POLITECNICO
MILANO 1863

Software Engineering 2 - Mandatory Project
AY 2019/2020

SafeStreets
DD

Version 1.0 – 9/12/19

Maserati Alessandro - 944593

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to provide details about the design of SafeStreets application. This will be done illustrating the chosen architecture, giving the description of all the components that form the system, representing the related run-time processes and showing which patterns are used to develop the system. Furthermore, the requirements illustrated in the RASD are mapped on the relative architecture's components. Lastly, the implementation, the integration and the test plan are identified and described in detail.

1.2 Scope

Here a brief review of what is the scope of SafeStreets application, referring to what has been described in the RASD.

SafeStreets' main goal is allowing users to report traffic violations, taking one or more pictures and providing a textual description, to authorities. Authorities, on the other side, with their knowledge, can check these reports, in order to store into the database only the correct ones. End users are also allowed to visualize the list of personal reports, with the response of the authority that checked them.

SafeStreets offers to customers other functionalities. Both end users and authorities are allowed to consult the Traffic Violation Map: exploiting a MapService, the system is able to build a map that highlights the various streets with different colours, depending on how many traffic violations occur per day. Furthermore, only authorities are able to retrieve the list of most notified vehicles. In addition, filters are available for these two functionalities, in order to visualize only certain information.

1.3 Definitions, acronyms and abbreviations

1.3.1 Definitions

- *Authority*: public institution related to street safety (e.g. municipality, local police).
- *End user*: people (unrelated with authorities) using Safe Streets application with the aim of report traffic violations and know the streets where the most violations occur.
- *My Reports*: section visible on End User mobile app in which all the reports made by that end user are shown, underlying the current state of each report (it can be Unchecked, Confirmed or Rejected depending on the decision of the authority).
- *Traffic Violation Map*: it's a map, build exploiting a Map Service, that highlights the streets with different colours, depending on how many traffic violations have been notified there.

1.3.2 Acronyms

- RASD: Requirements Analysis and Specification Document
- DD: Design Document
- API: Application Programming Interface
- DB: Data Base
- MVC: Model View Controller

1.3.3 Abbreviations

- [Rn]: n-th requirement

1.4 Revision history

- 09/12/19: Version 1.0
    - First release

1.5 Reference documents

- Specification Document "*SafeStreets Mandatory Project Assignment*"
- 1016.1-1993: IEEE International Standard

1.6 Document structure

This DD is composed by 5 sections:

Section 1 is an introduction containing the purpose, the scope, the revision history and the structure of the document and terminology conventions.

Section 2 represents the core of the document. It shows the architectural design with a detailed description of components, relations and interactions between them. This description is made exploiting component, deployment and sequence diagrams. Then, the patterns used in the design are listed.

In **Section 3**, an explanation on how the requirements, identified in the RASD, map on the design elements, illustrated in Section 2, is given.

Section 4 contains the identification of the order in which the implementation, the integration of subcomponents of the system and the test of them is planned.

In this document, a section that provides an overview on how the user interfaces will look like is not provided because that topic is presented in detail in Section 3.1 of the RASD.

2. Architectural design

2.1 Overview

SafeStreets application is thought to be developed on a client-server three-tier architecture. The three logical levels (presentation, application and data) have their own dedicated tier. In this way the system is more scalable and more flexible, because the three tiers can be developed separately by different teams and future updates that involve only one tier can be made without touching the others.

With regards to presentation layer, both end users and authorities exploits the same mobile application, obviously, depending on the login process, the application will show the correct functionalities. For what concerns the application layer, the application server communicates with the DB in two ways: synchronously when it must retrieve information and asynchronously when it has to store information.

It is worth pointing out that the application works as a client-server one, except for the notification process. More details will be given in the following sections.

2.2 Component view

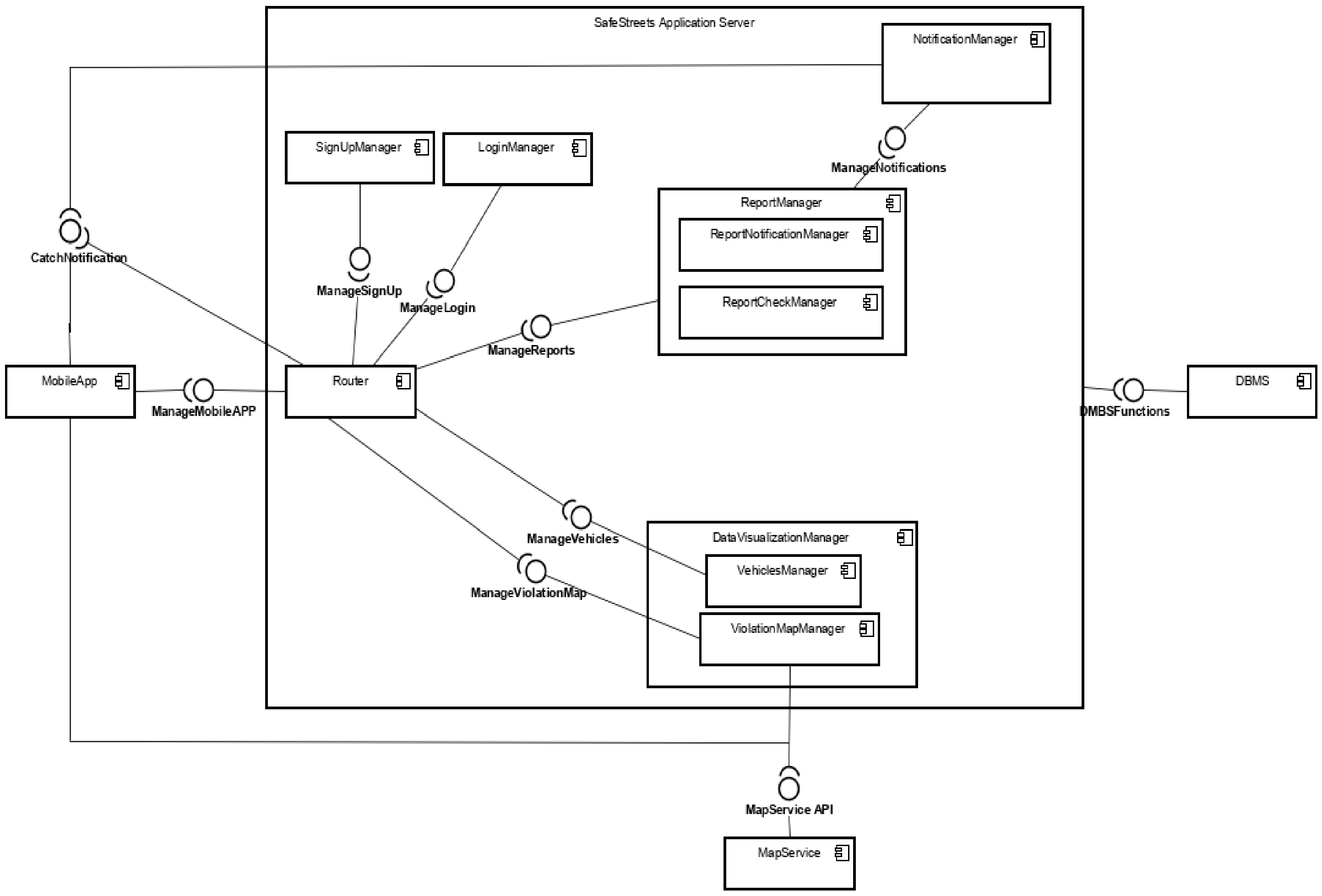


Figure 1: SafeStreets Component Diagram

Figure 1 represents the component diagram of the application. Because of simplicity reasons, the application server (the core of the application) is the only subsystem described and the interactions between application's components and the DB are omitted as they are illustrated in section 2.2.1 Components functionalities.

2.2.1 Components functionalities

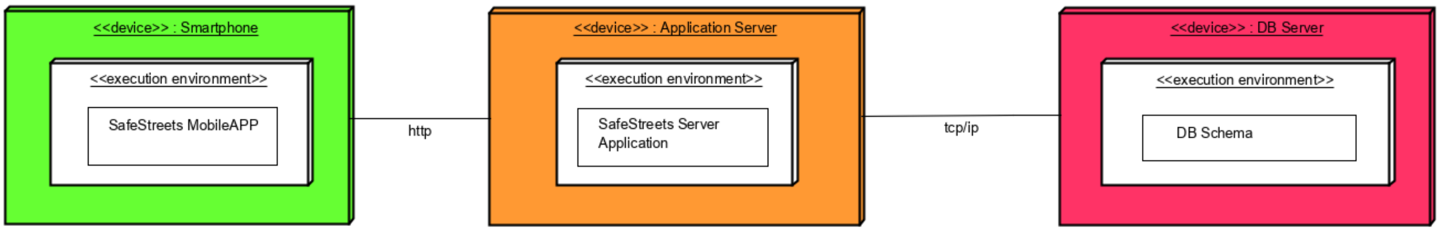
- **Router**: it interacts directly with user's mobile applications and it manages all the function calls from them. Depending on the result of the signup process, it forces the mobile app to show to the end user the right functionalities. It has also the task of acting as intermediary between the various application subcomponents.
- **SignUpManager**: it contains the procedures that allow a user to register to the system (both end users and authorities). It communicates with the DB to store registrations data and to perform changes on them (changes on credentials).
- **LoginManager**: it contains the procedure that allow users to log into the system. It interacts with the DB in order to check the inserted credentials.
- **ReportManager**: it manages all the processes that involves traffic violation reports. It is divided in two subcomponents: **ReportNotificationManager**, that controls the function of reporting a new traffic violation report from an end user and the visualization of *MyReports* section, and **ReportCheckManager**, that queries the DB for unchecked violation reports in order to show them to authorities. It interacts directly with the DB in order to extract the right report instances and with NotificationManager. Notice that ReportNotificationManager, after the submission of the report, runs the algorithm able to retrieve the license plates visible from the picture entered by the end user.
- **DataVisualizationManager**: it contains the applicative logic that handles users' requests of visualize data mined from

the DB. It has two subcomponents: the **ViolationMapManager**, that, interacting directly with the MapService, "builds" the TrafficViolationMap asked by both type of users, and **VehiclesManager** that queries the DB for the list of most reported vehicles. Obviously, these components manage also the filtered requests from users.

- **NotificationManager**: it contains the applicative logic for pushing notifications to users' mobile app. It comes into play in two different situations: when an end user reports a new traffic violation report to the system, pushing a notification to all authorities and when an authority checks a traffic violation report, sending the response to the end user that notified it.

It must be noticed that the only components that exploit the MapService API are ViolationMapManager, that creates the map using the information stored in the DB, and the Mobile App, in order to visualize the map in a correct way.

2.3 Deployment view



<<device>> : Smartphone
<<execution environment>>
SafeStreets MobileAPP
http
<<device>> : Application Server
<<execution environment>>
SafeStreets Server Application
tcp/ip
<<device>> : DB Server
<<execution environment>>
DB Schema

Figure 2: SafeStreets Deployment Diagram

The diagram shown in Figure 2 represents how the software components are distributed.

All the presentation logic must be deployed in the first tier. Both end users and authorities are provided with a mobile application on their mobile devices (the application must be available for both iOS and Android devices, so most users can exploit it).

The business logic must be deployed in the second tier, the Application Server handles all the requests from users and provides all the available functionalities of the system. All components, whose functionalities are described in the previous section, are distributed into the Application Server (the 2nd tier of the system). It's important to highlight that, as described in Section 4, the Router component must be replicated (more copies of this component are "available" in the Application Server), in order to manage the great number of requests from the MobileApp and to make the whole system more reliable.

In the third tier, the data access must be deployed. The DB Server is conceived to execute a Relational DataBase, in order to store and extract the information needed by the Application Server.

2.4 Runtime view

In this section, some of the most relevant use cases (from Section 3.2 Of the RASD) are described using sequence diagrams, to highlight the interactions between the involved design components and sub-components. In order to keep the sequence diagrams readable, the following sequence diagram doesn't contain the alternative scenarios where, for instance, incorrect inputs are entered by the user. These scenarios are described in the RASD Use Case Section, because the behaviour is more or less always the same, with some error messages shown to the user.

Registration and Login

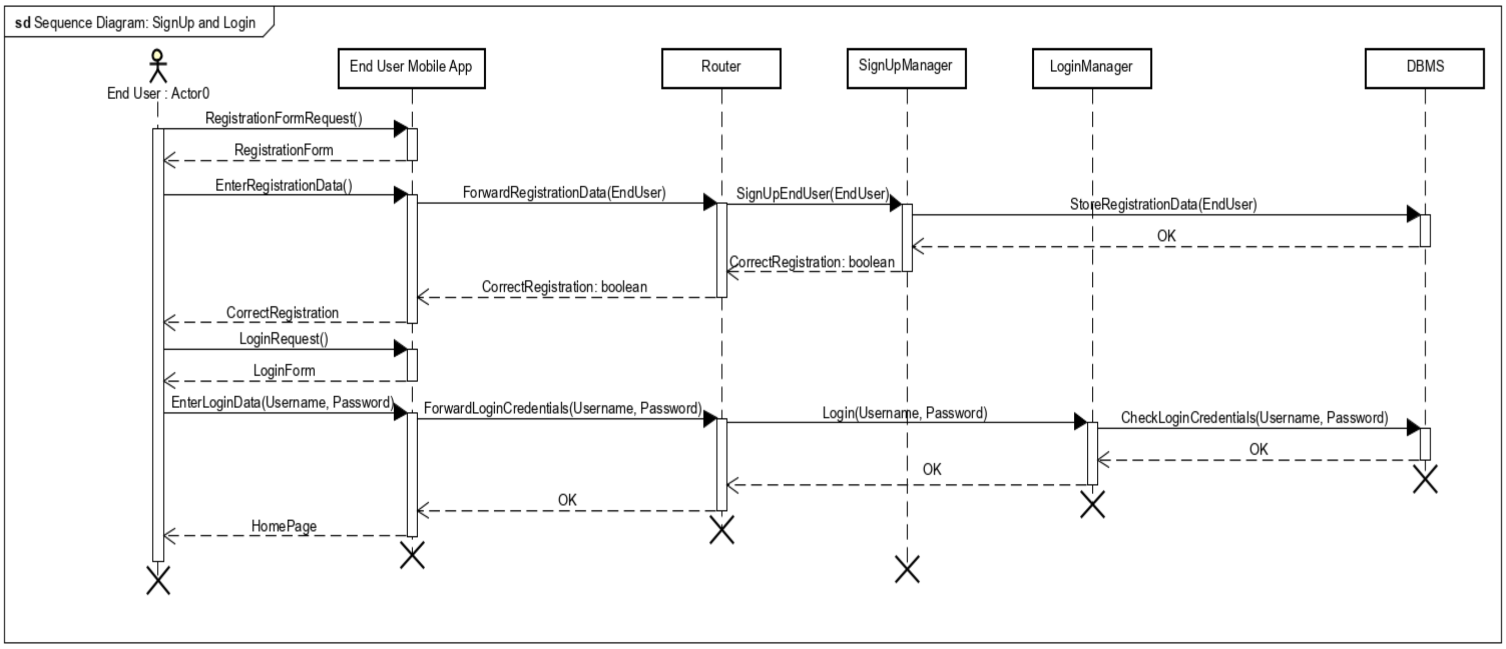

sd Sequence Diagram: SignUp and Login
End User : Actor0
End User Mobile App
Router
SignUpManager
LoginManager
DBMS
RegistrationFormRequest()
RegistrationForm
EnterRegistrationData()
ForwardRegistrationData(EndUser)
SignUpEndUser(EndUser)
StoreRegistrationData(EndUser)
OK
CorrectRegistration: boolean
CorrectRegistration: boolean
CorrectRegistration
LoginRequest()
LoginForm
EnterLoginData(Username, Password)
ForwardLoginCredentials(Username, Password)
Login(Username, Password)
CheckLoginCredentials(Username, Password)
OK
OK
OK
HomePage

Figure 3: Registration and Login Sequence Diagram

Data entered by the End User in the registration process are not listed for the sake of simplicity. The messages coming from the End User are to be understood as simple input data (clicking on App buttons). If the information entered contains some mistakes or the credentials are incorrect (after the DB has checked that) an error message is reported to the End User (as described in the relative Use Case in the RASD). Notice that the same process is also valid for authorities, the only difference is that they must provide their ID Number, instead of the Fiscal Code.

Reporting traffic violation

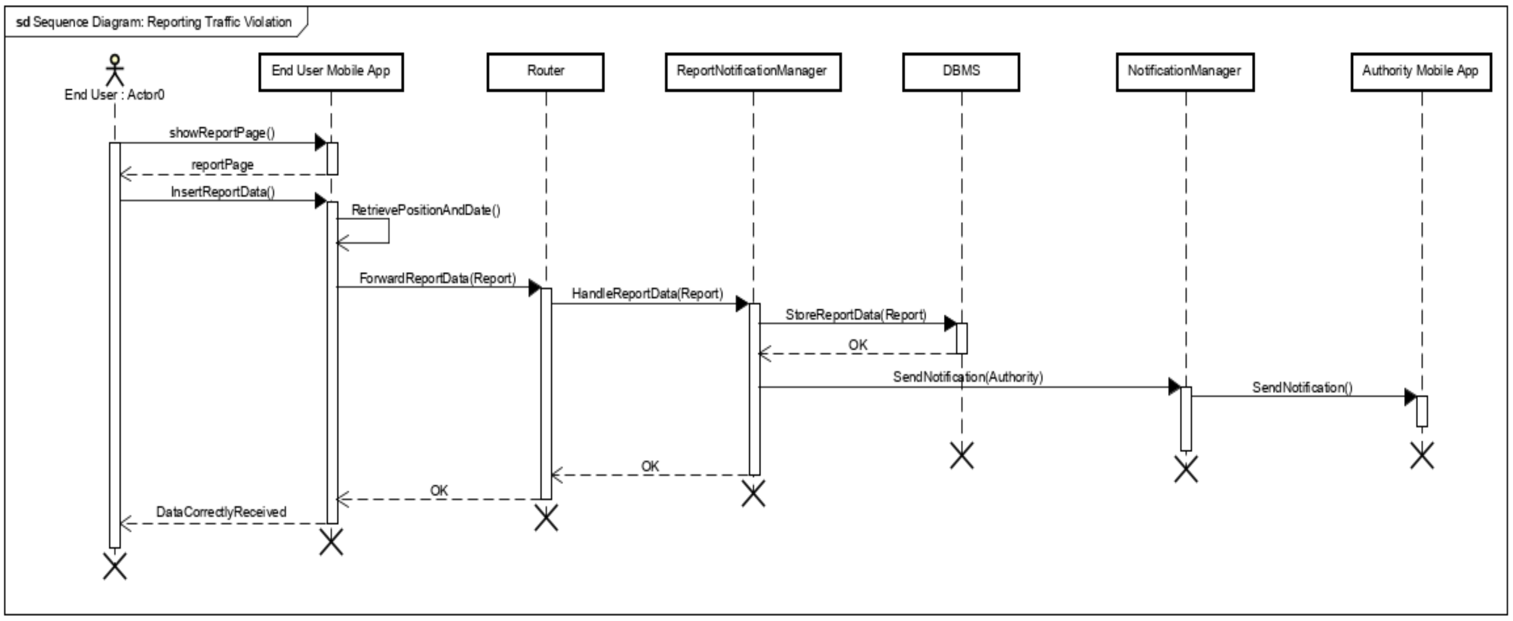


Figure 4: Reporting Traffic Violation Sequence Diagram

Also in this Sequence Diagram, the parameters that a Traffic Violation Report requires are not listed, they consists of one or more pictures (taken calling the device’s camera) and a textual description of the violation. It’s important to notice that, after the storage of the data provided by the End User, the ReportNotificationManager “calls” the NotificationManager in order to send a notification to all registered authorities, that can now check the report.

Traffic Violation Checking Process

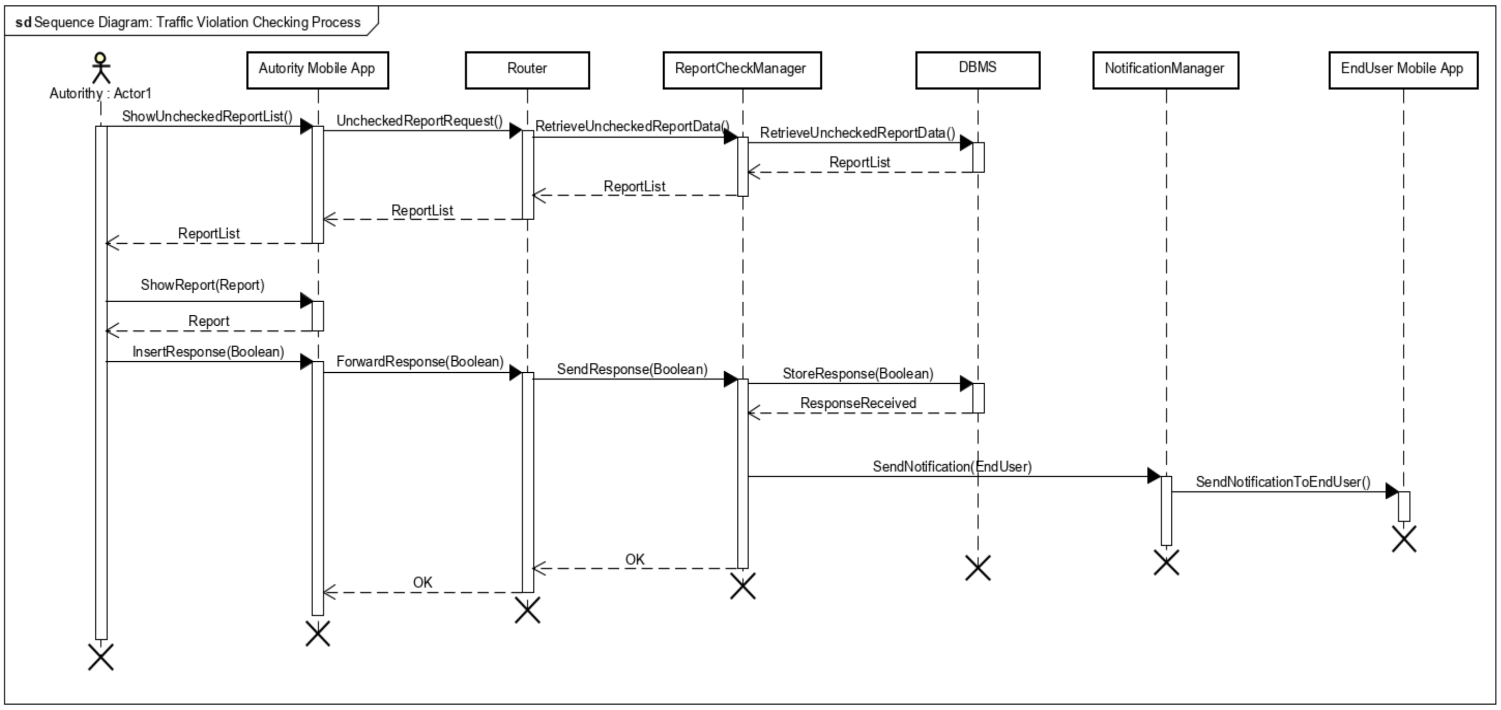


Figure 5: Traffic Violation Checking Process Sequence Diagram

The Authority Mobile APP retrieves the list of Unchecked reports from the DB, so the Authority can choose which is the report to check. After the storage of the response, that report is no more visible in the list. Furthermore, a notification, containing the response, is sent to the End User who made that report.

Visualizing the Traffic Violation Map and focusing on a specific street

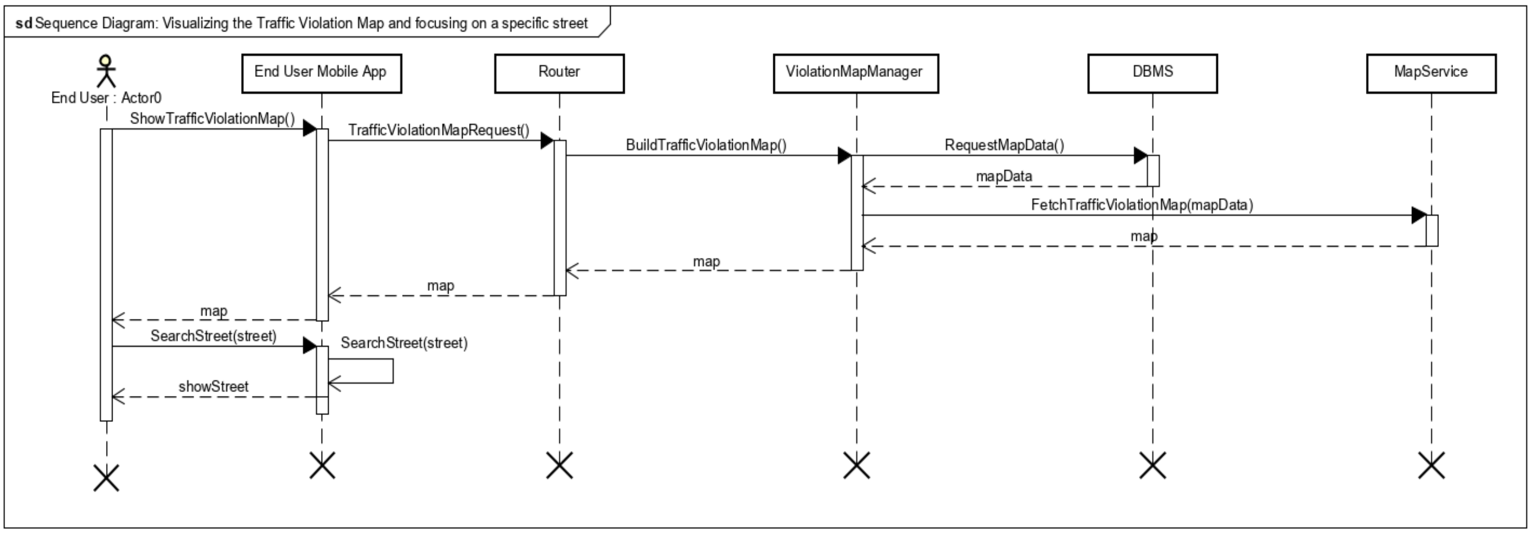

sd Sequence Diagram: Visualizing the Traffic Violation Map and focusing on a specific street
End User : Actor0
End User Mobile App
Router
ViolationMapManager
DBMS
MapService
ShowTrafficViolationMap()
TrafficViolationMapRequest()
BuildTrafficViolationMap()
RequestMapData()
mapData
FetchTrafficViolationMap(mapData)
map
map
map
map
SearchStreet(street)
SearchStreet(street)
showStreet

Figure 6: Visualizing the Traffic Violation Map and focusing on a specific street

When a user wants to visualize the Traffic Violation Map, ViolationMapManager retrieves all the necessary information from the DB. Interacting with the MapService and the MobileAPP, ViolationMapManager can show to the user the requested map. When the user enters the name of the street that he wants to visualize, the call to ViolationMapManager is not necessary, because the information is already available from the App. If the street's name doesn't exist, the MobileAPP notifies the error. Notice that if the user wants to exploit some filters, another call to ViolationMapManager and to the MapService is necessary, in order to restrict the information used to build the new map.

The "OK" messages are simple acknowledgement sent to the requesting component, in order to detect communication errors, when this detection is necessary.

2.5 Component interfaces

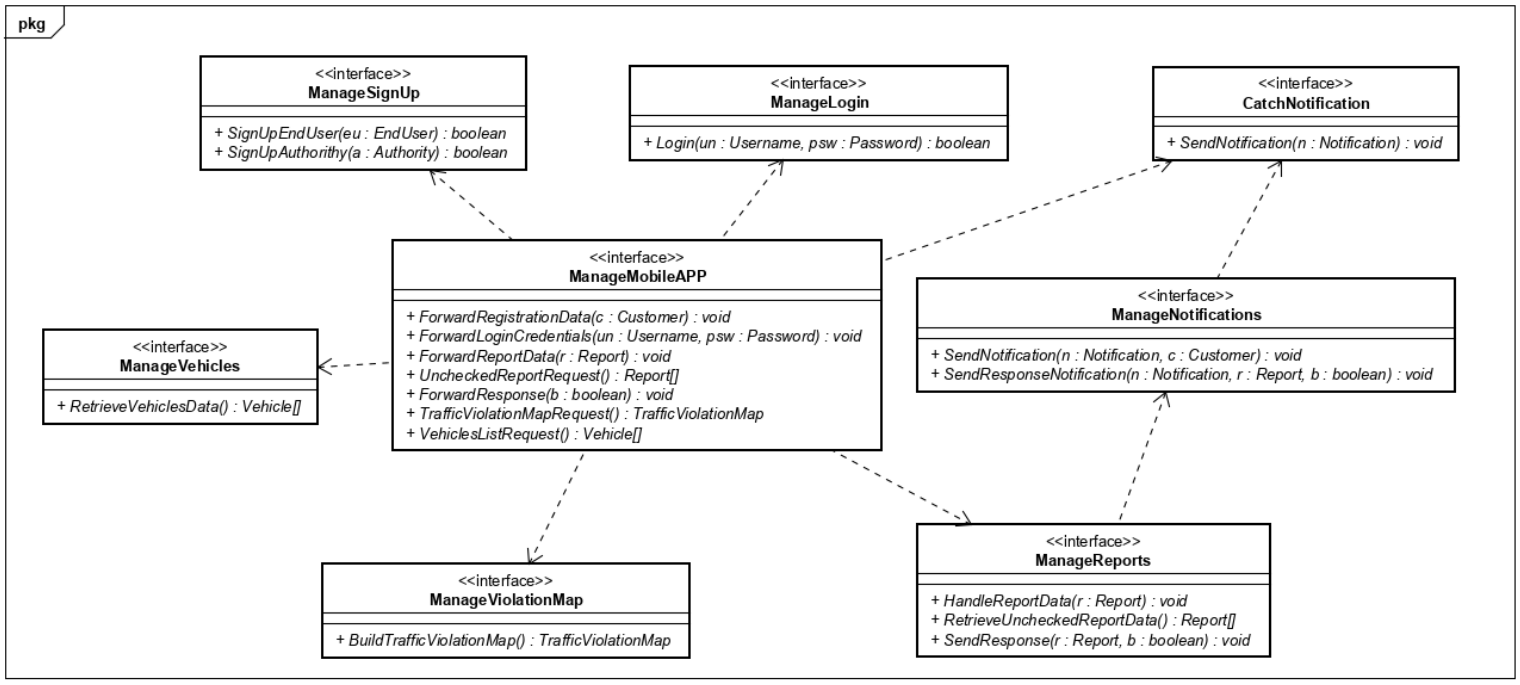

pkg
<<interface>> ManageSignUp
+ SignUpEndUser(eu : EndUser) : boolean
+ SignUpAuthorithy(a : Authority) : boolean
<<interface>> ManageLogin
+ Login(un : Username, psw : Password) : boolean
<<interface>> CatchNotification
+ SendNotification(n : Notification) : void
<<interface>> ManageMobileAPP
+ ForwardRegistrationData(c : Customer) : void
+ ForwardLoginCredentials(un : Username, psw : Password) : void
+ ForwardReportData(r : Report) : void
+ UncheckedReportRequest() : Report[]
+ ForwardResponse(b : boolean) : void
+ TrafficViolationMapRequest() : TrafficViolationMap
+ VehiclesListRequest() : Vehicle[]
<<interface>> ManageNotifications
+ SendNotification(n : Notification, c : Customer) : void
+ SendResponseNotification(n : Notification, r : Report, b : boolean) : void
<<interface>> ManageVehicles
+ RetrieveVehiclesData() : Vehicle[]
<<interface>> ManageViolationMap
+ BuildTrafficViolationMap() : TrafficViolationMap
<<interface>> ManageReports
+ HandleReportData(r : Report) : void
+ RetrieveUncheckedReportData() : Report[]
+ SendResponse(r : Report, b : boolean) : void

Figure 7: SafeStreets Component Interfaces

Figure 7 shows the interfaces offered by the Application Server components.

ManageSignUp contains methods exploited by the Router in order to successfully register a new user, the necessary information to identify a generic user is passed as parameter.

ManageLogin offers the method that allows a user to log into the system. Also this interface is exploited by the Router, that “forwards” the credentials to the DB, in order to check their correctness. The method returns a Boolean value, depending on that a message of success or failure is shown to the user.

The **ManageReports** interface offers all the methods that handles Traffic Violation Reports: HandleReport(Report) allows the storage of a new report to the DB; RetrieveUncheckedReportData() returns the list of all reports that haven’t been checked yet, with all the necessary information to take a decision about them; SendResponse(Report, Boolean) notifies the NotificationManager in order to send a notification of the decision to the Report’s creator. It must communicate with the **ManageNotification** interface in order to send notification to the customer MobileAPP when a new unchecked report is stored into the DB or when an authority takes a decision about one of it. The **CatchNotification** interface, offered by the MobileAPP, is exploited to do this. This is one of the two cases in which the client manages a “request” by the server.

The **ManageVehicles** and **ManageViolationMap** interfaces are those allowing the visualization of information by a user. They offer methods called by the Router, that, after consulting the DB, respectively return the notified vehicles list and the TrafficViolationMap requested by the user.

Lastly, the **ManageAPP** interface consists of methods called by the User MobileAPP, when a request from a user occurs.

Notice that if the input entered by the user is incorrect (e.g. the name of a street doesn’t exist or a vehicle’s license plate isn’t in the DB), the router must forward an error message, coming from the right Application component, to the MobileAPP (as described in

Section 3.2 of the RASD). This is possible exploiting the CatchNotification interface offered by the MobileAPP.

2.6 Selected architectural styles and patterns

Client-Server architecture

The system will be developed on a three-tier client-server architecture. The distribution of the entire application logic across three tiers helps optimize the overall application access and layer/tier level development and management.

Mediator Design Pattern

The Router component must be developed following the Mediator behavioural pattern. In this way, clients are able to communicate only with the Router component and they know nothing about the other components. It promotes loose coupling by keeping objects from referring to each other explicitly, and it allows their interaction to be varied independently. Obviously, the Router becomes the most crucial component of the application.

MVC Design Pattern

Using this design pattern, the software application is divided into three interconnected parts. This pattern guarantees a particular reusability of code and also incentivises the parallel development as much as possible. In this case the front-end is represented by the clients, the controller by the Application Server and the model by the DB.

2.7 Other design decisions

MapService API's exploitation

Considering that the app will be a cross-platform native mobile application, different APIs will be used to render the TrafficViolationMap on each operating system. For instance, Apple Maps APIs will be exploited on iOS while GoogleMaps APIs on Android.

3. Requirements traceability

In this section, the requirements identified in the RASD are mapped on the components (identified in section 2) that have the task of fulfil them.

- [R1] – End Users must be able to register to the system, providing personal information (and identified by Fiscal Code).
    - **SignUpManager**: it contains the logical procedures necessary to register to the system an end user.
- [R2] – Registered End Users must be able to login, using their credentials.
    - **LoginManager**: it contains the logic function necessary to log an end user into the system, checking the information stored into the DB.
- [R3] – End Users must be able to take pictures of traffic violations, opening their device's camera from the app.
    - **ReportNotificationManager**: it contains the functionality of opening the device's camera in order to take a picture of the violation and, after the submission of the report, to store the information into the DB.
- [R4] – End Users must be able to specify a textual description of the traffic violations.
    - **ReportNotificationManager**
- [R5] – The system must offer the possibility of being informed of the "Rules for a well-formed traffic violation report" to the user.
    - **ReportNotificationManager**: if the end user wants to know those kinds of rules, it provides them a textual list of the rules.
- [R6] – The system is able to read the notified vehicles' license plate from the reported pictures.
    - **ReportNotificationManager**: it runs the algorithm able to read a license plate from the picture taken by end users.

- [R7] – Authorities must be able to register to the system, providing personal information (identified by ID Number).
    - **SignUpManager**: it contains the logical procedures necessary to register to the system an authority.
- [R8] – Registered Authorities must be able to login, using their credentials.
    - **LoginManager**: it contains the logic functions necessary to log an authority into the system, checking the information stored into the DB.
- [R9] – The system must be able to show a map reporting the number of violations that have occurred in every street. This must be made highlighting the streets with different colours.
    - **ViolationMapManager**: exploiting the MapService API, it builds the TrafficViolationMap requested by users, using data stored into the DB.
- [R10] – Users are allowed to filter the data, by date and type of violation, which are used to “build” the map.
    - **ViolationMapManager**: exploiting the MapService API, it builds the TrafficViolationMap requested by users, using data stored into the DB.
- [R11] – Users are allowed to ask for information about a specific street and the system must show the corresponding map.
    - **ViolationMapManager**: exploiting the MapService API, it searches the requested street and shows the map, highlighting that street.
- [R12] – The system must be able to show (only to authorities) the list of vehicles that have been reported one or more time in a traffic violation report.
    - **VehiclesManager**: retrieving information from the DB, it shows the list of reported vehicles to authority.
- [R13] – Authorities are allowed to filter the list of notified vehicles by date, street and type of violation.

    - **VehiclesManager**: retrieving information from the DB, it shows the list of requested reported vehicles to authority.
- [R14] – Authorities must be able to know everything (pictures, license plate, date, street, textual description) about the reports made by end users.
    - **ReportCheckManager**: retrieving information from the DB, it shows to authority all the information about reports.
- [R15] – Authorities are allowed to consult and investigate the list of unchecked traffic violation reports notified by end users.
    - **ReportCheckManager**: retrieving information from the DB, it shows to authority the list of actually Unchecked reports.
- [R16] – Authorities are allowed to take decisions about any traffic violation report, after consulting it.
    - **ReportCheckManager**: it contains the logic to change the status of a report, after the decision of authorities. It works interacting with the NotificationManager.
- [R17] - The system must notify that response to the related user.
    - **NotificationManager**: it handles the notifications to send, after the checking process made by an authority, to the correct end user.
- [R18] – The possibility to visualise all personal previous reports, with the corresponding authority check decision, must be offered to End Users.
    - **ReportNotification**: retrieving information from the DB, it shows all the reports made by a specific end user.

Notice that the Router has never been mentioned not because it’s not involved in fulfil any requirement, but because its role is to route to the right component all the messages coming from the client side (application of the Mediator pattern).

4. Implementation, integration and test plan

SafeStreets' components must be implemented, integrated and tested following a bottom-up approach.

The following table helps to understand the decisions taken in this section showing the list of functionalities offered by SafeStreets application with the relative importance and difficulty of implementation.

Feature	**Importance**	**Difficulty of implementation**
SignUp and Login	Low	Low
Reporting a Traffic Violation	High	Medium
Checking a Traffic Violation Report	High	Medium
Visualize the TrafficViolationMap	Medium	High
Visualize the list of notified vehicles	Medium	Medium

Figure 8 shows the implementation and testing order of SafeStreets' Application Server components.

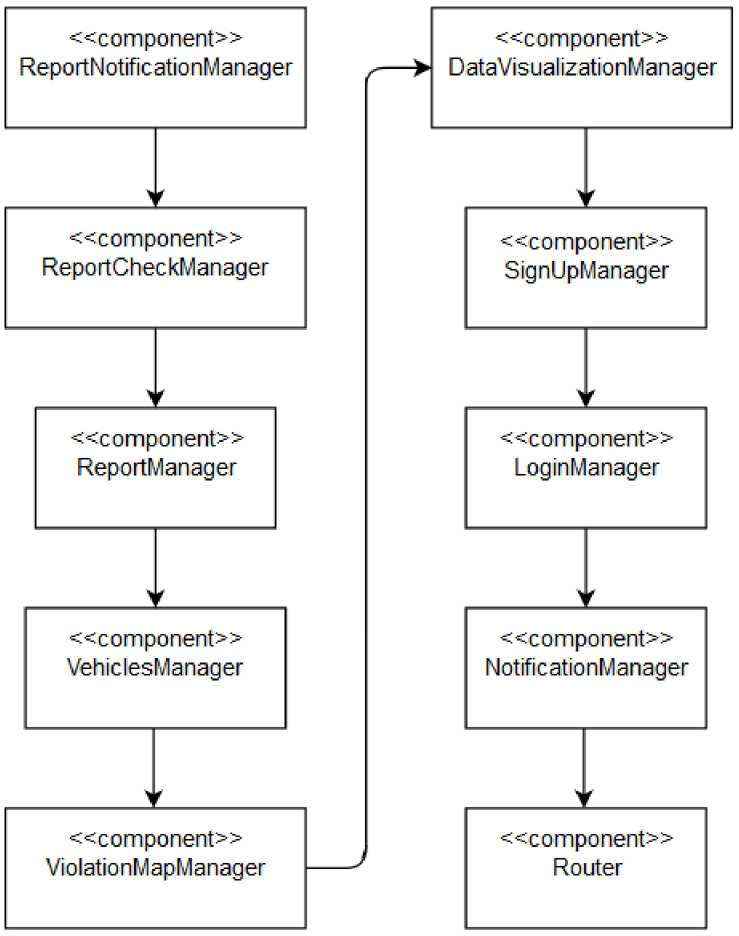


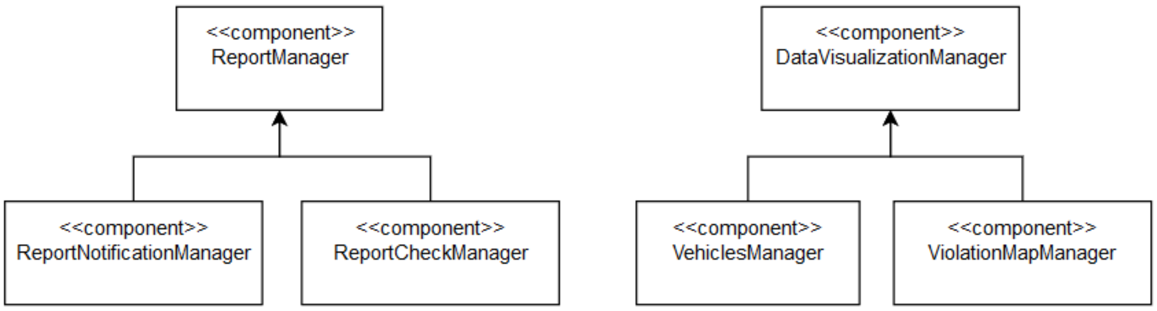
Figure 8: Order of components implementation and testing

The implementation must start with the components related to the functionality of notifying traffic violation reports, because this is the main one of the system. After the building and testing of ReportNotificationManager and ReportCheckManager, these two sub-components must be integrated to realize ReportManager component, that must be tested as well. The same process must be followed to realize the DataVisualizationManager, composed by VehiclesManager and ViolationMapManager. This latter has to be developed exploiting the MapService API, so particular attention is required. Then, the less challenging components related to the SignUp and Login functionalities must be implemented and tested. After that, NotificationManager component has to be developed and tested individually and then integrated with the component that interacts with him, i.e. ReportManager. Finally, the Router component has to be implemented and tested and then its

integration with the rest of the components in the application server must be performed. This is the most crucial component of the application, because of the central role that it's playing: every message and request from the MobileAPP passes through it to be redirected to the right component (if it goes down, the whole application goes down as well). For this reason, this component must be replicated, more than one copies of the Router must be available to the clients (in this way a greater number of requests is satisfiable).
In the next section, the integration plan is provided in detail.

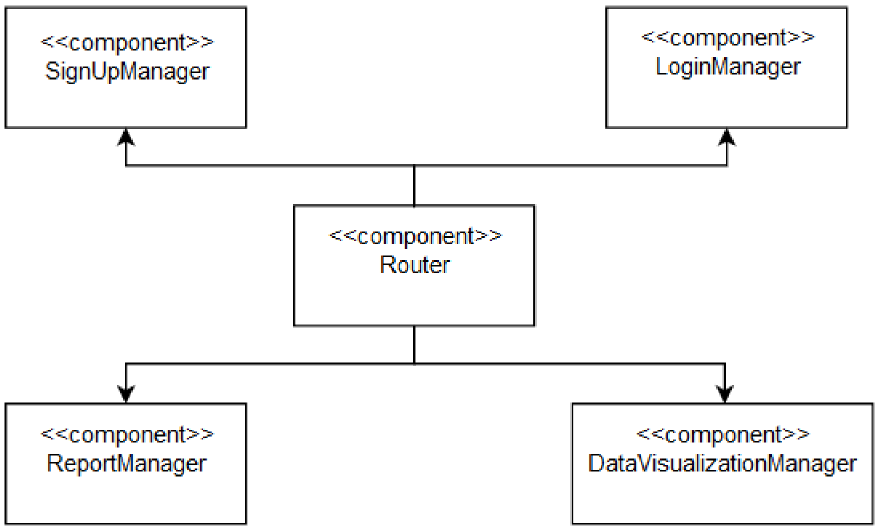
4.1 Integration plan

Application Server components' integration



<<component>> ReportManager
<<component>> DataVisualizationManager
<<component>> ReportNotificationManager
<<component>> ReportCheckManager
<<component>> VehiclesManager
<<component>> ViolationMapManager

Figure 9: Sub-components Integration



<<component>> SignUpManager
<<component>> LoginManager
<<component>> Router
<<component>> ReportManager
<<component>> DataVisualizationManager

Figure 10: Router Integration

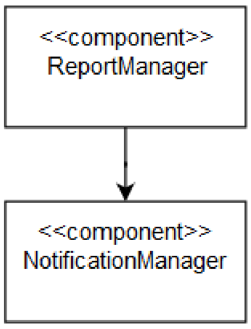


Figure 11: Notification Integration

The integration (and testing) of the frontend (MobileAPP) with the backend (Application Server) must take place after the complete integration of the two parts (an intermediate testing of the MobileAPP with the Router component is necessary in order to ensure the correct sending of requests).