Software engineering project

SafeStreets

DD : Design Document.

Authors:
Mattia Micheloni,
Francesco Montanaro,
Amedeo Pachera.

Professor:
M.Rossi.

Version 1.0
10/12/19

Contents

1 Introduction **1**
- 1.1 Purpose . . . 1
- 1.2 Scope . . . 2
- 1.3 Definitions, acronyms, abbreviations . . . 3
  - 1.3.1 Definitions . . . 3
  - 1.3.2 Acronyms . . . 3
  - 1.3.3 Abbreviations . . . 4
- 1.4 Revision history . . . 4
- 1.5 Reference documents . . . 4
- 1.6 Document's overview . . . 4

2 Architectural design **6**
- 2.1 Overview: High-level components and their interaction . . . 6
- 2.2 Component view . . . 7
- 2.3 Deployment view . . . 10
- 2.4 Runtime view . . . 10
  - 2.4.1 Report a violation . . . 11
  - 2.4.2 Validate a violation . . . 11
  - 2.4.3 Report an accident . . . 12
  - 2.4.4 Request statistics . . . 12
- 2.5 Component interfaces . . . 12
- 2.6 Selected architectural style and patterns . . . 14
  - 2.6.1 Client - Server architecture . . . 14
  - 2.6.2 MVC : model-view-controller . . . 14
- 2.7 Other design decision . . . 15
  - 2.7.1 License plate recognition algorithm . . . 15
  - 2.7.2 Statistics computation . . . 15

3 User interface design **29**

4 Requirements traceability **35**

5 Implementation, integration and test planning **40**
5.1 Team 1 41
5.2 Team 2 42
5.3 Flow of Integration 43

6 Effort spent **48**

Chapter 1

Introduction

This chapter explains the purposes and scopes of this document and gives some keys of interpretation of it (abbreviations, definitions..).

1.1 Purpose

The purpose of this document is to give a more detailed description of the architectural choiches made to perform into the software system all the functionalities described into the RASD document.
Therefore, if the RASD describes all the features that the system must indulge according to the stakeholders needs, the DD describes how it has been thought to convert all the high-level system requirements into the design structure.
It provides guidance and material which is intended to assist technical staff to perform system testing and implementation and, moreover, to facilitate future maintenance and features extension.
List of topics covered by the document:

- High-level architecture.
- Main Components, interfaces and deployment.
- Runtime behavior.
- Design Patterns.
- UI details.
- Mapping between requirements and architecture.
- Implementation, integration and testing plan.

1.2 Scope

Here a short review of SafeStreet scope referred to what just stated in the RASD document.
Safestreet is a software thought to make users able to report traffic violations to authorities. Anyone can report violations simply opening SafeStreet application and taking several pictures of them. For each violation is required at most 6 pictures in which 1 of the involved vehicle license plate.
In order to guarantee to authorities the reliability of data gathered, which means that pictures can not be altered or modified, the latter can be only taken in the application itself.
To make it easier the reporting process all other information are automatically mined by the system, such as time, date, geographical location and license plate; the last one is extracted by the system from the license plate picture.
Finally, user is asked to choose the violation type from a predefined set and confirm the above listed data that will be sent.
If no errors occur, the system stores the violation, otherwise it is refused.
Authorities, in order to access information concerning reports made by users, must log into the SafeStreet website.
They are allowed to discard, modify and validate violations sent by users, in order to take into account only pertinent violations.
Thanks to the data granted by the SafeStreet initiative and the validation performed by authorities, they can even decide to generate traffic tickets from validated violations.
Authorities can also upload information about accidents that occur on the municipality.
The software provides also a service to offer both users and authorities statistics computed on the available data. Obviously, in order to make statistics consistent and reliable, the latter are generated only by information validated by authorities.
Statistics highlight streets or areas with the highest frequency of violations, the vehicles that commit the most ones or the infractions trend that occurs over time.
Crossing data gathered by users and authorities, the system can identify potentially unsafe areas and suggest interventions in order to improve streets safety.

1.3 Definitions, acronyms, abbreviations

This section gives some definitions in 1.3.1, acronyms in 1.3.2 and abbreviation in 1.3.3 which will be use in the document, in order to explain some concept and help the general understanding.

1.3.1 Definitions

- User: The customer of the application who can upload reports concerning traffic violations and have access to the system's statistics services.
- Authority: The costumer of the application who can access the information on reports made by the users in order to validate or modify them, and eventually generate tickets. They can also upload on the system information about accidents that occur on the territory and have access to the system's statistics services.
- Violations: The set of all the infractions that occur on the territory which are recognized by the system.
- Metadata: The information extracted by the system from a report such as license plate, geographical position, data, time and traffic violation type.

1.3.2 Acronyms

- API: Application Programming Interface
- GPS: Global Positioning System
- UI: User interface
- OCR: Optical Character Recognition
- DBMS: DataBase Management System
- MVC: Model View Controllorer
- AM:Authentication Manager
- RSM: Retrive Statistics Manager
- CSM: Compute Statistics Manager
- MSM: Mobile app Statistics Manager

- WSM: Website Statistics Manager
- VG: Violation Generator
- WVM: WebSite Violation Manager
- WAM: WebSite Authentication Manager
- WAccM: WebSite Accidents Manager
- GVM: Get Violation Manager
- UVM: Upload Violation Manager
- UAM: Upload Accident Manager
- EVM: Edit Violation Manager

1.3.3 Abbreviations

- R_n : nth Requirement.

1.4 Revision history

- Version 1.0 : 10/12/2019

1.5 Reference documents

- Design document: "Mandatory Project Assignment AY 2018/2019".
- SafeStreets RASD document.
- Lecture slides of professor M.Rossi and E.di Nitto of Politecnico di Milano.

1.6 Document's overview

The DD document is structured by seven chapters as describes below:

- Chapter 1: It presents an introduction. It is formed by a general description of the purpose and scope of the document and a list of definitions, acronyms and abbreviations used in the document.

- Chapter 2: It describes the architecture of the system with different representations, beginning with an overview, following with the component, sequence and interface diagrams to describe all the components and their interaction. In the end describes also the architectural and deployment choices.
- Chapter 3: It presents the mockups of the user interfaces that are close to the final layout.
- Chapter 4: It describes a mapping between the requirements listed in the RASD document and the components that manage them.
- Chapter 5: It describes the main guideline to the implementation and testing adopted for SafeStreets, in particular it specifies the flow of implementation and the flow of integration of each subsystem.

Chapter 2

Architectural design

This chapter contains the description of the main architectural design choices. The section 2.1 describes the architecture in an high level, the section 2.2 describes the component of the system, the section 2.3 shows the deployment diagram to describe how the system will be physical deployed, the section 2.4 shows with sequence diagrams the flow of each components, the section 2.5 describe the main interfaces of the components, the section 2.6 describe the selected architectural style and patterns and the last section 2.7 describes other decisions about algorithms and data structures.

2.1 Overview: High-level components and their interaction

The software to be is designed as a client-server system with a 3-tier architecture: clients, application server and a database. In particular, the clients provide an interface both to the users and authorities, and they incorporate only a part of the application's logic. The main logic features of the software are delegate to the server, which transitively communicates with the database, whose purpose is to store the data used by the application itself. This architecture has been chosen in order to make the software scalable and flexible to future integrations and modifications, without giving up a fair robustness and a good user experience. The communication between the various components is managed through the exchange of requests and responds, so, some of the main design choices have been taken in order to minimize the response time and optimize the expenditure of the available resources. As mentioned before, the clients provide an interface both to the users (a mobile application) and authorities (a website) in order to allow them to exploit, in a simple way, all the services of the application. The purpose

is to make them as efficient and intuitive as possible. The client software is of limited complexity, acting in many cases only as an interface to the server. The server is used to manage the main application logic; moreover, it cointains the management of some of the functions that users do not need to access, or the ones that require an high expenditure of resources. One of the main purposes of the server is to send requests to the database; in particular, by communicating with its DBMS it can handle, when necessary, all the operations of reading, writing and uploading of data.

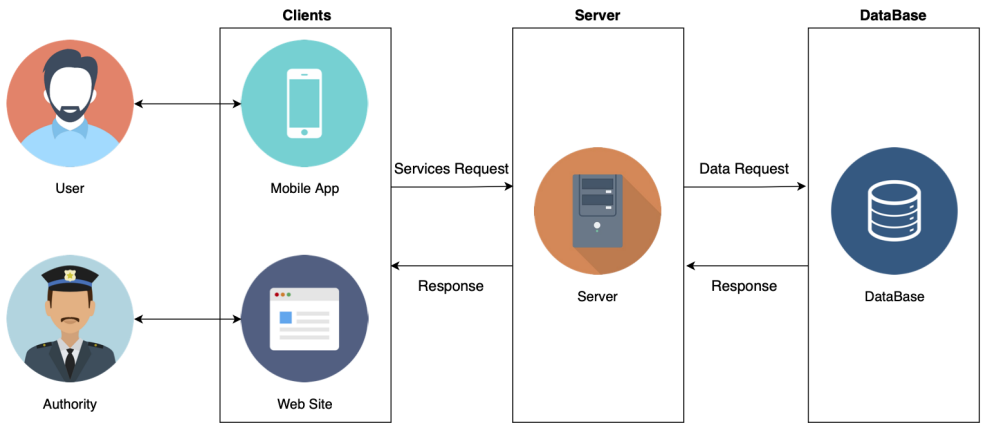


Figure 2.1: High level view of the application architecture

2.2 Component view

In this paragraph are presented the logical and physical components of the system, how they are structured and the way in which they interact with each other. In particular, the components will be described first in a general way in order to highlight how they are integrated into the main structure of the system and how they are connected with the others. Then, a more detailed description is provided for each one in order to identify their subcomponents and their logical features. In figure 2.2 is shown the component diagram of the system.

The system is based on three main physical components: User interfaces, Server and Database. The entire system depends on the server component which is the core of the application. Its purpose is to handle the main logic features such as the authorities'authentication, the management of statistics and the operations on the reports made by the users. The available data

are stored into a database which is managed by a DBMS that allows the creation, manipulation and querying of data. The UI is characterized by two subcomponents which are a mobile application (made for the users) and a website (made for the authorities); in particular in order to reach the prefixed purposes they require the integration of some additional component: A camera interface, for the Mobile App, to take pictures of the violations and a GPS to get users'position, moreover and an external API providing a mapping service is required for both The UIs. Google Maps API is chosen instead of other services due to its versatility, its ease of use, and its libraries suitable for many devices and programming languages. As mentioned before, the communication between components is managed by the exchange of requests and responses. In particular, The UI communicates with the server in order to get the main application services, while the server communicates with the DBMS in order to upload, get, or edit the data. More details regarding the logical subcomponents of each main component are described below:
In figure 2.3 is shown the explosion of the component diagram with all server'subcomponents.

- **Authentication manager**: The purpose of this component is to collect the requests coming from the website UI containing the credentials inserted by authorities and to check for their correctness. Then, it sends back either a positive or negative response which determines the possibility to log into the system and the access to the services made for them.

- **Retrieve Statistics manager**: When it receives a request coming from the UI concerning statistics it sends in turn a request to the DBMS component in order to get them. Then, it sends back a response containing the information requested.

- **Compute Statistics manager**: Its task is to query the database by sending requests to its DBMS in order to get the data on which to build statistics. Then, through appropriate algorithms it computes all of them and stores the new information into the database.
  In order to optimize the expenditure of resources and minimize the response time, the invocation of this component is scheduled only once a day (which is also the minimal time granularity settable by the users).

- **Edit Violation manager**: The purpose of this component is to collect the requests coming from the website UI to edit a violation report previously made by the users. Then, it communicates with the DBMS in order to upload the information stored in it.

- **Get Violation manager**: The purpose of this component is to collect the requests coming from the website UI to get the violation reports made by the users. By communicating with the DBMS, it gets the information and sends them back to the UI.

- **Upload Violation manager**: The purpose of this component is to collect the requests coming from the mobile application UI to upload a violation report. By communicating with the DBMS, it writes the information on the database.

- **Upload Accident manager**: This component collect the requests coming from the website UI containing the information on an accident that occurred on the territory of the municipality. By communicating with the DBMS, it writes the information on the database.

In figure 2.4 is shown the explosion of the component diagram with all UI'subcomponents.

- **Violation Generator**: This component contains all the functions that allow users to generate a violation report. Its purpose is also to verify if a report has been correctly formulated otherwise it prevents the user to upload it into the system. Then, it sends a request to the server to store the information.

- **Mobile App Statistic Manager**: It sends a request to the server component to get the statistics, providing also an interface through which is possible to filter the information to retrieve. When a response is returned it plots the data and displays them to the user.

- **WebSite Authentication Manager**: This component allows authorities to input their own credentials in order to log into the system. Then, it sends a request to the server component and waits for its response.

- **WebSite Violation Manager**: This component sends a request to the server in order to get the violation reports made by the users. When a response is returned it displays them in a correct way.

- **WebSite Statistics Manager**: It sends a request to the server component to get the statistics, providing also an interface through which is possible to filter the information to retrieve. When a response is returned it plots the data and displays them to the authorities.

- **WebSite Accidents Manager**: This component provides an interface to the authorities which allows them to insert information concerning

an accident that occurred on the territory. Then, it sends a request to the server containing the data to upload.

2.3 Deployment view

The deployment diagram in figure 2.5 describes the execution architecture of the software system; it includes nodes such as hardware or software execution environments and the middleware connecting them. The purpose of this section is to visualize how the system will be physically deployed on the hardware and the UML deployment diagram is the one that mostly outlines the hardware topology of a system. As usual in standard UML specification the «executable» and «document» represents artifacts respectively for executable files and tables in DB. The three tiers contain:

- **Tier 1:** The first tier contains the presentation logic for Users and Authorities, in fact as shown in figure 2.5 the MobileAPP makes the User able to simply reports violations and retrieve statistics of each type. On the other hand authorities have to log in on the SafeStreet website to perform operations on restricted information, such as violations validation and report accident. All the operations both MobileAPP and WebAPP side are possible through the communication via server, that retrieves all kind of data needed.

- **Tier 2:** Here is deployed the logic to retrieve and store data on the DB, which means all the violations, accidents and statistics. The server contains also executables to daily compute all kind of statistics and store them on the DB, the reason is to optimize the available resources and to minimize the response time.

- **Tier 3:** The DB Server is conceived to execute a non-relational DBMS on a NoDB. The choice of a non-relational DB allows a lighter computation than a relational one, without risks for data integrity, why data are not split as in a relational DB. Moreover it allows horizontal scalability of data and it is suitable for large recorded amount of data that have to be retrieved.

2.4 Runtime view

This section describes the flow of interaction between the components of SafeStreets using sequence diagrams.

In particular four use-cases are taken in consideration explaining the manner in which the components are used (report a violation in 2.4.1, validate a violation in 2.4.2, report an accident in 2.4.3 and request statistics in 2.4.4). The behavior of the system with the other use-cases is similar to the ones described below.

2.4.1 Report a violation

The sequence diagram in figure 2.6 shows the flow of reporting a violation.
The flow starts from the mobile application which asks to the Violation Generator to create a report. The generator check if the device settings are acceptable (no fake-GPS, internet connection available..) and sends an error message in negative result. If settings are accepted the flow proceeds taking pictures from the mobile camera and the GPS location and finally reporting to the server subcomponent UploadViolation Manager all the pictures and metadata. This last subcomponent interacts with the DBMS to insert in the database the new data of the violation.
This operation can be done in loop.

2.4.2 Validate a violation

The sequence diagram in figure 2.7 shows the flow of validating a violation.
The flow starts from the website which asks to the web authentication manager to login into the system. This subcomponent forward the request to the server authentication manager which interact with the DBMS to verify the correctness of credentials and return an error message in negative result.
After the authentication, a request to the website violation manager is made, which send it to the Get Violation manager on server. This last subcomponent interacts with the DBMS to query the database (a notification is sent in the case of empty list).
The result is bark-propagated to the website in which a violation can be modified. After the update, a request is made to the web violation manager passing the new violation and the update, this data are forwarded to the Update Violation Manager on the server which interacts with the DBMS to patch the violation.
This operation can be done in loop.
This flow is similar in the case of deleting the violation, with the difference of Delete Violation Manager instead of Update Violation Manager on the server.

2.4.3 Report an accident

The sequence diagram in figure 2.8 shows the flow of reporting an accident. The flow starts from the website which asks to the web authentication manager to login into the system. This subcomponent forward the request to the server authentication manager which interact with the DBMS to verify the correctness of credentials and return an error message in negative result.
After the authentication, a request to the website accident manager is made, passing all the data needed, which send them to the upload accident manager. This last subcomponent interacts with the DBMS to insert in the database the new data of the accident.
This operation can be done in loop.

2.4.4 Request statistics

The sequence diagram in figure 2.9 shows the flow of requesting a statistic. Firstly all the type of statistics are computed by the Compute Statistic Manager on the server. This subcomponent request data to the DBMS, makes all the computations and insert the new data to the database.
Then in the mobile app a request to the Mobile Statistics Manager (on the website the process is the same) passing the type of statistics desired. This subcomponent forward the request to the Retrieve Statistic Manager on the server which interacts with the DBMS to get data from the database and back-propagate the results to the mobile application.
In the case of request a map-typed statistic the UI component request to GoogleMapsAPI the map passing the list of coordinates to plot.
This operation can be done in loop.

2.5 Component interfaces

In this paragraph are described all the interfaces of each component of the system. The methods presented are the ones considered the most relevant in order to exploit the main critical aspects of application behavior, components dependencies and internal interactions.
The figure 2.10 shows the interface diagrams of the system.
All the messages presented in the sequence diagram that do not have a corresponding method in the interfaces are described below, for instance exceptions and notifications. Moreover a short description of their main behaviors is provided.

- **CheckDeviceSettings() :bool**: the method in the violation generator

interface checks if device settings are valid to generate a violation, so if they are all satisfied it returns a positive message, otherwise it raise an exception. This exception is thrown through a bool value to warn the user that the violation cannot be created.

- **Login(user,password) :bool**: the method in the Web Authentication Manager calls the Login function of the Authentication Manager on server side that checks and returns an exception through a bool value to warn Authority on the incorrect submission.

- **Login(user,password) :bool**: in the server Authentication Manager the Login method retrieves authorities credentials from the DBMS to compare them with the ones submitted by them, if their correctness is satisfied a positive response is returned, otherwise an exception is thrown.

- **requestListUnvalidatedViolation(): list<Violation>**
  requestListValidatedViolation(): list<Violation>:
  The Web Violation Manager can request to the Get Violation Manager the complete list of validated or not yet validated violations. If the list is not empty it is returned otherwise an exception in thrown.

- **requestUpdateViolation(violationID,type): bool**
  requestDeleteViolation(violationID): bool:
  The Web Violation Manager can update the type of a violation or delete it by calling the respective method in the Edit Violation Manager. Both the functions throw an exception in case an error occurs, if the violation is not correctly updated or deleted, and they are handled with the respective method in the Web Violation Manager.

- **WebSite Statistic Manager**
  App Statistic Manager:
  These Interfaces handle the communication with the server for retrieving statistics according to the chosen type. Both interfaces are linked with the Retrieve Statistics Manager that communicates the DBMS in order to get them. RequestStatisticData() contains a getStatisticType() method that allows users and authorities to set the information in which they are interested. Then, it invokes the corresponding method on the server(getStatistics(type)) to get them from the database. Finally showStatistics() method is invoked. If an error occurs the server throws an exception which is handled by the RequestStatisticData ().

- **DBMS**: The DBMS contains all the methods to retrieve, insert and upload data. All the server interfaces interact with it and handle the error messages of their own requests.

2.6 Selected architectural style and patterns

This section describes decision about styles and patterns used in SafeStreets. In particular the system adopts a Client-Server architecture (see 2.6.1) and a MVC pattern (see 2.6.2).

2.6.1 Client - Server architecture

SafeStrees is built with a three-tier client-server architecture: each component (client, server, database) are physically separated in order to guarantee the robustness and reusability of them.
This layered architecture guarantee also the bottom-up development strategy: each layer can be developed separately with the right interfaces and can be also modified without compromising the other components.
The three-tier architecture is show in figure 2.11.

2.6.2 MVC : model-view-controller

SafeStreets'architecture uses a MVC design pattern to manage the components and their interaction.
This design pattern is adopted in the following way (see also figure 2.12):

- **Model**: The model is composed of the database in which all the data are stored.

- **VIew**: The view is composed of the MobileApp UI and the WebSite UI: in particular the view receives statistics, violation and suggestion (website only) lists from the controller and shows them on charts and maps.

- **Contoller**: The controller is the logic of the system, it's composed of all the server subcomponents and part of UI subcomponents that manage data.
  The controller takes in input data about violations and accidents and

insert/update them in the model, takes in input the chosen type of statistics requested by users and sends the related data to the view and also manages the computation of statistics interacting with the model.

2.7 Other design decision

This section describes decision about algorithms and data structures used in SafeStreets. In particular to manage the license plate recognition (see 2.7.1) and the statistics computation (see 2.7.2).

2.7.1 License plate recognition algorithm

Tesseract is a free OCR engine, it can recognize words of approximately every language. It offers a huge variety of API for any type of programming languages and also a C/C++ library to developers who want to build their own application on it. Tesseract is very powerful in recognizing black on white text so it's perfect for license plate recognition, and it has also an API for Android and iOS developers so it's optimized to be run on mobile devices, which make easy to SafeStreets to use it directly on devices instead of running it in the server

2.7.2 Statistics computation

To satisfy every requests of statistics by both user and authority the Server Statistics Manager has to return the desired data when needed. To manage the problem of scalability and to avoid huge number of request to the DBMS, the server computes and updates every type of statistics every day at midnight, and the day after returns the same values at each request.
To reduce also the computation complexity, the database of statistics data is built with a Tree structure, to make the search algorithm with a polynomial time complexity ($O(n)$).
The figure 2.13 shows the structure of the Tree: each node contains an attribute with the value of that statistic. To compute that value the algorithm needs only to sum the values of its child without querying the database every time.
When the computation ends, new values are stored in the structure; to visit the tree in order to get data to show in the user interfaces the system runs a breath-first-search algorithm to speed up the search (for example to get

data about violations with time granularity “year” only the second level of the tree is needed).

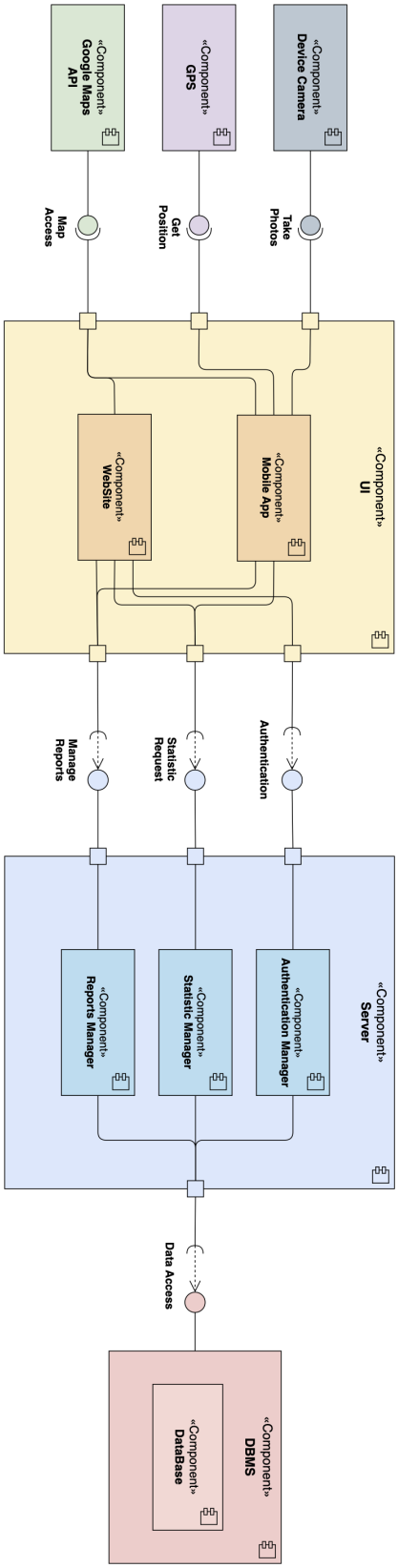


Figure 2.2: Component diagram

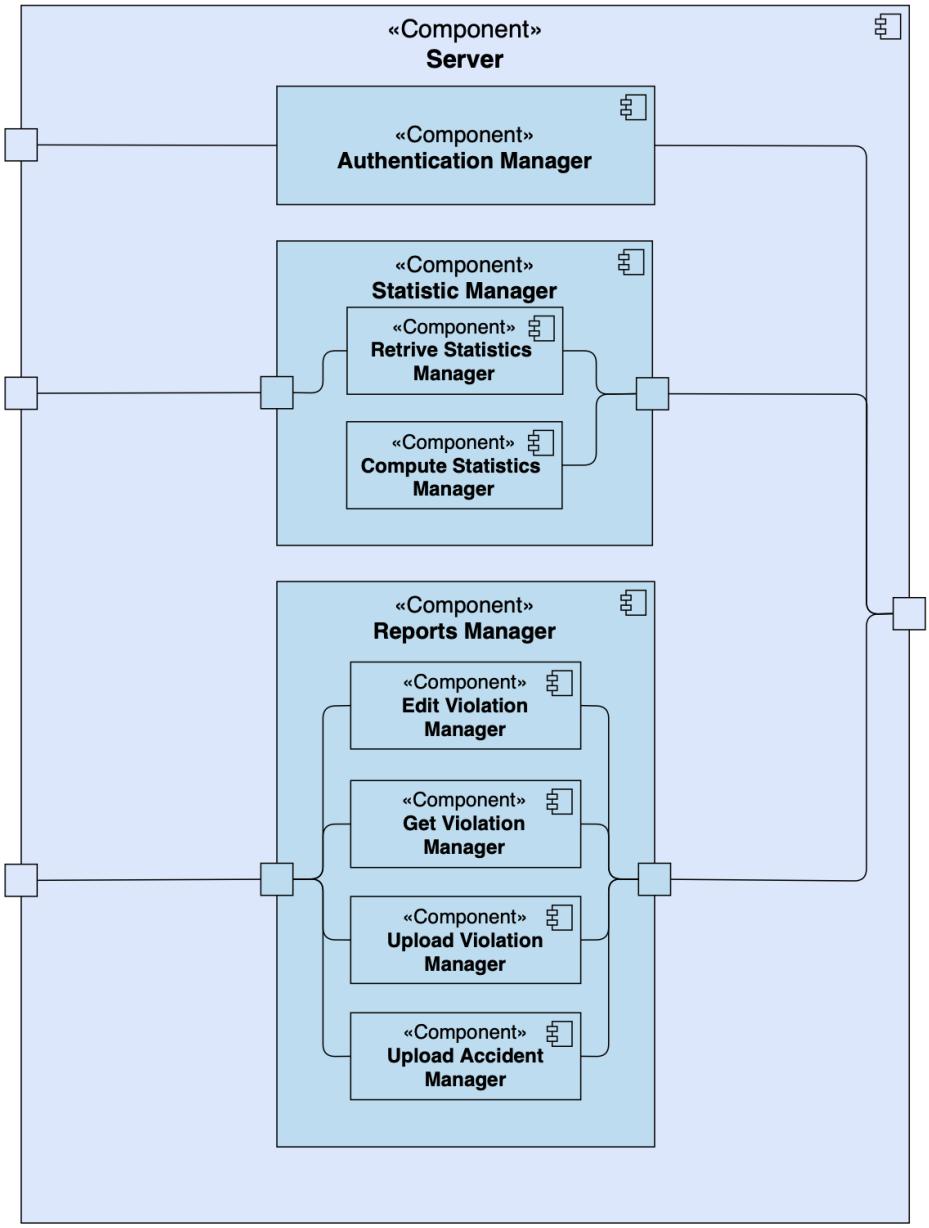


Figure 2.3: Component diagram of the Server

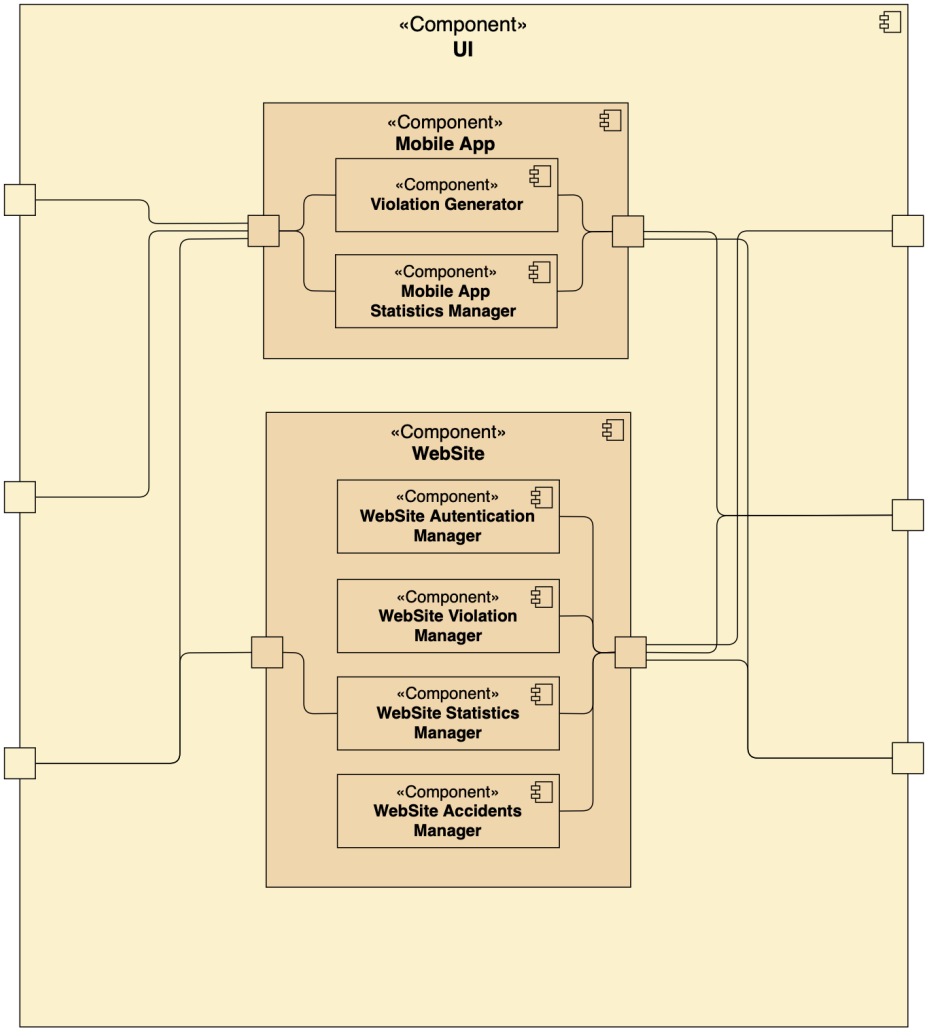


Figure 2.4: Component diagram of the UI

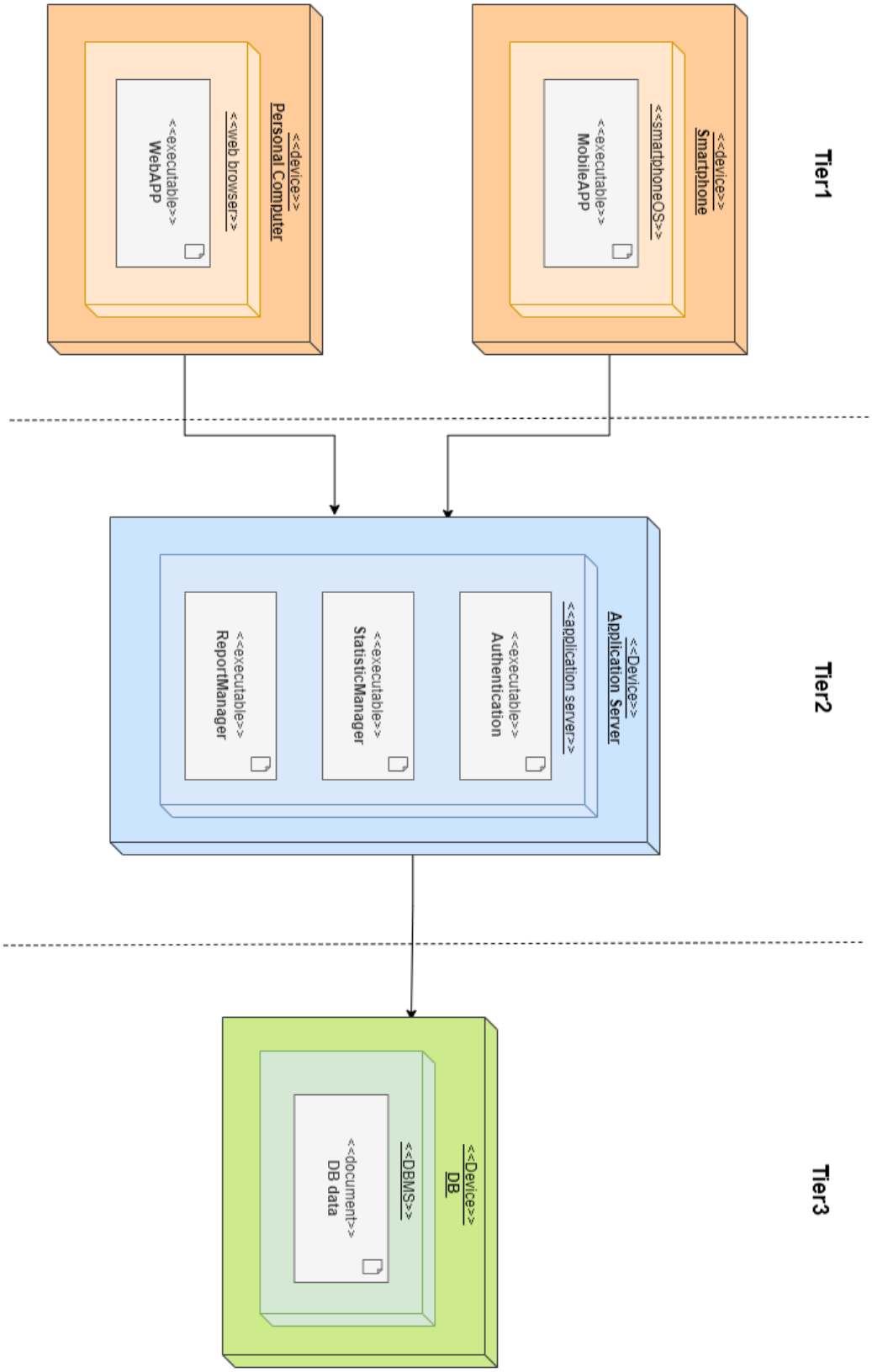


Figure 2.5: Deployment diagram

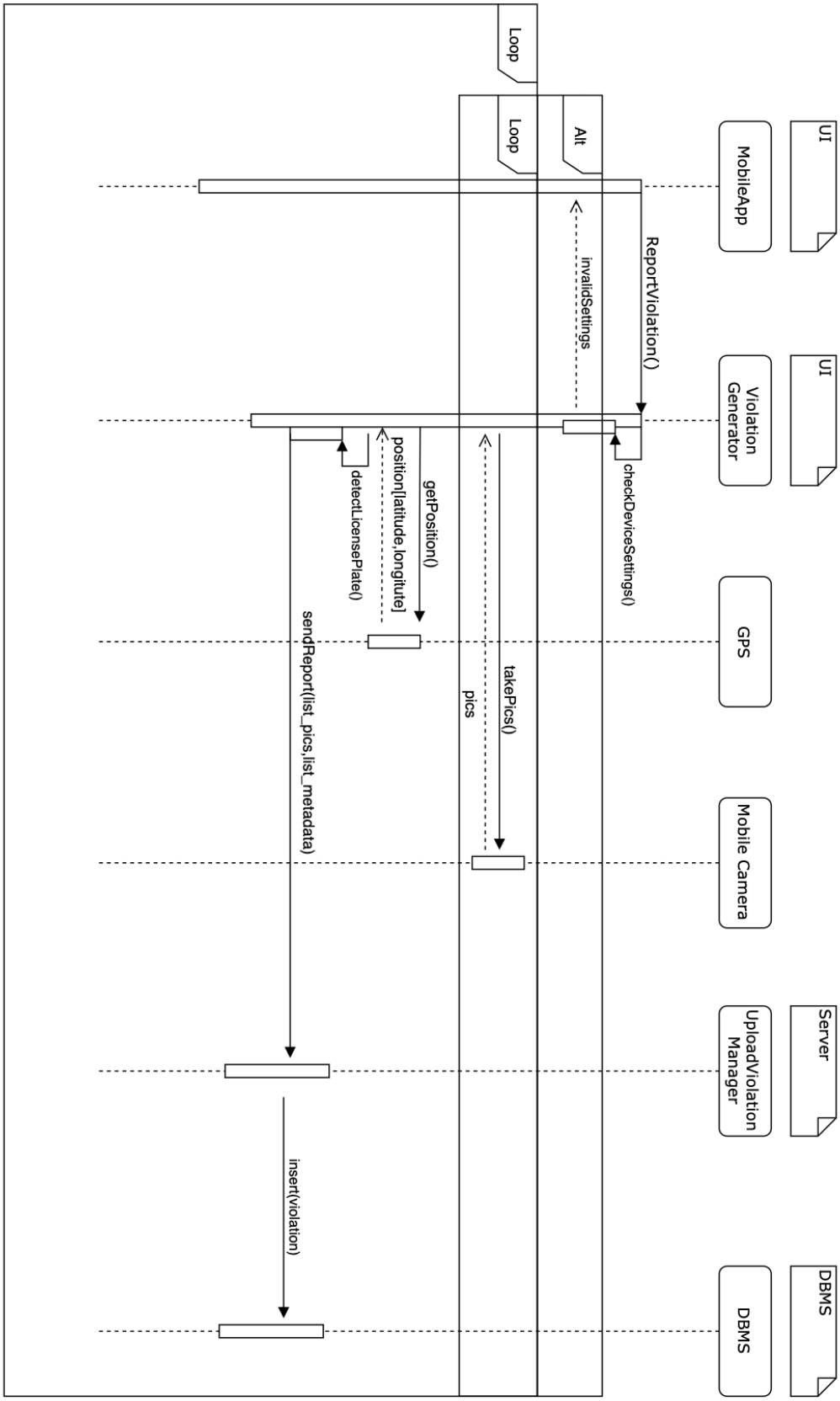


Figure 2.6: Sequence diagram of reporting a violation

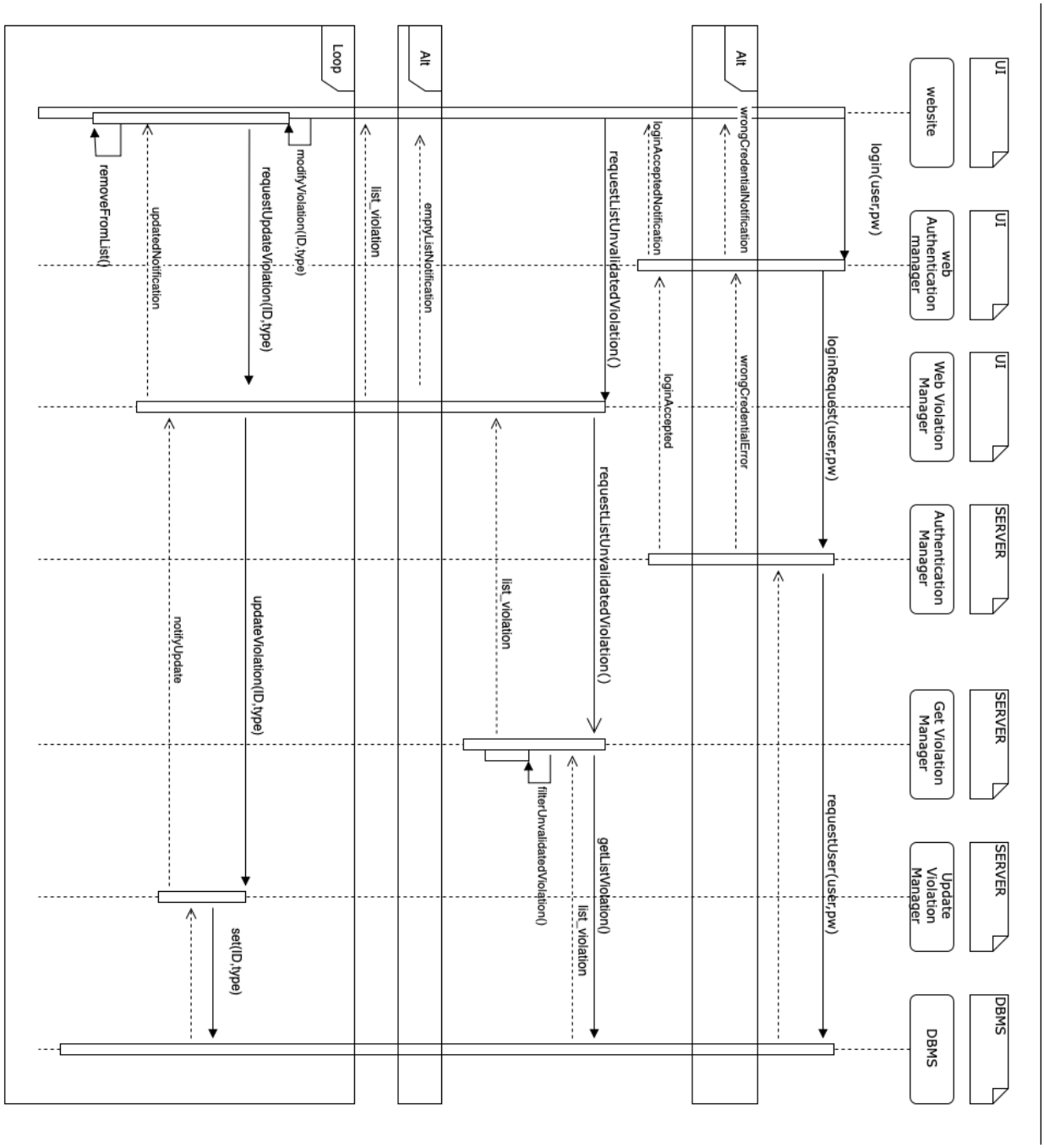


Figure 2.7: Sequence diagram of validating a violation.

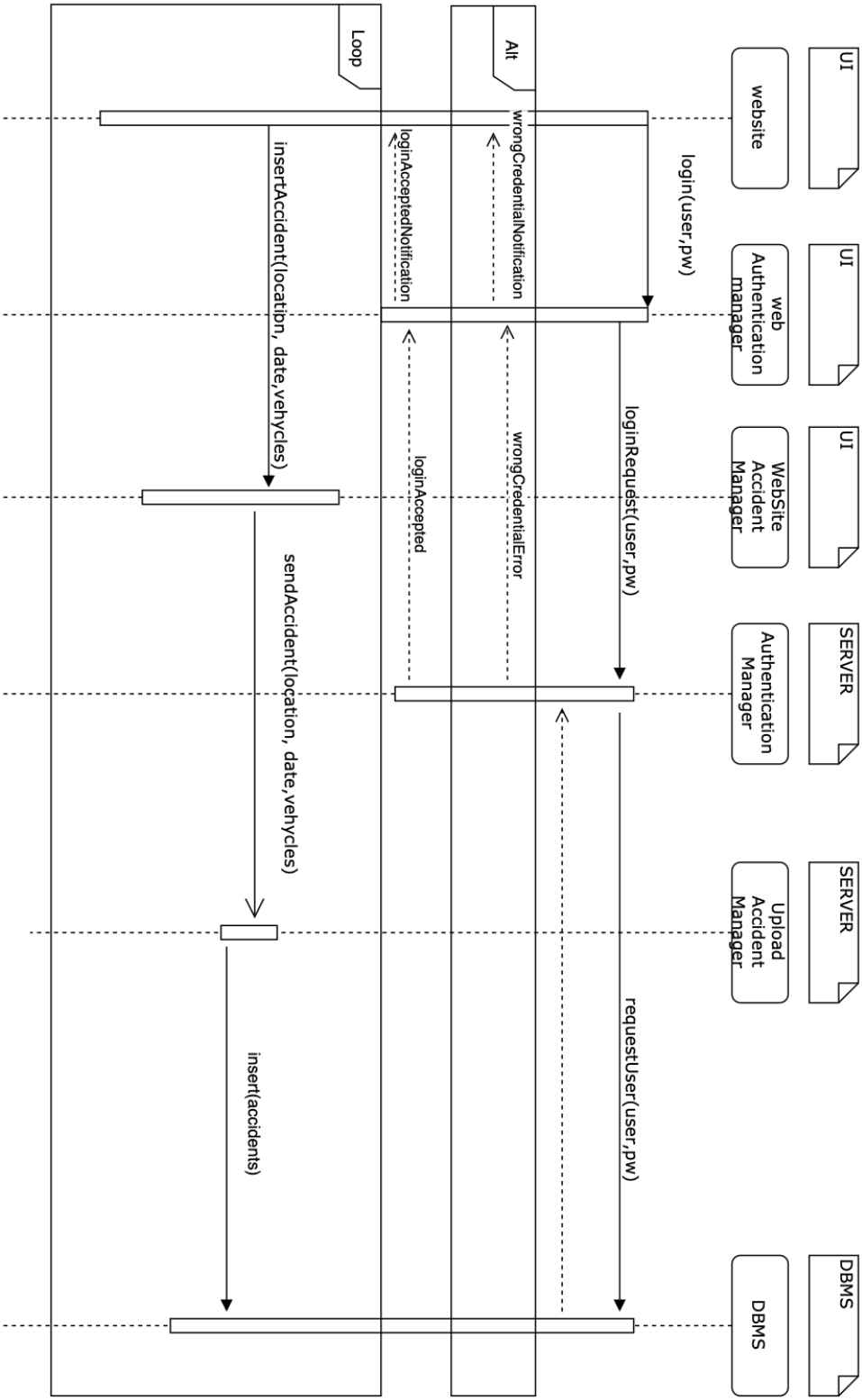


Figure 2.8: Sequence diagram of reporting an accident.

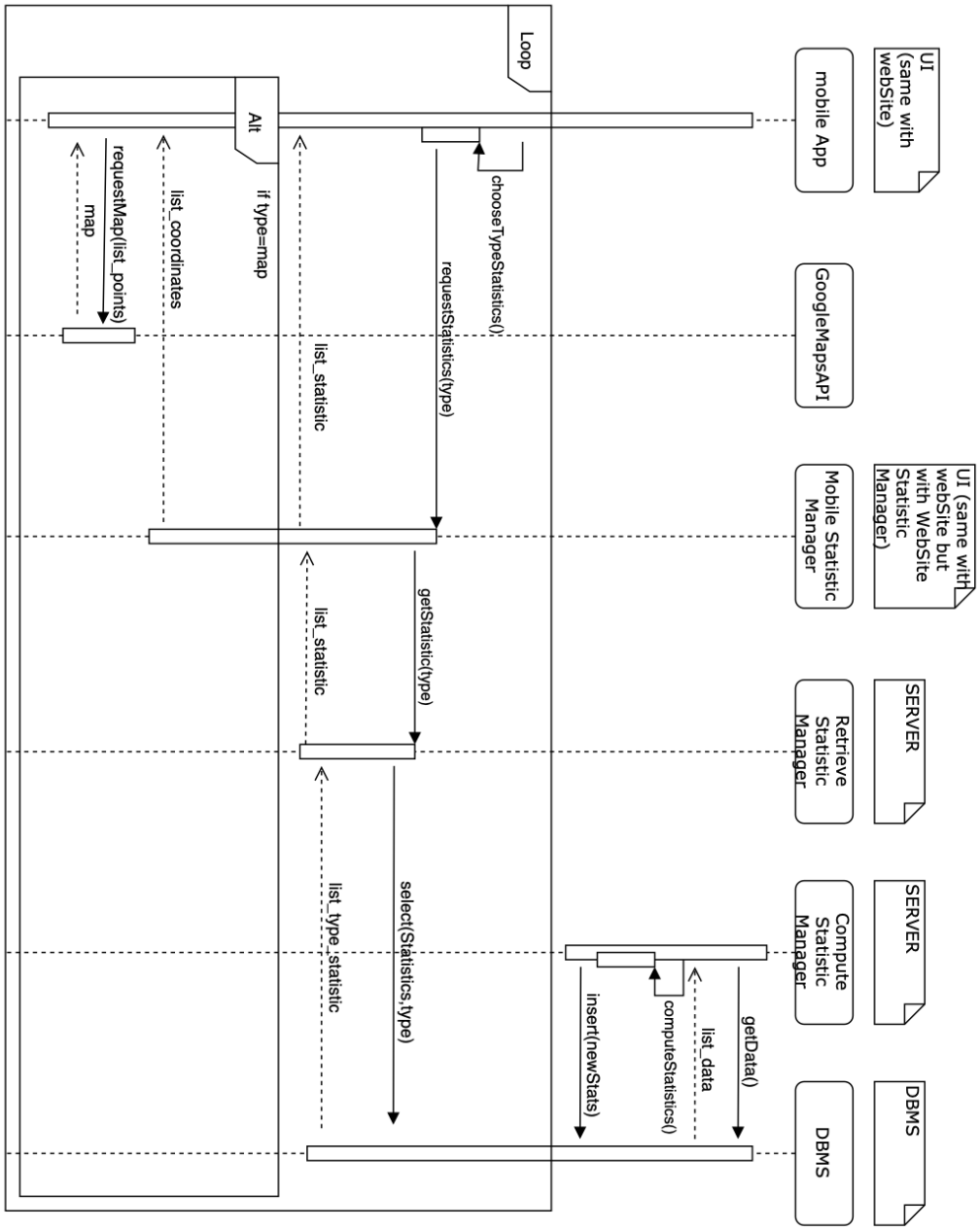


Figure 2.9: Sequence diagram of requesting statistics.

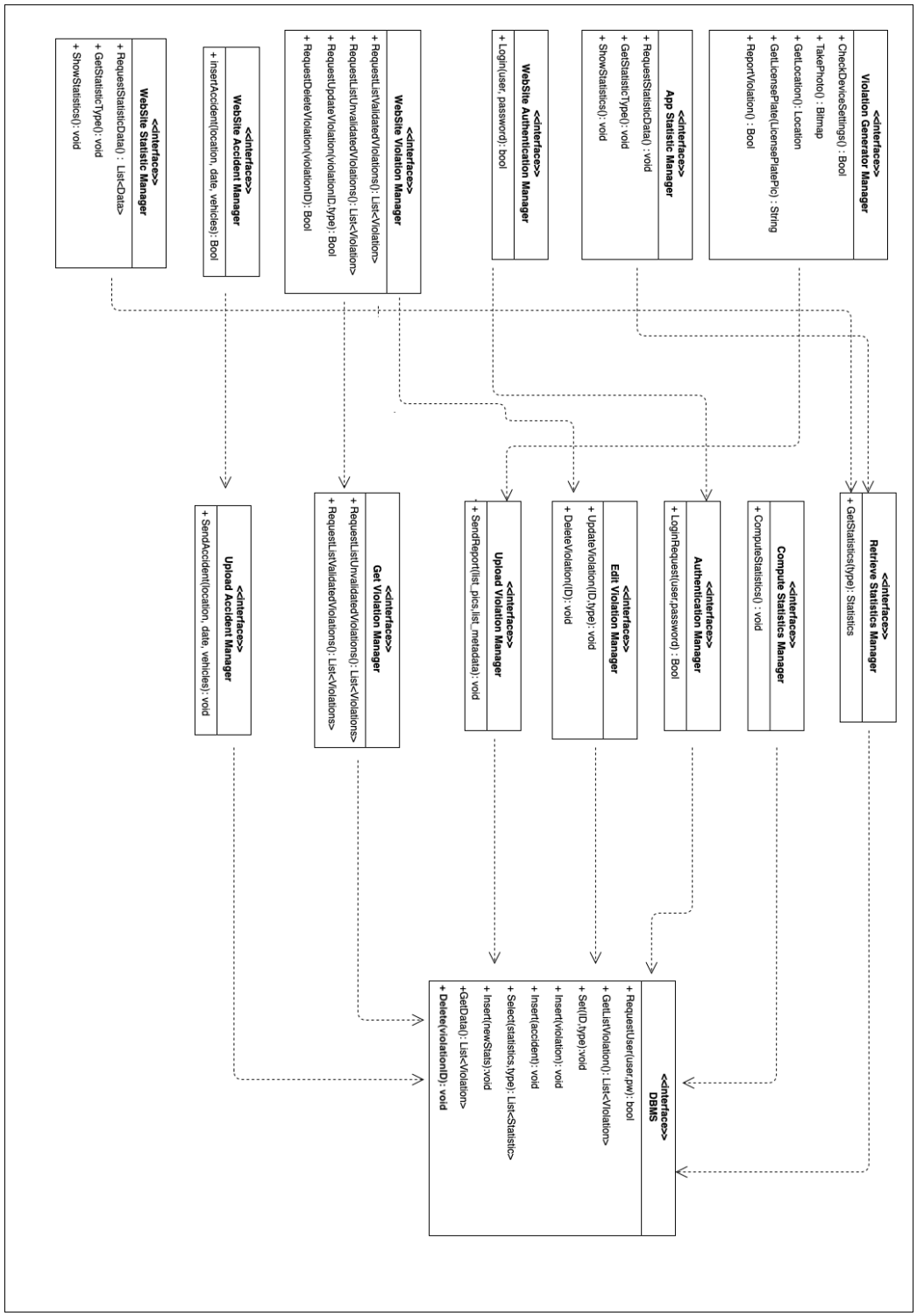


Figure 2.10: Interface diagram.

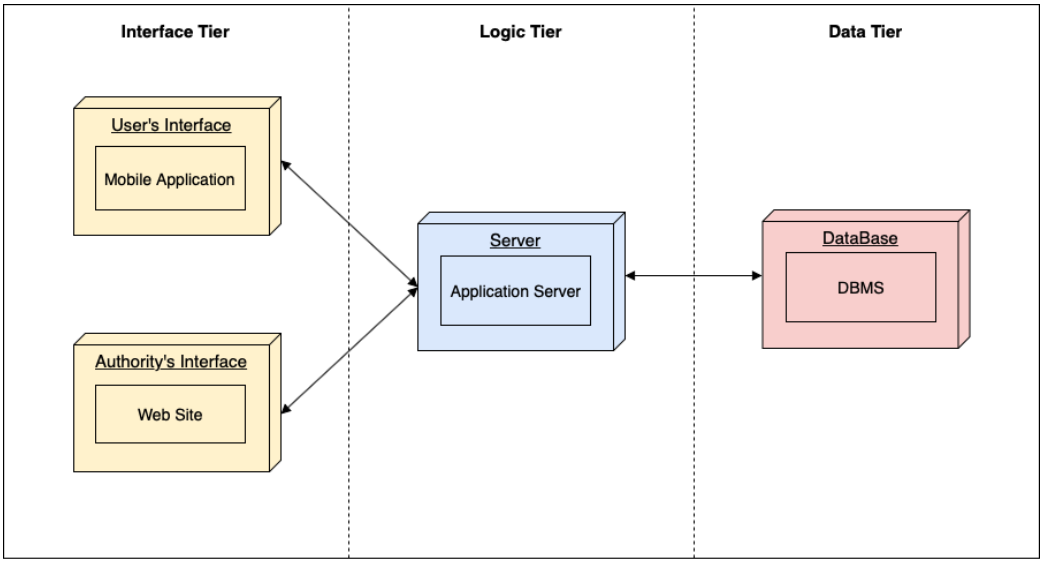


Figure 2.11: Client-server three-tier architecture.

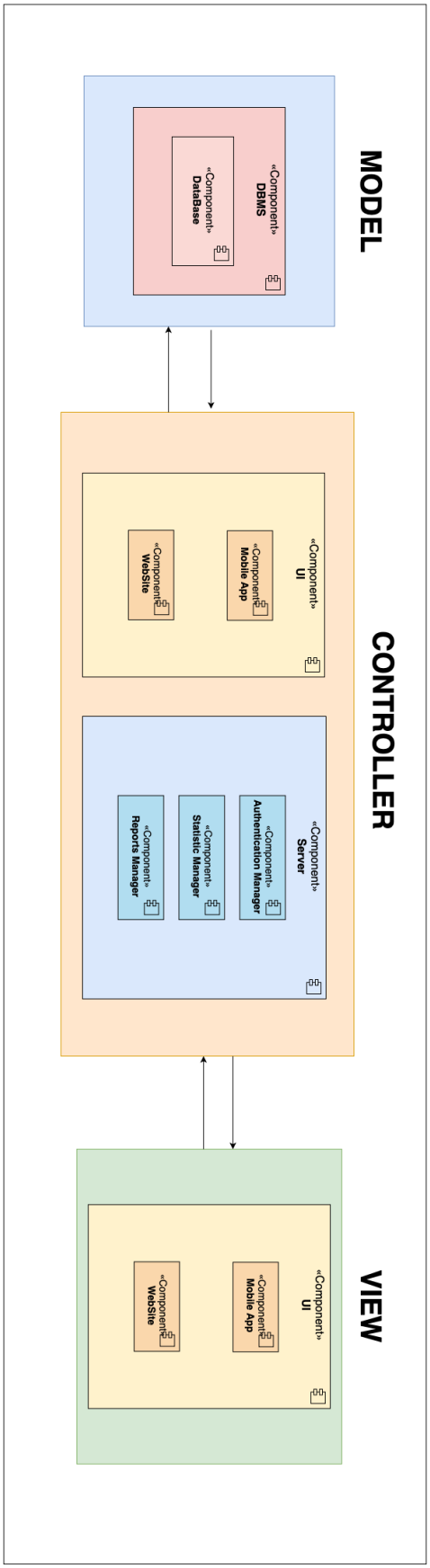


Figure 2.12: Model-view-controller diagram

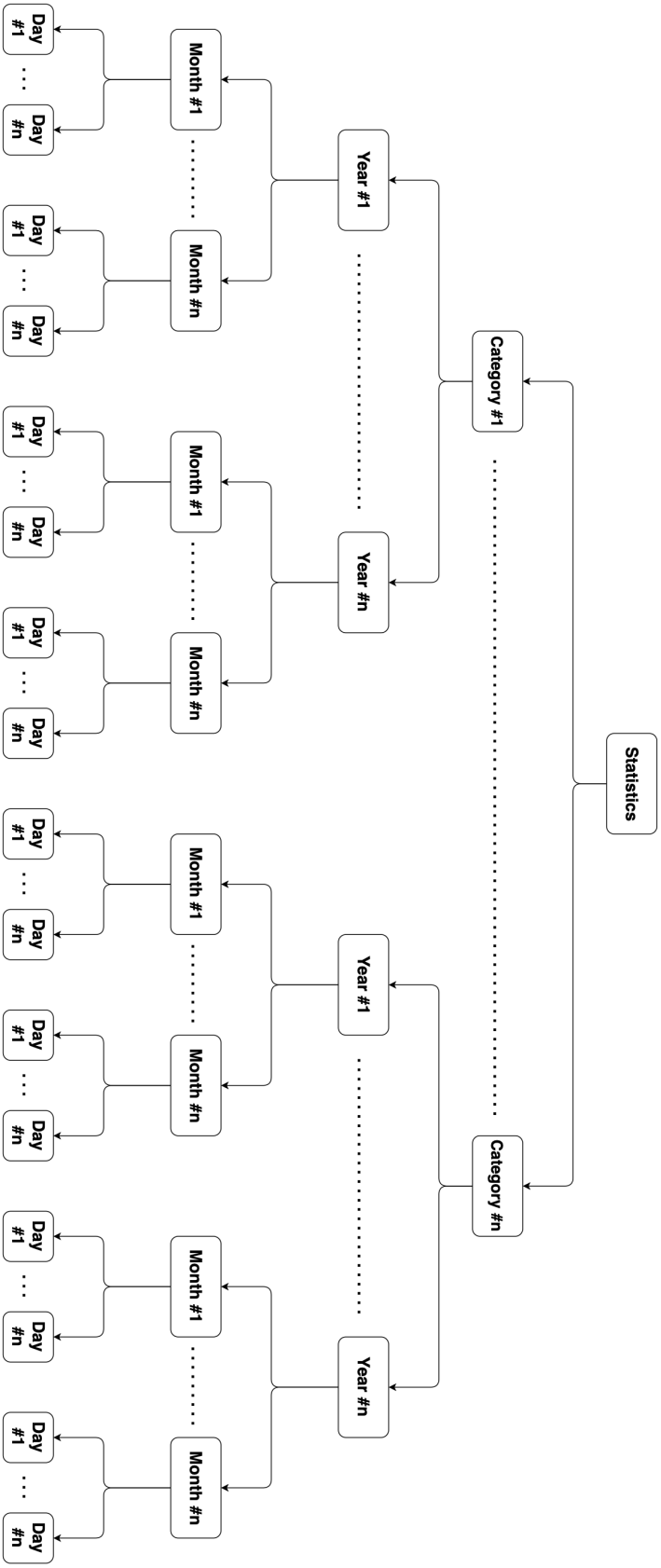
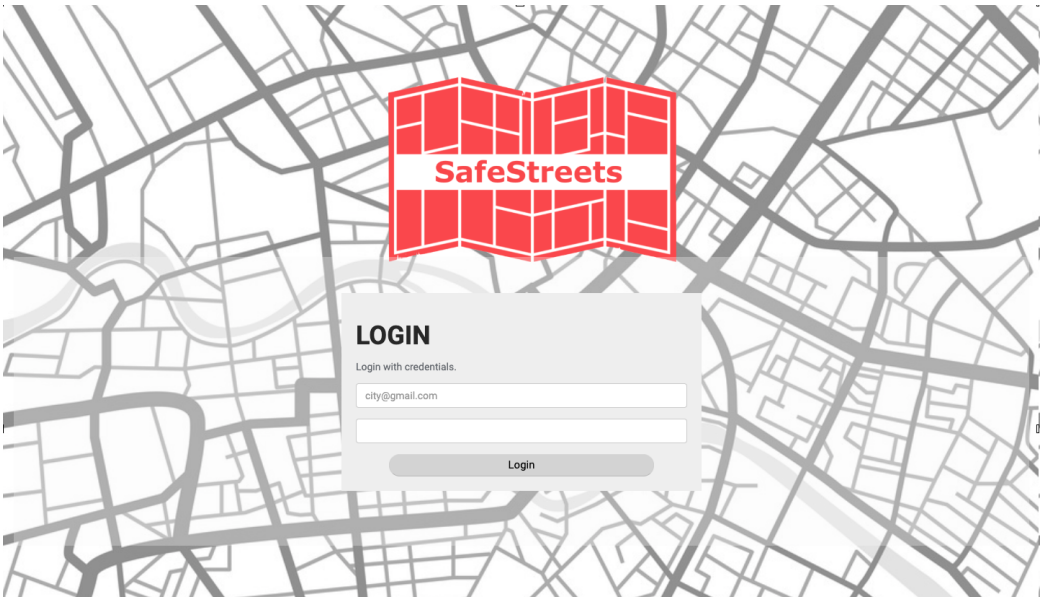


Figure 2.13: Statistics tree structure.

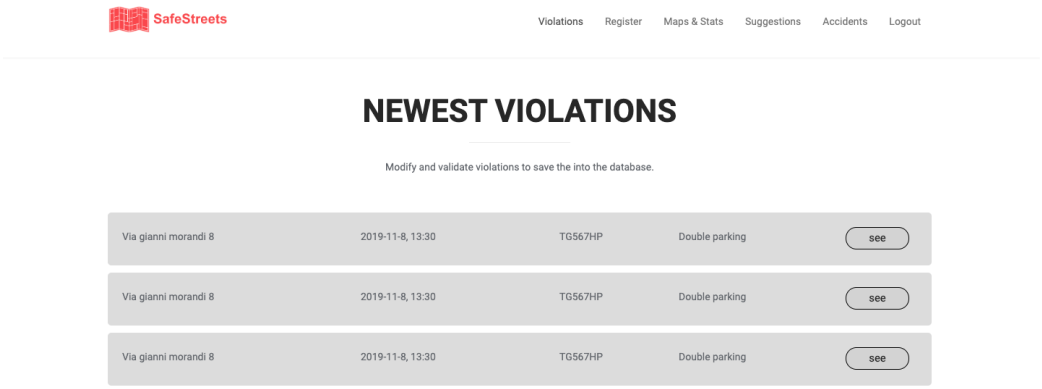
Chapter 3

User interface design

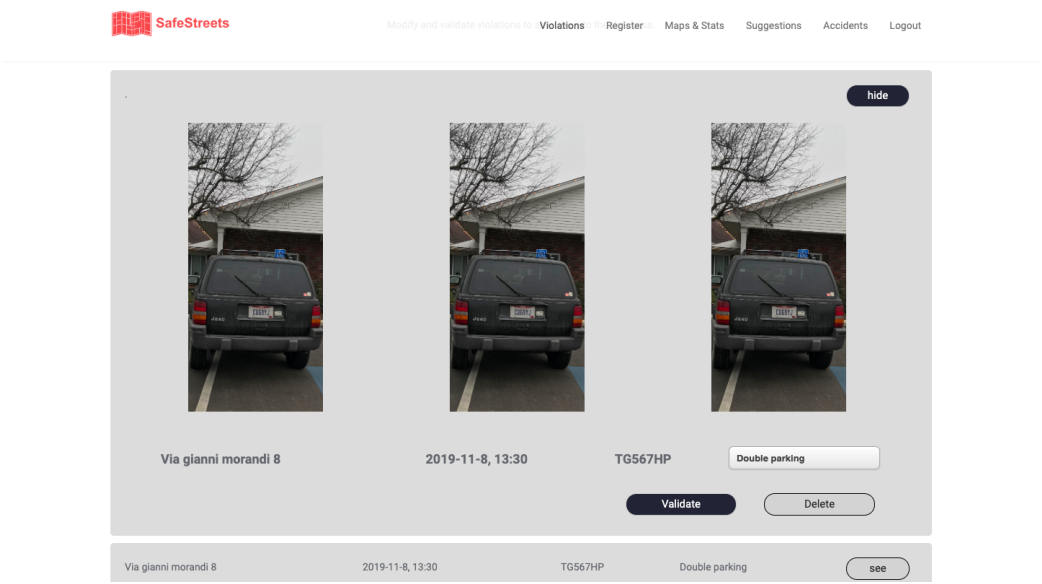
The mockups presented in the RASD document represented the main functionality of the software; the following mockups instead are more detailed and precise, the layout design is also close to the final implementation.
The UI interfaces are still made of a Mobile application and a website, the application has 3 main pages (report violation, maps, statistics), the website has instead 6 pages as shown below.



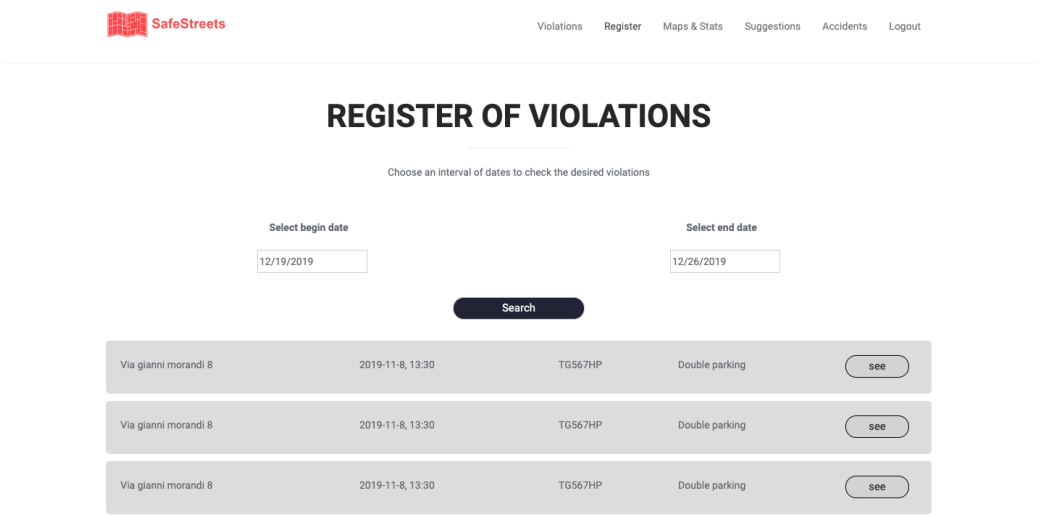
(a) Login page of Authority Website.



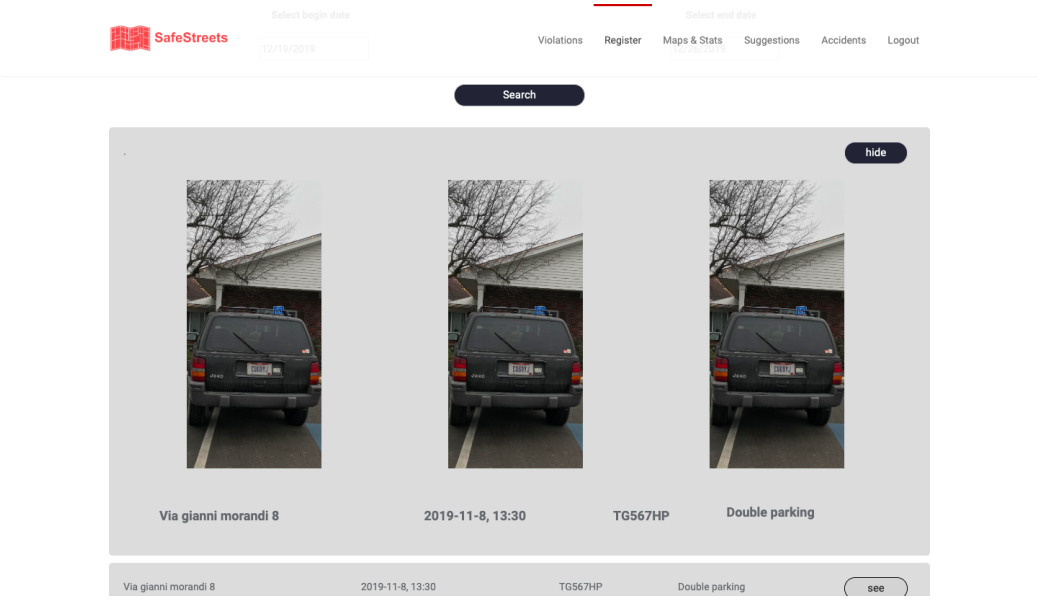
(b) Website dedicated to violation validation.



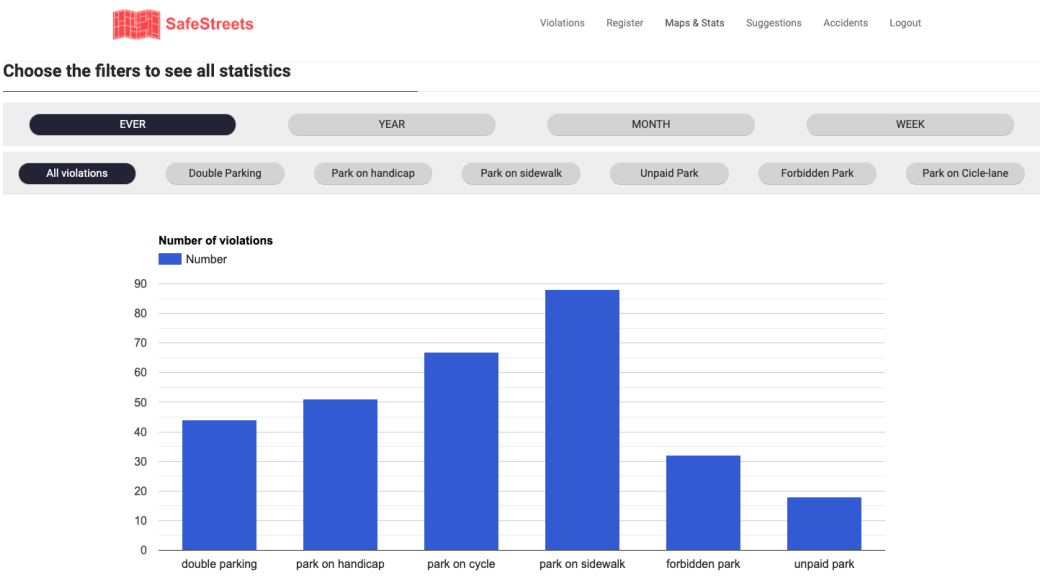
(c) Menu dedicated to violation validation.



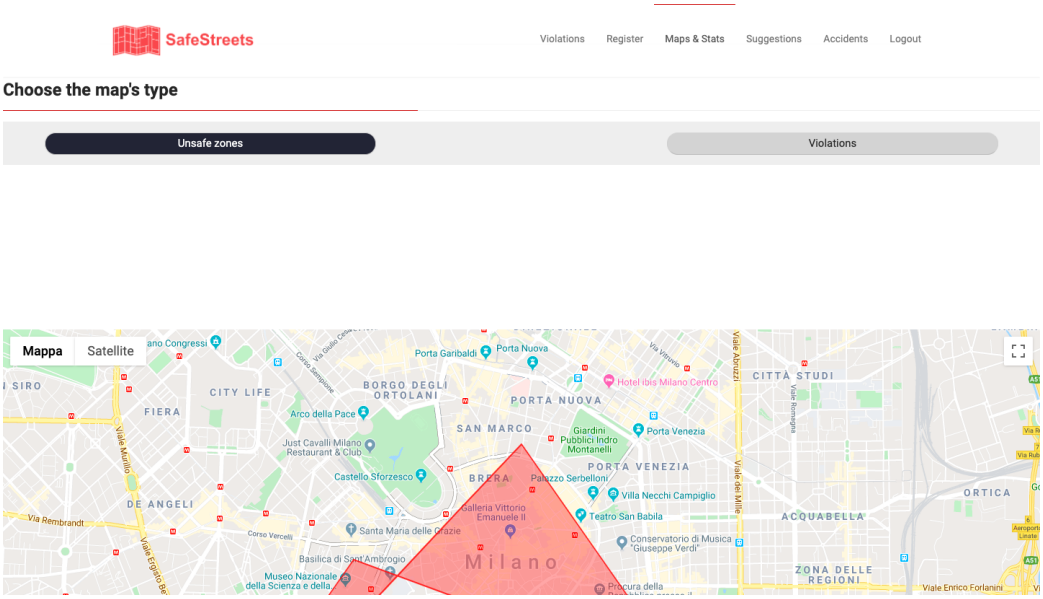
(a) Website dedicated to violation register.



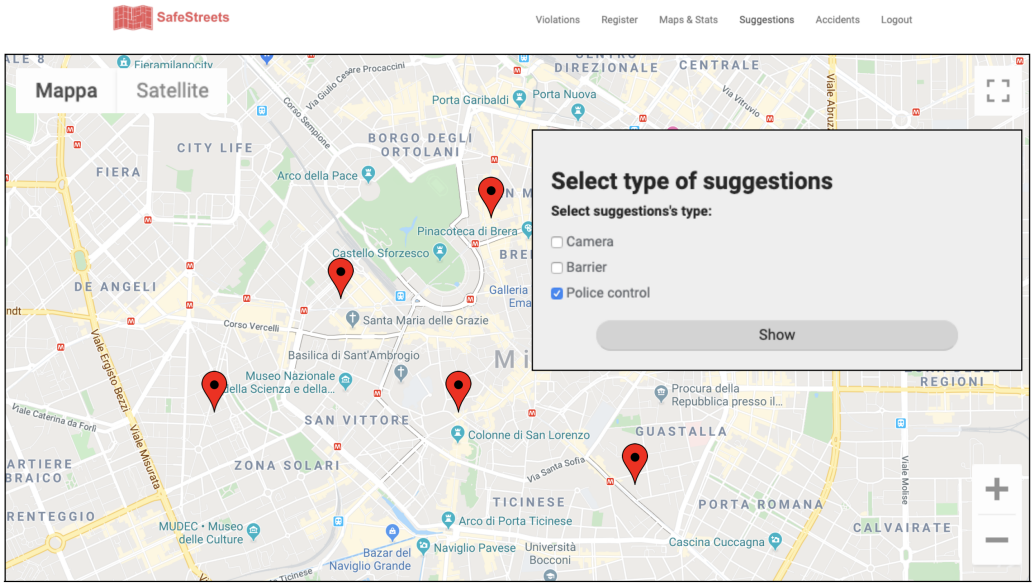
(b) Menu dedicated to violation register.



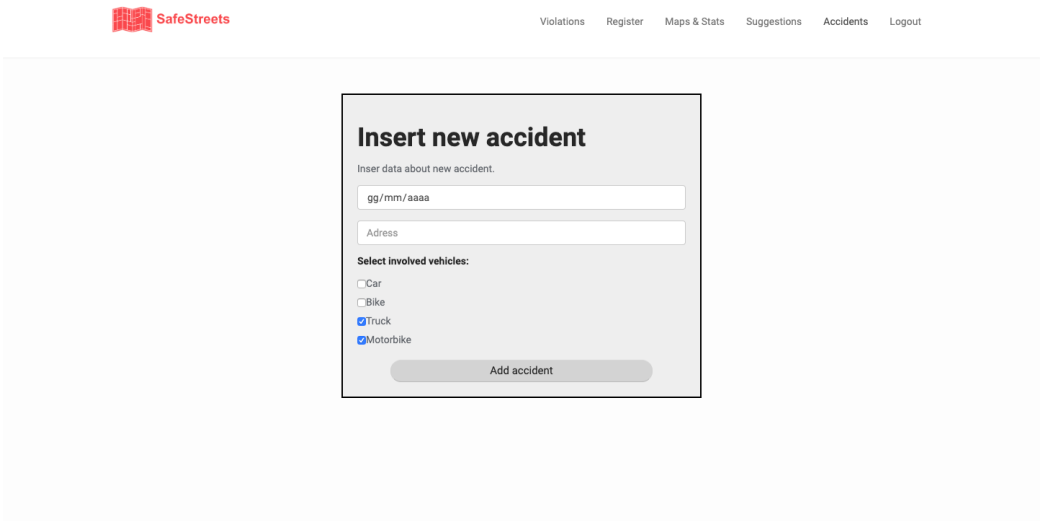
(c) Website page dedicated to statistics (charts).



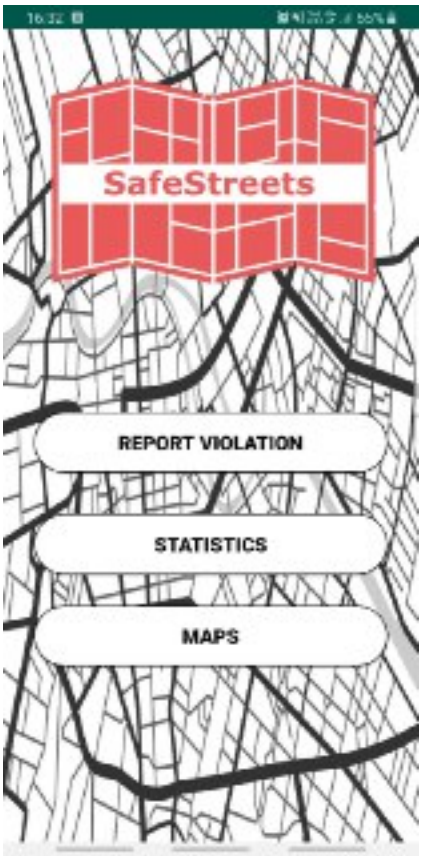
(a) Website page dedicated to statistics (maps).



(b) Website page dedicated to suggestions.

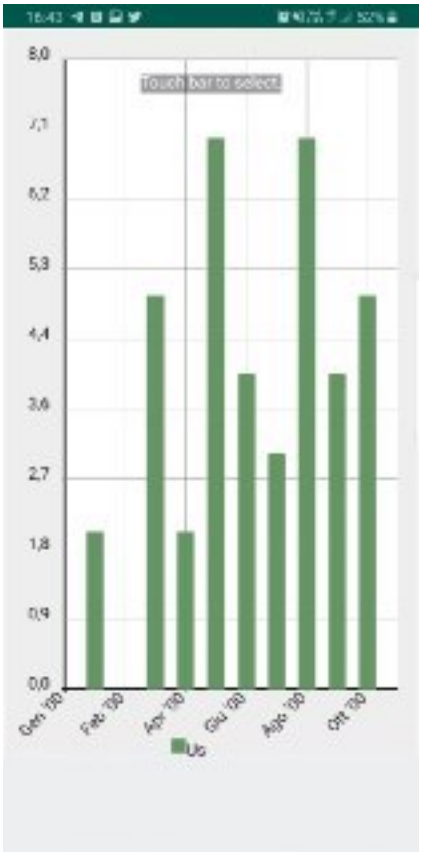


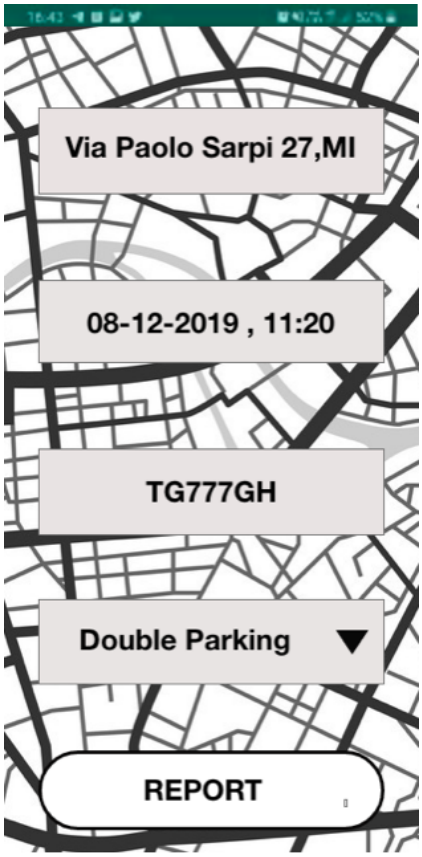
(c) Website page dedicated to accidents.



(a) User Interface of the mobile app: Menu and report violation.

(b) User Interface of the mobile app: continue report violation and map page





(a) User Interface of the mobile app:statistics page

Chapter 4

Requirements traceability

This chapter explains which of the components are used to meet the requirements defined in the RASD document (in order to achieve the prefixed goals). It contains also a general description on how each component should work and how they interact with the others.

- R_1: The system allows to take pictures only within the software itself.
  $UI \rightarrow MobileApp \rightarrow ViolationGenerator$: This component allows users to take pictures only within the Mobile Application itself, through the device's camera interface.

- R_2: The report can be made only if the internet connection is available.
  $UI \rightarrow MobileApp \rightarrow ViolationGenerator$: Before making a report, this component checks if the device's internet connection is available and works properly. Otherwise it prevents the user from proceeding with the report.

- R_3: The system imposes the user to include a picture of the license plate of the vehicle that committed the violation.
  $UI \rightarrow MobileApp \rightarrow ViolationGenerator$: During the process of taking pictures this component ensures that at least one of them contains the license plate of the vehicle involved in the violation.

- R_4: The system gets date and time from its internal clock when the picture is taken.
  $UI \rightarrow MobileApp \rightarrow ViolationGenerator$: when the first picture is taken, a request is sent to the server component to obtain the current date and time from the system'clock.
  $Server \rightarrow ReportsManager \rightarrow UploadViolationManager$: It receives a request from the UI and responds with its current date and time.

- R_5: The system obtains the violation's position from the device's GPS.
  $UI \rightarrow MobileApp \rightarrow ViolationGenerator$: When the first picture is taken, this component retrieves the geographical positioning of the user's mobile device from its GPS.

- R_6: The system converts the geographic coordinates obtained in the corresponding address.
  $UI \rightarrow MobileApp \rightarrow ViolationGenerator$: Before sending a report to the system this component converts the geographical positioning of the user's mobile device in the corresponding address. Then, it integrates it in the reporting information.

- R_7: The system automatically detects the vehicle's license plate number.
  $UI \rightarrow MobileApp \rightarrow ViolationGenerator$: When the license plate picture is taken this component launches an algorithm to extract the license plate number. If for any reason an error occurs, it will impose the user to take the picture again.

- R_8: Access to the violation reports is only permitted to authorities.
  $UI \rightarrow WebSite \rightarrow WebsiteViolationManager$: This component allows authorities to access to the Violation's Information. In particular, it sends a request to the server component and awaits for its response.
  $Server \rightarrow ReportsManager \rightarrow GetViolationManager$: This component receives the request to access the violations from the UI and queries the database in order to get them. Then, it will send a response to the UI containing the information requested.

- R_9: The system stores violations data.
  $UI \rightarrow MobileApp \rightarrow ViolationGenerator$: If a report has been formulated correctly, this component sends a request to the server containing the report information in order to upload them.
  $Server \rightarrow ReportsManager \rightarrow UploadViolationManager$: This component receives the request to store the violation from the UI and write it into the database.

- R_{10}: Authorities must log in to the system.
  $UI \rightarrow WebSite \rightarrow WebsiteAuthenticationManager$: This component allows authorities to insert the credentials in order to access to the Website's services. Then, it will send a request to the server and awaits for its response.
  $Server \rightarrow AuthenticationManager$: It collects the requests coming from the Website UI containing the credentials and checks for their correctness. Then, it sends back a positive or negative response.

- R_{11}: The system checks that a fake GPS service is not active on the user's device.
  $UI \rightarrow MobileApp \rightarrow ViolationGenerator$: Before making a report, this component checks if the user has not a fake GPS service activated. Otherwise it prevents the user from proceeding with the report.

- R_{12}: Authorities can modify, discard or validate the violation reports present on the system.
  $UI \rightarrow WebSite \rightarrow WebsiteViolationManager$: This component allows authorities to edit the information of a violation with an appropriate interface. Then, it sends a request to the server component containing the information to update.
  $Server \rightarrow ReportsManager \rightarrow EditViolationManager$: It collects the requests coming from the UI component and uploads the information stored into the database.

- R_{13}: Authorities can insert data concerning accidents that occur on the territory.
  $UI \rightarrow WebSite \rightarrow WebsiteAccidentsManager$: This component provides an interface through which authorities can insert the information on an accident that occurred on the territory. Then, it sends a request to the server containing the information to upload.
  $Server \rightarrow ReportsManager \rightarrow UploadAccidentManager$: It collects the requests coming from the UI interface concerning the insertion of a new accident and writes in the database.

- R_{14}: The system builds statistics based on validated data.
  $Server \rightarrow StatisticManager \rightarrow ComputeStatisticsManager$: This component queries the database in order to get all the validated violations and computes statistics on it. Then, it uploads them into the database.

- R_{15}: The system produces statistics concerning possible unsafe areas built on the accidents information.
  $Server \rightarrow StatisticManager \rightarrow ComputeStatisticsManager$: This component queries the database in order to get the accidents uploaded by authorities and computes ‚Äúunsafe areas‚Äù statistics. Then, it uploads them into the database.

- R_{16}: Authorities and users can set the temporal granularity and the category of the statistics that they want to consult.
  $UI \rightarrow Website \rightarrow Websitestatisticsmanager$: It provides the tools through which authorities can filter the statistics they are interested

in. Then, it sends a request to the server containing the information to retrieve and awaits for its response.
$UI \rightarrow MobileApp \rightarrow MobileAppStatisticsManager$: It provides the tools through which users can filter the statistics they are interested in. Then, it sends a request to the server containing the information to retrieve and awaits for its response.
$Server \rightarrow StatisticManager \rightarrow RetrieveStatisticsManager$: This component collects the requests concerning the statistics to retrieve coming from the UI component and queries the database in order to get them. Then, it sends back a response containing the information requested.

- R_{17}: Authorities can consult the interventions suggested by the system.
  $UI \rightarrow WebSite \rightarrow WebsiteStatisticsManager$: It allows authorities to send a request to the server component in order to retrieve the information concerning the intervention suggestions built by the system. Once a response is returned it display the information in a correct way.
  $Server \rightarrow StatisticManager \rightarrow RetrieveStatisticsManager$: This component collects the requests coming from the UI component concerning the intervention suggestion and queries the database in order to get them. Then, it sends back a response containing them.

- R_{18}: The system has an associated suggestion for each type of violation.
  $Server \rightarrow StatisticManager \rightarrow ComputeStatisticsManager$: This component check if the number of each violation overcome some threshold and suggest an intervention related to that violation.

- R_{19}: The system places a marker on the map where the intervention is suggested.
  $UI \rightarrow WebSite \rightarrow WebsiteStatisticsManager$: This component place a marker on the map in the zone with high number of violation labelled with the type of suggestion chosen.
  $Server \rightarrow StatisticManager \rightarrow RetrieveStatisticManager$

- R_{20}: The system allows to filter the suggestions on the map by category.
  $UI \rightarrow WebSite \rightarrow WebsiteStatisticsManager$: It allows authorities to set the suggestion category to retrieve. Then, it sends a request to the server component in order to retrieve the information and awaits for a response.
  $Server \rightarrow StatisticManager \rightarrow RetrieveStatisticManager$: This component collects the requests coming from the UI component concerning

the intervention suggestion category and queries the database in order to get them. Then, it sends back a response containing them.

Chapter 5

Implementation, integration and test planning

The main guidelines of the software system implementation, integration and testing are listed below.
The system, as previously described, is composed by several subsystems and by a client-server architecture:

- MobileApp
- Website
- Authentication Manager
- Statistic Manager
- Reports Manager

The implementation flow mainly follows a bottom-up strategy, which means that all components will be firstly implemented and tested and after integrated and re-tested in their own subsystem. Once the first phase ended, all the subsystems will be in turn integrated and tested up to the final product.
Moreover the implementation process must take into account the most critical parts of the system, which in this case concern the violation reporting logic into the MobileApp and the authority monitoring and validation into the Website.
Building statistics is not complex as the reporting logic, moreover it relies on data of violations (by Users), accidents (by Authorities) and violations validation by Authorities; therefore statistics are less relevant than the reporting process implementation.
User experience is a key factor, why it is thought to make the reporting

process as simple as possible to promote SafeStreet from the beginning. Therefore it is required to develop the user interfaces as soon as possible to get feedbacks and consequently improve them.
To sum up the main tasks and their respective priorities:

- Statistics: medium.
- Website UI and MobileApp UI: medium.
- "Violation monitoring and validation" and "violation reporting logic": high.
- Implementation strategy: mainly bottom up.

Knowing all these, the tasks are paralleled and assigned to two different teams:
Team1 → Task 1 and 2.
Team2 → Task 3.

5.1 Team 1

Here the flow of implementation for team1:
MobileApp and Website UI: First of all the implementation is focused on the two applications front end with all the necessary logic to get feedbacks on practical use, so it will be possible to improve them during all the implementation flow. For this purpose also communication interfaces between MobileApp and Website with respect to the Server and external APIs must be implemented:

- Manage Reports «i».
- Statistic Request «i».
- Authentication «i».
- Device camera «i».
- GPS«i».
- Maps API«i».

Finally also the Authentication Manager on the server side for the authority credentials verification is implemented:

- Authentication Manager (AM).

Once this first part is completed is possible to turn to Statistics.
Statistic:
Statistics compose of several components both on Server and UI side: Server:

- Retrive Statistics Manager (RSM).
- Compute Statistics Manager (CSM).

UI:

- Mobile app Statistics Manager (MSM).
- Website Statistics Manager (WSM).

The most important statistics component is CMS, its task is to daily compute all the statistics and store them on the DB. The reason why is to increase the access speed to statistics and not to compute them at each requests.
So if the CSM is the first component realized, the second is the RSM to complete statistics on server side. Once the server side is completed is possible to proceed with the UI side and develop the MSM and WSM components. The first one is the WSM, why the WebSite component is composed by more components than the MobileApp, therefore it might produce more troubles during integration and testing.
Once it is finished and tested, it is possible to proceed with MSM.

5.2 Team 2

Team2 must deal with both UI and Server logic: it must implement all that concerns making and reporting violations process and their validation and making and reporting accidents process.
Components to be implemented:
UI:

- Violation Generator (VG).
- WebSite Violation Manager (WVM).
- WebSite Authentication Manager (WAM).
- WebSite Accidents Manager (WAccM).

Server:

- Get Violation Manager (GVM).

- Upload Violation Manager (UVM).
- Upload Accident Manager (UAM).
- Edit Violation Manager (EVM).

The flow of implementation starts with the Reports Manager component on server side, why potentially it might be the most problematic, due to the high number of components. Therefore EVM, GVM, UVM and UAM are implemented and unit tested and once finished they can be integrated and retested.
Then the WebSite internal components are implemented, which means: WVM, WAM and WAccM. At this point of the implementation team1 should have finished the WebSite Statistics Manager and so it is possible to integrate and retest all the UI WebSite subcomponents.
Finally VG into the MobileApp component is implemented and after integrated with the Mobile Statistics Manager realized by the team1.

5.3 Flow of Integration

Here is listed the integration flow according to the planned implementation flow described above, which means that the integration is performed following the chronological order in which the components are implemented. So the integration will take place only once all the sub-components of a macro-components are implemented and unit tested. Obviously at the end of each step of integration macro-components will be retested and if necessary fixed.
Statistic Manager
Statistic Manager is completely implemented, integrated and tested by team1 (see figure 5.1).
Reports Manager

Figure 5.1: Integration of Statistic Manager diagram.

ReportsManager is completely implemented, integrated and tested by team2 (see figure 5.2).
Server

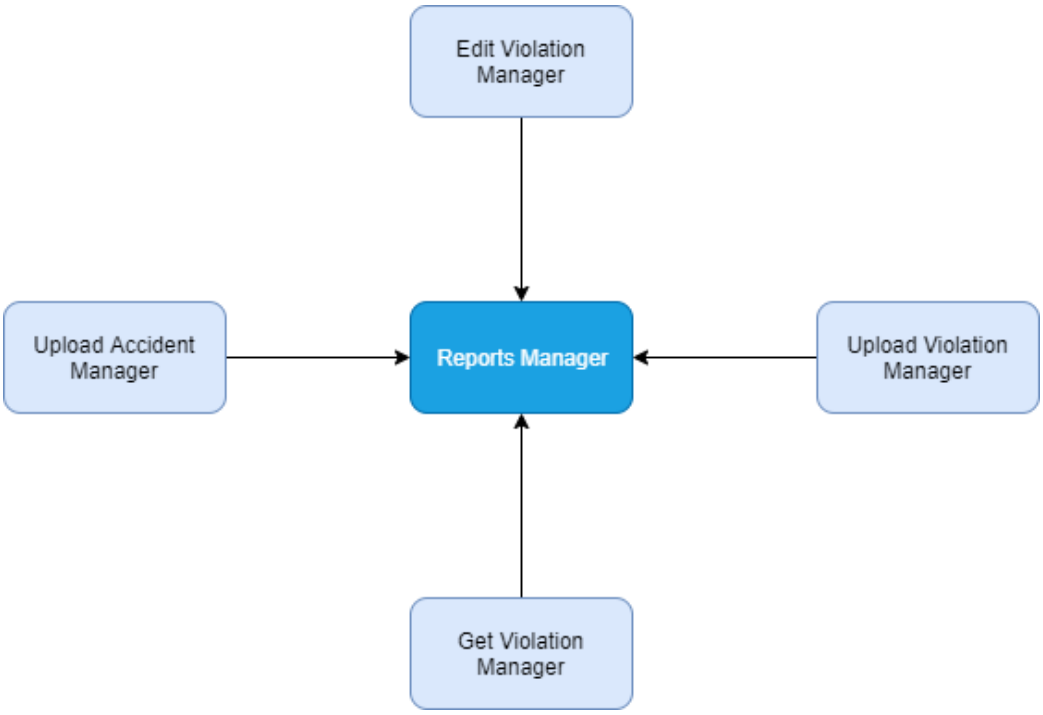


Figure 5.2: Integration of Reports Manager diagram.

The Server macro-component is integrated and tested only once all the sub-components are completed (see figure 5.3). It requires that:

- Team1 implements and tests Statistic Manager, Authentication Manager and Interfaces.
- Team 2 implements and tests Reports Manager.

WebSite
The WebSite component is integrated and tested only once all the sub-components are completed (see figure 5.4). It requires that:

- Team1 implements and tests WebSite Statistics Manager.
- Team 2 implements and tests WebSite Authentication Manager, WebSite Accident Manager and WebSite ViolationManager.

MobileApp
The MobileApp component is integrated and tested only once all the sub-components are completed (see figure 5.5). It requires that:

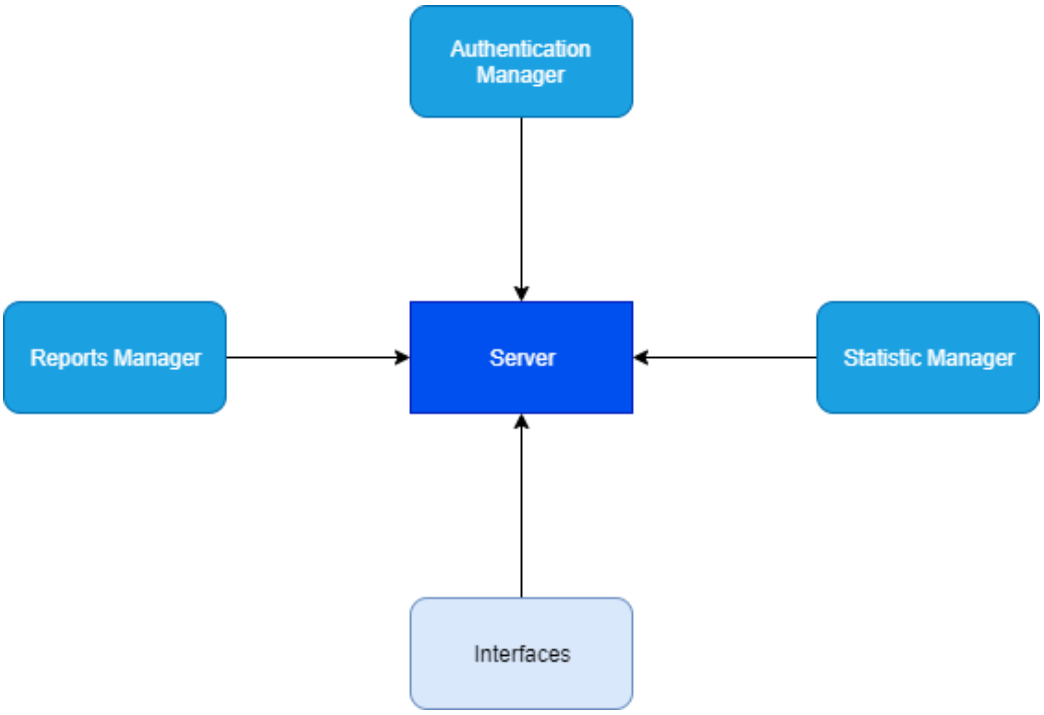


Figure 5.3: Integration of the Server diagram.

- Team1 implements and tests MobileApp Statistics Manager.
- Team 2 implements and tests VIolation Generator.

UI: The UI macro-component is integrated and tested only once all the sub-components are completed (see figure 5.6). It requires that the Website and MobileApp component are correctly tested and implemented and the External API are successfully integrated into the system (they include Maps API, Device Camera API and GPS API).

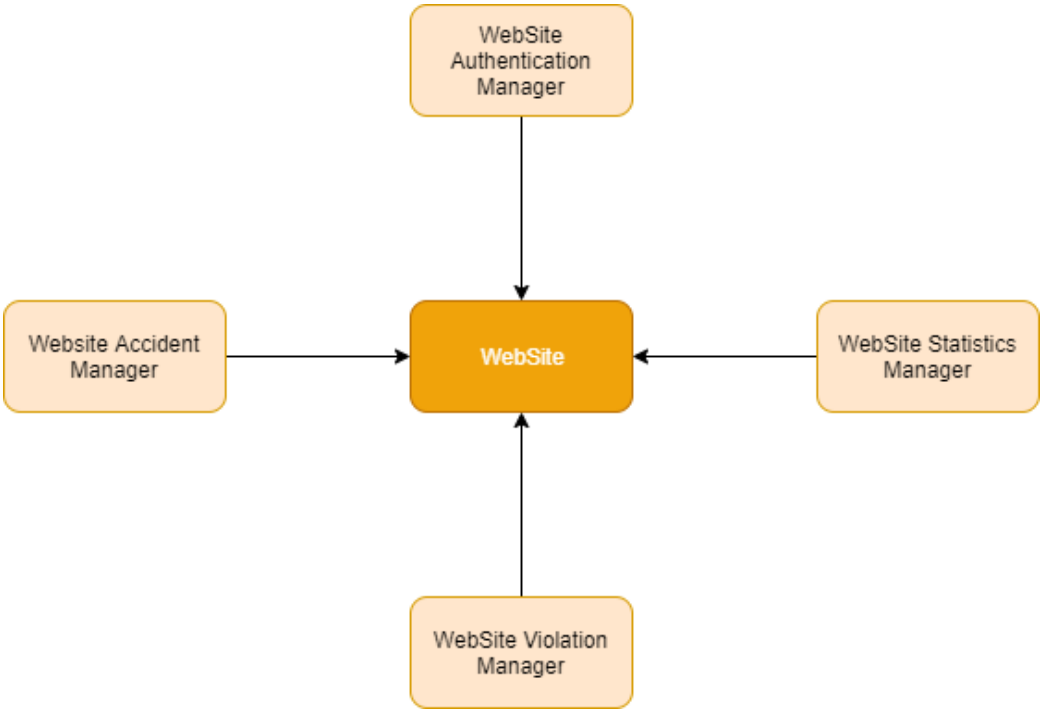


Figure 5.4: Integration of the WebSite diagram.

Figure 5.5: Integration of the MobileApp diagram..

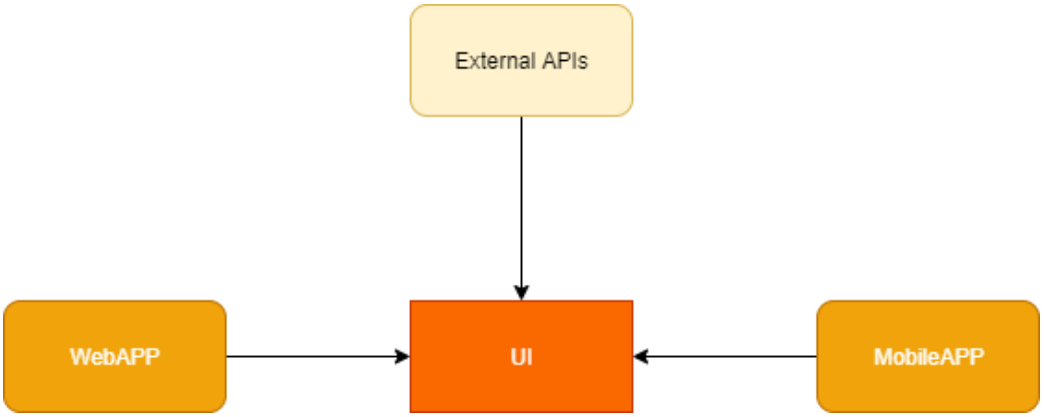


Figure 5.6: Integration of the UI diagram.

Chapter 6

Effort spent

Mattia Micheloni

TASK	HOURS
Ch.1	2
Ch.2	11
Ch.3	3
Ch.4	4
Ch.5	19

Francesco Montanaro

TASK	HOURS
Ch.1	0.5
Ch.2	20
Ch.3	7
Ch.4	6
Ch.4	9

Amedeo Pachera

TASK	HOURS
Ch.1	1
Ch.2	16
Ch.3	10
Ch.4	2
Ch.4	10