output	Correctly formatted response with result code 200
Description	The response has the expected format and its content is a detailed list of all the reports created by the requesting user

4.

Name	Web report detail
Components	Web API, Reports Module, Reports Database
Input	Existent username, Correct password, Report ID of a report created by user
Output	Correctly formatted response with result code 200
Description	The response has the expected format and its content is a complete description of the report

5.

Name	Web nonexistent report detail
Components	Web API, Reports Module, Reports Database
Input	Existent username, Correct password, Report ID of a report which does not exist
Output	Correctly formatted response with result code 400
Description	The response has the expected format and it signals that the requested report does not exist

2 Further integration testing and system testing

The testing phases which followed testing of the APIs have been carried out by the developers, without the support of any tool.
The following test cases cover both **Integration testing** (between the Presentation tier components and the APIs) and **System testing**, as they evaluate the execution of the system as a whole.

2.1 Mobile application

1.

Name	Login
Components	Mobile application, Accounts API, Accounts Module, Accounts Database
Input	Existent username, Correct password
Output	Successful login
Description	The application correctly contacts APIs to make sure the credentials are correct and logs him in

2.

Name	Signup
Components	Mobile application, Accounts API, Accounts Module, Accounts Database
Input	Email address, Username, Fiscal Code, First name, Last name, Password, Picture
Output	Successful signup
Description	The application correctly contacts APIs to create a new user, which is correctly added to the system

3.

Name	Restore credentials
Components	Mobile application, Accounts API, Accounts Module, Accounts Database
Input	Existent username
Output	The user's password is substituted with a randomly generated one and an email is sent to its email address to inform him
Description	The application correctly contacts APIs to start the password restore process

4.

Name	Restore credentials
Components	Mobile application, Accounts API, Accounts Module, Accounts Database
Input	Existent username, Correct password, New password
Output	The user's password is substituted with the one provided
Description	The application correctly contacts APIs to change the user's password with the one he provided

5.

Name	Logout
Components	Mobile application
Input	-
Output	The user is logged out of the application
Description	All data regarding the user's credential is deleted from local storage and the user will be asked to enter them again to access the application functions

6.

Name	Report list
Components	Mobile application, Reports API, Reports Module, Reports Database
Input	Existent username, Correct password
Output	List of the user's past reports
Description	The application correctly contacts APIs to acquire a list of the reports sent by the logged in user and shows them

7.

Name	Detailed report view
Components	Mobile application, Reports API, Reports Module, Reports Database
Input	Existent username, Correct password, Desired report ID
Output	View of the complete data about the report
Description	The application correctly contacts APIs to acquire the details of the report sent by the logged in user, and shows them in a dedicated page

8.

Name	Detailed report view
Components	Mobile application, Reports API, Reports Module, Reports Database
Input	Existent username, Correct password
Output	Map showing streets safety information
Description	The client correctly contacts APIs to acquire a list of streets for which the system has safety information and shows them on the map

2.2 Web client

1.

Name	Login
Components	Web client, Accounts API, Accounts Module, Accounts Database
Input	Existent username, Correct password
Output	Successful login
Description	The web client correctly contacts APIs to make sure the credentials are correct and that the user has a sufficient role, then logs him in

2.

Name	Reports list
Components	Web client, Reports API, Reports Module, Reports Database
Input	Existent username, Correct password
Output	List of all the reports in the system
Description	The client correctly contacts APIs to acquire a list of the reports in the system and shows them in a table

3.

Name	Report details
Components	Web client, Reports API, Reports Module, Reports Database
Input	Existent username, Correct password, Desired report ID
Output	Detailed view of the requested report and the user who sent it
Description	The client correctly contacts APIs to acquire and show complete data about the requested report and the user who submitted it

4.

Name	Users list
Components	Web client, Accounts API, Accounts Module, Accounts Database
Input	Existent username, Correct password
Output	List of all the users in the system
Description	The client correctly contacts APIs to acquire a list of the users in the system and shows them in a table

5.

Name	User details
Components	Web client, Accounts API, Accounts Module, Accounts Database
Input	Existent username, Correct password, Fiscal code of the user
Output	Detailed view of the selected user with the ability to change its role and suspend him
Description	The client correctly contacts APIs to acquire and show complete data about the requested user

6.

Name	User acceptance
Components	Web client, Accounts API, Accounts Module, Accounts Database
Input	Existent username, Correct password, Username of the user to accept
Output	The user is accepted and removed from the list of the users to accept
Description	The client correctly contacts APIs to set acceptance data for the user, to allow him to login for the first time

7.

Name	User suspension
Components	Web client, Accounts API, Accounts Module, Accounts Database
Input	Existent username, Correct password, Username of the user to suspend
Output	The user is suspended and the suspension timestamp is set
Description	The client correctly contacts APIs to set suspension data for the user, to reject every request he makes until the account is restored

8.

Name	User restore
Components	Web client, Accounts API, Accounts Module, Accounts Database
Input	Existent username, Correct password, Username of the user to restore
Output	The user is restored and the suspension timestamp is set to `NULL`
Description	The client correctly contacts APIs to unset suspension data for the user, to allow him to make new requests

9.

Name	Change user role
Components	Web client, Accounts API, Accounts Module, Accounts Database
Input	Existent username, Correct password, Username of the user whose role will change, New role code
Output	The user's role is changed
Description	The client correctly contacts APIs to change the requested user's role

10.

Name	User creation
Components	Web client, Accounts API, Accounts Module, Accounts Database
Input	Email address, Username, Fiscal Code, First name, Last name, Password, Picture
Output	Successful creation of a new user
Description	The client correctly contacts APIs to create a new user, which is correctly added to the system

11.

Name	Logout
Components	Web client
Input	-
Output	The user is logged out of the client
Description	All data regarding the user's credential is deleted from local storage (cookies) and the user will be asked to enter them again to access the client's

3 Municipality Stub

With the aim of testing and providing a fully functioning product, a stub simulating the municipality endpoint for data sending and retrieval has been implemented.
The stub consists in a `.php` file which acquires data about new reports through a *POST https* request (saving the last retrieved data in the `data.txt` file for logging purposes) and simulates the retrieval of data about randomly generated tickets and accidents through a *GET* request.
The stub's URL is `https://safestreets.altervista.org/municipalityStub/index.php`

Part VI
Installation instructions

The following instructions include all the steps needed to deploy the SafeStreets solution or one of its modular components.
The source code of the system can be found in the GitHub repository at `https://github.com/EliaBattiston/AspesiBattistonCarabelli`. In particular, folders referenced in the following sections will be found in its Implementation folder.

0.0.1 Official deployed system

To try the system without deploying it privately, a working version of the Database, APIs and Web client is automatically deployed to `safestreets.altervista.org`. The website on this domain has no homepage, since it was not included in the scope of this prototype.

- ▷ `https://safestreets.altervista.org/api` is the official API endpoint
- ▷ `https://safestreets.altervista.org/web` is the login page of the Web client for authorized users
- ▷ The database deployed on the same domain for the APIs is not publicly accessible

0.0.2 Compiled application

An already compiled version the Mobile application can be found in the DeliveryFolder of the repository, and is called `SafeStreets_Android.apk`

1 Database installation

1.1 Requirements

- ▷ The SafeStreets database does not depend on any other component, so it can be installed as the first module of the system.
- ▷ A relational DBMS with the ability of importing `.sql` files is needed

1.2 Steps

1. Import `Database/safestreets.sql` in the desired DBMS.
   The SQL file includes the definition of the database schema and data needed to start running the solution, for example the `system` user.

2 APIs installation

2.1 Requirements

- ▷ APIs depend on the deployment of the SafeStreets database, so it has to be installed first to be able to correctly configure and run APIs
- ▷ A web server with PHP support is needed to run and serve the API

2.2 Steps

- ▷ Place the contents of the `API` folder in a directory served by your web server.
  From now on, the URL at which the API is served will be called *endpoint*, and the folder where the files have been placed will be called *endpoint directory*
- ▷ Make sure that PHP and the web server have write permissions to the `reportPictures` and `userDocumentPhotos` folders in the endpoint directory

▷ Change the values in the included `config.php` file to configure access to your deployed database

Parameter	Value to be set
`$_CONFIG['host']`	Name of the host where the database is set
`$_CONFIG['user']`	Username of the database account to access the database
`$_CONFIG['pass']`	Password of the account
`$_CONFIG['dbname']`	Name of the deployed database. If the database `.sql` file is imported correctly, this value should be set to "`safestreets`"

Table 1: Database configuration parameters

3 Web Client installation

3.1 Requirements

▷ The web client depends on a functioning deployment of the SafeStreets APIs, so they have to be installed first to be able to correctly configure and run the client

▷ A web server with PHP support is needed to run and serve the web client

3.2 Steps

▷ Place the contents of the `WebClient` folder in a directory served by your web server.
From now on, the URL at which the client is served will be called *website*, and the folder where the files have been placed will be called *website directory*

▷ Change the values in the included `config.php` file to configure access to deployed APIs

Parameter	Value to be set
`$endpoint`	URL of the deployed APIs endpoint. `"https://safestreets.altervista.org/api"`, automatically deployed from the repository, can be used. The value **must not have** a trailing slash.

Table 2: API configuration parameters

4 Mobile application build

4.1 Requirements

▷ Microsoft Visual Studio with Xamarin component installed

▷ Internet connection for the download of the needed NuGet packages

▷ (iOS build only) MacOS device running MacOS Mojave or higher

4.2 Steps

▷ Open the solution (`.sln`) file contained in the Xamarin folder

▷ Select "Generic device" as project destination in Release mode

▷ Open the project structure tab in Visual Studio

4.2.1 Android

- ▷ Right-click on the Android project and click “Recompile”
- ▷ When the compilation is completed, right click on the Android project and then click on “Archive”
- ▷ When archiviation is completed, click on the archive and then on “Distribute”
- ▷ Select “Ad Hoc” and select a KeyStore to sign the APK
- ▷ Export the APK to the desired folder
- ▷ Send to an Android device and install

4.2.2 iOS

- ▷ Right-click on the iOS project and click “Recompile”
- ▷ When the compilation is completed, right click on the iOS project then click on “Archive”
- ▷ When archiviation is completed, click on the archive and then on “Distribute”
- ▷ Follow the long series of shown steps. At the end you’ll be able to publish the application for TestFlight deployment or for AppStore publishing

Part VII
Effort spent

Part	Hours
1. API development	0
2. Web client development	0
3. Mobile application development	15
4. DevOps	0
5. Document	3

Table 3: Andrea Aspesi's effort spent

Part	Hours
1. API development	9
2. Web client development	11
3. Mobile application development	0
4. DevOps	3
5. Document	4.5

Table 4: Elia Battiston's effort spent

Part	Hours
1. API development	12
2. Web client development	0
3. Mobile application development	0
4. DevOps	1
5. Document	2

Table 5: Alessandro Carabelli's effort spent