DD

Carlo Dell'Acqua, Adriana Ferrari, Angelica Sofia Valeriani

November 27, 2019

Contents

Chapter 1

Introduction

1.1 Purpose

1.1.1 Purpose of the Platform

SafeStreets is a service that aims to improve the safety of the general traffic. This is achieved by creating a community of users who are able to report any violation they see while the system manages all the aspects of data validation and statistical analysis. Different services contribute to this purpose:

- The first service offered by the front end application is the report service. Any registered user can submit violation reports and SafeStreets will validate them as described in the following sections with the help of the community.
- The second service offered by the front end application is the the Unsafe Areas Map. SafeStreets will provide statistics about areas that have a higher risk of violations based on the reports it receives and the danger of the infractions. Data can also be collected from public services if available to increase accuracy.
- The third service is the ticket generation. Traffic policemen will have access to a dedicated section of the application where SafeStreets will collect validated reports. This will enable any registered policeman to take actions against those violations.

1.1.2 Purpose of this document

In this document a more detailed approach with respect to the RASD will be provided to explain how we intend to build the platform. The following paragraphs will describe the architecture of the physical system and its abstract software components, how they interact with each other and how they create the services we described in the RASD.

1.2 Scope

1.3 Definitions, acronyms and abbreviations

1.3.1 Definitions

- **Safe (or Unsafe) Area**: A geographical region, usually a set of streets, where less (or more) accidents occur than the average based on neighboring regions
- **Report**: A set of data containing all the information about a traffic violation
- **System/Platform**: The SafeStreets platform
- **Web Application**: A Rich Internet Application that enables users to access the functionalities of the system through a modern web browser without having to manually install any other software
- **Reverse Proxy**: A server that acts as an intermediary between the client and the application server.

1.3.2 Acronyms

- **DD**: Design Document
- **API**: Application Programming Interface
- **GPS**: Global Positioning System
- **HTTP**: HyperText Transfer Protocol
- **HTTPS**: HTTP over TLS
- **JSON**: JavaScript Object Notation
- **SQL**: Standard Query Language
- **HTML**: HyperText Markup Language
- **CSS**: Cascading Style Sheets
- **JS**: JavaScript
- **DBMS**: DataBase Management System
- **RDBMS**: Relational DBMS
- **UI**: User Interface

1.3.3 Abbreviations

- **Web App**: Web Application

Version	Major changes
1.0.0	First release

1.4 Revision History

1.5 Reference documents

- **Assignment:** SafeStreets Mandatory Project Assignment
- **Previous project example:**
  - **Assignment:** Mandatory Project Assignment AY 2018-2019
  - **Example document:** DD to be analysed AY 2019 2020
- OpenAI Web Source - Description of GPT-2 algorithm and basic machine learning features

1.6 Document Structure

1. **Introduction**: The first section is a general description of the system's scope and purpose. It also includes references of the document and definitions, abbreviations and acronyms used along the paper
2. **Architectural Design**: The second section describe the architecture of the platform from different views.
3. **User Interface Design**: The third section includes an overall description of the user interface, explaining how the user will interact with the system and how the user interface will help them through the different interaction scenarios
4. **Requirements Traceability**: The fourth section provides a traceability matrix that allow keeping track of the requirements
5. **Implementation, Integration and Testing**: The fifth section describes how the system will be implemented, how the integration with existing system will be made and how the test will ensure the stability of
6. **Effort Spent**: The sixth section includes the detailed information about the time spent for each part of the document and how the work has been divided between group members

Chapter 2

Architectural design

2.1 Overview

2.2 Component View

In this section, the individual components will be presented in terms of their functionalities, their role and the needed sub-parts, as well as how the sub-elements interface one with another within the overlaying component. Moreover, in this section it will be specified which of the considered sub-elements are in charge of interfacing with the other components of the system.

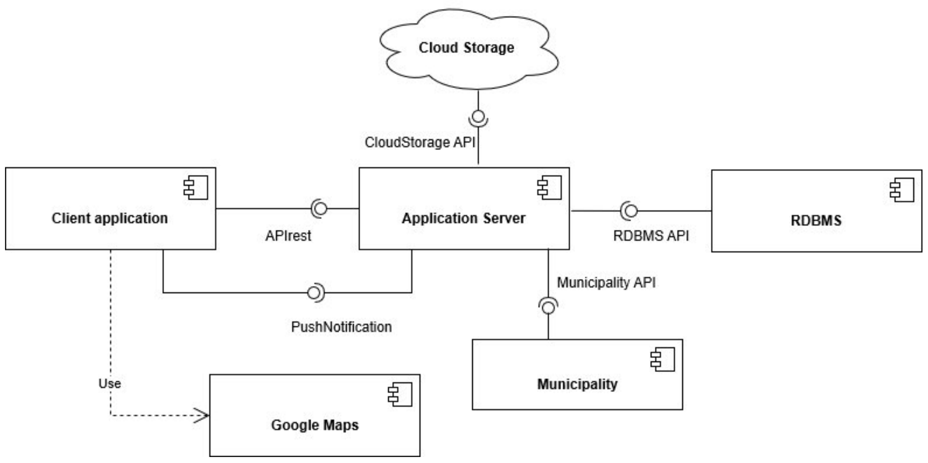


Figure 2.1: *Component View of the application*

2.2.1 Database

The database layer must include a DBMS component, in order to manage operations like insertion, update or deletion as well as logging of transactions on data inside the storage memory. The DBMS must guarantee the correct functioning of concurrent transactions and the ACID properties. As the application doesn't require a more complex structure than the one provided by the relational data

structure, the database has to be a RDBMS. The data layer will be exclusively accessible through the Application Server via a dedicated interface. Besides, the Application Server must provide a persistent unit to handle dynamic behavior of all the contained application data.
The system also provides an external Cloud Storage in order to memorize the pictures sent by the users that notify a violation. There is a dedicated service for this task as the amount of pictures that can be collected can be very huge. After saving pictures in the Cloud Storage, there is a unique identifier for every memorized data.
Sensible data, fundamentally passwords, have to be hashed properly and salted before being stored. The users will be granted access only after verification of correct and valid credentials.
Pertinence and integrity are granted, as well as isolation in concurrent transactions and atomicity, all ACID properties are satisfied.

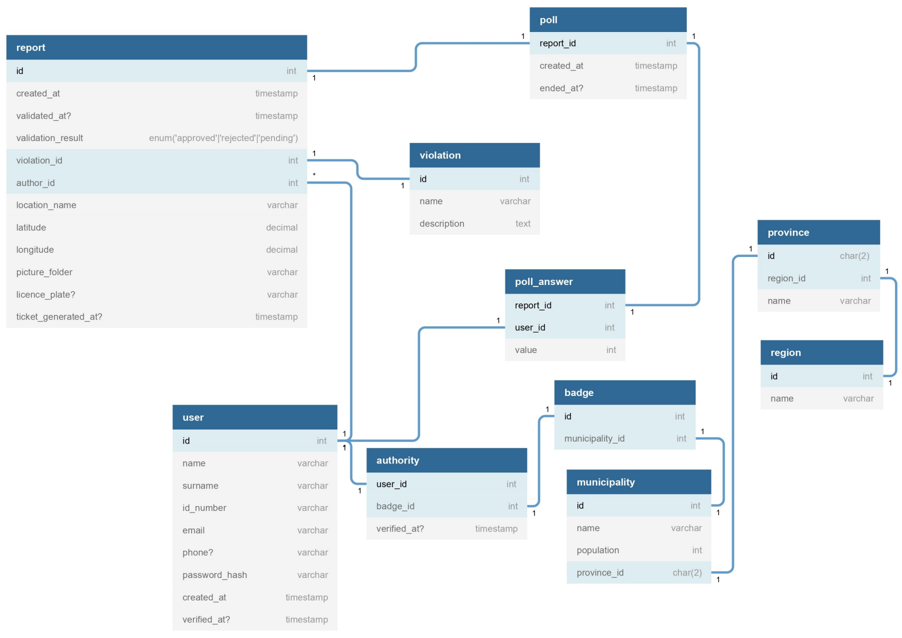


Figure 2.2: *Relational Model of RDBMS*

2.2.1.1 Notes about Relational Model

The diagram doesn't include fields that are obtainable with specific queries. An example is the score of a pull, that can be obtained with a view, or a materialized view (through a selection on the sum). Another attribute, $validation_result$, in Report entity, is inserted in the table, even if it could be computed with an equality, because it is not granted that threshold value will be constant in time. In fact, this value will depend on the number of active users that is monthly updated, or on other elements that make it variable.

- *"AttributeName?"* indicates an attribute that can assume NULL value. Most of the attributes are mandatory fields, only a few can be nullable

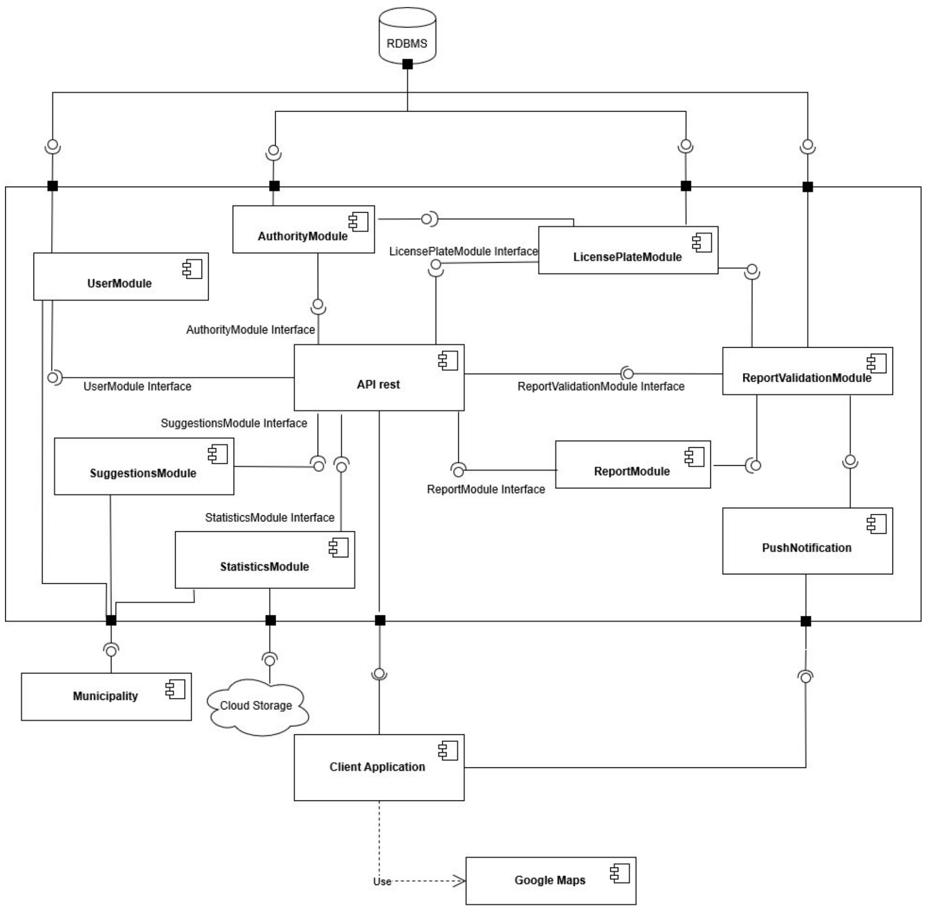


Figure 2.3: *Component View of the application - Focus on Application Server*

2.2.2 Application Server

This layer must handle a great part of the application logic, together with the connections with the data layer and the multiple ways of accessing the application from different clients. The main feature of the Application Server is the set of specific modules of logic, that describe rules regarding violations, their identification, and the following procedures. Work-flows for each of the functionalities provided by the application itself are also provided in this component.
The interface with the data layer must be handled by a dedicated persistent unit, that will be dedicated also to the dynamic data access and management, besides the object-relation mapping. In this way the only Application Server is granted to have access to the database.
The Application Server must also provide a way to communicate with external systems, by adapting the application to already existing external infrastructures.

The main logic must include:

- **UserModule**: This module will manage, through AccountModule component, all the logic involved with the user account, management, and operations like registration, login, and also profile customization and update, as allowed to authority, that can register with special permission after they are authorized. Furthermore, the module will deal with the generation and provision of user credentials, connected to the optional two-factors authentication, that can be requested by the client. Besides, this module has to manage user accounts; it has to store all the data related to user accounts in the RDBMS, all the data related to users and to show stored data in case of personal data requests.

- **StatisticsModule**: This module contains the logic used to locate violations and users; besides it deals with the definition of the Unsafe Area boundaries.This module must provide useful data to the logic that is at service of the unit that deals with Authorities, since it can need localization information, in order to perform its task.

- **AuthorityModule**: This module contains all the logic that grant access to an authority to its specific functionality: generation of traffic tickets and access to personal information and sensible data regarding offenders. Besides, the module is also devoted to the fundamental task of keeping the chain of custody of a certain notification with license plate, in order to preserve the integrity of information.

- **ReportModule**: This module provides the function of collecting all the reports, it doesn't depend on the validity of notification itself. The module is used principally as a general collector, that provides the logic to collect reports that are provided by the user in a correct form. The reports are then sent to the ReportValidationModule in order to validate or reject them.

- **ReportValidationModule**: This module provides the logic behind the report of violations, with particular focus on timing restrictions and on the confirmation of the report by other users. This module is also in charge to detect multiple reports of the same violation. This module is useful to filter the reports, validating or rejecting them.

- **LicensePlateModule**: This module includes the logic needed by other components to set the license plate status. It must also be useful as an interface with the external communication with the municipality, as it forwards a certified license plate, after it has been validated under the supervision of an authority.

- **SuggestionsModule**: This module implements the algorithm to automatically generate suggestions in respond to the violations notified and validated by SafeStreets. It uses the validated report, and provides suggestion to be sent to Municipality, hat is an external agent to the system.

- **PushNotification**: This module is used as a gateway from all the modules that need to interact by sending an email to the clients. Its task is managing the logic behind the email notification services.

- **APIrest**: This module is used as an interface for the client to communicate with server and to send general requests. This module holds the general request of the client providing him the correct service he asked for.

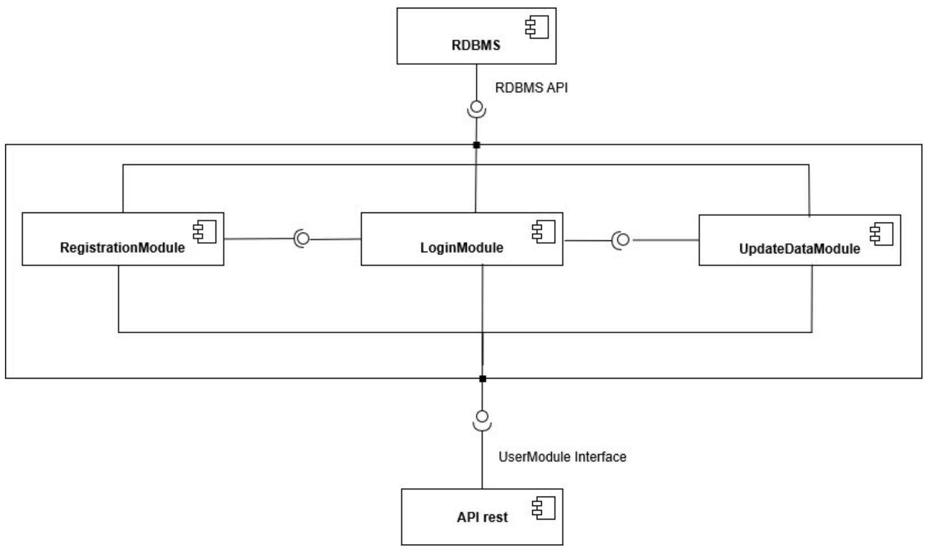


Figure 2.4: *Component View of the application - Focus on UserModule of Application Server*

2.2.3 Client Application

The client must be designed in such a way to make the communication with the Application Server easy and not dependent on the implementation on both sides of communication. In order to reach this goal, adequate APIs must be defined and used to manage the interaction between the two components. The mobile application UI must be designed following the procedures provided by Android and iOS systems. The provided interface is suitable either for Mobile Client Application or Web Client Application, as the client can connect to SafeStreets in both ways.
The client application must provide a software module that manages GPS connection of the device and helps with the identification of location. This software will provide collected data to the Application Server to be processed. This is useful, but not fundamental for the application as the user is able to use SafeStreets also in absence of GPS functioning; in this case the location is manually inserted by the user.

2.3 Deployment View

The SafeStreets platform can be divided into the following Tiers that separate different aspects of the application.

- Client: it represents the client application. It will be the Web Application running in the browser or the native mobile application running on the target OS

- Reverse Proxy: it represents the intermediary between the client and the server application. It will provide basic functionalities such as data compression and caching.
- Server Application: it represents both the static file serving service and the application that provides the REST APIs for accessing and creating dynamic information
- Database: it represents the data storage solution. This tier identifies both the RDBMS and the media storage

This is more of a logical separation due to the fact that we'll be using a Cloud Infrastructure Provider, thus the real architecture may vary based on the specific company offering the service.

In the following page a simplified graphical representation of the architecture previously described is provided.

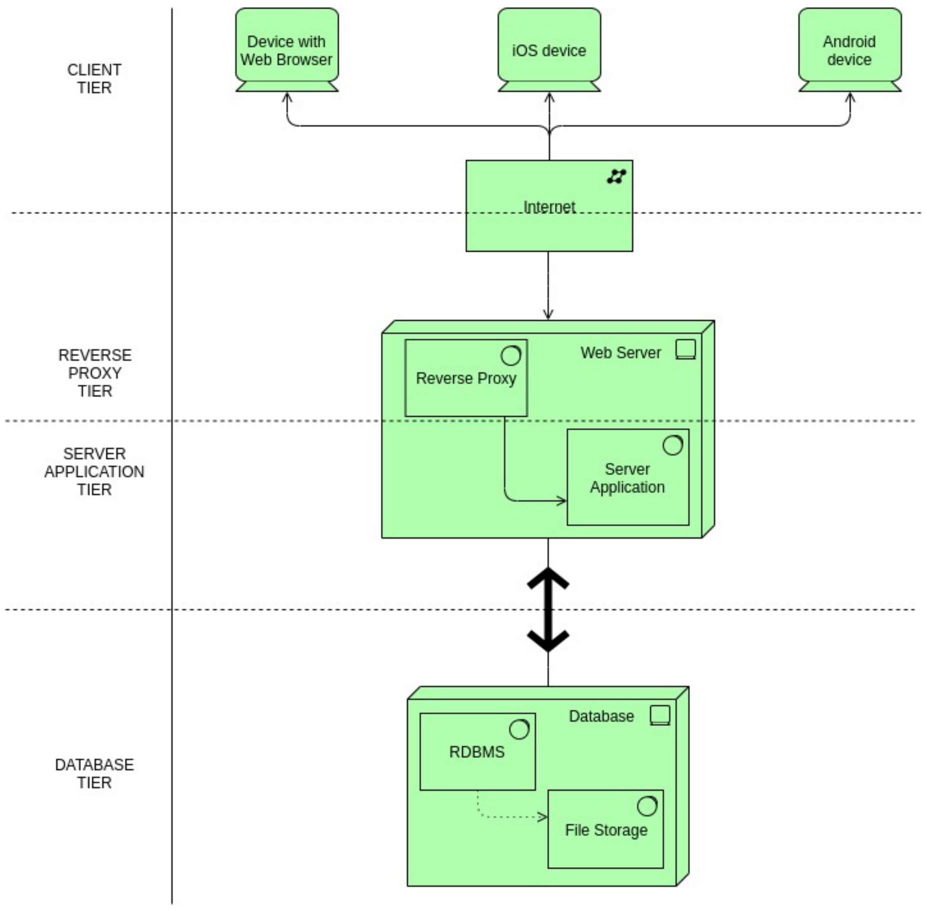


Figure 2.5: *Simplified Architecture diagram*

2.4 Runtime View

In this section the runtime view of the system will be analyzed. It shows in detail how the system works in every use case, presenting the interaction between the internal components, discussed in the section 2.2, talking about Component View.

2.4.1 Registration

The diagram shows the workflow of the application during the registration. The behavior of the system is the same for normal user and authority, because authority first registers as a normal user, then updates the profile. The main component is the **UserModule** that contains the logic to check correctness of data for registration. The module communicates with RDBMS to check if creation of user account is possible. In positive case the Client Application can choose the way to confirm account creation (whether mobile phone number or email), and only after confirmation of data, the module inserts new user data into the RDBMS. In case of failure, an error message is sent back to the Client Application to communicate it.

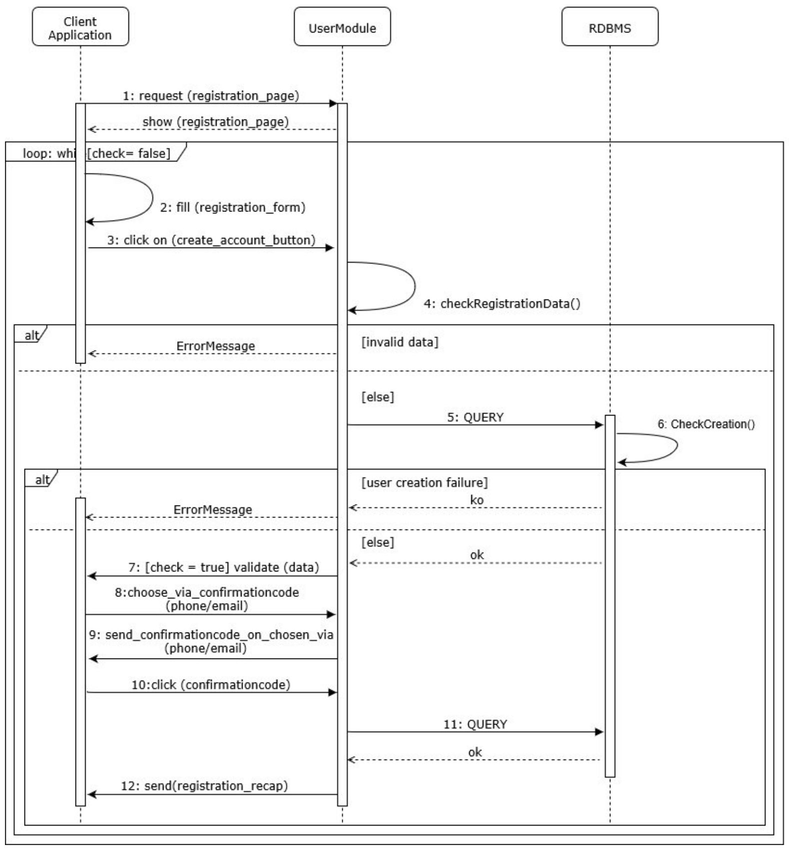
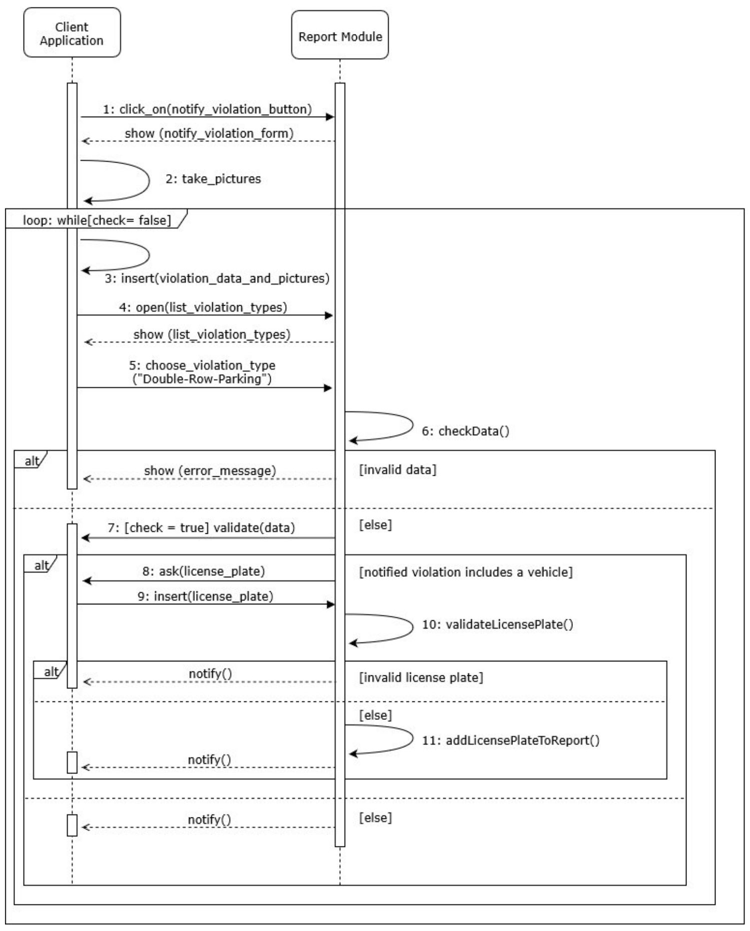

Client Application
UserModule
RDBMS
1: request (registration_page)
show (registration_page)
loop: while [check= false]
2: fill (registration_form)
3: click on (create_account_button)
4: checkRegistrationData()
alt
ErrorMessage
[invalid data]
[else]
5: QUERY
6: CheckCreation()
alt
[user creation failure]
ko
ErrorMessage
[else]
ok
7: [check = true] validate (data)
8:choose_via_confirmationcode (phone/email)
9: send_confirmationcode_on_chosen_via (phone/email)
10:click (confirmationcode)
11: QUERY
ok
12: send(registration_recap)

Figure 2.6: *Sequence diagram 1 - Registration*

2.4.2 Submit of report

The diagram shows the workflow of the application during the sending of a report from Client Application. The main component is the **ReportModule** that contains the logic to check correctness of data regarding report, and has the task to collect all data regarding the provided report. The module communicates with ReportValidationModule for further confirmation of data. In this workflow the syntactic correctness of the received report is checked.



Client Application
Report Module
1: click_on(notify_violation_button)
show (notify_violation_form)
2: take_pictures
loop: while[check= false]
3: insert(violation_data_and_pictures)
4: open(list_violation_types)
show (list_violation_types)
5: choose_violation_type ("Double-Row-Parking")
6: checkData()
alt
show (error_message)
[invalid data]
7: [check = true] validate(data)
[else]
alt
8: ask(license_plate)
[notified violation includes a vehicle]
9: insert(license_plate)
10: validateLicensePlate()
alt
notify()
[invalid license plate]
[else]
11: addLicensePlateToReport()
notify()
notify()
[else]

Figure 2.7: *Sequence diagram 2 - Submit of report*

2.4.3 Validation of report

The diagram shows the workflow of the application during the validation of a report. The system shows the interaction between two core modules, the **ReportModule** and the **ReportValidationModule**. The report, once accepted by the ReportModule, is submitted to the ReportValidationModule, that contains all the logic to validate the report and communicate with the Client Application through the Notification Push, to send all Client Applications the poll to answer to validate the report. Once the report has been validated the ReportValidationModule registers the violation, inserting new data into the RDBMS. In case of absence of confirmation by the Client Applications to the poll, if the report is proved to be inconsistent and wrong, as rejected by a sufficient number of other clients, the ReportValidationModule updates RDBMS data assigning a penalty to the client who submitted the report.

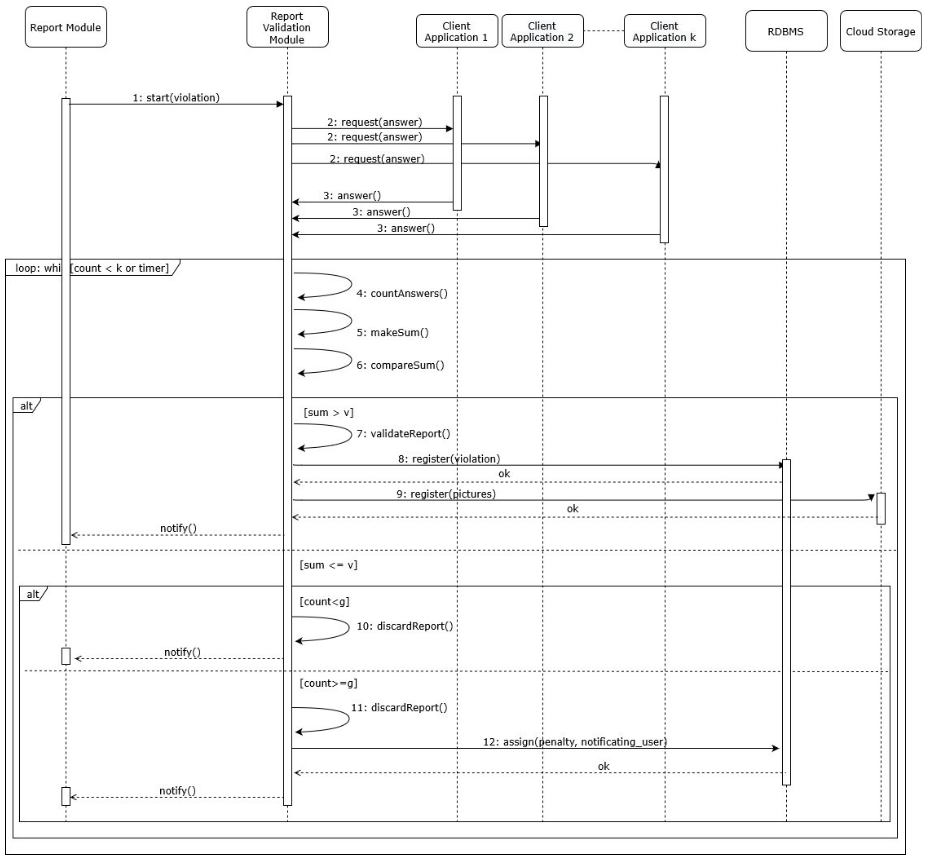
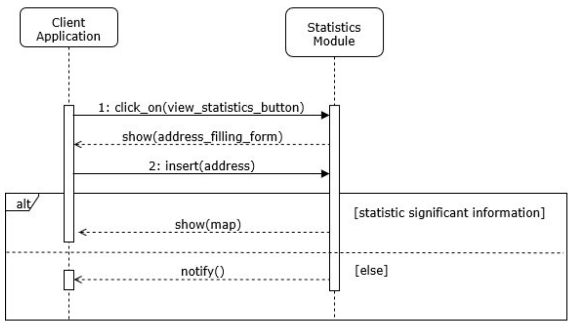


Figure 2.8: *Sequence diagram 3 - Validation of report*

2.4.4 Statistic Display

The diagram shows the workflow of the application during the visualization of statistics. The main component is the **StatisticsModule**, which contains the logic to access statistics data. The module contains the logic to show the user statistics when required by the Client Application by inserting location of the desired area.



Client Application
Statistics Module
1: click_on(view_statistics_button)
show(address_filling_form)
2: insert(address)
alt
[statistic significant information]
show(map)
notify()
[else]

Figure 2.9: *Sequence diagram 4 - Statistics Display*

2.4.5 Display to Authority Module of details of violations

The diagram shows the workflow of the application during the display of data to AuthorityModule. The main components are the **LicensePlateModule** and the **AuthorityModule**. Once a license plate has been validated, the License-PlateModule updates the number of violations connected to that license plate in the RDBMS. After this update, the AuthorityModule is called, to notify the validated license plate, and, as a consequence, it queries the RDBMS asking for history of the validated license plate. After the query, the AuthorityModule generates the traffic ticket.

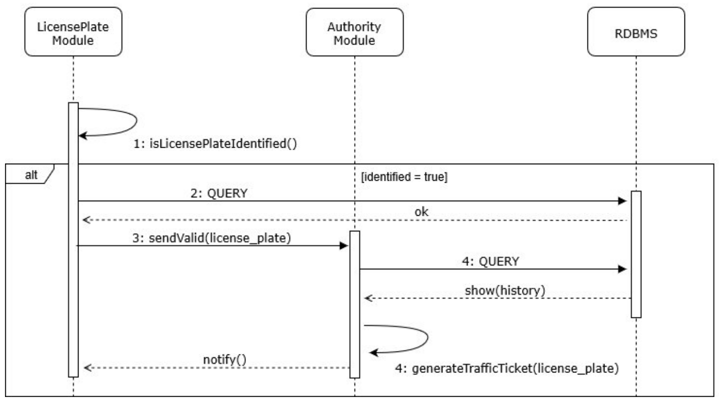


Figure 2.10: *Sequence diagram 5 - Display to Authority Module of details of violations*

2.4.6 Update Profile to Authority

The diagram shows the workflow of the application during the update of profile to authority. The main component is the **UserModule** that collects the data sent by the Client Application who asked for profile update, and interact with the Municipality to check data authenticity. When data are confirmed by the Municipality, the UserModule updates data on RDBMS regarding the Client Application who asked for profile update.

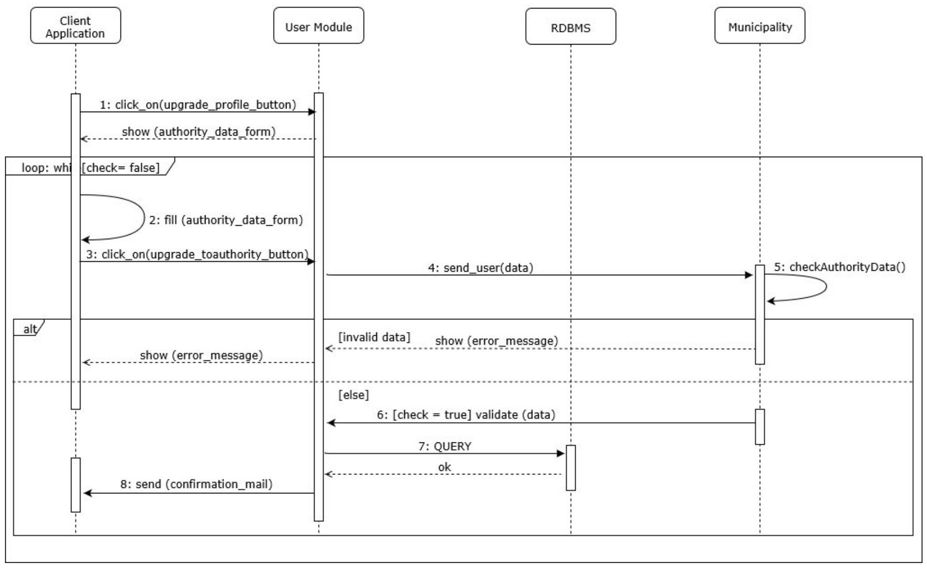

Client Application
User Module
RDBMS
Municipality
1: click_on(upgrade_profile_button)
show (authority_data_form)
loop: while [check= false]
2: fill (authority_data_form)
3: click_on(upgrade_toauthority_button)
4: send_user(data)
5: checkAuthorityData()
alt
[invalid data]
show (error_message)
show (error_message)
[else]
6: [check = true] validate (data)
7: QUERY
ok
8: send (confirmation_mail)

Figure 2.11: *Sequence diagram 6 - Update Profile to Authority*

2.4.7 Update Statistics

The diagram shows the workflow of the application during the update of statistics. The main component is the **StatisticsModule**, that contains the logic to update statistics, collecting information by the Cloud Storage, at the first step, where information about pictures of violations are memorized. Then the module interacts also with the Municipality to increase the information collected by SafeStreets. In particular, the Municipality can provide more detailed information about car accidents, and other type of violations like this.

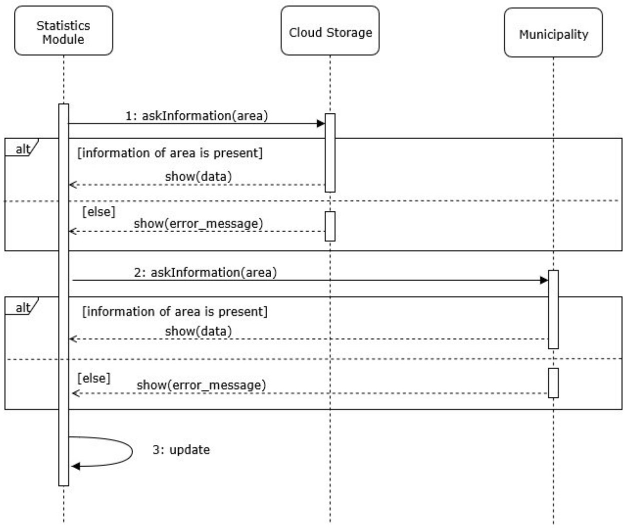


Figure 2.12: *Sequence diagram 7 - Update Statistics*

2.4.8 Traffic Ticket Generation

The diagram shows the workflow of the application during the generation of the traffic ticket. The main components are the **LicensePlateModule** and the **AuthorityModule**. Once a license plate has been validated, the License-PlateModule queries the RDBMS to create a specific record for the new data. Subsequently, the AuthorityModule queries the RDBMS to have the license plate, and checks if some information is missing. The module contains the logic to check integrity and completeness of data and to insert information, in case data are not complete. After the query, the AuthorityModule generates the traffic ticket.

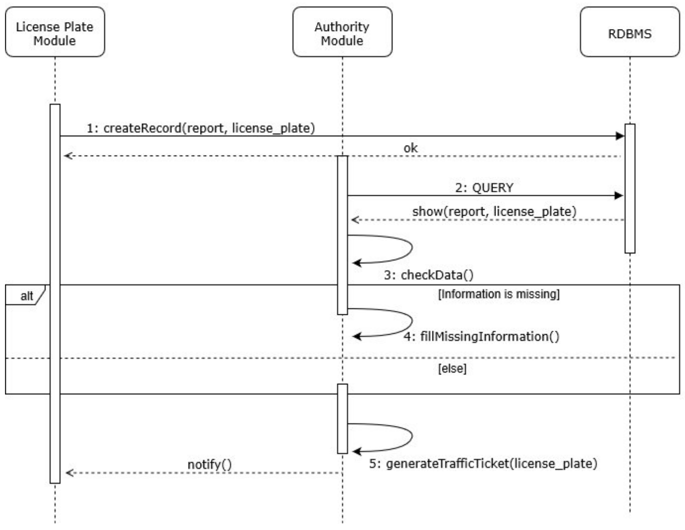

License Plate Module
Authority Module
RDBMS
1: createRecord(report, license_plate)
ok
2: QUERY
show(report, license_plate)
3: checkData()
alt
[Information is missing]
4: fillMissingInformation()
[else]
notify()
5: generateTrafficTicket(license_plate)

Figure 2.13: *Sequence diagram 8 - Traffic Ticket Generation*

2.5 Component Interfaces

2.5.1 Backend APIs

In this section the main APIs that the back-end application will expose are shown and commented. They are grouped according to the entities they are mostly related to.

2.5.1.1 User APIs

These APIs will handle the data related to the User entity, therefore their route will be prefixed by "/user". They'll retrieve all necessary data to display the profile of a user, including the points obtained in SafeStreets reward system. Some of these APIs will be accessible only to the user they are related to, others will also be available to authorities.

2.5.1.2 Login APIs

These APIs will manage the access to the system for both standard users and authorities. Their route will be prefixed by "/login". Login APIs are accessible to anyone who is not already logged in, as it is their role to check the validity of user credentials and, in case they are correct, generate tokens that allow the logged user to access other areas of the application.

2.5.1.3 Registration APIs

These APIs will allow visitors to register to SafeStreets and authorities to upgrade their profile. Their route will be prefixed by "/register". Registration APIs are accessible to all visitors that are not logged in the application.

2.5.1.4 Report APIs

These APIs will manage the creation and validation of reports. All APIs belonging to this group will be prefixed by "/report". Report APIs are only available to users that have a verified profile; some of these APIs will only become available to specific profiles (e.g. the APIs that allow to vote for the approval of a report).

2.5.1.5 Ticket APIs

These APIs will handle the generation of tickets and therefore all their routes will be prefixed by "/ticket". These APIs are only available to verified authorities.

2.5.1.6 Statistics APIs

These APIs will retrieve all necessary data to display the maps of safe and unsafe areas. Their routes will be prefixed by "/stats". The majority of these APIs are accessible to anyone, but a few are available only to authorities as they retrieve specific data about offenders.

In the following page an overview of the main APIs and respective server-side controllers is provided.

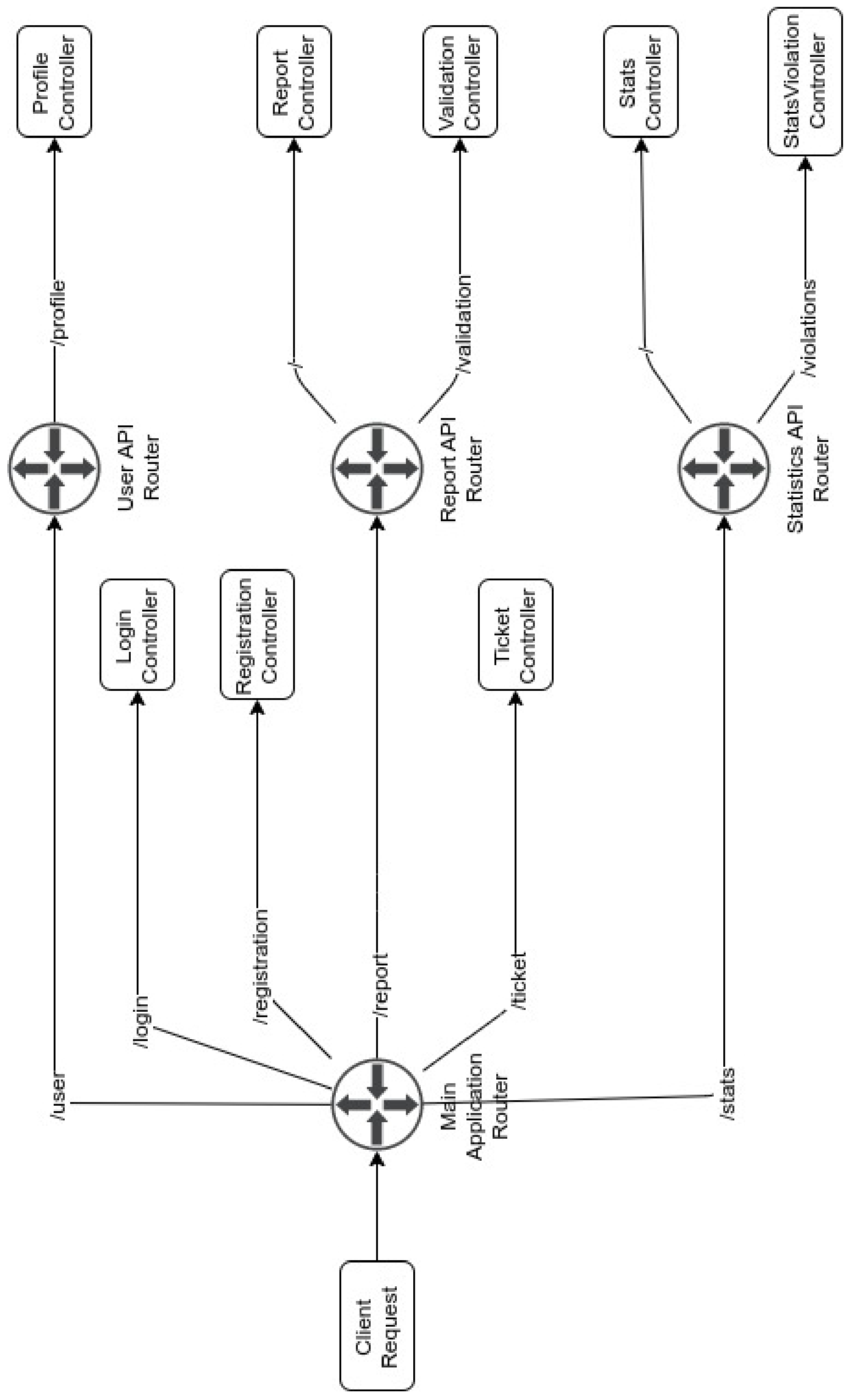


Figure 2.14: *Interface Diagram*

2.6 Selected Architectural Styles and Patterns

SafeStreets will be built using a RESTful architecture. The web application and the mobile application will interact with the server using HTTP endpoints communicating using the JSON format. The main advantages of these choices are:

- separation of the presentation layer from the application logic at a physical level: delegating all the presentation logic to the client device will reduce the load of the server (no Server Side Rendering)
- the JSON format will ensure the compatibility with all the platforms: being a standard communication format guarantees the existence of a variety of libraries for all client platforms

The server application will be structured using the concept of middlewares. Middlewares are chainable components in the HTTP Request-Response flow and guarantee a good level of isolation among the different functionalities they offer, thus providing a simple pattern that encourages the employment of the Single Responsibility Principle.

The middlewares we'll be using are:

- CompressionMiddleware: it reduces the overall bandwidth needed by the server using a compression algorithm that decreases the size of the data sent (and received) by (and to) the server. This middleware will be provided by the Reverse Proxy
- AuthenticationMiddleware: it has the purpose of filtering requests, rejecting those which lack the permissions to access a specific resource
- LoggingMiddleware: it logs the server traffic. This can be useful for debugging purposes and statistical analysis on further improvement of the infrastructure

The SafeStreets APIs will be developed using **express**, a lightweight library for Node.js that provides an easy programming interface for building middlewares and REST applications. The main language we'll be using is TypeScript due to its statically-typed nature. Moreover, being it a superset of JavaScript, it guarantees the compatibility with a large amount of web-oriented libraries and first-class Functions. Functional programming fits well with the stateless concept of a RESTful architecture, we'll be following a Functional style for our code, trying to avoid state whenever possible to reduce the possibility of error caused by mutability.

The database system we'll be using is PostgreSQL, a common RDBMS, on which we'll store all kind of data except for multimedia content. This content, being non-relational and prevalent on our platform will be stored on a Cloud Storage solution (such as Google Cloud Storage) and linked to the RDBMS using its own identifiers.

2.7 Other Design Decisions

2.7.1 Algorithm for generating possible interventions

This algorithm is used to automatically generate possible interventions in response to certified violations, in order to prevent them. It is an algorithm that uses the principles of Machine Learning and started from an input of a b GB of text, extracted from n (order of thousands) web pages. It has also in input some parameters; particularly, it is focused on keywords. It takes as the main keywords the ones taken from the category of certified violation, provided by the user, then as auxiliary keywords, to be used if the collected material in the first step is not sufficient, the ones taken from the brief description inserted by the user in the report. Thanks to this information a simple and middle-length text suggesting possible interventions is produced with a procedure similar to algorithm GPT-2.

Chapter 3

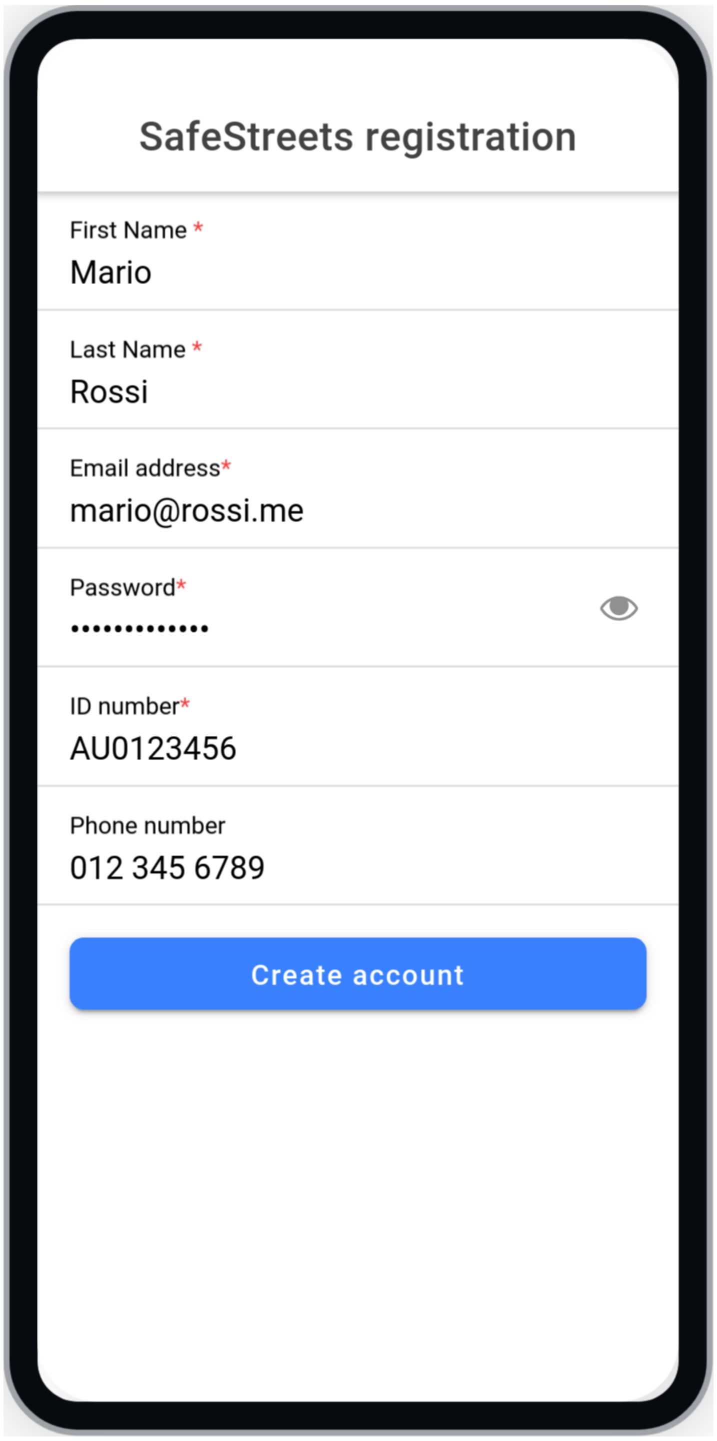
User interface design

3.1 User Interface Design

The design of the web application will use the Google Material UI elements to give out a more native look on Android Devices, which we expect to be the majority of our user base. The design of the native applications will, instead, comply with the target platform. This process will be handled by the framework (e.g. Ionic framework) we'll use to port the Web App to Android and iOS.

In the following pages some mock ups are provided with a description of how the user will interact with them.

3.1.1 User Registration



SafeStreets registration

First Name *
Mario

Last Name *
Rossi

Email address*
mario@rossi.me

Password*
•••••••••••••

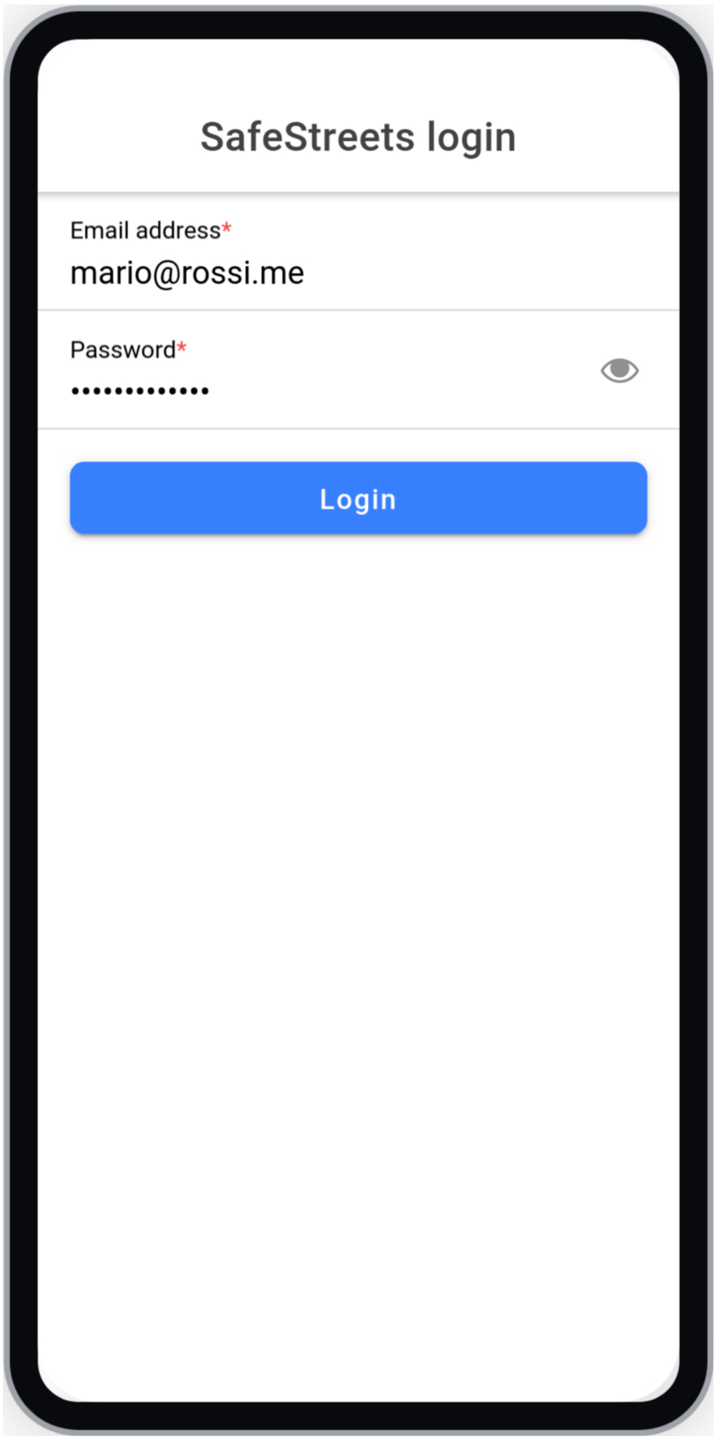
ID number*
AU0123456

Phone number
012 345 6789

Figure 3.1: *User Registration*

The registration process asks the user for personal information about who they are and require them to enter their identification number.

3.1.2 User Login



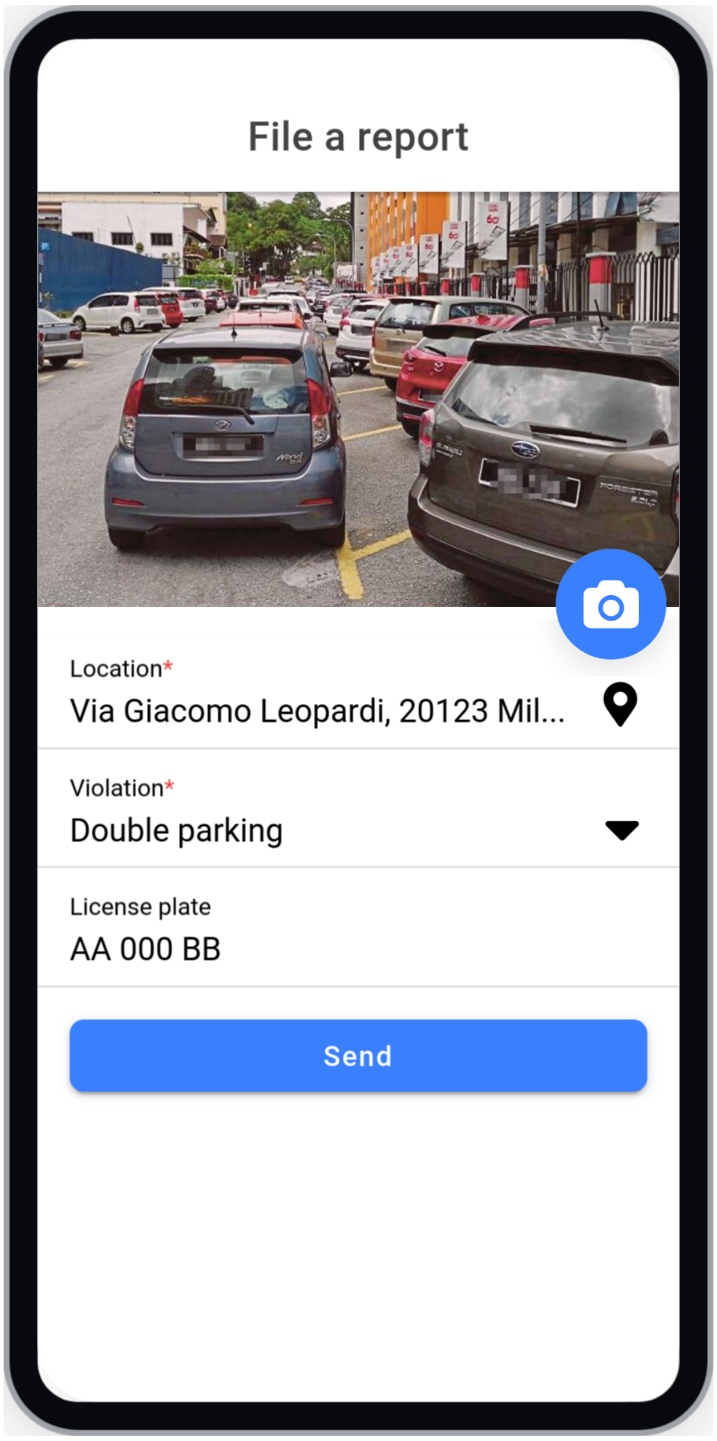
mario@rossi.me

•••••••••••••

Figure 3.2: *User Login*

The login phase asks for the email and the previously selected password like a common login process.

3.1.3 Report a Violation


File a report

Location*

Via Giacomo Leopardi, 20123 Mil...

Violation*

Double parking

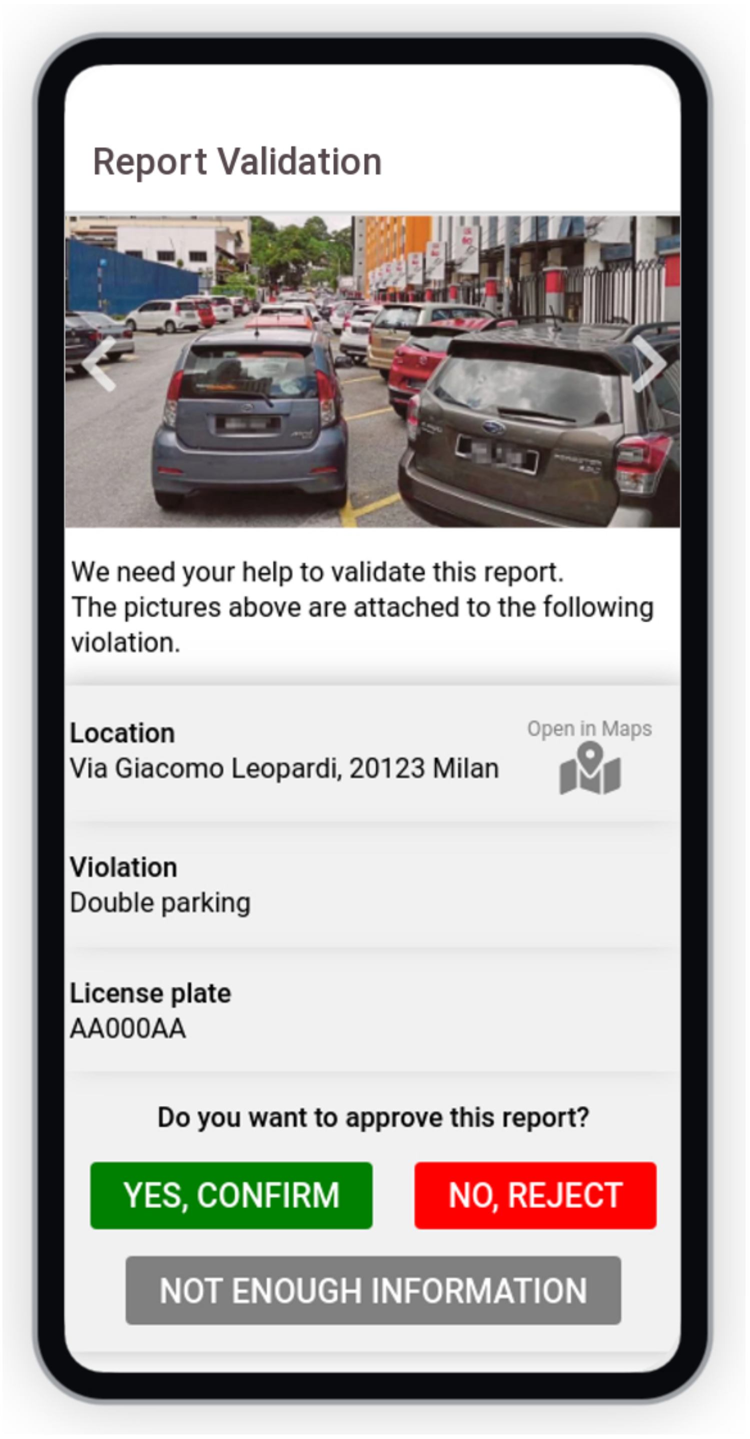
License plate

AA 000 BB

Figure 3.3: *Report a Violation*

In this mock up it can be seen that the user is required to insert images regarding the violation they want to report, the location where the violation happened and the category of violation identified. Optionally the user can add a license plate to ease the verification process.

3.1.4 Report Validation


Report Validation

We need your help to validate this report.
The pictures above are attached to the following violation.

Location
Via Giacomo Leopardi, 20123 Milan

Violation
Double parking

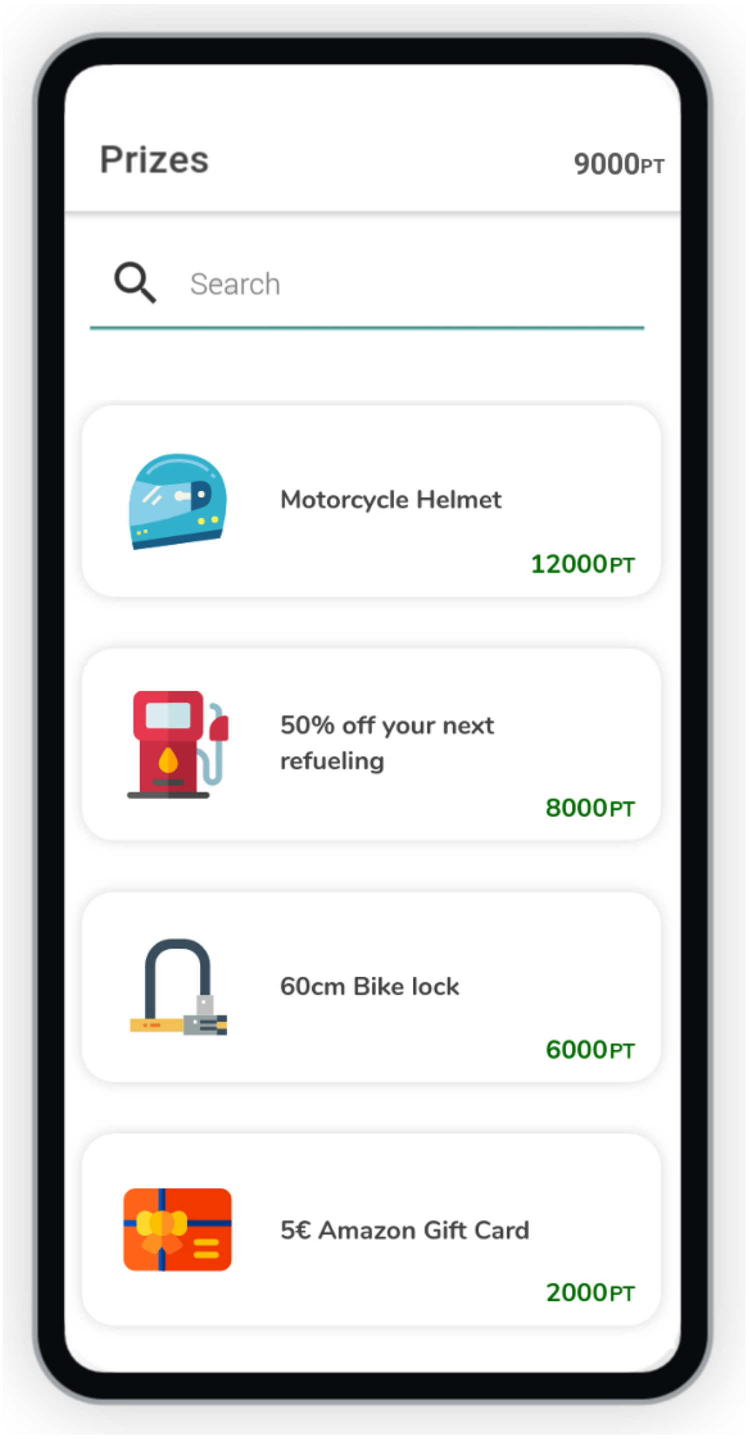
License plate
AA000AA

Do you want to approve this report?

Figure 3.4: *Report Validation*

Reports are validated by the community, so every time a new violation is submitted to SafeStreets some users will be notified and will see this screen. The report is anonymous, so the users that validate it don't see who sent it, but only the relevant details about the notified infraction. The user is therefore asked to click one of the three buttons to either validate, reject or ignore the report.

3.1.5 Prize Catalog


Prizes 9000PT

Search

Motorcycle Helmet 12000PT

50% off your next refueling 8000PT

60cm Bike lock 6000PT

5€ Amazon Gift Card 2000PT

Figure 3.5: *Prize Catalog*

Prizes are an incentive for users to use the application. Users can be assigned points when they file or evaluate a report. In the mock up above some example of rewards are provided. Prizes, being an addendum to the platform, will be provided to our users by external partners.

Chapter 4

Requirements traceability

Chapter 5

Implementation, integration and test plan

5.1 Overview

The implementation of the different features will follow a priority list. This way we can schedule the development process and make adjustments on less important aspects of the platform to better fit the key functionalities it will offer.

We identified the following features and assigned to them a priority based on the importance it will have for our customers and an expected time of implementation and graphic design:

Feature	**Priority (1 low - 5 high)**	**Man-days**
User notifications	5	8
Violation report form	5	5
Report validation form	5	4
App Store & Play Store page	5	4
Registration form	5	4
Prize catalog	4	10
Authority registration	4	10
Landing page	4	6
Intervention suggestions	3	40
Ticket generation	3	20
Login form	3	1
Unsafe area map	2	25
User profile page	1	5

5.2 Implementation

The implementation process will parallelize the development of the server application and the client web application. The client web application, using simple mock up components for retrieving example data, can be developed independently of the server application. Once the web application is completed, we can start porting it to native mobile platforms, i.e. Android and iOS, using the chosen framework tools. The server application will be implemented using a simple structure. This is made possible by the thick client architecture that delegates all the presentation and basic application logic to the client, leaving the server managing only the business logic.

The database structure must be completed before developing the server application to reduce the integration overhead.

The following reference Gantt diagram shows the precedences and the parallelization process we want to achieve with a team made of:

- 2 Back end developers
- 2 Front end developers
- 1 Native developer

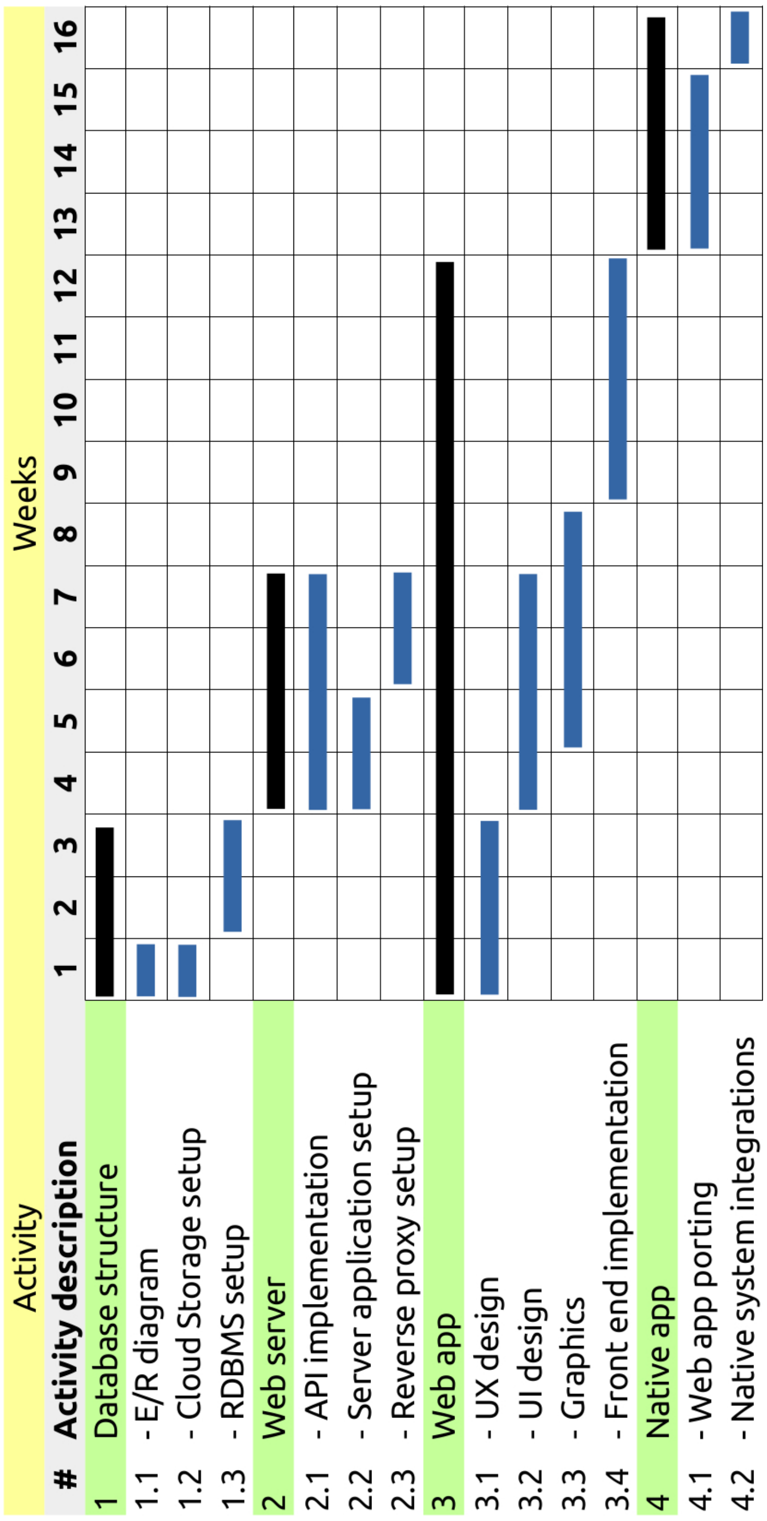


Figure 5.1: *Reference Gantt*

5.3 Integration

5.3.1 Parallelization

The integration of the various components of our platform will be realized at the end of the development process. This way we can parallelize the implementation, having different teams independent from each other till the completion of their respective components. All components will be unit tested to prevent common errors from being overlooked.

Whenever two components are integrated, specific integration tests will be written to verify the end result.

Once the server and client application have been completed they will be integrated using REST APIs. The process won't need any refactoring of the client application, as it will be developed using mock up classes and stubs for the server calls.
The client-server integration will need specific tests and human tests. We can take into consideration the possibility of an early release of the application on the mobile application stores as a beta distribution to collect a large number of feedbacks, regarding not only bugs and glitches of the client application but also suggestions about possible improvements of the UX.

5.3.2 External services

Once our server application is completed and all its components have been successfully integrated, we can start integrating external services to it.

We'll focus on the Cloud Storage solution first, because it's necessary to launch the client application. The Cloud Storage solution will provide an HTTP API to access, modify and create multimedia content, making our server a proxy to its data storage.

To integrate partners APIs we'll need to make specific agreements with each one of them. These agreements will discuss the technicalities of their systems which may vary significantly from one another.

Any possible municipality API for the retrieving of statistical data about accidents will be integrated individually, due to the several differences that can be found in each information system.

On the client-side a service that will be integrated in the first release is the map. Server-side we only need to store the coordinates of the reports, latitude and longitude, while on the client-side these points will be presented visually through a map service that accepts these parameters. This integration can be made independently of the server implementation process.

5.4 BPMN

The implementation, integration and testing process is summarized in the following BMPN diagram.

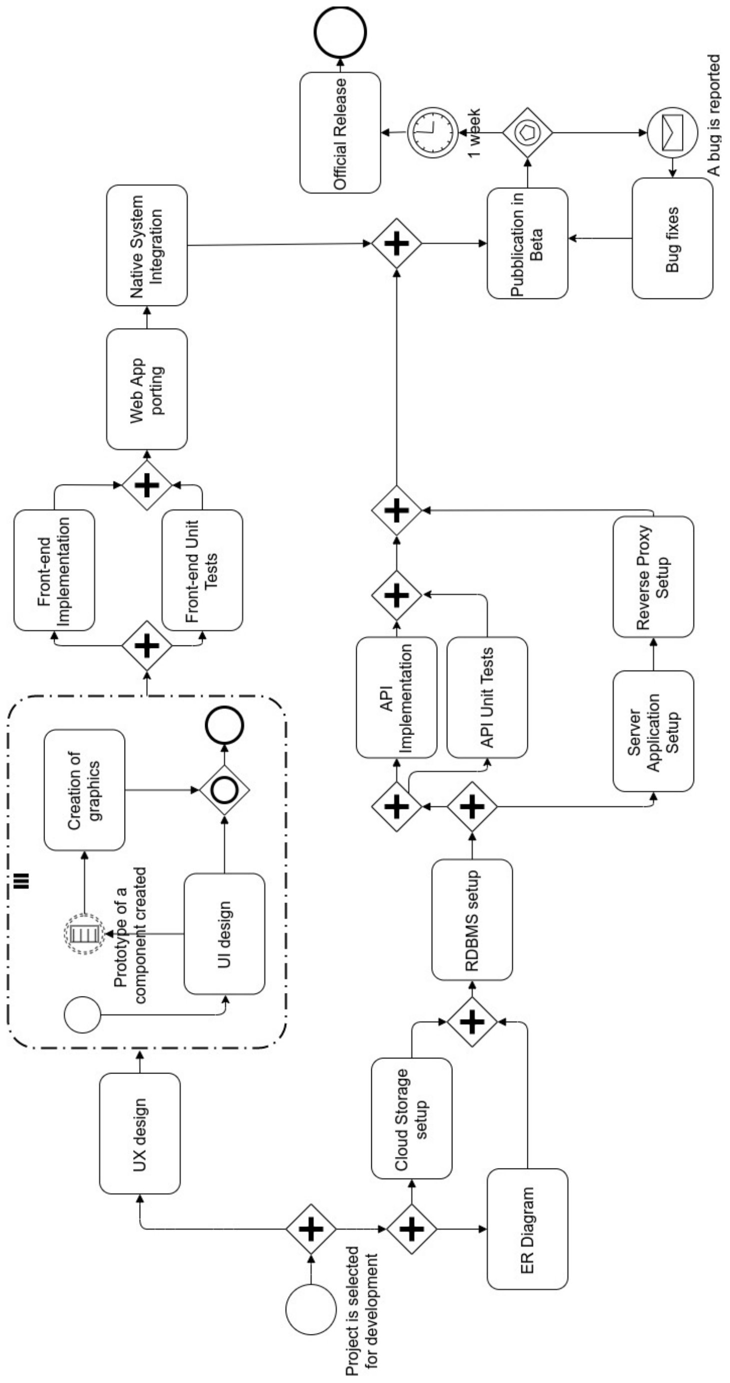


Figure 5.2: *BMPN Diagram*

Chapter 6

Effort spent

Carlo Dell'Acqua

Task	**Time spent (hours)**
Project setup	0.5
Introduction	1
Architectural Styles	1
Deployment View	1.5
User Interface Design	2
Implementation, Integration & Testing	6
Architectural Design	1

Adriana Ferrari

Task	**Time spent (hours)**
Architectural Styles	1
Deployment View	1.5
User Interface Design	4.5
Component Interfaces	1.5
Implementation, Integration and Testing	2

Angelica Sofia Valeriani

Task	**Time spent (hours)**
Component View	2.5
Other Design Decisions	1
Component View Diagrams	2.5
Runtime Diagrams	2
Runtime Section	1.5
Other Diagrams	2
Relational model and descriptions	1.5

References