POLITECNICO
MILANO 1863

Ivan Cavadini (941927).
Nicolò Molinari (942404).
Luigi Pederzani (943023).

SafeStreets: Design Document

Deliverable:	DD
Title:	Design Document
Authors:	Ivan Cavadini, Nicolò Molinari, Luigi Pederzani
Version:	7.1
Date:	09-December-2019
Download page:	github.com/Pederzh/CavadiniMolinariPederzani
Copyright:	Copyright l' 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani All rights reserved

Contents

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

List of Figures

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

1 Introduction

1.1 Purpose

The purpose of the DD is to provide a strategic guide to develop the project in the correct way, avoiding the errors and the waste of time that could be generated without studing the problem and the possible solutions. The document will be given to the development team.

1.1.1 Scope

The web platform will provide a system that manage the reports of the parking violations and elaborates the data to retrieve information about streets and cars:

- Report violation
- Report validation
- Update statistics
- Manage streets data
- Different usage between end user and authority

1.2 Definitions, Acronyms, Abbreviations

1.2.1 Definitions

- End-user: The end-user is the common user of the application, that takes pictures of violations and send it with all the report data to the system.
- Authority: Authorities are public entities, e.g. municipality, they use SafeStreets in order to see statistics, consult possible interventions and get information about traffic violations.

  Furthermore they can retrieve data in order to generate traffic tickets with their own systems.
- Web Server: A Web server uses HTTP (Hypertext Transfer Protocol) to provide the files that compose the Web pages to users. The requests are forwarded by their clients HTTP.

  Dedicated computers and appliances may be referred to as Web servers as well.

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

- Application Server: An application server is a type of server designed to install, operate and host applications and associated services for end users, IT services and organizations.

1.2.2 Acronyms

- API: Application Programming Interface
- DD: Design Document
- JDBC: Java DataBase Connectivity
- RASD: Requirements Analysis and Specifications Document
- UX: User Experience
- ANPR: Automatic number-plate recognition
- CUU: Code Unique Unitary
- JSON: JavaScript Object Notation
- REST: Representational State Transfer
- RDBMS: Relational DataBase Management System

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

2 Design architecture

2.1 Overview

The architecture we have decided to adopt, is the three-tier architecture. It is divided into three levels: Presentation, Application and Data Access.

The presentation tier contains the user interface of the web platform, it displays to the end-user the useful data like statistics.

The application level contains the core of the platform, it is composed by: the application server with the application installed and the web server. It contains the functions that elaborate the data (e.g. Check report validity, license plate export from image). The application server also allows multiple clients to work simultaneously.

Data access layer contains the database, which store all the reports, streets and users data (end-users and authorities). Some communications between presentation and application layers are asynchronous. This structure grants speed because the team can improve every single layer without having impact on the other tiers. It improves the efficiency because it brings separation between independent tasks.

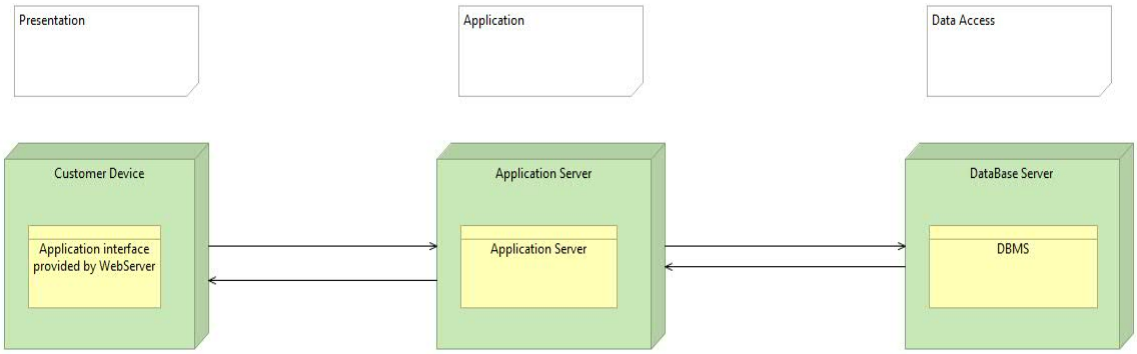

Figure 1: High level system architecture

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

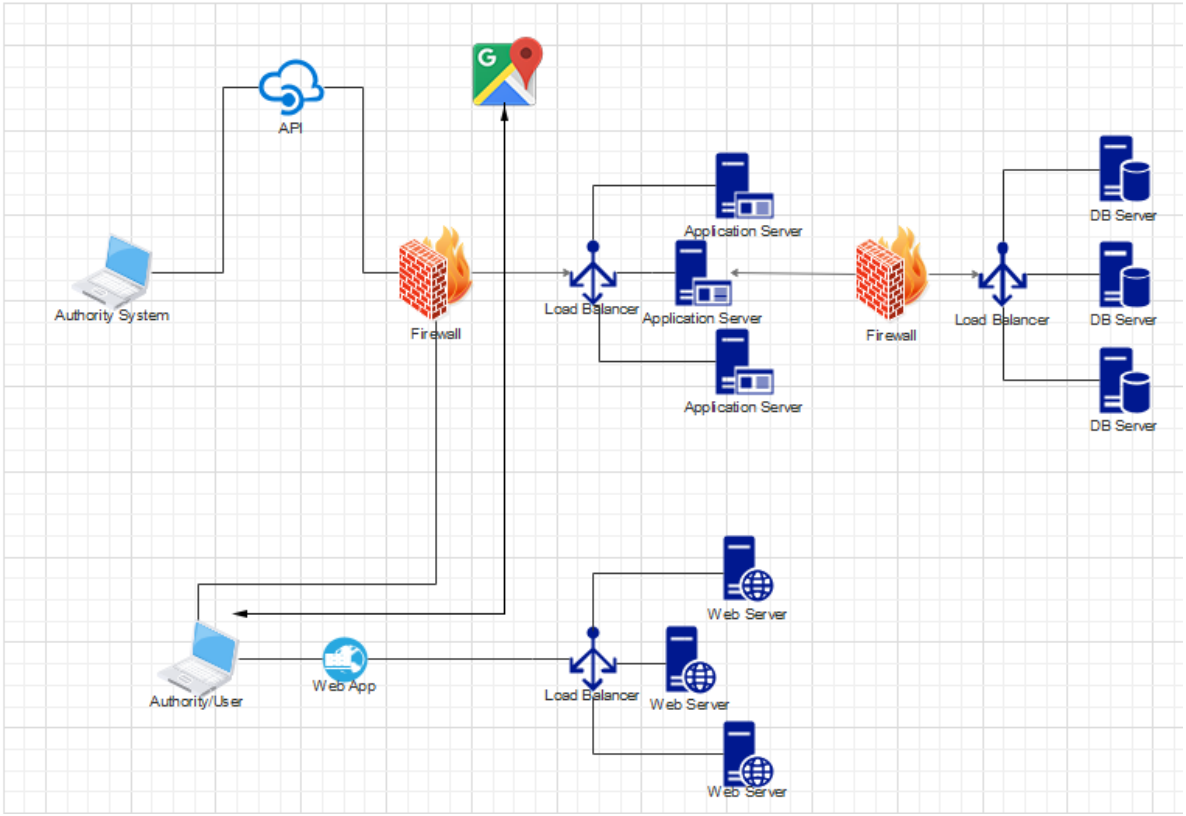

Figure 2: Network Architecture

The network architecture illustrates how the three-tier is applied to SafeStreets. The presentation level is all information that are shown on the web-app UI.

The authority own systems, that are not incorporated in the SafeStreets system, represent the software used to generate traffic tickets. They receive the data (type of violation, license plate, name, surname etc.) from the API that SafeStreets provides.

The application tier contains the web server and the application server. The web server provides the HTML pages and the JavaScript logic, to the web clients.

The web server does not communicate with the application server because the contents of the web pages will be filled dynamically with the JavaScript calls between the authority/user device and the application server.

With JavaScript asynchronous calls the client make and API call to some specific endpoints, by making a *promise*.

The promise is resolved when the client receives data, in our case in JSON format, or the server communicate the HTTP status of the request made. Contents of the pages can be updated when the JSON object is received.

The data access level contains the database server, that communicates with the application server. The latter will interrogate the database server to retrieve, update or remove data.

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

The application server is connected with two firewalls. They guarantee and improve security of the network and provide more controls on the packets that are sent through the network.

In order to grant better performances, every server is connected to a load balancer to distribute the request avoiding slowdowns.

The end devices exploit the browser cache to grant less requests.

2.1.1 Database structure

The database will have a structure according to the following one:

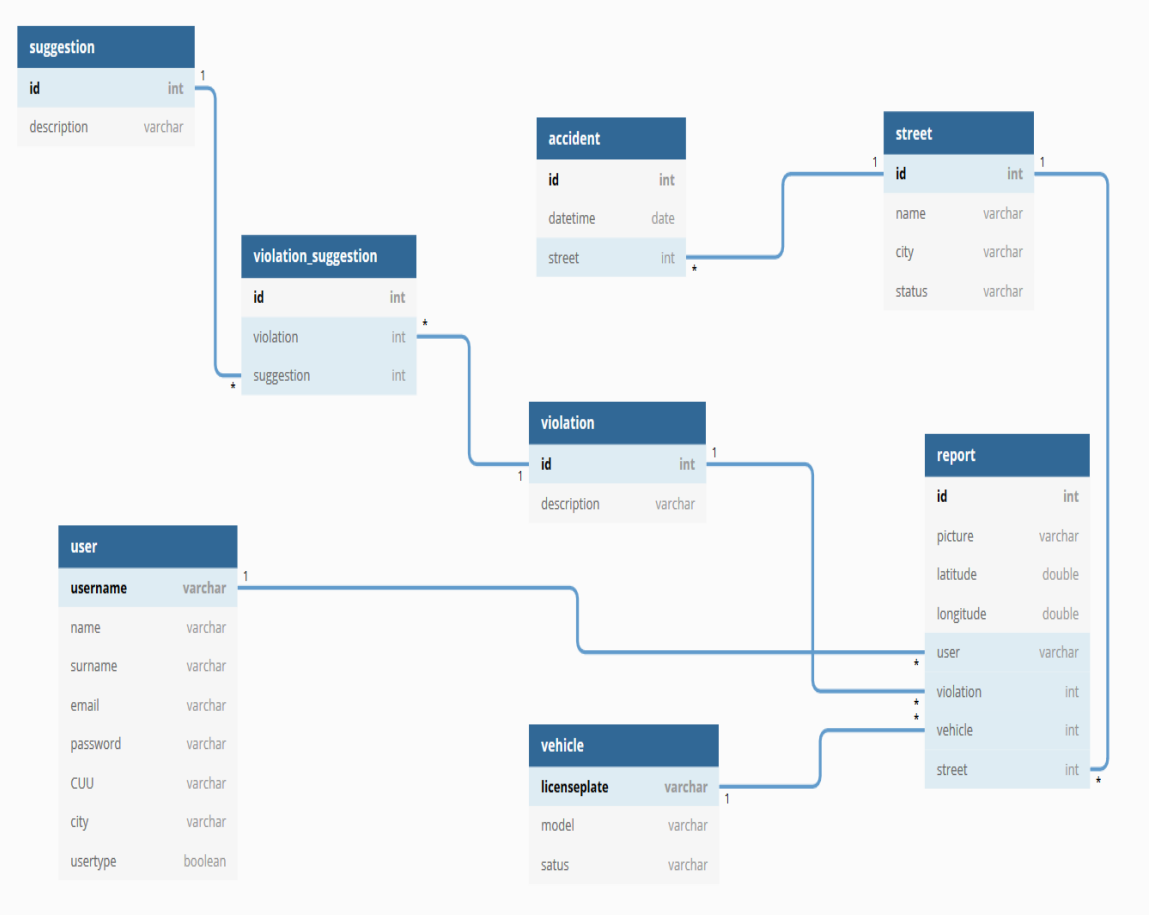

Figure 3: Database structure

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

2.2 Component view

In order to give a better description of the component of the system, the class diagram below shows how the application is designed.

All this description regards only the application tier of the system.

Communication between client and the application is made by an interface exposed (SystemMangerInterface).

The client, used by end users and authorities that want to see statistics, does not contain any portion of the logic of the system (except for the GPS tracking).

Thus it is a thin client and no class diagram has been made for it.

In diagram notes, it is possible to see which design patterns has been implemented (facade and singleton).

The data are stored in the database, are accessible using the manager classes. Their function is to query the database and save the data retrieved in the appropriate objects (for every entity of the database a specific class has been designed).

All the request arriving from clients are managed by the application server. Then it calls the appropriate method of facade class SystemMananger using the interface SystemMangerInterface. This class uses the methods of other classes to respond to the request.

A SystemManager method, used for obtaining information about accidents occurred, is called periodically. It retrieves the information and then stores them in the database.

During the registration process of an authority the specific method calls a web service to check the authenticity of the CUU code.

Moreover, an interface is exposed to authorities systems. In this way they can call a method that provide the reports data and generate traffic tickets on them.

Before sending any type of information the identity of the authority is checked using the appropriate method of UserManager class.

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

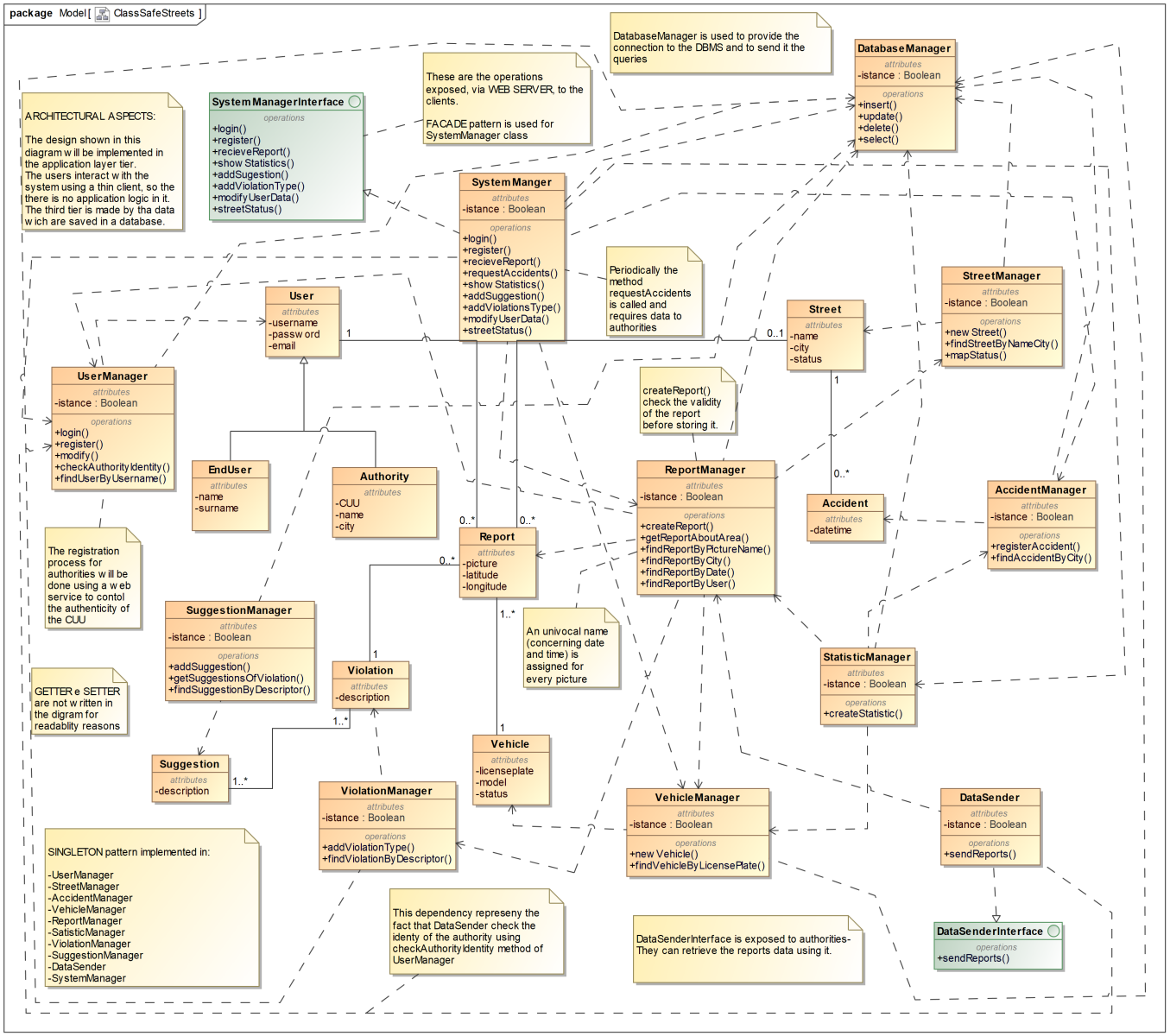

Figure 4: Class diagram

The component diagram below describes the implementation of the classes described before in terms of components.

The macro-component *SafeStretsApplication* represents the Java application running on the application server. It exposes two interfaces: the first one is designed to communicate with the client. The second is exposed to authority systems in order to offer information that can be used to generate traffic tickets.

There are also sub-components that perform specific operations and interact with the database:

- SystemManager: is the component that conveys all the requests to appropriate sub-components and periodically activates the AccidentManager.
- AccidentManager: is the component that calls the specific service to retrieve the data about accidents occurred. These information will be used by StatisticManager.
- StatisticManager: is the component designed to create statistic crossing data coming from report and accidents.
- PositionManager: is the component used to organize data about position and streets.
- UserManager: is the component used to manage user operations (for instance login,registration...).
- ReportSender: it is used to send data about reports stored to authority system. These information will be used to generate traffic tickets. This sub-component exposes directly an interface to the external environment. Naturally before sending the data, the method check the identity of the authority using the UserManager.
- VehicleManager: is the component designed to manage the data about vehicle.
- ViolationAndSuggestionManager: it is used to store and modify the type of violation and suggestion.
- ReportManager: is the component that store reports in the database and retrieve information on it.
- DatabaseManager: it is used to keep the connection to the DBMS and to send queries to it.

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

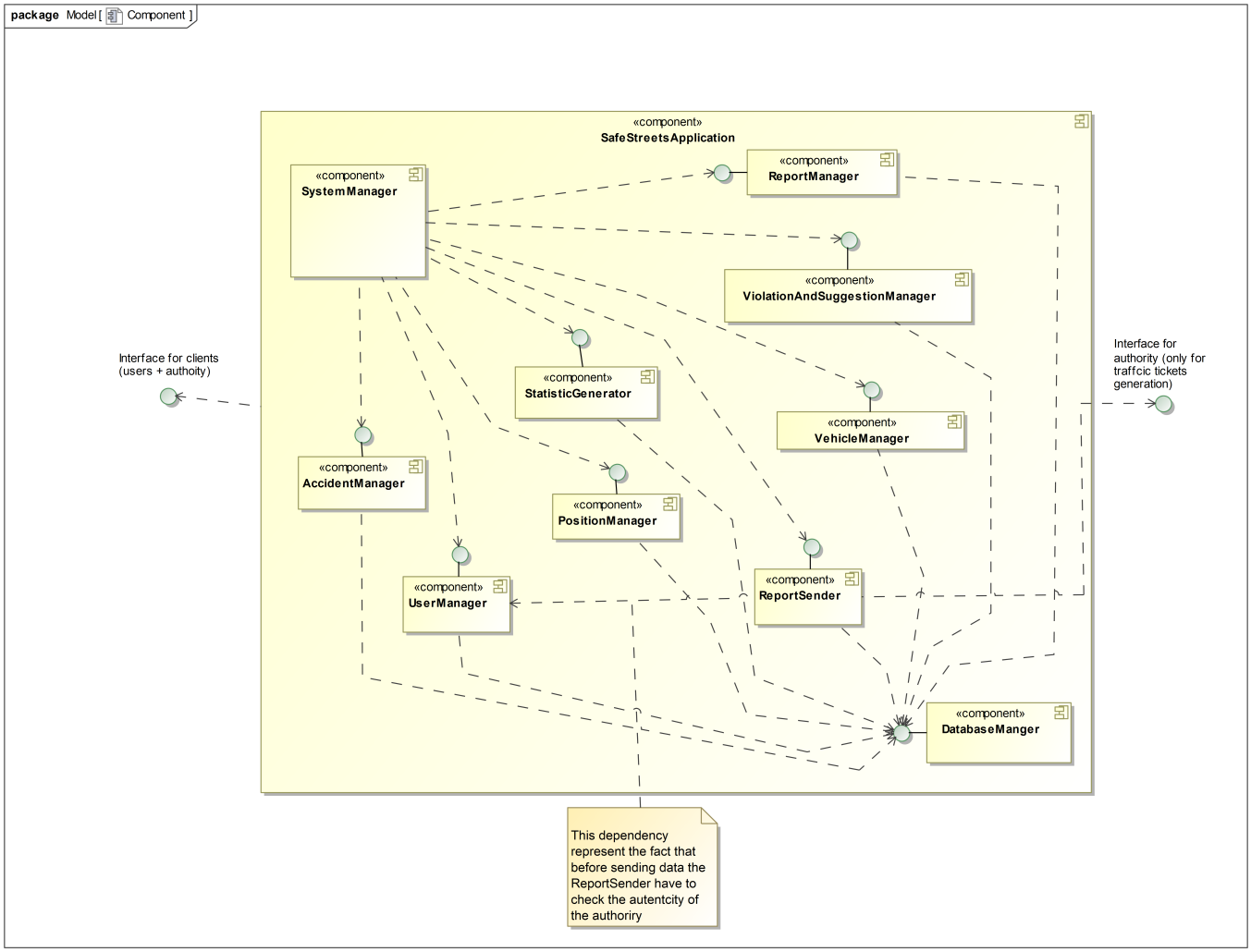

Figure 5: Component diagram

2.3 Deployment view

In the deployment diagram below it is possible to see the physical implementation of the system. SafeStreets is composed by 3 nodes:

- First tier: represent the thin client that make HTTP requests. It is used by users for sending reports and by authorities in order to see statistics.
- Second tier: is made by the web server and the application server. The first one is responsible of the catching of the HTTP requests coming from the clients. The web server read the requests and call the appropriate method exposed by the application server. The real computation of the requests is done here. When data stored are required, the application server communicates with the third tier.
- Second tier: this tier represents the database of the system.

In the diagram there is also another node that corresponds with the authority system in charge of retrieving reports to generate traffic tickets. The application server exposes a interface that provides this function.

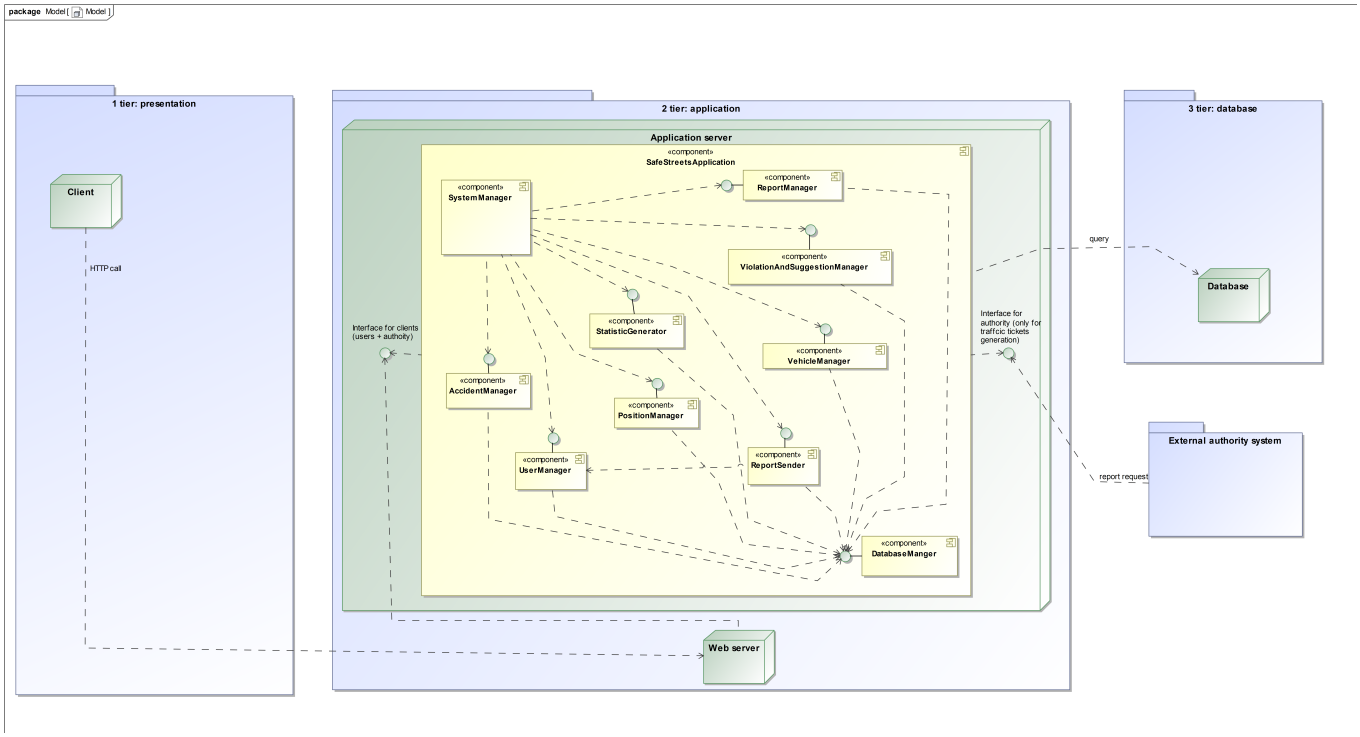

Figure 6: Deployment diagram

2.4 Runtime view

2.4.1 End-user registration

The sequence diagram below shows how it is developed the registration of a new user. The SystemManager class , through its interface, calls the method "register()" of the UserManager, giving three parameters: username, mail, password. The first two parameters must be unique otherwise the registration failed. UserManager will call the method "insert()" of DatabaseManager that will recieve the query to insert the User into the database.

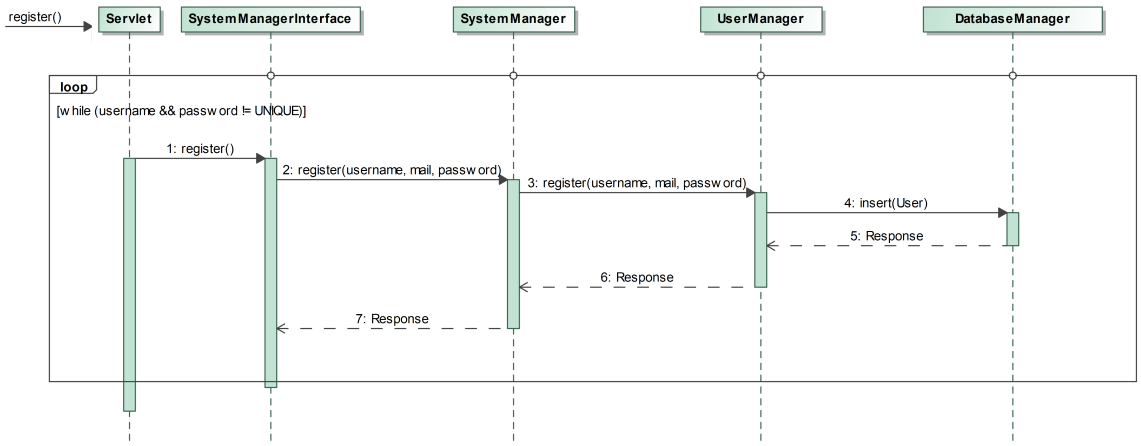

Figure 7: Register

2.4.2 User reports a violation

SystemManager calls the method createReport() that will create a Report object with the data received. The report is created and stored only if it is valid and consistent.

Once the report is received, if they are not already present in the database, the street and the vehicle will be added in the database, through DatabaseManager, using the different manager, StreetManager and VehicleManager with respectively: newStreet() and newVehicle().

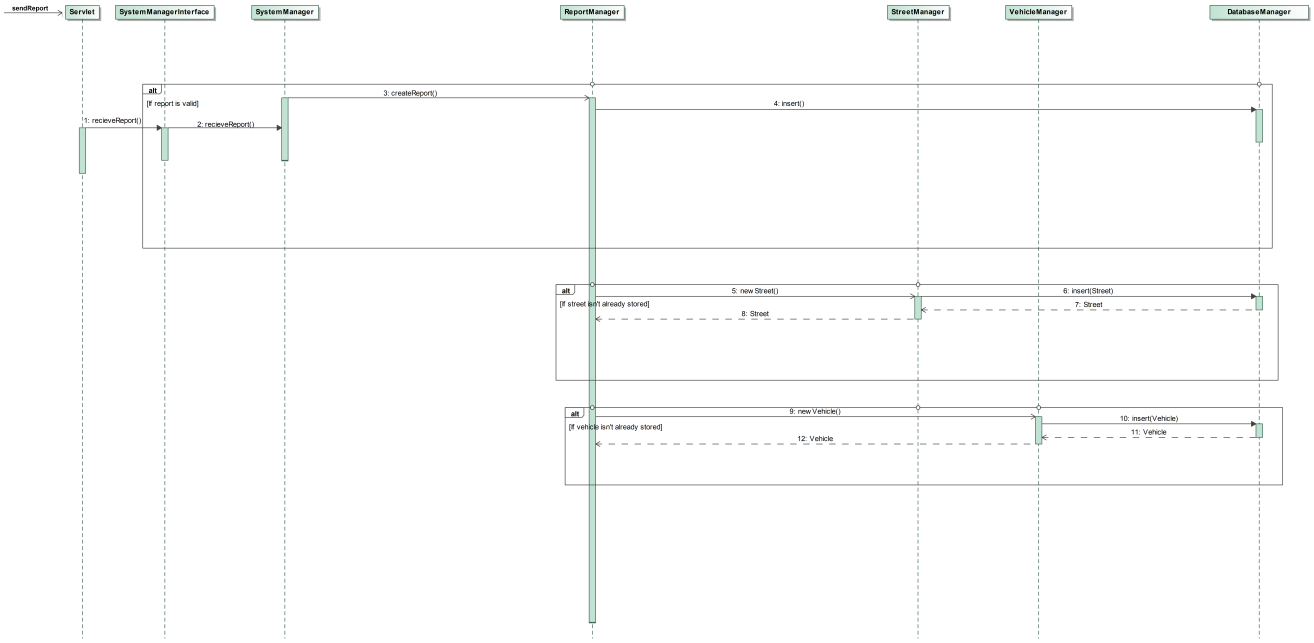

Figure 8: User reports a violation

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

2.4.3 Generate statistic

SystemManager receives the client call showStatistic() and will uses the method createStatistic() of StatisticManager class.

This method uses DatabaseManager to retrieve data from the database and generate statistics regarding a specific area.

Statistic contains: information about number of violations and accidents, most unsafe areas and most involved vehicles.

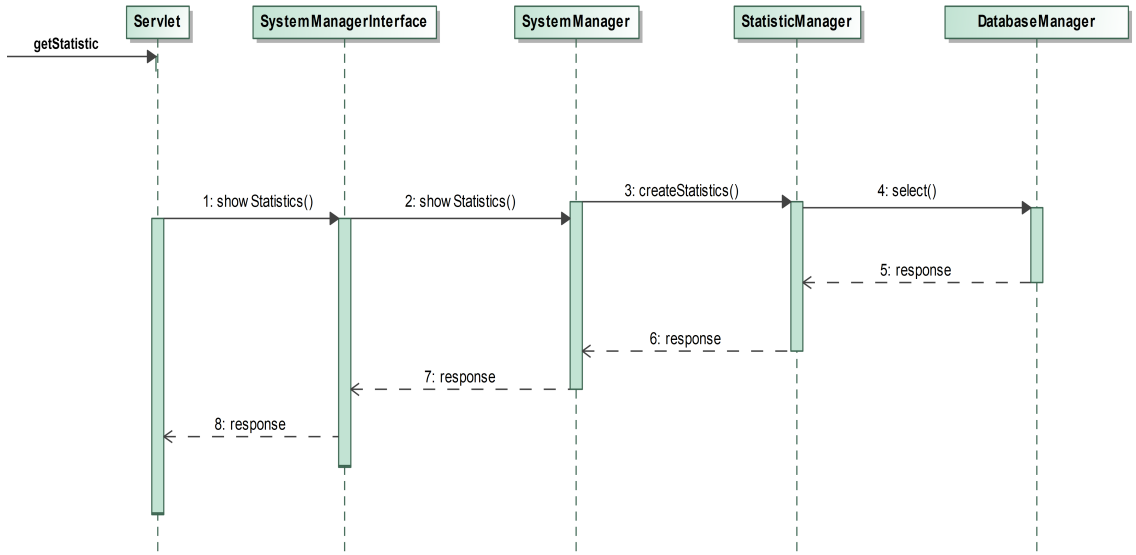

Figure 9: Insert accident and generate statistic

2.4.4 Check street status

SystemManager is called by the method streetStatus and uses a proper method of StreetManager to retrieve the information about the status of the street in a specific area. In this process is involved also DatabaseManager that communicates with StreetManager, which receives the data from the database.

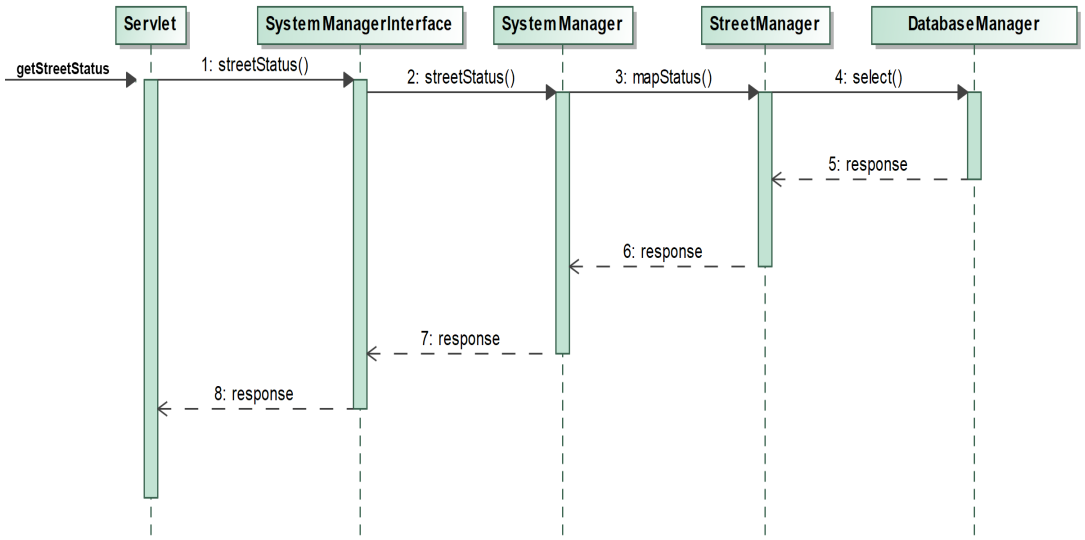

Figure 10: Check street status

2.4.5 Data sender to Authority

DataSender receives an external call from the authority own system, the latter is in charge of the generation of traffic tickets. DataSender calls getReportAboutArea() of ReportManager, which returns the reports received in a specific area.

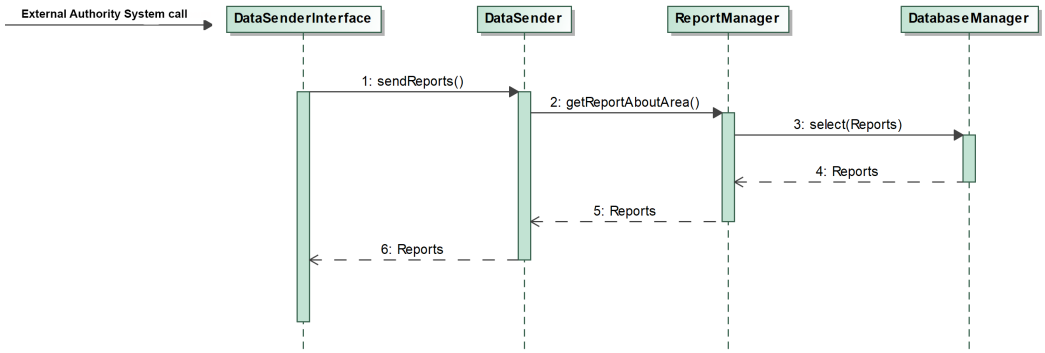

Figure 11

2.5 Component interfaces

The interfaces exposed are mainly two: SystemManagerInterface and DataSender-Interface. Their role was explained in the previous sections of this document.

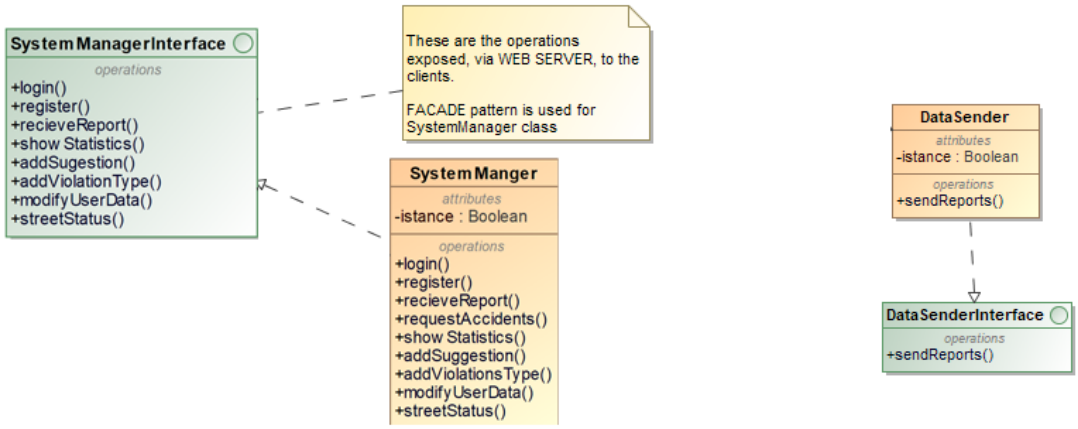

Figure 12: Component interfaces

Here the focus is on the feature provided by the methods exposed.

2.5.1 SystemManagerInterface

Every method exposed by this interface will correspond to a servlet that is able to catch the request and return data to the client.

- User login(String user, String password)

  It is the method in charge of checking the identity of the user/authority that want to log in the application. It returns the object User if the login process was successfully done, otherwise the object is null.

  The application server checks the type of User object returned and then generates the proper response. The method works by invoking methods of UserManger class.

- User register(String user, String password, String mail, String name, String surname)

This method does the registration of a normal user and returns the User object as login method. Since the username has to be univocal the process can abort, in this case a null object is returned. Reading the returned object, the application server will generate the proper response. The method works by invoking methods of UserManger class.

- User register(String user, String password, String mail, String name, String CUU, String city)

  This method does the registration of an authority user and returns the User object as login method. Since the username has to be univocal and the CUU is checked here (via Web Service), the process can abort, in this case a null object is returned. Reading the returned object, the application server will generate the proper response. The method works by invoking methods of UserManger class.

- Bool recieveReport(String picture, Double latitute, Double longitude, String Violation, String violation, String licenseplate, String user, String city, String streetName)

  This method creates the report. During this process all the checks concerning the authenticity of the picture are performed and the coordinates are transformed into street name and city. In case the operations ends successfully, the proper objects (Report, Vehicle, Street) are created or associated (Violation, User) and then all data will be stored in the database. The picture name will be renamed with an univocal key made on date, time and user. A true value is returned. In case of bad result on the checks of the authenticity, a false value will be returned. According to the value returned by the method, the application server will generate the proper response. These operations will be performed using methods of ViolationManager, UserManager, ReportManager, VehicleManager and StreetManager.

- List<Integer> showStatistics (String user, String city)

  This method before generating statistics regarding the city, checks the identity of the user. If the username does not correspond to an authority, it returns a null value. Otherwise it generates statistics using data of reports and accidents referred to the city. According to the result of the method the application server generates the proper response. This method works using UserManager and StatisticManager classes.

- Bool addSuggestion (String user, String description)

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

This method before adding the suggestion, checks the identity of the user. If the username does not corresponds to an authority, it returns a false value. Otherwise it adds the suggestion into the list and returns true. According to result of the method the application server generate the proper response. This method works using UserManager and SuggestionManager classes.

- Bool addViolationType(String user, String description)

  This method has the same behavior of the previous one regarding the suggestions.

- Bool modifyUserData(String oldPassword, String email, String newPassword)

  The method updates the data if the username exists and the old password is correct, then a boolean value is returned. According to result of the method, the application server generate the proper response. The method use UserManager class to perfom that operton.

- List<String> streetStatus(Street city)

  This method analyze the status of a city, especially the status of routes. It returns the color of the street that indicates if there happened so much violations and accidents or not.

2.5.2 SystemManagerInterface

The only method of this interface is sendReports:

List<Report> sendReport (String username, String password, String city, Date date)

The method checks the user's identity using username and password. If it results a profile of an authority, a list of report done in the city on that date is provided to the caller. Otherwise a null value is returned. The process works using ReportManager class and without involving the web server.

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

2.6 Selected architectural styles and patterns

2.6.1 RESTful architecture

SafeStreets adopts RESTful architecture. The main advantage of this architecture is the total separation between the user interface, from the server and the data storage.

RESTful is the architectural style of the web itself, for this reason developers with knowledge in web architecture will naturally develop in the RESTful architecture which is also simple and lightweight.

Client context is never stored on the server between requests, for this reason each request has to contain all the necessary information. A stateless server improves scalability because it allows the server to quickly free resources.

Using a RESTful architecture implies the adoption of the constraints illustrated below. By operating within these constraints, the system gains desirable non-functional properties

Stateless It means that the necessary state to handle the request is contained within the request itself and server would not store anything related to the session. In REST, the client must include all information for the server to fulfill the request whether as a part of query params, headers or URI.

Uniform interface This constraint is fundamental to the design of any RESTful system and it is divided into: resource identification in requests and individual resources are identified in requests (e.g. using URIs in RESTful Web services).

Cacheable Caching, that can be implemented on the server or client side, shall be applied to resources when applicable and these resources have to declare themselves cacheable.

Caching partially or completely eliminates some clientserver interactions, further improving availability and performance.

Layered system An application architecture needs to be composed of multiple layers. Each layer does not know anything about any layer except for the immediate one.

Intermediary servers may improve system availability by enabling loadbalancing and by providing shared caches.

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

Code on demand It is an optional feature. According to this, servers can provide executable code to the client.

ClientServer REST application must have a client-server architecture. Client does not need to know anything about business logic and server does not need to know anything about front-end, so they can change independently.

2.6.2 Client-Server architecture

Clientserver model is an application structure that separates tasks between the providers of resources or services called servers, and multiple entities that request resources or services called clients.

In our case, client is the web-app from which users and authorities can access to SafeStreets through the browser, both mobile and desktop.

As described above, Client-Server architecture is a constraint that defines a RESTful system.

2.6.3 MVC pattern

This pattern is used to separate application concerns.

- **Model**: It is the application data structure and directly manages the data, logic and rules of the application.
  In our implementation:
  - End User
  - Authority
  - Violation
  - Suggestion
  - Vehicle
  - Accident
  - Street

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

- **View**: View represents the visualization of the data that model represents.
  In our implementation:
    - Login form
    - Registration form
    - User send violation report
    - Map showing violations
    - Suggestion for possible interventions
    - Violations by license plate
- **Controller**: It communicates data back and forth between the Model objects and the View objects.
  In our implementation:
    - login
    - register
    - receviereport
    - showstatistics
    - addsuggestion
    - addviolation
    - modifyuserdata
    - streetstatus

Because MVC divides the various components of an application, developers are able to work in parallel on different components without blocking others.

SafeSteets is divided between the front-end and the back-end. Back-end developers design the structure of the data and how the user interacts with it without requiring the UI. Instead, front-end developers are able to design the UI of the web-app prior to the data structure being available. Cons: Knowledge on multiple technologies becomes the norm. Developers using MVC need to be skilled in multiple technologies.

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

2.6.4 Facade pattern

Facade pattern is used in SafeStreets class diagram: *SystemManagerInterface* and *DataSenderInterface*.

It is used to cover the complexities of a large system and therefore provides a simple interface to the client.

2.6.5 Singleton pattern

Singleton pattern is used in SafeStreets class diagram: *UserManager, StreetManager, AccidentManager, VehicleManager, ReportManager, StatisticManager, ViolationManager, SuggestionManager, DataSender, SystemManager*.

By definition it is a pattern design that restricts the number of instances of a class to one object, so it is used when we need to have only one instance of our class.

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

2.7 Other design decisions

2.7.1 Front-end

React For the front-end, SafeStreets uses React JS, a modern JavaScript framework developed by Facebook.

React allows developers to create large web applications which can change data, without reloading the page. The main purpose of React is to be fast, scalable, and simple.

JSX is used for templating, instead of using regular JavaScript. It is simple JavaScript which allows HTML quoting and uses these HTML tag syntax to render components.

2.7.2 Back-end

Spring Boot For the back-end SafeStreet uses Spring Boot, is an application framework for the Java platform. It is an already configurated Spring solution for creating stand-alone, Spring-based applications. It is an HTTP- and servlet-based framework for web applications and RESTful web services.

2.7.3 Database

MySQL SafeStreets uses MySQL as RDBMS (*Relational DataBase Management System*) to store its data. Structured relationships between rows and columns in a table, SQL databases are best when you need ACID compliance.

ACID stands for:

- Atomicity: each transaction either succeeds completely or is fully rolled back.
- Consistency: data written to a database must be valid according to all defined rules.
- Isolation: when transactions are run concurrently, they do not contend with each other, and act as if they were being run sequentially.
- Durability: once a transaction has been committed to the database, it is considered permanent, even in the event of a system failure.

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

3 User Interface Design

3.0.1 User interfaces

While in the RASD document UI mocks were generically described showing just the position of the components inside the screens, in the DD document are shown as they are in production environment.

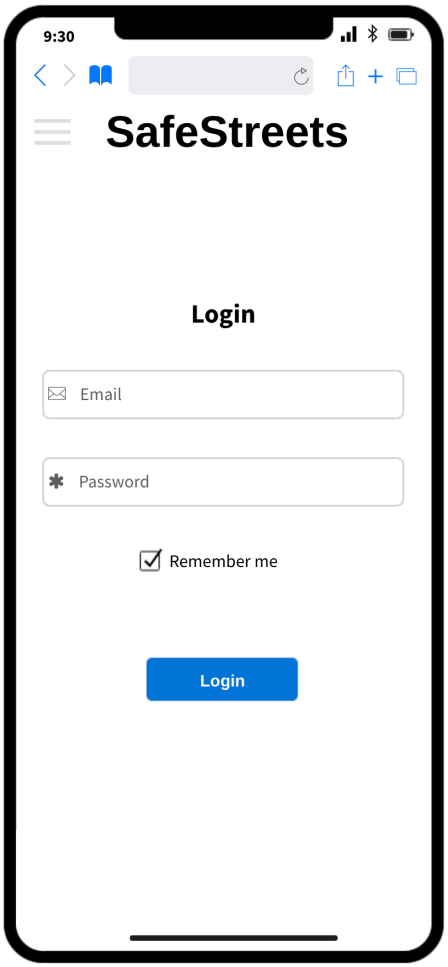

Figure 13: Login form

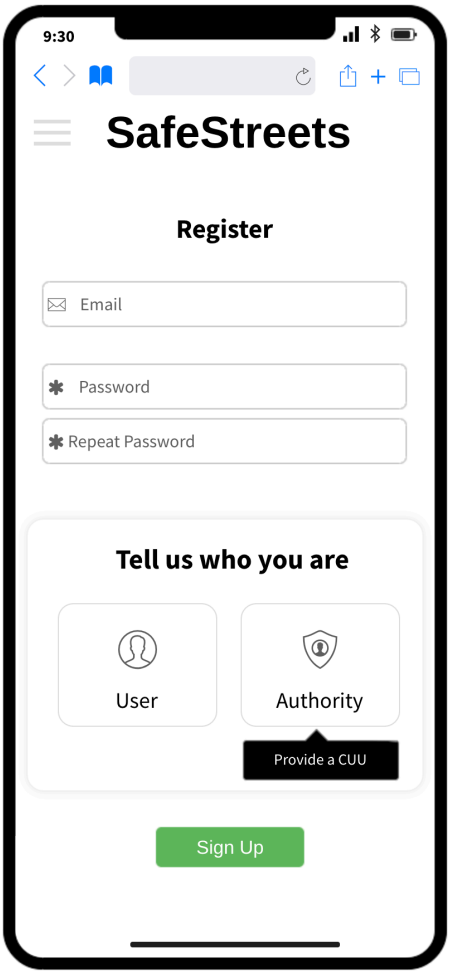

Figure 14: Registration form

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

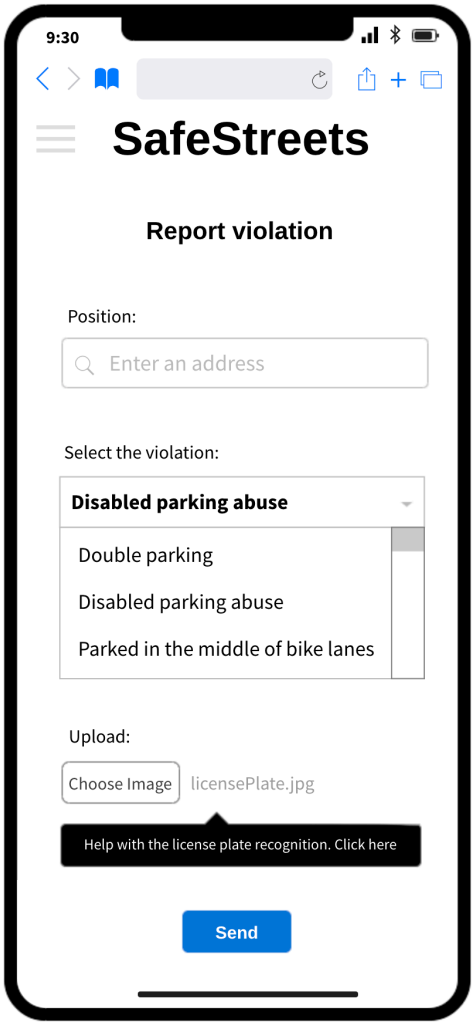

Figure 15: User send violation report

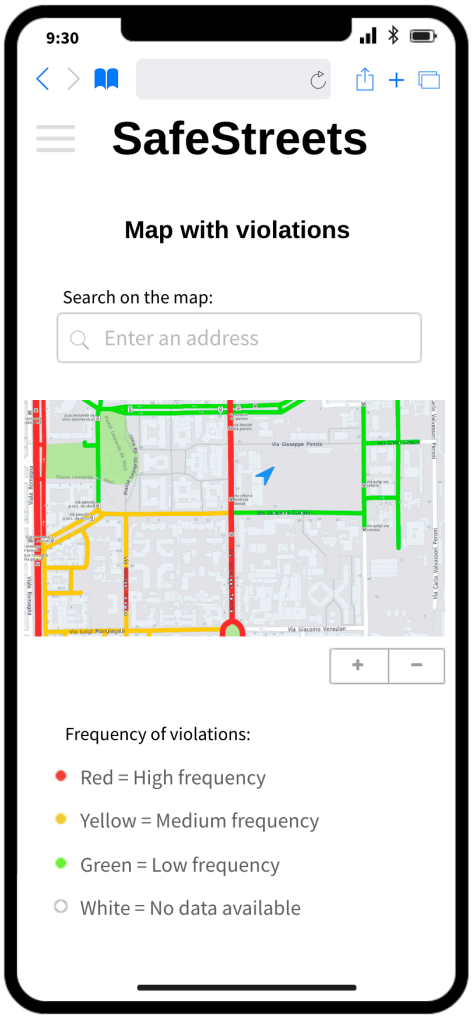

Figure 16: Map showing violations

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

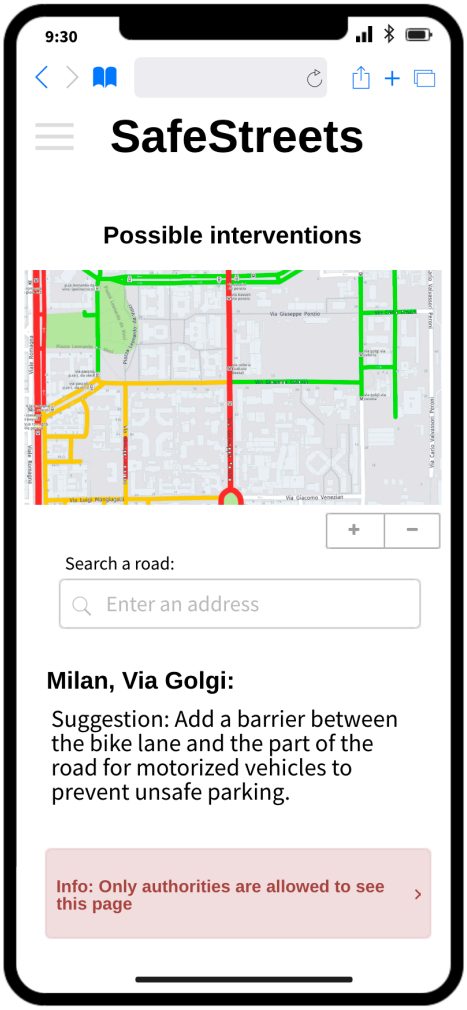

Figure 17: Suggestion for possible interventions

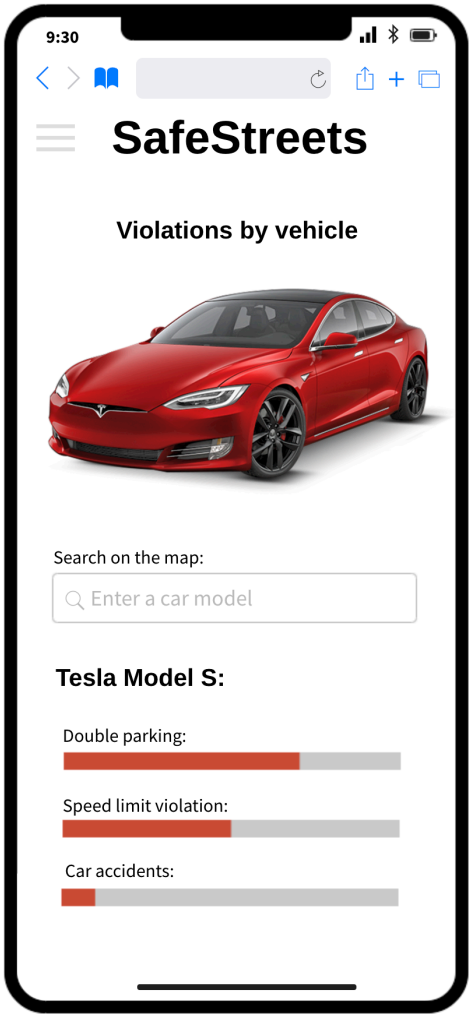

Figure 18: Violations by vehicle (license plate)

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

3.0.2 Screens routing

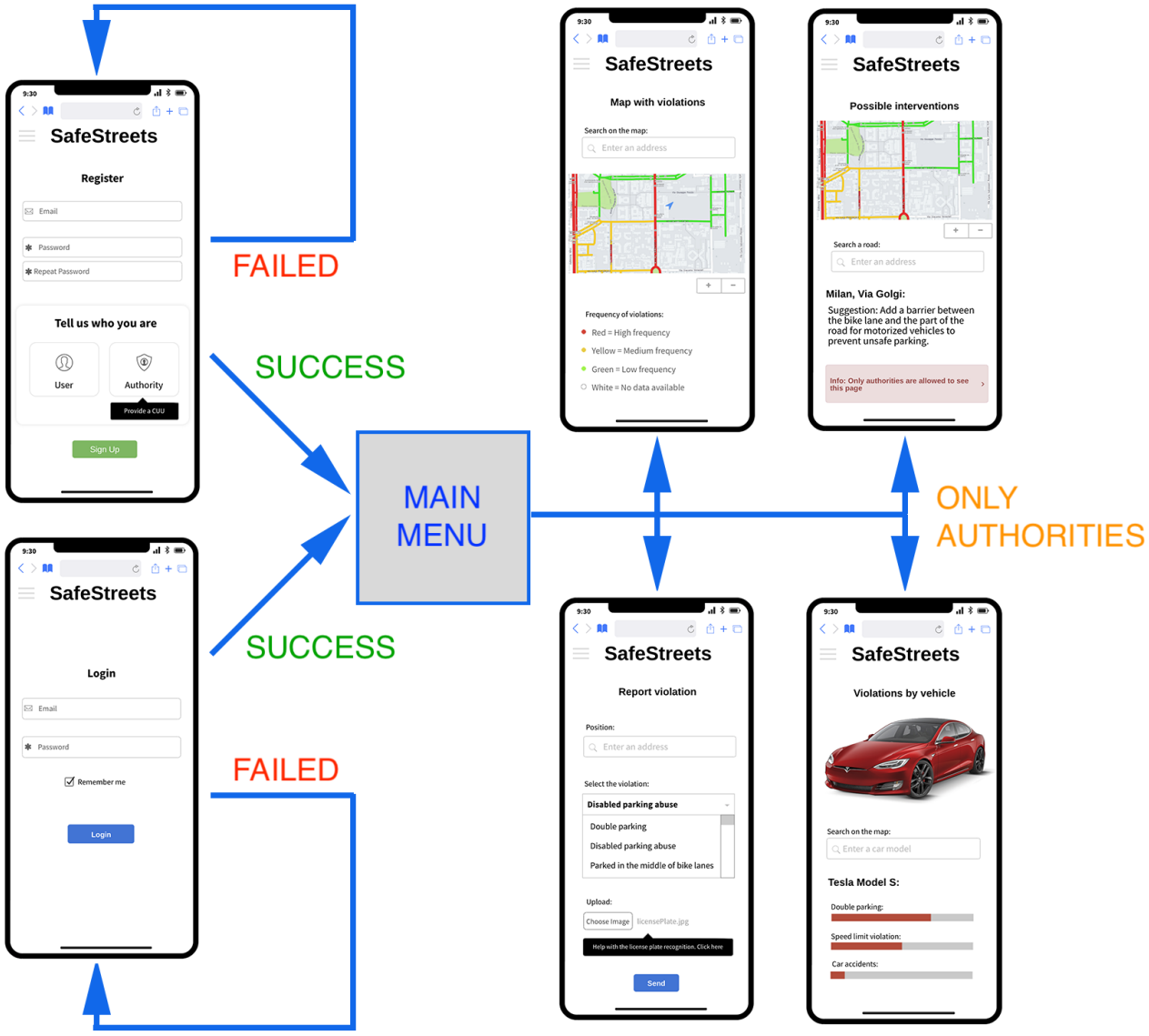

Figure 19: Screens routing

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

4 Requirements traceability

- **R1**: The system have to allow users to send pictures and information about traffic violations.
  ReportManager: it creates the report using the method createReport(). The user will log into the web platform and will insert all data about the violation.

  A report will contain: the type of violation, the license plate of the vehicle that commit the violation and the street where it happened.

- **R2.1**:The system must be able to retrieve license plates from pictures.
  ReportManager: it contains all the methods to retrieve the license plate from pictures and/or from user input.

  Using the method createReport(), the picture sent by the user will be analyzed. In addition the user can insert the license plate into a textbox to help the recognition. The system will match the license plate retrieved by the algorithm with the license plate suggested by the user.

- **R2.2**: The platform have to retrieve the street where the violation occurred.
  ReportManager: it has all the methods to retrieve the street from the GPS and/or from the user input. It will use the method createReport() in which is contained the position sent by the GPS or the position inserted by the user.

- **R2.3**:The platform must be able to ensure the picture reliability.
  ReportManager: it contains all the methods to retrieve the license plate from pictures and/or from the user input. Using the method createReport() will be analyzed the picture sent by the user. In addition the user can insert the license plate into a textbox to help the recognition. The system will match the license plate retrieve from the alghoritm with the licenseplate insert by the user. To ensure reliability if they are the same the report will be created.

- **R3.1**:The system have to locate the most unsafe areas.
  StatisticManager: it contains the method to show the most unsafe areas with the data provided by the **ReportManager** and by the **AccidentManager**.

- **R3.2**:The system must be able to show the impact of the application in terms of number of violations.

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

StatisticManager: using the method createStatistic will be showed the impact of the application. It allows to see the number of violations with their type and the streets where they happened.

- **R3.3**:The platform have to generate other type of statistics like the most egregious offenders.
  StatisticManager can display statistics of the most offenders, data can be retrieved through the **ReportManager** that permit to get the license plate of the most offenders.

- **R4**:The platform must be able to suggest possible interventions.
  SuggestionManager: provides suggestion based on the type of violation committed by offenders. It will receive the type of violation, and the SuggestionManager will provide the related suggestions for that violation.

- **R5**:The system have to guarantee data integrity.
  SystemManager guarantees the data integrity, because it stores and maintains the data only if they are complete and consistent.

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

5 Implementation, integration and test plan

SafeStreets is composed by these subsystems:

- Web App for end user
- Web App for authorities
- Application back-end
- Report provider for authorities

Furthermore, the system works with external services such as:

- Google Maps
- CUU authenticator
- DBMS

Each subsystem will be implemented and tested independently with a top down approach. At the end of this phase a test of the whole system will check the integration between the components.

The external components will be used directly without any kind of test because they are assumed to be reliable.

5.1 Features

In this section we consider all the features provided by SafeStreets to users in order to understand their importance for the costumer related to the difficulty of implementation.

FEATURE	IMPORTANCE	DIFFICULTY
login	medium	low
reception of the report	high	medium
reception of the report	high	medium
license plate recognition	high	high
unsafe area individuation	medium	medium
impact evaluation	medium	low
statistics generation	medium	medium
suggestion providing	low	low

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

5.2 Implementation

The implementation of these features will be done by implementing first the classes that provides methods to other classes.

One of the first task will be to create the DatabaseManager that will be used by the application to interact with the database. Naturally, before doing this task the database have to be created and configured.

The next step will be to implement all the others managers excepted from the SystemManager, that works on these. After this, the SystemManager will be implemented.

The last implementing task concerns the creation of the interfaces exposed to the web app (servlets) and to the authorities.

The creation of the front-end of the web app can be done simultaneously with the other tasks.

At the end in the front-end will be implemented the management of data fetched and sent through API calls, *in the image below is represented as "javascript part"*.

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

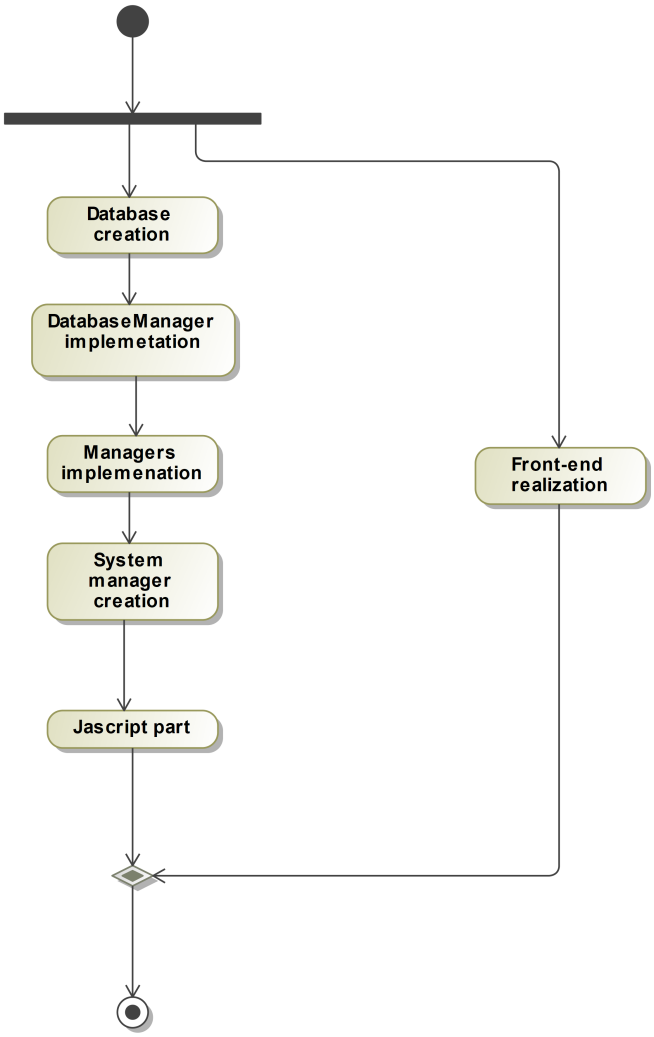

Figure 20: Tasks flow

5.3 Testing

5.3.1 Unit tests:

Tests of the system components will be done on the single piece of code in order to unsure the reliability of their behavior.

By definition, Unit tests should have no dependencies on code outside the unit tested, for this reason the database wont be docked to tests, in order to avoid any hard dependency on external data.

The tool used for Unit testing is **JUnit**.

In conjunction with JUnit, we use **Mockito**, a mock framework that allows you to create mock objects and define their behavior.

A mock object simulates the data source and ensures that the test conditions are always the same.

5.3.2 Integration tests:

Integration tests focus on integrating different layers of the application. No mocking is involved, so we introduce actual dependencies like databases.

Arquillian is an integration testing framework designed for JEE. Tests are written with a syntax similar to JUnit, used to execute test cases against the container.

API testing is part of integration testing, used to determine if they meet expectations for functionality, reliability, performance, and security.

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

6 Effort Spent

6.0.1 Ivan Cavadini

TASK	TIME
Purpose	2h
Scope	2h
Definitions, Acronyms, Abbreviations	1h
Overview	3h
Runtime view	10h
Requirements traceability	1h
Various	5h
TOTAL	24h

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

6.0.2 Nicolò Molinari

TASK	TIME
Component view	12h
Deployment view	3h
Revision history	0.5h
Component interfaces	3h
Implementation, integration and test	3h
Database structure	1h
Various	4h
TOTAL	26.5h

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

6.0.3 Luigi Pederzani

TASK	TIME
Revision history	2h
Selected architectural styles and pattern	8h
Other design decisions	4h
User characteristics	4h
User Interface design	5h
Various	1.5h
TOTAL	24.5h

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

6.1 Versions

VERSIONS	DESCRIPTION
1.0	Introduction creation
1.1	Introduction review
2.0	Overview draft
2.1	Database structure
2.2	Added class diagram
2.3	Added component view
2.4	Added runtime view (sequence diagrams)
2.5	Creation of component interfaces
3.0	Writing out Selected architectural styles and patterns
3.1	Added description of Other design decisions
4.0	Creation of UI mocks
4.1	Added screens routing
5.0	First draft of Requirements traceability
5.1	Requirements traceability correction
6.0	Writing out Implementation
6.1	Writing out Test
7.0	Document review
7.1	Final version

6.2 Used Tools

- TeXstudio 2.12.14
- Magic Draw 19.0
- MockFlow
- Dropbox Paper

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani

7 References

- Specification document:"SafeStreets mandatory project assignment"
- DD sample from previous year
- Slides from the course Software Engineering 2

Copyright 2019, Ivan Cavadini, Nicolò Molinari, Luigi Pederzani