POLITECNICO
MILANO 1863

SafeStreets - DD

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 DD is the Design Document of the SafeStreets project.
The objective of this document is to provide a more detailed analysis of the problem than the one outlined in the RASD, while specifying the proposed design and implementation aspects of the final product.
The chosen architecture of the system, the components it is made of and their interactions are completely described with the purpose of guiding the system's development.
The addressees of this document are developers working on the implementation of the solution and the customer.

2 Scope

2.1 Description of the given problem

SafeStreets is a community service that aims to provide the users of the possibility to notify parking and traffic violations. It can be done providing proofs of the accident like pictures.
The goal is to raise awareness in drivers that driving and parking can make other's life harder. With a little effort, instead, it can be made easier.
The service will be served through a mobile application where users can report new violations and see the ones they reported before. Every attached picture comes with the metadata provided by the smartphone (like: time, position...) and with valuable information given by the user whom can describe the violation. More pictures of the same violation will be connected crossing the spacial and time information so that more viewing angles will be available.
To ensure the chain of custody, the backend infrastructure will run an AI algorithm to ensure the quality of the data given (e.g. no edited picture) before considering them valuable.
Authorities have access to all the data provided by the users, plus:

- Suggestions from the system on ways the municipality can improve the safety of the streets, directly from their dashboard. These are computed by crossing SafeStreets reports with the municipality accident and traffic tickets data;
- Automatic acquisition of reports to produce traffic tickets from the data collected;
- Maps of the less safe roads in the city;
- Vehicles that commit the most violations and other statistics for their municipality.

2.2 Goals

- [G1] Allow citizens to register providing basic information that certifies their identities and become certified users;
- [G2] Allow authorities to register through a verification procedure;
- Enable certified users to:
  - [G3] report new traffic or parking violations;
  - [G4] see their own past reports;
  - [G5] access the map showing the most reported streets.
- Enable authorities to:
  - [G6] access all certified users services;
  - [G7] access all report data (including data about the user who created it);
  - [G8] acquire report data in their own systems to generate tickets;

  - [G9] access data analysis results with suggestions.

- ▷ Enable administrators to:
  - [G10] certify authorities;
  - [G11] block or remove users;
  - [G12] assign roles to users.

3 Definitions, Acronyms, Abbreviations

3.1 Definitions

- ▷ *Violation*: illegal behaviour executed by the Offender
- ▷ *Violation report / Report:* information regarding a single violation, sent by a Registered User through SafeStreets
- ▷ *Server:* SafeStreets server which elaborates the reports and communicates to the municipality and clients through custom APIs

Please refer to the RASD (5) for the definitions of actor such as Offender and Registered User.

3.2 Acronyms

- ▷ *GDPR:* General Data Protection Regulation
- ▷ *API:* Application Programming Interface
- ▷ *GPS:* Global Positioning System
- ▷ *UI:* User Interface
- ▷ *UX*: User eXperience
- ▷ *ID:* Identification Document
- ▷ *SLA*: Service Level Agreement
- ▷ *DBMS:* DataBase Management System
- ▷ *RDBMS:* Relational DBMS
- ▷ *VM:* Virtual Machine
- ▷ *ER*: Entity Relationship
- ▷ *REST:* REpresentational State Transfer
- ▷ *OS*: Operating System
- ▷ *BLOB:* Binary Large OBject
- ▷ *SDLC*: Software Development Life-Cycle
- ▷ *BL*: Business Logic

3.3 Abbreviations

- ▷ *[Gn]:* Goal n
- ▷ *[Dn]:* domain assumption n
- ▷ *[Rn]:* Requirement n

4 Revision history

- *Version 1.0:*
  Initial release

5 Reference documents

- Specification document: "SafeStreets Mandatory Project Assignment"
- SafeStreets RASD
- AWS services reference:
  - Firewall Manager: "https://aws.amazon.com/firewall-manager/"
  - Elastic Load Balancing: "https://aws.amazon.com/elasticloadbalancing/"
  - EC2: "https://aws.amazon.com/ec2/"
  - RDS: "https://aws.amazon.com/rds/"
  - S3: "https://aws.amazon.com/s3/"
- AWS Service Level Agreement: "https://aws.amazon.com/legal/service-level-agreements/"

6 Document structure

This document consists of the following parts:

1. Brief description of this document, its nomenclature and the description of the project.
2. Description of the architecture of the system to be developed, focusing on its component, their dependencies and interactions, how they will be deployed.
3. Specification of the system's user interfaces.
4. Maps components of the system to the requirements they fulfil.
5. Description of the order in which various parts of the system will be developed based on dependencies and importance.
6. Reporting of the time spent by each team member while redacting the document.

Part II
Architectural design

1 Overview

The SafeStreets service is a distributed system of which the inner workings can be logically sliced in four layers. Each layer only interact with the "next" and "previous" layers, going from the components the user interacts with to the business logic.
This approach guarantees high modularity and inherent security at the same time, since a layer doesn't have a way to see anything beyond the layers it contacts directly.
The aforementioned layers, described in detail in 2, are the following:

- ▷ Client components are the ones directly interacting with the platform's users, and which will be directly delivering all of its features
- ▷ REST API components provide an interface to the core of the system to be used by the clients, exposing only what is needed to every category of customer
- ▷ The Database layer represent the storage of every piece of data archived, generated or elaborated by the system
- ▷ The Business Logic components feed the database using external sources of relevant data or by computing statistics on data the system generated or acquired in the past

2 Component view

The following diagrams show the components of the SafeStreets system. Each diagram shows a layer of interaction between logical levels, starting from the interfaces used by different kinds of users, the private REST APIs, the database and the logic used to populate it.

2.1 Client components

The following are the components used by clients to access the services provided by SafeStreets.
They can be found in Figure 2, where their relationships with REST APIs is showed.

- ▷ The *Mobile application*, the only available interface of the system for common users, lets them use the main features of SafeStreets:
  - It is possible to register a new account or log in an existing one
  - Users can send reports and view the ones sent by the currently logged in user
  - Information about the safety of particular streets can be obtained
- ▷ The *Web client* interface lets privileged users like Officers and Administrators carry out their respective jobs
  - Authorized users can log in
  - Authorized users can see and filter every report on the platform, sent by any user
  - Administrators can obtain data on all user accounts, change their roles or administratively create new account for the municipality's employees
  - Officers can read the suggestions generated by the system

2.2 REST API components

These are the private APIs used to achieve communication between client interfaces and the service's data, and represent the transactional operations of the system.
They can be found in Figure 2, where their relationships with REST APIs is showed, and in Figure 3, where their relationships with database components is showed.

- ▷ The *Mobile application REST API* provides all characteristic operations of the mobile application
  - The *Reports module* manages the upload of new reports and provides access to all previous reports by the same user
  - The *Street safety* module gives detailed information of every street the user wishes to view
- ▷ The *Accounts REST API* provides transactional operations on user accounts
  - The *Registration module* manages all steps of the creation of new accounts
  - The *Login module* is contacted when a user wants to authenticate in order to access SafeStreet's services
  - The *AccountDetails module* provides methods to retrieve and change account details, in particular it lets administrator change an account's role
- ▷ The *Web client REST API* provides access to features exclusively accessible by authorized users
  - The *AccountsData* module is used to show a list of all users of the system. This component is part of the Web client API rather than the Accounts API to avoid exposing this method to an endpoint which is also accessible to common users, since returned data is highly sensitive (it includes fiscal codes and identification document pictures).
  - The *Suggestions module* gives access the the suggestions generated by SafeStreets
  - The *Reports module* lets administrators and authorities view the whole collection of reports in the database

2.3 Database components

The database components are logical sections of the system's data storage system, which can be logically split in different databases.
Each of them contains data relevant to different aspects of the system's structure.
They can be found in Figure 3, where their relationships with client components is showed, and in Figure 4, where their relationships with business logic components is showed.
Databases are passive components only, so they provide interface without depending on any other.

- ▷ The *Reports database* stores data from all the reports
- ▷ The *Safety&Suggestions database* stores the results of SafeStreet's engine elaboration
- ▷ The *UserAccounts database* stores details of every user registered on the platform
- ▷ The *Accidents&Tickets* database stores data received form the municipality's systems to be used for mining purposes. This database has no contact with the REST "layer", since this information is highly confidential and can only be exposed in aggregated form in the results contained by the *Safety&Suggestions database*

2.3.1 Entity Relationship diagram

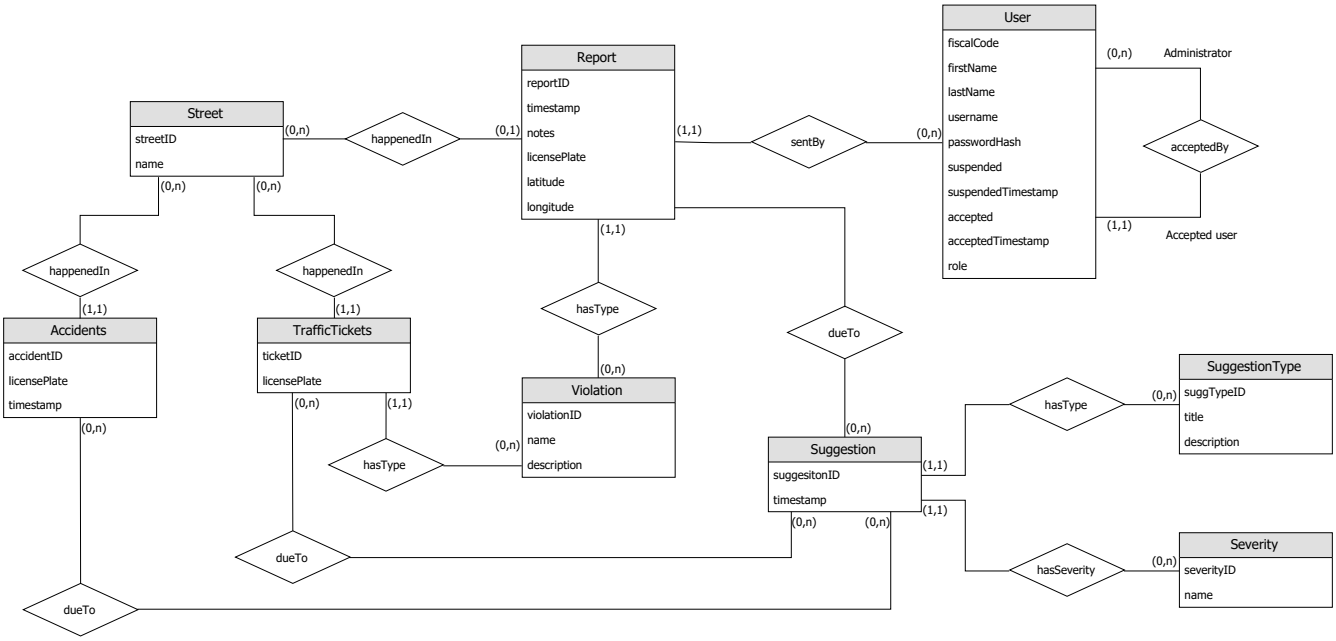


Figure 1: Database ER diagram

2.4 Business logic components

The following components are the ones responsible for operations of the system which are not driven by user interaction.
They can be found in Figure 4, where their relationships with database components is showed.

- The *Safety&Suggestions engine* is responsible for mining accidents, tickets and reports data to provide suggestions and safety ratings for different streets
- The *Municipality APIs adapter* manages access to the municipality's APIs by periodically sending and receiving needed data
  - The *Reports pusher* periodically sends reports generated after the latest push to the municipality's systems to create traffic tickets
  - The *Accidents&Ticket puller* periodically retrieves traffic contravention data provided by the municipality, feeding the corresponding local database

2.5 Component diagrams

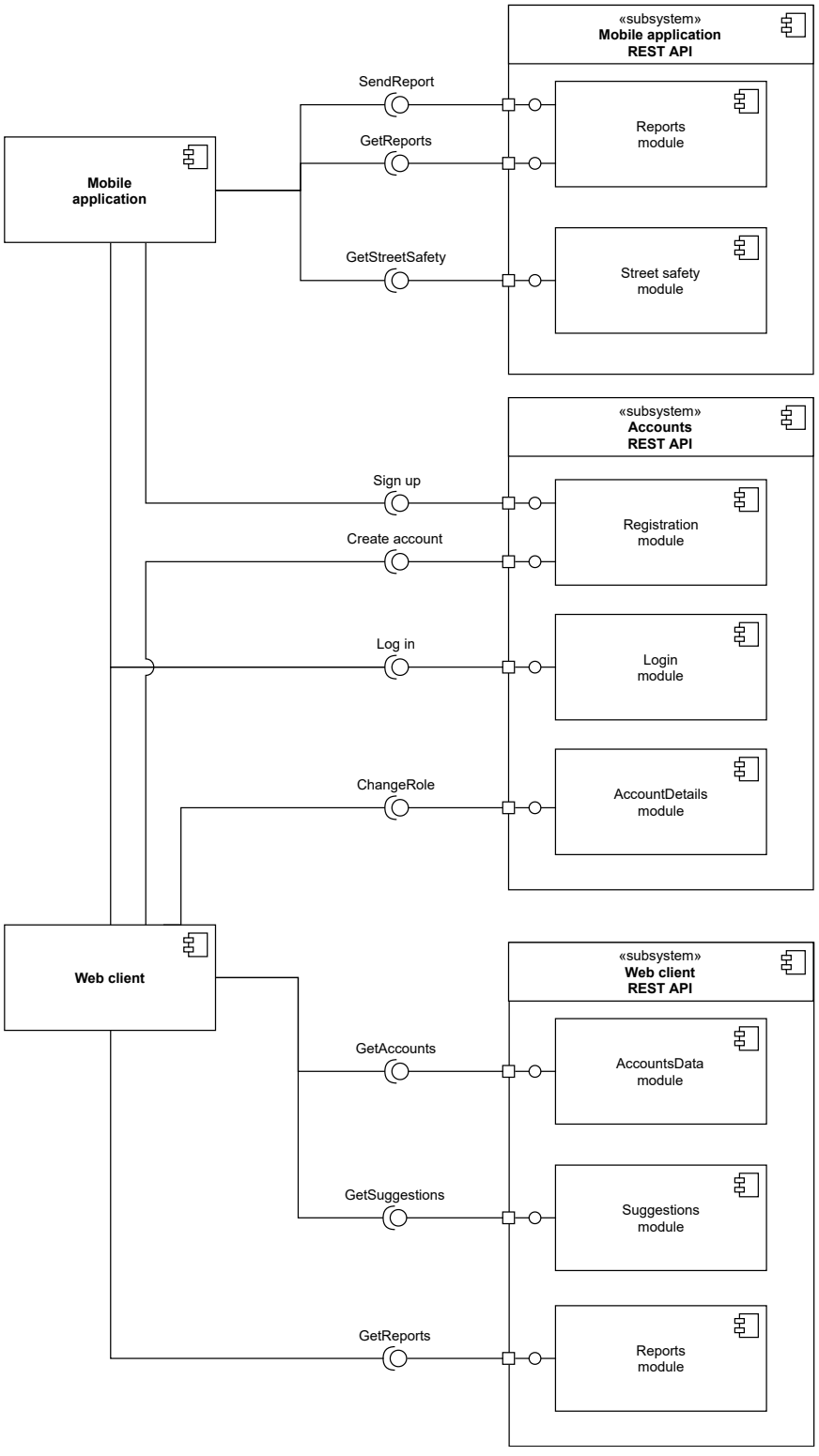


Figure 2: Client and API components

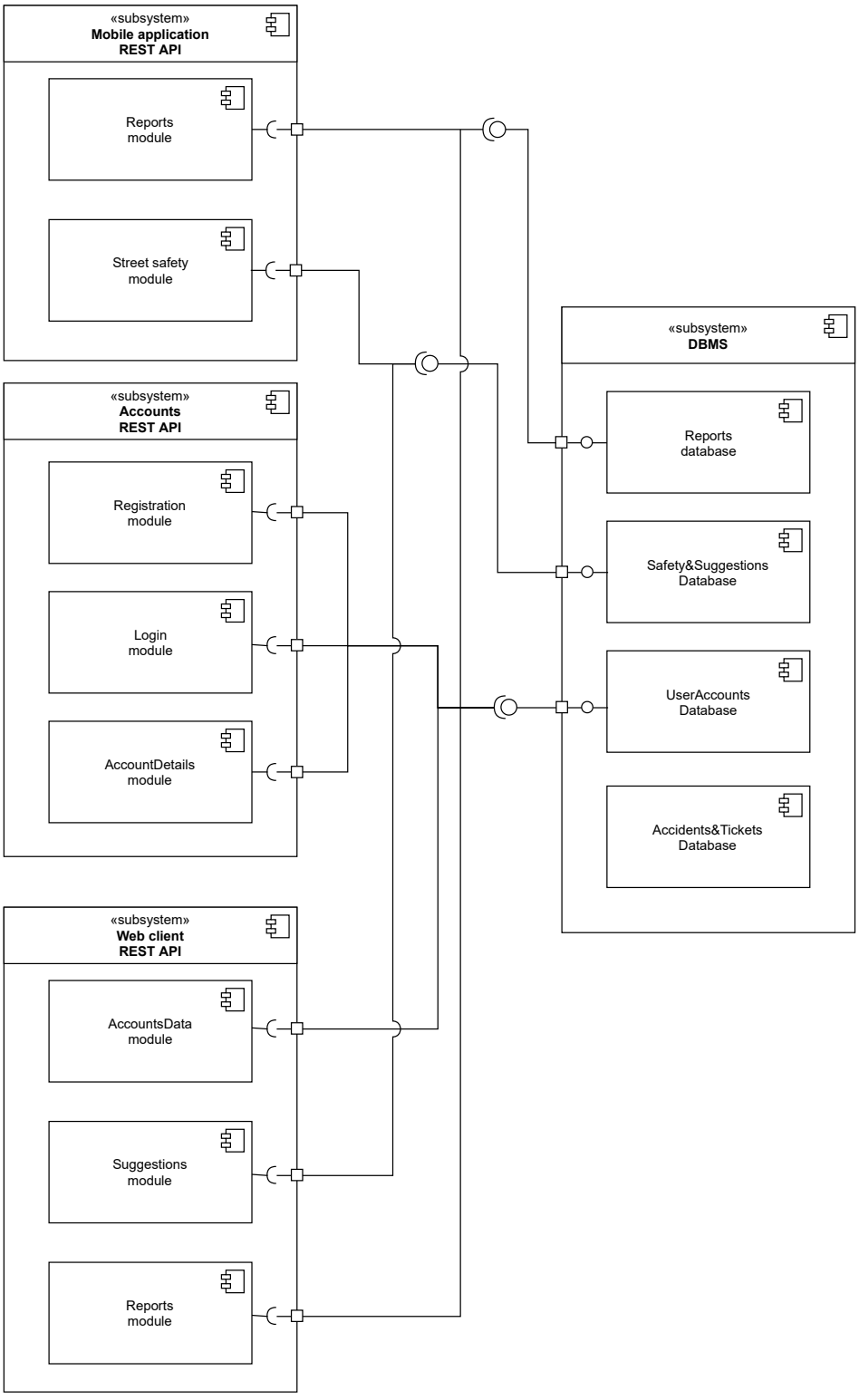


Figure 3: API and Database components

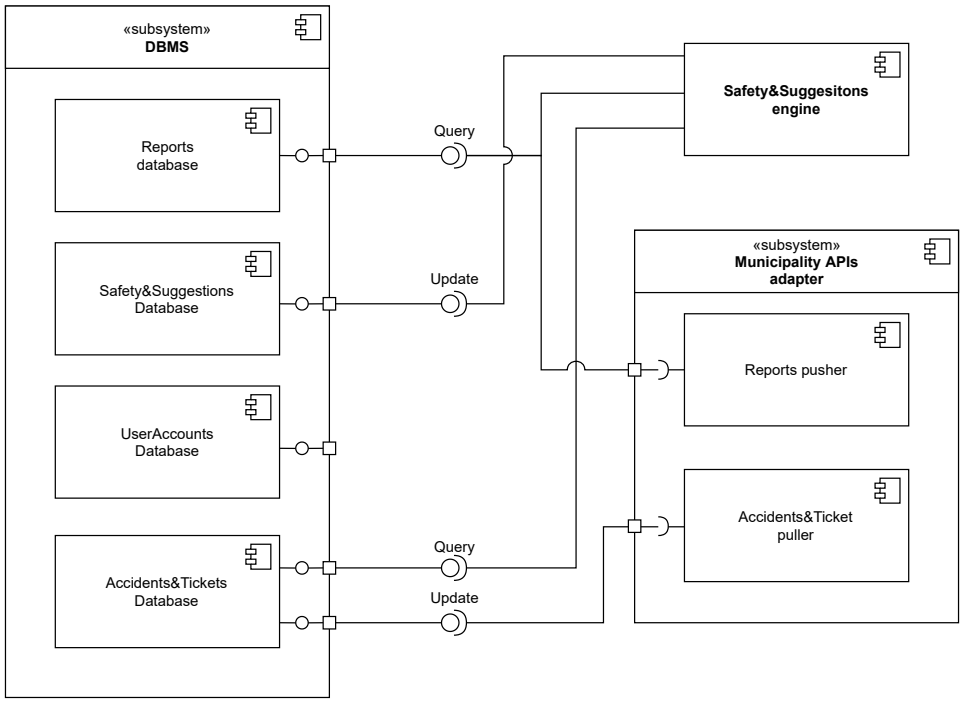
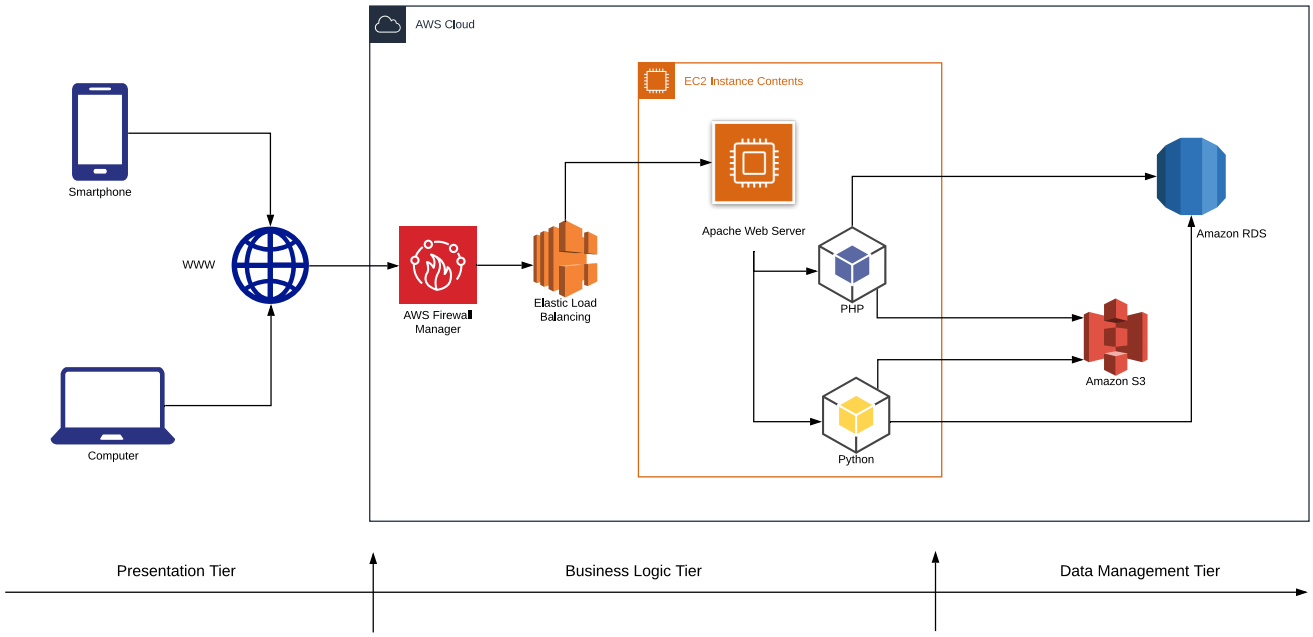


Figure 4: Database and Logic components

3 Deployment view

The architecture of SafeStreets is composed of three tiers:

- ▷ **Presentation Tier**: it is composed by the mobile application and the web application. This is the Tier that directly interacts with the user.
  - Mobile App: used by any user to access reporting features;
  - Web App: used by municipality Officers (they can see reports by users and statistical data) and Administrators (they can manage accounts and use all other features).
- ▷ **Business Logic Tier:** it is composed by the Apache web server, the PHP and Python interpreters. It handles the logic on the backend side.
  - Handles all RESTful APIs requests coming from the presentation tier;
  - Provides the Web Application hosting;
  - Runs the *Municipality APIs adapter* that handles dedicated municipality features;
  - Runs the *Safety&Suggestions engine* for data mining.
- ▷ **Data Tier:** it is composed by the database and the pictures hosting. This is the container of the data collected from the user.
  - A RDBMS is used to store structured data (everything but pictures);
  - A Data Storage is used to manage pictures uploaded by users.



3.1 Implementation

▷ **Presentation Tier:**

- *Mobile App*: the mobile application will be implemented using the cross platform framework *"Xamarin Forms"*. The App will be developed using a shared C# codebase from which both Android and iOS native applications can be built.
  Xamarin has been chosen because it makes implementation much faster than making two different applications while maintaining the same experience for both the OSs.
- *Web App*: the web application is going to be written in PHP, HTML, CSS, JS, using the jQuery library and the SemanticUI development framework for its frontend.
  The data will be served following an Ajax approach, accessing data from SafeStreets' APIs.
  It will be served through a secured TLS connection (HTTPS) to ensure privacy and security of the communication channel.
  A Web approach has been chosen to provide an always-available on any-platform service that can be easily accessed in Offices where there are company PCs .

▷ **Business Logic Tier:** on a virtualized Linux x64 machine, potentially replicated by the Elastic Load Balancer:

- Apache web server: handles HTTPS requests over the network for both the APIs and the Web App.
- PHP 7.4 runtime:
  - runs scripts implementing the *Mobile application REST API*, *Accounts REST API* and *Web client REST API*;
  - runs the Web App dynamic pages.
- Python: runs the *Safety&Suggestions engine*:
  It has been chosen for its libraries on data mining, geocoding and image manipulations.

▷ **Data Tier:** implemented with Amazon RDS and S3 services:

- Maria DB on Amazon RDS: Relational DBMS used to store data (apart from pictures) such as users data, reports data and location ones.
  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.

– Amazon S3: used as storage for all the users pictures.
  Pictures are stored outside the DB because are BLOBs that have no advantages in being collected in a DB and that can slow it down. The DB contains the path to the picture related to an event.

3.2 Infrastructure

Amazon AWS has been chosen as the cloud services provider for the infrastructure of the system.
Adopting this ecosystem has multiple advantages:

- ▷ Using AWS purchase, maintenance and running (power and thermal management) costs can be up to 80% less compared to an equivalent on-premises solutions. Maintenance costs are calculated with awstcocalculator.com
- ▷ Data replication and safety is handled by the service provider, including off-site backups and almost instantaneous failure recovery
- ▷ The Service Level Agreement of chosen components guarantees an availability of 99.99%
- ▷ Security is handled by the service provider with regular patching on most of the components, since they are provided as PaaS. The only exception to this are EC2 instances (IaaS) where the user has to manage the software running on them
- ▷ Choosing a single vendor for the whole cloud infrastructure guarantees ease of integration between components

3.2.1 Virtual machines

Our virtualized Linux x64 Machines will be deployed on AWS on EC2 instances.
Estimations of load and required performance brought to the deploy of the following machine:

- ▷ BL Tier: 8 v-cpu 32GB ram and 256GB SSD Linux machine;

In cases of intense load on the system (during peak hours or unexpected events), an AWS Elastic Load Balancing is used to dynamically spawn replicated copies of the instance. This way scalability can be easily and automatically managed by the infrastructure, and needed resources will be billed only when they are needed.
Concurrent access of the replicated EC2 instances to the Data Tier is discussed in 3.2.2.

3.2.2 Storage

Amazon RDS will be used instead of MariaDB running on a dedicated VM for two reasons:

- ▷ In case of replications of the Business Logic Tier, access to the database will be seamless and will scale with load. Otherwise, it would be be necessary to manually increase the computing power of the database's host or to create multiple instances of it, making it necessary to take into consideration the problem of keeping replicated data consistent.
- ▷ Resources used by a dedicated machine would be billed regardless of real use of the database.
  By using RDS, instead, billing is evaluated on real use. Thanks to this it's possible to lower the cost of the infrastructure by exploiting periods of low load (e.g. night-time).

The same reasons apply to the choice of Amazon S3 for the storage of pictures.

4 Runtime view

In this section the main functions' sequence diagrams are reported. Diagrams for common functions, like login or signup, have been omitted as they follow a standard implementation.
Each reported sequence is a more detailed explanation of the ones reported in the RASD document, as they explain the logical and functional steps behind the functionalities of the system.

4.1 Create a report

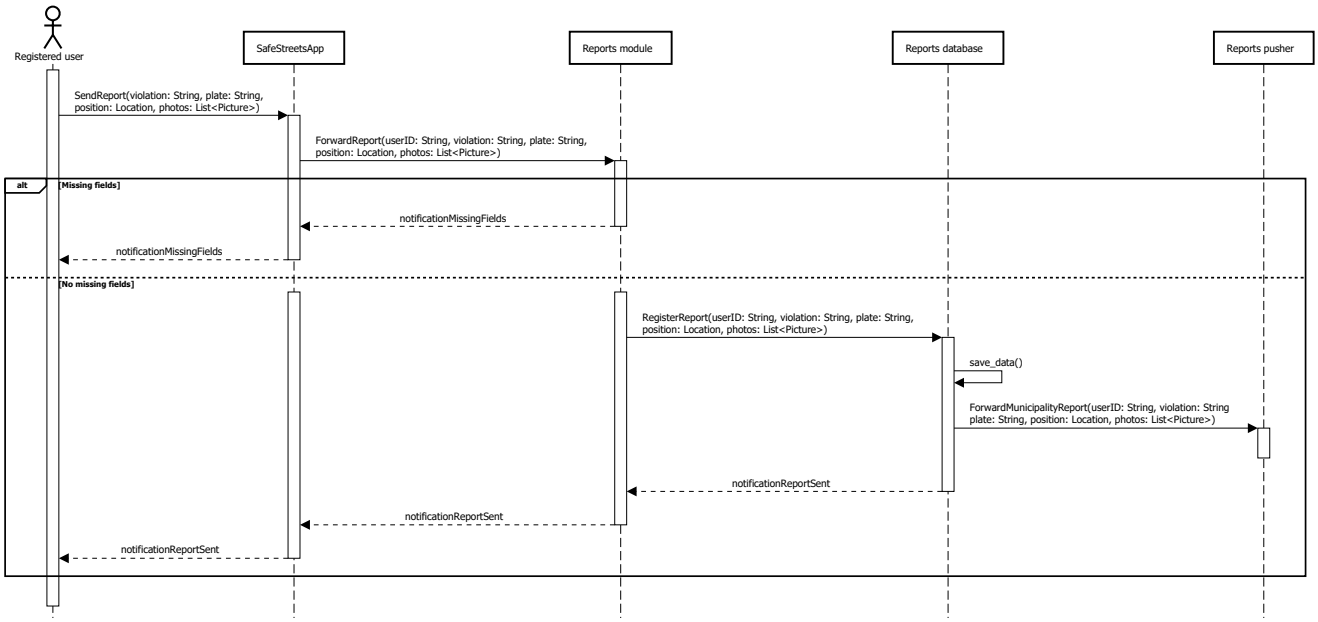


Figure 5: Create report - sequence diagram

This sequence diagram explains the backend procedure for handling a report creation.

- SafeStreets App adds the user ID to the report and sends all information to the *Reports module* Mobile application REST API.
- The Reports module forwards the informations to the *Reports* DBMS, which saves the information about the report and forwards them to the municipality through the *Reports pusher* Municipality APIs adapter.
  If there are missing data pieces in the request, the *Reports module* doesn't forward the information to the DBMS and generates a notification for missing parameters.

4.2 See past reports

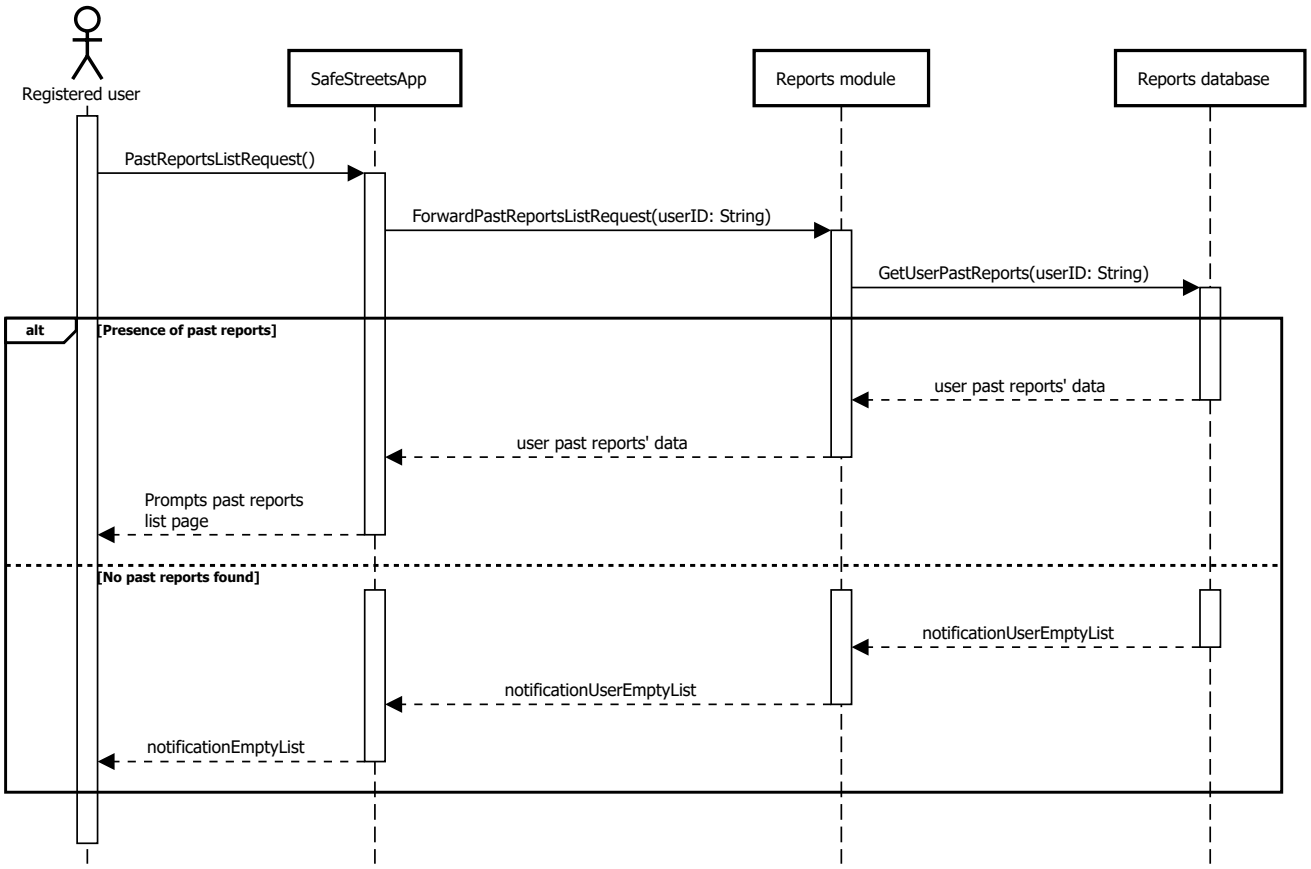


Figure 6: See past reports - sequence diagram

This sequence diagram explains the backend procedure for showing the user's past reports.

- ▷ The SafeStreets App adds the user ID to the report and sends all informations to the *Reports module* Mobile application REST API.
- ▷ The Reports module forwards the informations to the *Reports* DBMS, which retrieves the data and send them back in the chain.
  If no past reports are found, an Empty list notification is created by the DBMS and sent to the user through the *Reports module* and SafeStreets App.

4.3 See Street Safety

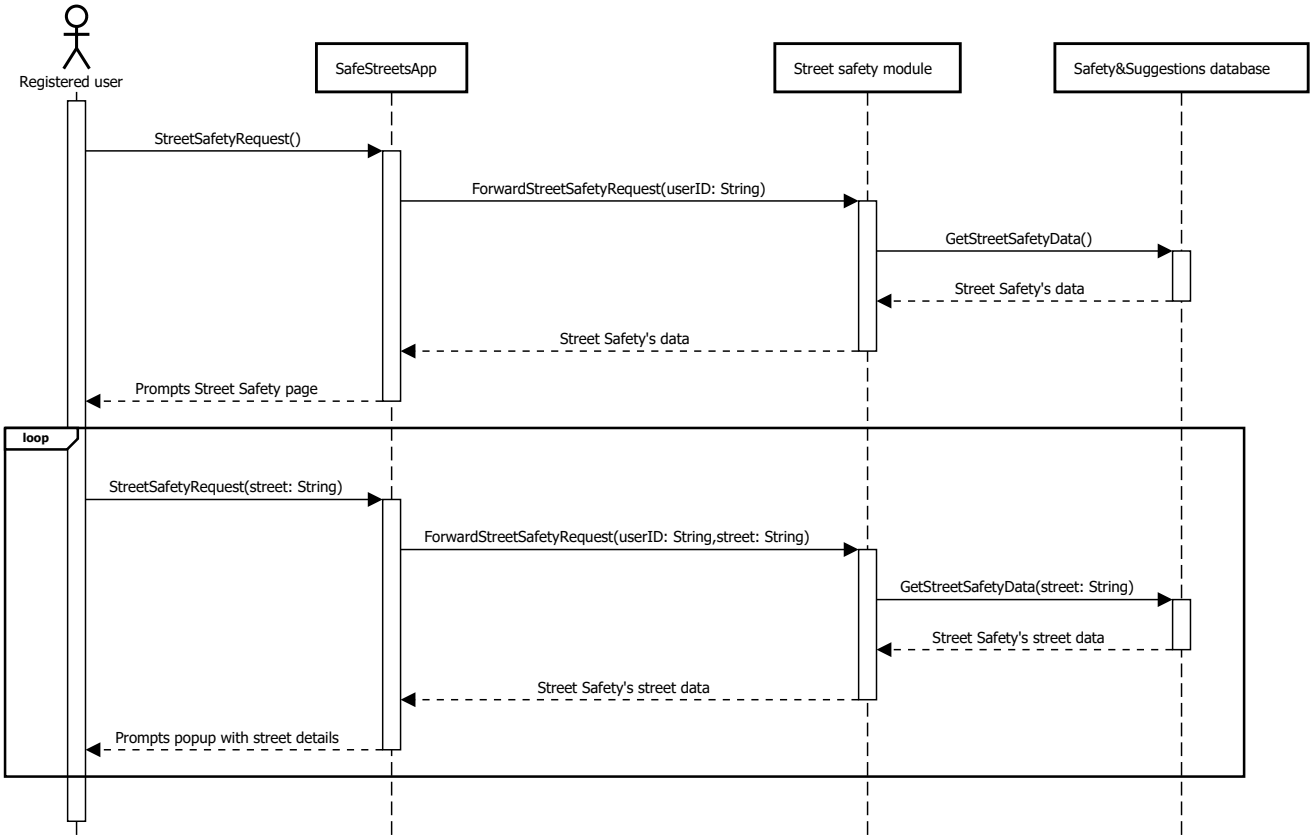
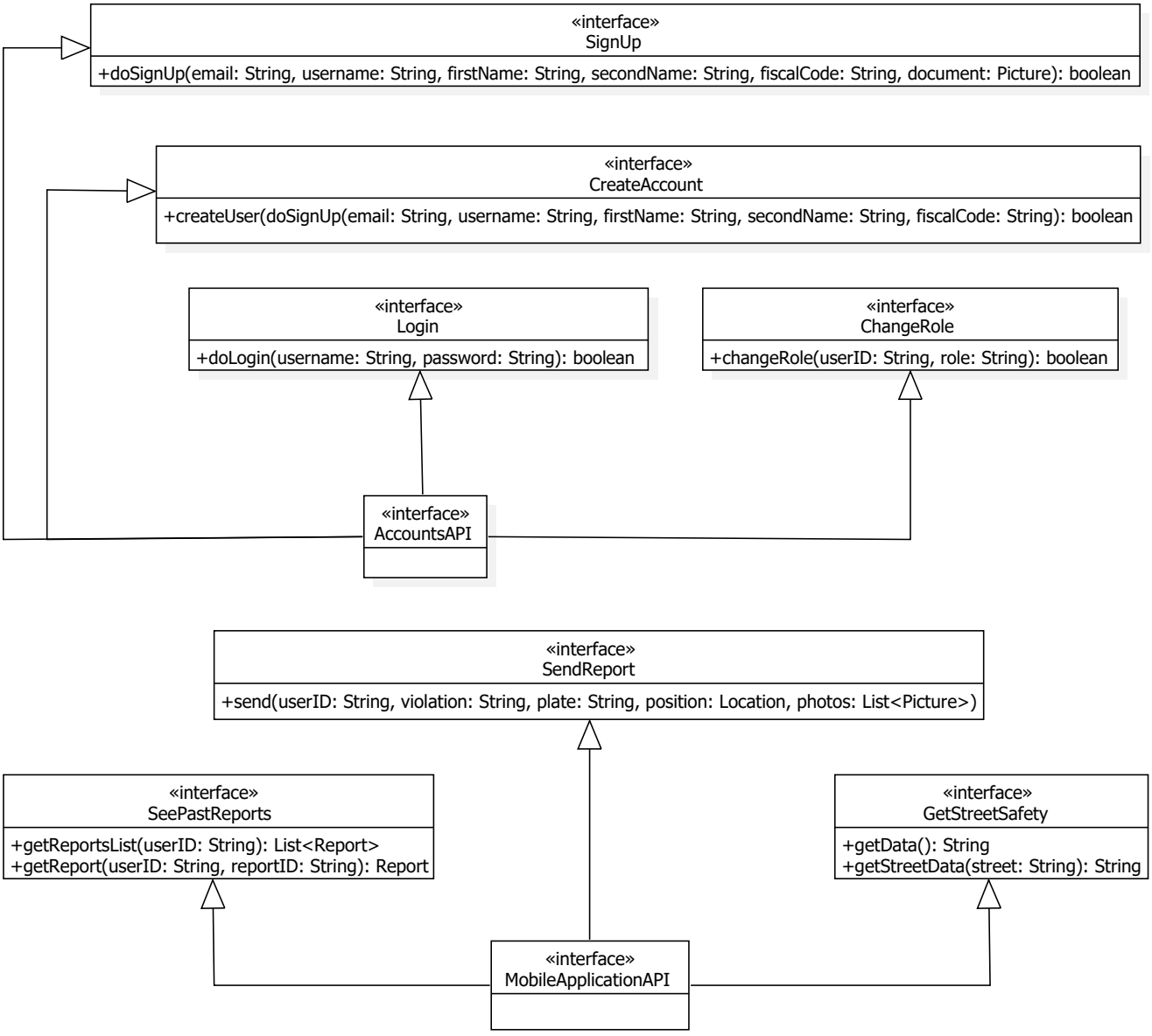


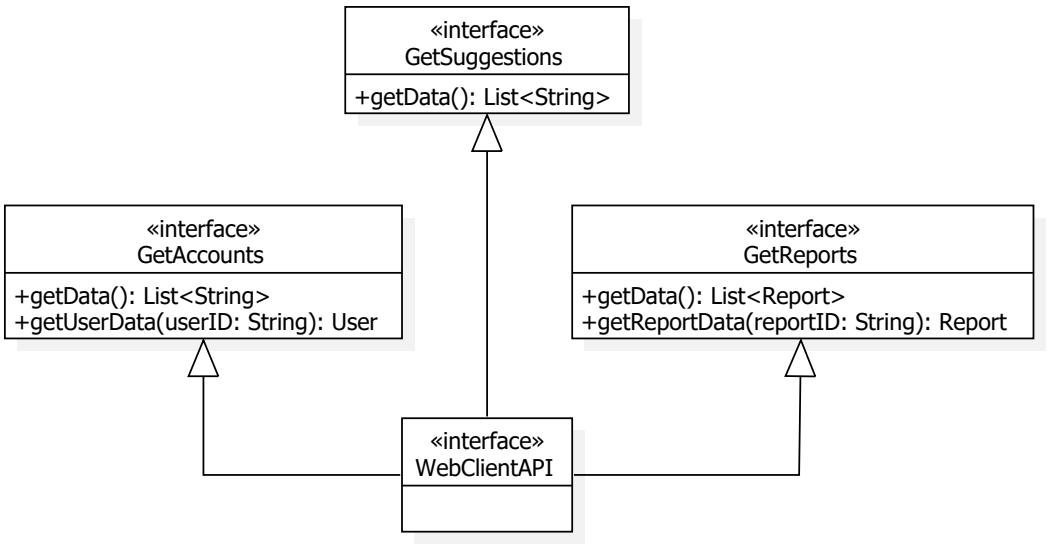
Figure 7: See Street Safety - sequence diagram

This sequence diagram explains the backend procedure for showing Street Safety information.

- ▷ The SafeStreets App adds the user ID to the report and sends all informations to the *Street safety module* Mobile application REST API.
- ▷ The Street safety module forwards the informations to the *Safety&Suggestions* DBMS, which retrieves the data and send them back in the chain.
- ▷ Whenever a street is clicked, a specific request for data concerning that street is sent in the same way, adding details about the specific street to the request.

5 Component interfaces





This diagram shows the main methods belonging to the interfaces of the components.
In particular:

- *Accounts API* interface is the interface for Accounts REST API subsystem, providing all methods needed for signup, login, third person account creation and role changing. The login method is used for both mobile application and web platform.
- *Mobile Application API* interface is the main backend interface for mobile platform, as it gathers all methods needed for the application functions, such as report sending, past reports view and provisioning of Street Safety elaborated data. It provides all methods from Mobile Application REST API subsystem.
- *Web Client API* interfaces Web Client REST API subsystem to the web application used by Authorities and Administrators. The interface provides methods for accounts, reports and suggestions retrieval. It is important to highlight that the method for accounts data retrieval is in the Web Client interface and not in Accounts to avoid personal data retrieval by common users.

6 Selected architectural styles and patterns

The adoption of well know architectural styles such as RESTful and three tier (architecture) can be seen throughout all the SDLC (software development life-cycle).
Each of them has been chosen between different options for their advantages in the Agile project management of SafeStreets.

6.1 RESTful architecture

RESTful APIs has been chosen to grant a flexible, but still completely defined endpoint for all the client-server interactions.
Thanks to the fact that API interfaces have been completely defined before the start of development, the Mobile Application and the APIs implementation can be built in parallel to save time.
The use of RESTful APIs also provides an easy to consult implementation of the communication protocol. APIs' responses come in form of JSON data structures.
Another advantage is the sharing of the client-server communication standard between the Mobile App and the Web App. In future versions third-parties could also easily access our APIs, under predefined constraints, in order to connect them with their systems.
To prevent external unwanted behaviours from possible hackers, the APIs are going to be served through CORS policies only to our Web and Mobile Applications.
Other RESTful advantages are the stateless operations, the uniform access interface and the definition of a well layered system.
All the request will be served over the HTTPS protocol in order to guarantee the security the user is expecting.

6.2 Three Tier Architecture

The Three Tier Architecture is a client-server approach where presentation, business logic and data are divided.
Its key feature is the possibility to divide data from the logical computation and the user representation, so that different security measures can be implemented to improve the privacy of the user informations.
It forces developers to work on isolated modules where components have well defined interfaces.
The complexity of the system has also been decreased thanks to the adoption of the three tier approach.

6.3 RDBMS

In order to easily manage users' data, with fast and complex but easy-to-implement statistical queries, the use of a RDBMS was desirable. Using a non relational database could have brought better scaling but the flexible data model and the absence of the SQL language would have been an harder obstacles to overcome.

6.4 Thin Client

To create a lightweight Mobile Application on both new and old devices, with fast or slow internet connection, it has been decided to adopt a Thin Client approach.
The client is developed to work only as an input/output for all the communications to the server, where the business logic is deployed.
This approach makes the User Interface faster on old devices and is a good option for slow connections too, making the User eXperience enjoyable.

6.5 Model View Controller

The MVC approach has been chosen for the development of the Mobile Application to ensure the division of objectives for each part of the code.
The Model is the component that interacts with the server and retrieves data through the RESTful APIs.
The View shows the data from the Model to the user, with native interfaces on both platforms.
The Control is responsible for handling user interactions through the View and to modify the state of both the Model and the View.

6.6 Cross Platform

Developing a widespread Mobile Application requires the creation of apps for both iOS and Android.
Building dedicated applications for each platforms is a good option for big teams. The constraints of the SafeStreets project brought to the adoption of a cross platform approach that could decrease implementation price and time while giving dedicated UI and experiences on each platform.

Part III
User interface design

User interface mockups for both the mobile application and the web client have been showed and described in section 1.1 of the RASD of this project, and are not reported here for this reason.
The following User Experience diagrams show how the user will be able to navigate pages in both clients.

1 Diagrams

1.1 Mobile application UX diagram

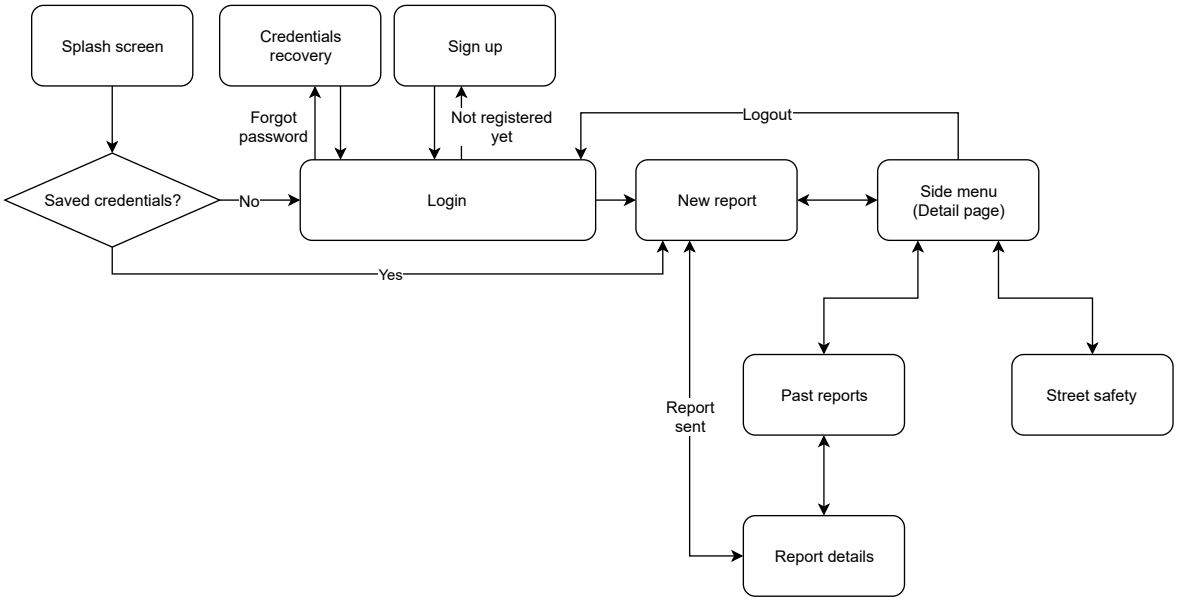


Figure 8: Mobile application UX diagram

1.1.1 Start-up

While the application is starting, a *Splash screen* showing its logo is presented to the user until all resources are loaded.
At this point, the application checks local storage for saved credentials and tries to authenticate with them.

- ▷ If the process is successful the application skips the *Login page* and goes to the *New report* page
- ▷ If no previous authentication is found, or if the login attempt with saved credentials fail, the application switches to the Login page

1.1.2 Login

From the *Login page*:

- ▷ Registered users can input their credentials to authenticate, or switch to the *Credentials recovery page* if they forgot their username and/or password
- ▷ Unregistered users can go to the *Sign up page*, input all required information and send a registration request

1.1.3 New report

This is the default master page in the master-detail layout, described in the previously cited section of the RASD. All other pages can be accessed by opening the *Side menu* by dragging from the left side of the screen.
The *New report page* allows users to fill a form with information regarding a violation and submitting it. After the report is successfully sent, a *Report details page* with a summary of its content will be displayed. Tapping the "back button" will bring the user back to the *New report page*.

1.1.4 Side menu

This is the detail page in the master-detail layout, described in the previously cited section of the RASD.
The *Side menu* allows the user to navigate between the main functions of the application

- ▷ Tapping the "Report" item will open the *New report page*
- ▷ Tapping the "Past reports" item will open the *Past reports page*
- ▷ Tapping the "Street safety" item will open the *Street safety page*
- ▷ Tapping the "logout icon" will clear locally saved credentials and bring the user back to the *Login page*

1.1.5 Past reports page

The *Past reports page* consists in a list of brief summaries of previously sent reports from the currently logged in user. Every entry shows the type of violation, location and time of submission (date or time, depending on the fact that the report was submitted in the same day the page is consulted or not).
Tapping on a report opens the *Reports details page*, which shows a more extensive summary of the report's content. Tapping the "back button" will bring the user back to the *Past reports page*.

1.1.6 Street safety

The *Street safety page* consists of a map of the user's surrounding area where the streets are highlighted with different colours depending on the level of safety determined by SafeStreet's algorithms. Tapping on a street will bring up a popup containing more detailed information on relevant events happened in that street.

1.2 Web client UX diagram

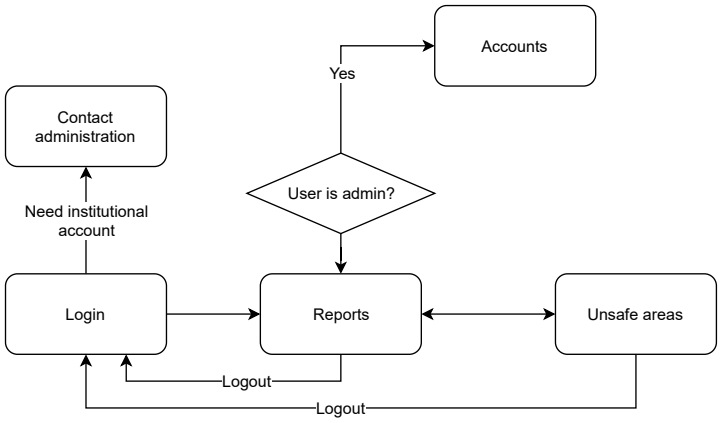


Figure 9: Web client UX diagram

1.2.1 Login

Accessing the web client's URL will bring the user to the *Login page*, consisting of a simple form.

- ▷ Registered and privileged users, such as Administrators and Authorities, can insert their credentials and log in to access the *Reports page*

- ▷ Officers or municipality employees who need access to the web client can click the "Contact us" link to access the *Contact administration page*, which will provide all information needed to contact SafeStreets n behalf of the municipality to request an institutional account

1.2.2 Reports

The *Reports page* is the default page for both Administrators and Authorities. It shows a detailed table of every report submitted by the service's users.
From the sidebar of this page, all other functions of the Web client can be accessed.

1.2.3 Sidebar

- ▷ Clicking on the "Accounts" button (only visible to Administrator users) the *Accounts page* will open
- ▷ Clicking on the "Reports" button the *Reports page* will open
- ▷ Clicking on the "Unsafe areas" button the *Unsafe areas page* will open
- ▷ Clicking on the "Log out" button, the user's authenticated session will be terminated and the *Login page* will open

1.2.4 Accounts

The *Accounts page* lets Administrators manage user accounts. From this page:

- ▷ The role and details of each account can be changed
- ▷ Registered but not yet approved accounts can be approved
- ▷ New users can be administratively created

1.2.5 Unsafe areas

The *Unsafe areas page* lets privileged users consult a list of suggestions and safety notifications created by the Business logic layer of the system

Part IV
Requirements traceability

Objective of the Design Document is to achieve, through the development, the goals previously defined in the RASD with different requirements for each of them.
This part of the document defines how each requirement is going to be satisfied throughout the implementation of the different software components of the SafeStreets infrastructure:

- **[R1]** Every citizen that wants to register has a valid identification document and a smartphone (with GPS and camera systems):
  - *Mobile App*: in order to access the Mobile Application, the user needs to download it through the App Store or the Play Store.
  - *PHP backend*: checks the validity of the ID calculating the Fiscal Code.
- **[R2]** Every citizen that wants to register owns a valid e-mail address:
  - *Mobile & Web App*: the input text field (using a regex) ensures the email is in a valid format. An automatic email is sent with a verification link to the user, to enable its account, the link has to be opened.
- **[R3]** The municipality allows SafeStreets administrator to verify the belonging of the requesting officer to the traffic police (via an automated system or sending an official request):
  - *Web App*: when a request for an officer account has been received, an automatic email is sent to a dedicated email address (PEC email) of the Police with a link. In case the link is visited, the validity of the "Officer Position" is enabled.
- **[R4]** The user has his smartphone with him, is able to login in the application and is not suspended:
  - *PHP backend*: the login authenticator ensures that the user that tries to log in is registered and is not suspended.
- **[R5]** The user's smartphone has Internet connection enabled and functioning:
  - *Mobile App*: in case requests to SafeStreets servers fail, the Mobile App tries to ping two well known DNS servers. If one of the connections can be established, the internet connection of the user is working while our systems are down.
- **[R6]** The user's smartphone has GPS enabled and functioning:
  - *Mobile App*: the Mobile App triggers a request to the user to grant access to the GPS for the application. Every time the position is needed, it first checks that data is updated and still accessible to the app.
- **[R7]** The user is able to take at least one picture of the violation to correctly choose the type of violation he is reporting:
  - *Mobile App*: the application requires the permissions to access the camera.
  - *Mobile App*: suggest a group of possible violations between which the user chooses the correct one.
- **[R8]** The user has already made at least one report:
  - *PHP backend*: counts the number of reports made by a user.
- **[R9]** The officer access has the same base functions of the basic user access, plus specific ones:
  - *ALL*: the officer can access all the normal user's functionalities through the Mobile & Web Apps.
- **[R10]** The officer has a computer with Internet access and is able to login to the system:
  - *PHP backend*: the login authenticator ensures that the user that tries to log in is registered, not suspended and has the Officer role.

▷ **[R11]** The municipality and/or the authorities use the provided APIs:

- *PHP backend*: the municipality APIs are connected to the PHP backend

▷ **[R12]** The administrator has a computer with Internet access and is able to login to the system:

- *PHP backend*: the login authenticator ensures that the user that tries to log in is registered, not suspended and has the Administrator role.

▷ **[R13]** The administrator has the possibility to use an official communication channel with the municipality:

- *WebApp*: in the contacts page, the PEC email of the municipality is provided.

▷ **[R14]** The administrator has received a block/removal request from the authorities:

- *PHP backend / Web App*: tracks the block/removal requests coming from the authorities and give the Administrator the possibility to change a user status.

▷ **[R15]** The administrator has received a sign-up request from the authorities and/or from the municipality:

- *PHP backend / Web App*: tracks the signup requests from the authorities/municipality.

Part V
Implementation, integration and test plan

The whole system can be logically divided into subsystem, so that is possible to assign a deployment priority order for development. It is possible to identify these subsystems:

- ▷ WebServer
- ▷ WebClient
- ▷ MobileApplication

WebClient and MobileApplication refers mainly to the end-user application interfaces and does not contain the application logic; for this reason they can be implemented in parallel with the WebServer once the APIs for the services have been formally defined. Because of the user interface primary nature of these subsystems, their testing part will focus only over the communication part with the server. Integration will follow the implementation progresses of the functionalities in the WebServer.
WebServer subsystem instead contains the main logic of the application, with also the DBMS for data storage. It can be implemented incrementally following the feature implementation order described in a while. Once a functionality is fully implemented, the corresponding part of the user interface can be integrated in the system and tested in the whole.
External services such the municipality APIs does not need neither to be implemented nor to be tested, as it is assumed their reliability within the system.
The below table resumes the functionalities of the system in association with their priority level.

Functionality	Importance for the client	Implementation difficulty	Priority level
Signup and login	Medium	Medium	1 - High
Violation reporting	High	Low	1 - High
Own reports history	Low	Low	3 - Low
Street Safety visualization	Medium	High	2- Medium
Account managing	Medium	Low	1 - High
Whole reports history	Low	Low	3 - Low
Unsafe area analysis	High	High	2 - Medium

As shown, the implementation priority level doesn't follow client importance levels or implementation difficulty, because there are key components that need to be fully implemented and tested in order to use the other ones.
The following is the implementation and testing order:

1. **Signup and login**: these functionality has to be the first one to be implemented because all the system relies on the possibility to identify the user and to validate its identity before allowing system usage.
2. **Account managing**: as before said, the system relies on the identification and validation of users; for this reason the account managing functionality is essential both for users and for authorities for system access.
3. **Violation reporting**: because of it is the core functionality of the whole system, the violation reporting is the first active functionality that has to be implemented, in order to allow users starting using the system.
4. **Unsafe area analysis**: this is a core functionality for the municipality, so it has to be implemented as soon as the violation reporting is implemented and tested. For this feature there is also the need to implement the interface with the municipality through its APIs, to retrieve data about the made checks. This is the most complex functionality to be implemented, because it needs some heuristics for data analysis to produce effective suggestions for unsafe areas.
5. **Street Safety visualization**: differently from the precedent functionality, these one is a little easier to implement because collects data only from the internal dataset. However it has a lower importance for the client, so it is implemented after Unsafe area analysis.

6. **Whole reports history**: it is a low importance functionality as it provides only a list of past reports, with only the possibility to visualize it without any other possible action. It has a low implementation difficulty level as it is just a data retrieval.

7. **Own reports history**: as for the whole reports history, it is a low importance functionality. It is implemented after the whole reports history because it is the same functionality with only a reduction in the retrieved data.

For each functionality, the procedure for full implementation includes the testing and verification phase towards all the implementation phase, so that at the end of the functionality implementation the test part is also ready.
As soon as a functionality is implemented, it has to be integrated in the system and with the previous implemented functionalities, also with dedicated unit tests.

Part VI
Effort spent

Part	Hours
1. Introduction	0
2. Architectural design	9
3. User interface design	0
4. Requirement traceability	3
5. Implementation, integration and test plan	0

Table 1: Andrea Aspesi's effort spent

Part	Hours
1. Introduction	1
2. Architectural design	8.5
3. User interface design	2.5
4. Requirement traceability	0.5
5. Implementation, integration and test plan	0.5

Table 2: Elia Battiston's effort spent

Part	Hours
1. Introduction	0.5
2. Architectural design	8
3. User interface design	0.5
4. Requirement traceability	0.5
5. Implementation, integration and test plan	3

Table 3: Alessandro Carabelli's effort spent