POLITECNICO
MILANO 1863

AA 2019/2020

DD – Design Document

Version 1.1 - 09/12/19

Authors:

Giuseppe Italia

Giuseppe Asaro

Professor

Elisabetta Di Nitto

Table of contents

1. INTRODUCTION

1.1 PURPOSE

The purpose of the Software Design Document (DD) consists of giving more technical details than the RASD about SafeStreets. Indeed, if the RASD has as its objective to provide a more abstract view of the system with its functionalities, the Design Document goes deeper into detail about the design, providing an overall guidance to the architecture of the system.
In this document will be described how SafeStreets is designed and planned, identifying its main components and the interfaces between them. The documents also guides the developers through the architecture and the software project, stating what has to be implemented and how to do it.
Here all the components forming part of the system are described, with the related run-time processes and all the design choices are listed and motivated.

In particular, the following topics are touched by the document:
- The high-level architecture;
- The main components, their interfaces and deployment;
- The runtime behavior;
- The design patterns;
- Additional details about the user interface;
- A mapping of the requirements on the architecture's components;
- Implementation, integration and testing plan.

1.2 SCOPE

To have a deeper view of the application's scope, reader is invited to refer to the RASD. (RASD 1.2, page 5)

There are different classes of users: the citizen and the Local Police; the System allows the firstone to make a report violation, when they notice a traffic infraction, and eventually review them, they also can access to some stats build by the system through the analysis of past reports and data provided by municipality. On the other hand, Local Police can monitor user's report and if they are not thrutful delete them, they also can access to same stats of user.

1.3 Definitions, acronyms, abbreviations

Definitions

- **User**: general SafeStreets customer that can make a violation report with his device and can view all stats that the system provided.
- **Citizen:** actor that can only make report and see stats.
- **Local police:** actor of the system that intervenes checking all report and in some case removing it; furthermore can view all stats provided by SafeStreets.
- **Traffic violation:** a road infraction reported from user.
- **Municipality:** the entity that will provide accident information and will get suggestion from SafeStreets.
- **Accident:** sent by municipality, will involved into stats.
- **Report violation:** core element of the application that will be sent by citizen to the system.

Acronyms

- **API :** Application Programming Interface
- **UI :** User Interface
- **MVC**: Model View Controller
- **DMZ**: Demilitarized Zone
- **A:** Application
- **P:** Presentation
- **DB:** DataBase
- **DBMS:** DataBase Management System
- **REST:** Representational State Transfer
- **HTTP:** HyperText Transfer Protocol
- **SQL:** Structured Query Language
- **UX Diagrams:** User eXperience Diagrams
- **RASD:** Requirements Analysis and Specification Document
- **DD:** Design Document

Abbreviations

- **Stat:** Statistic

1.4 Revision History

- Version 1.0: first release
- Version 1.1: fix some grammar and coherence errors.

1.5 DOCUMENT STRUCTURE

- **CHAPTER 1:** It is simply an introduction to the DD, highlighting the differences with the RASD.

- **CHAPTER 2:** This is the core section of the document; this chapter provides at first an overview of the architecture of the system, then there is a deeper description of the design choices explaining the main hardware and software decisions, in particular: in the component view the software components are listed and is described the interaction between them, moreover these interactions became useful in the “RunTime view” part to define the sequence diagrams. The end of the chapter aims to illustrate design patterns and other design decisions.

- **CHAPTER 3:** In this chapter is illustrated how the customer can use the application, moving from an interface to another one.

- **CHAPTER 4:** In this chapter is described a mapping between requirements defined in the RASD and the design components in the application server.

- **CHAPTER 5:** Includes the description of the implementation plan, with also the implementation plan and testing plan.

- **CHAPTER 6:** In this chapter is shown the distribution of work between the group members.

- **CHAPTER 7:** This final chapter provides a list of the reference documents used to write this DD document.

2. ARCHITECTURAL DESIGN

2.1 Overview: High-level components and their interaction

In the following paragraphs a general overview of how the system is architected will be presented. The application to be developed is a distributed application: SafeStreets is supposed to be fully scalable and portable, so a layered architecture is the one that best fits these requirements. The three logic software layers are:

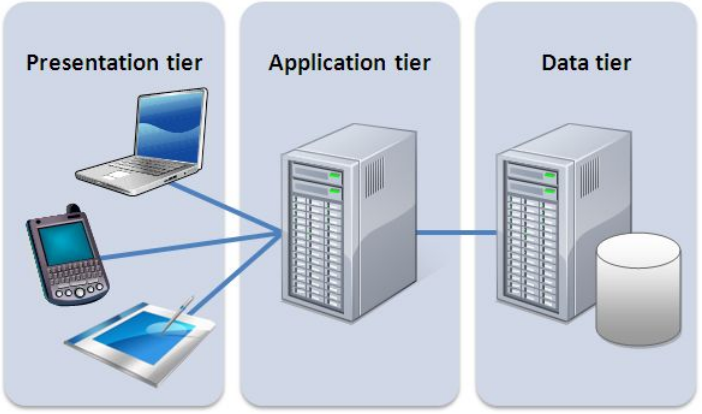
- Presentation: it manages the user interaction with the system and it is the only layer that users can directly access and interact with, it is the most external layer of the system, and it is responsible for handling all UI communication and logic. This layer does not handle data or process business rules.

- Application: it captures the business logic of the application and all its core functionalities, there are implemented all the main functions of the system and all the operations related to the analysis of traffic violations and elaboration of stats. This layer interacts with the APIs exposed by the Data Layer in order to store and retrieve data.

- Data Access: this layer manages the information with access to the DB. It also provides an API to the Business Layer that exposes methods of managing the stored data without creating dependencies on the data storage mechanisms, promoting the encapsulation of the persistence mechanisms and avoiding data exposition.

All these logic layers are thought to be divided onto three different hardware layers so that each logic layer has its hardware dedicated, so here we have a three-tier architecture.

Figure 1 : The following image show the high level architecture of the system, without providing any detail



There will be different kind of interaction between P & A; on the one hand citizen, that can access to the system through their mobile device, will communicate with synchronous message to retrieve their data, access to stats and specially to make reports, the messages will be synchronous because we aspect an as fast as possible reply from the system in order to have the possibility

to fill again the form of the report violation if something went wrong, moreover the connection must be transient, indeed it is necessary that receiver (the Application Server) is online.

On the other hand both Local Police (that can access via Mobile App and via WebApp) and Municipality communicate with the Application layer in an asynchronous and persistent way, they don't wait for an answer. Finally the Server in the application layer communicates synchronously with the DBServer to retrieve information or asynchronously to store information.

2.2 Component View

This part of the document aims to describe how the software components of the application server could interact between themselves.

The following diagram contains logical and physical components of the system showing their interactions. The ports that represent the external interface exposed by components are shown only among different subsystems.
In the diagram only the application server subsystem is analyzed in detail because it is the core component of the system: it contains the business logic (Application layer). The other components are represented (through their software components only) just to show their interactions with the application server.

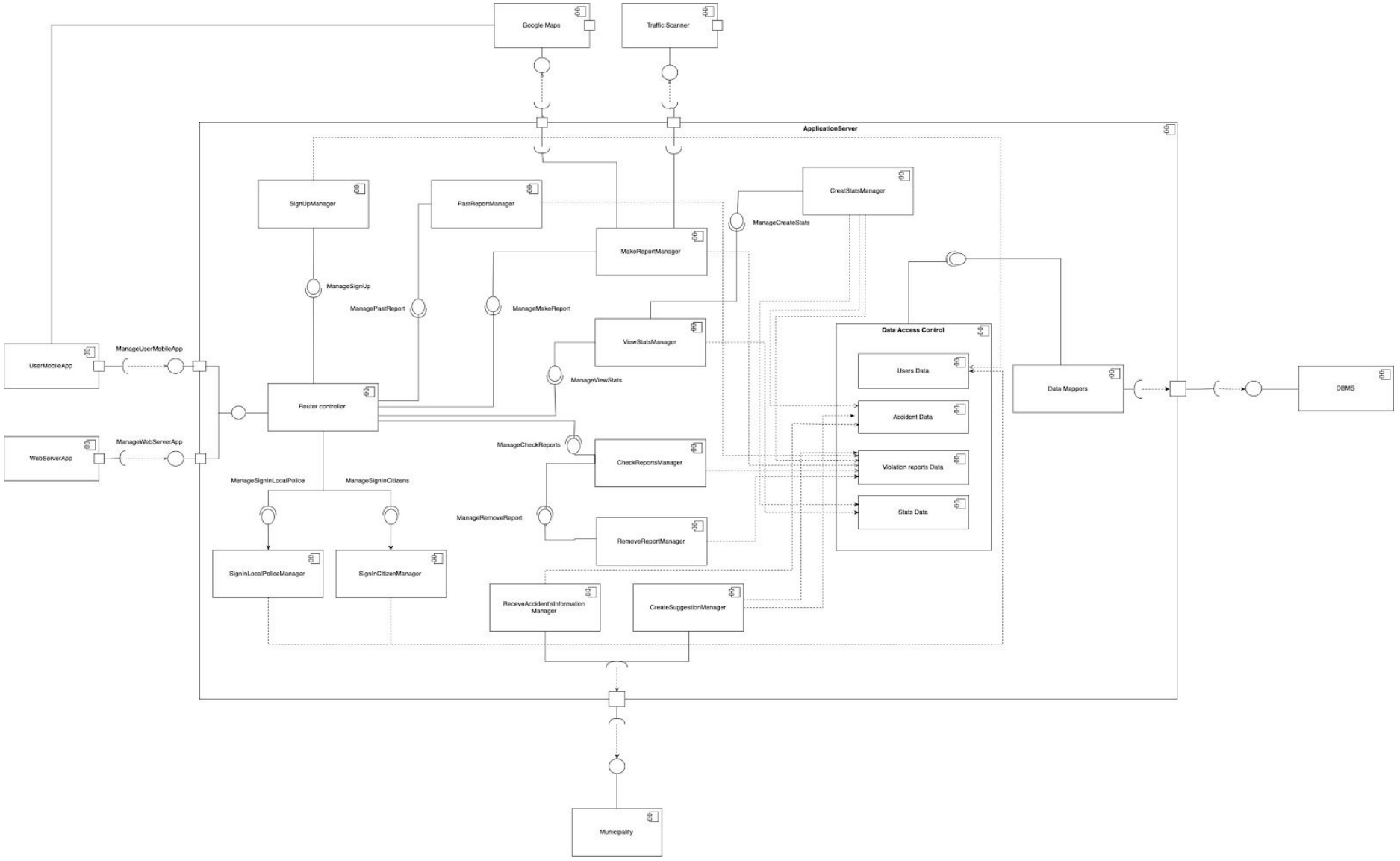


Figure 2 : Component-view diagram

The components' functions contained in the 'ApplicationServer' are described in the following:

• **Router**: it manages all the 'messages' and all the function calls coming from the other subsystems to pass the data to the correct component and eventually call the right method on it. The router is partitioned according to the type of component it has to interact with because the functionalities offered are quite different among them.

We have three different substructure of the router:

1. Stats router : it handles all interactions between the system and who wants view one stat; so he interacts only with logic component ViewStatsManager;
2. Report router : it interacts with logic Component MakeReportManager and CheckReportManager;
3. Sign In/Up router: it interacts with only logic component that allows login and registration on the system: SignUpManager, SignInLocalPoliceManager, SignInCitizenManager;

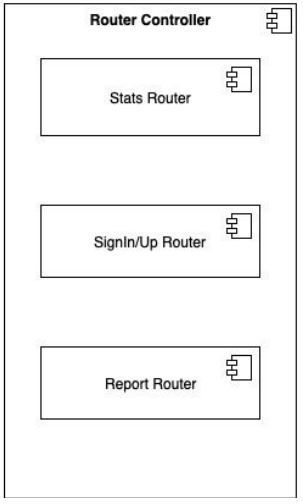


Figure 3 : Structure Router

- **SignUpManager**: this component contains all the procedures to allow citizens to register to SafeStreets. It has to interact with DataAccessControl to store data about the registration and performing controls about the chosen username and password;

- **SignInCitizenManager**: it manages all the logic inherent to the authentication of the citizens. It interacts with the DataAccessControl to check that the authentication parameters, username and password, match the stored ones;

- **SignInLocalPoliceManager**: it manages all the logic inherent to the authentication of Local Police. It interacts with the DataAccessControl to check that the authentication parameters, match the stored ones;

- **MakeReportManager**: it is one of the main components of the system; it allows citizens to report violations through their devices; so this component manages procedures like take a photo, get the type of violation, get the type of vehicle etc; this component interacts also with external component “GoogleMaps” to take the localization of violation and “Traffic Scanner” to check the traffic plate. Finally, MakeaReportManager interacts with the DataAccessControl to save all information about the reports received by Citizen;

- **ViewStatsManager**: this is another very important component, it allows citizens and Local Police to view the stats about the violations and the city 's safety that the system provides; he works only with the DataAccessControl because all information which he needs must be taken from the DB;

- **CheckReportManager:** this component allows Local Police(once logged) to check reports; it interacts with the DataAccessControl to receive the reports that policeman is interested in; finally, he can be called by the WebServerApp through Router and can dialogue with RemoveReportManager if the policeman wants to remove a particular report;

- **RemoveReportManager**: it manages the removal of the wrong report; this component can be called only by CheckReportManager component and managed by WebServerApp;

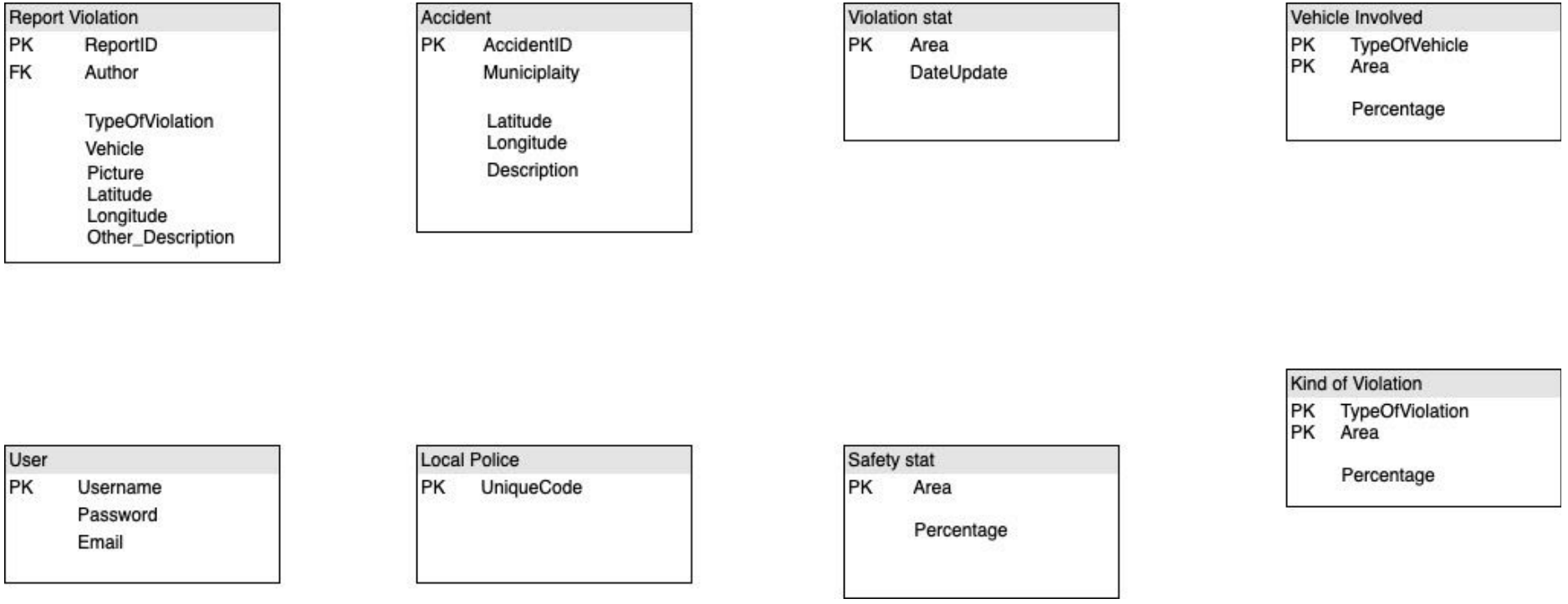
- **ReceiveAccidentInformationManager**: this component handles all weekly interactions between the system and the municipality; the information exchanged concerns the city accidents; he interacts only between external component “Municipality”, who provides the information requested, and the DataAccessControl which save the information received into the DBMS to allow later to “CreatStats” to analyzer them and create a stats;

- **CreateStatsManager**: this component manages the creation and the update (using also a timer to let the routine start) of all stats that the system provides to users; it interacts only with the DataAccessControl to retrieve the information and store the stats created;

- **PastReportManager**: this component manages the retrieve of user's past reports; it interacts with the "Violations report data" of "Data access control" passing to it the username of the user so that DataMapper will be able to get the correct data from the DB.

2.3 DATABASE DETAILED ANALYSIS

Now an high-level model of the Relational DataBase is shown: main tables are provided, followed by the SQL code to create them.



```
create table User (
        Username varchar (12) primary Key,
        Password varchar (12) not null,
        Email varchar (24) not null
)

create table LocalPolice (
        UniqueCode character (8) primary key
)

create table ReportViolation (
        ReportID int (12) primary Key,
         Author varchar (12) foreign key references User (Username)
```

```
                                    on delete cascade
                                    on update cascade,
        TypeOfViolation varchar (10) not null,
        Vehicle varchar (8) not null,
        Latitude  int (12) not null,
        Longitude  int (12) not null,
        Other_Description varchar (100)
        )
```

```
create table Accident (
AccidentID character (12) primary Key,
Municipality varChar(10) not null,
Latitude  int (12) not null,
Longitude  int (12) not null,
```

```
Description varchar(100)
)
create table SafetyStat (
Area varchar(20) Primary Key,
DataUpdate date
)

create table VehicleInvolved (
TypeOfVehicle varchar(8) ,
Area varChar(20)  foreign key references ViolationStat (Area)
                                    on delete cascade
                                    on update cascade,
,
Primary Key (TypeOfVehicle, Area),
Percentage int(3)
```

```
)

create table ViolationStat (
        Area varchar (20) Primary Key,
        DateUpdate date
)

create table KindOfViolation (
        TypeOfViolation varchar(8) ,
        Area varChar(20) foreign Key references ViolationStat (Area)
                                on delete cascade
```

```
                    on update cascade,

Primary Key (TypeOfVehicle, Area),
Percentage int(3)
)
```

2.4 Component interfaces

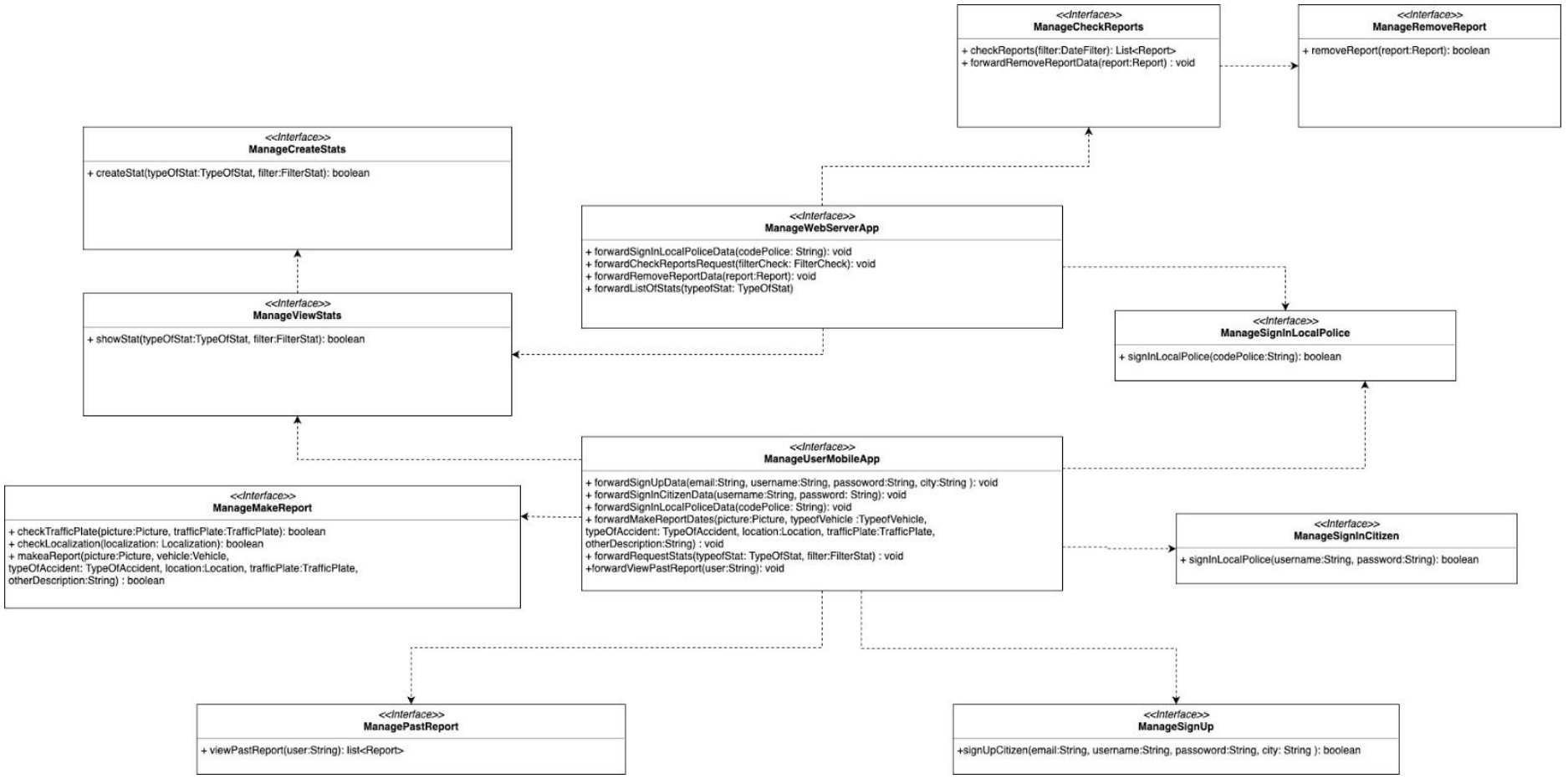


Figure 4: Component interface

In the above figure the component interfaces belonging to the application server, with reference to what was shown in the Component diagram, are represented. The arrows represent a dependency relation.

There are some things that have to be pointed out, in order to give a better understanding of how those component interfaces are intended:

- Only the interfaces offered by the application server are chosen to be represented.
- Most methods have a boolean as its return parameter to indicate if the resolution has finished correctly; moreover, some methods have a List<> as its return parameter (for instance when a policeman wants to check the reports and when the method "checkReport" is called is correct to receive one list of reports and not only one object).
- The manageCreateStats is never called by the user but only by manageViewStats when one policeman or citizen wants to browse and see one stat and it is not available because not present in the DBMS, so the component interface manageViewStats provides the result false with the method "showStat" and calls upon "manageCreateStat" to create the new stat(if it's possible). After the creation the stat will be stored in the DBMS and will be updated every two weeks.
- The manageRemoveReports is called only by the component manageCheckReport through the method "removeReport"; this happen because one policeman before to remove one report has to check if it's correct or not; so after checking the report, he can decide to keep the report because it's considered correct or remove it forwarding the request to the component RemoveReport through the method forwardRemoveReport of the manageCheckReport component.

2.5 DEPLOYMENT VIEW

Now let's illustrate how the general software architecture is distributed on the Hardware infrastructure considering the 3-tier structure described in the *GENERAL OVERVIEW*, so here we have a mapping between software-logical components and the hardwareones:

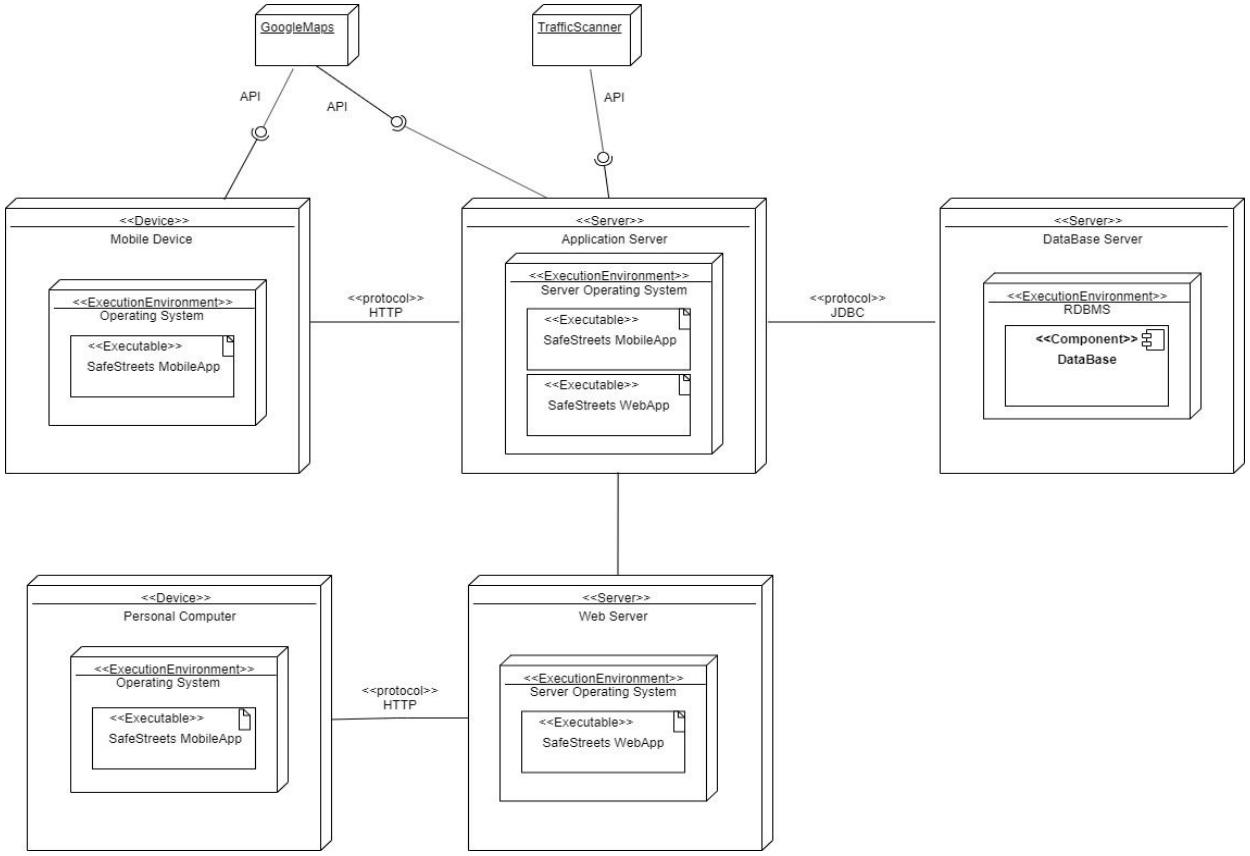


Figure 5 : deployment diagram

On the left-side we have the first tier with the MobileApp on a mobile phone and WebApp on a personal computer, at this tier correspond the presentation layer so the client in an architecture client-server: the client can be either a common citizen or Local Police, in the first case it uses the MobileApp, in the latter the webApp; both of them communicate, through the protocol HTTP (the one who best fits with a RESTful architecture) with a Server (Web Server or Application Server) that is the second tier, a passive component, waiting for the client invocation, that contains the business logic of the system (Application); obviously the server can not responds to the client requests without communicate with the third tier, that is the DataBase, through the protocol JDBC.

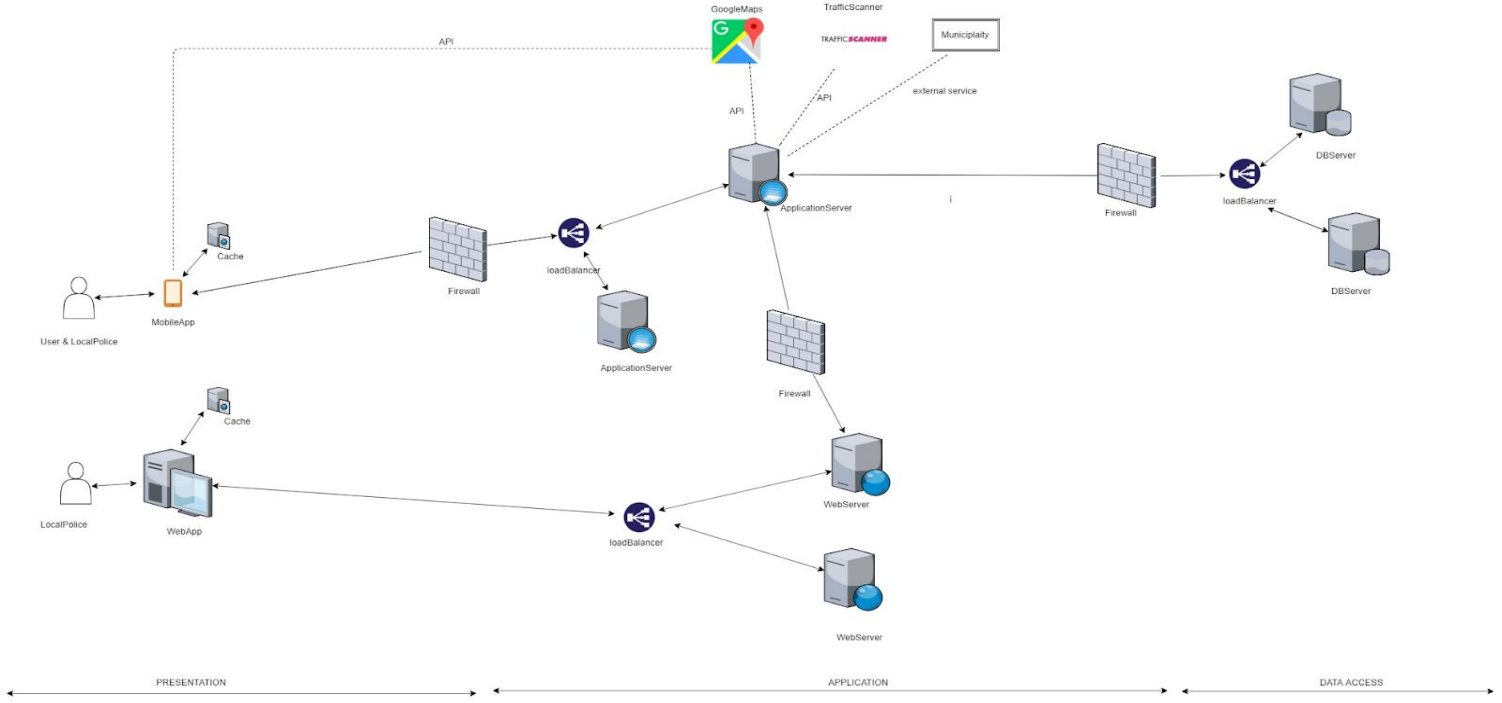


Figure 6: an informal view of the System is given

One of the most important design choice was to replicate nodes; there are two mainly reason to replicate a node: the first one is to increase the reliability of the system, for example if a node crashes it may be possible continue working by simply switching to one of the other replicas, moreover by maintaining multiple copies becomes possible to provide a better protection against corrupted data. The other reason to replicate is performance, replication begins important when the system needs to scales in numbers and geographical area.
Another important choice was the one to use a DMZ (Demilitarized Zone) to expose only the resources into this zone, so that there is an increase of the security level both at the WebServer and at the DBServer. The DMZ is bounded by some firewalls: essentially a firewall disconnect any part of a distributed system from the outside world, this firewall operates as a router and makes decisions as to whether or not to pass a network packet based on the source and destination address as contained in the packet's header.

In order to have a quicklier communication the idea is to use a cache mechanism, caches will store some data like the most recently stats asked to the system, so that the number of interaction with server side is reduced. It's important to underline that replicated node's can not use caches because is not possible knowing what have been requested from a certain server and what from another one, so it's not possible provide a mechanism that could prevent Servers to access to DB.

2.5 RUNTIME VIEW

At this point using sequence diagrams will be described the way components interact to accomplish some tasks yet described in the RASD at high level (make reports, delete reports, building & access to stats).

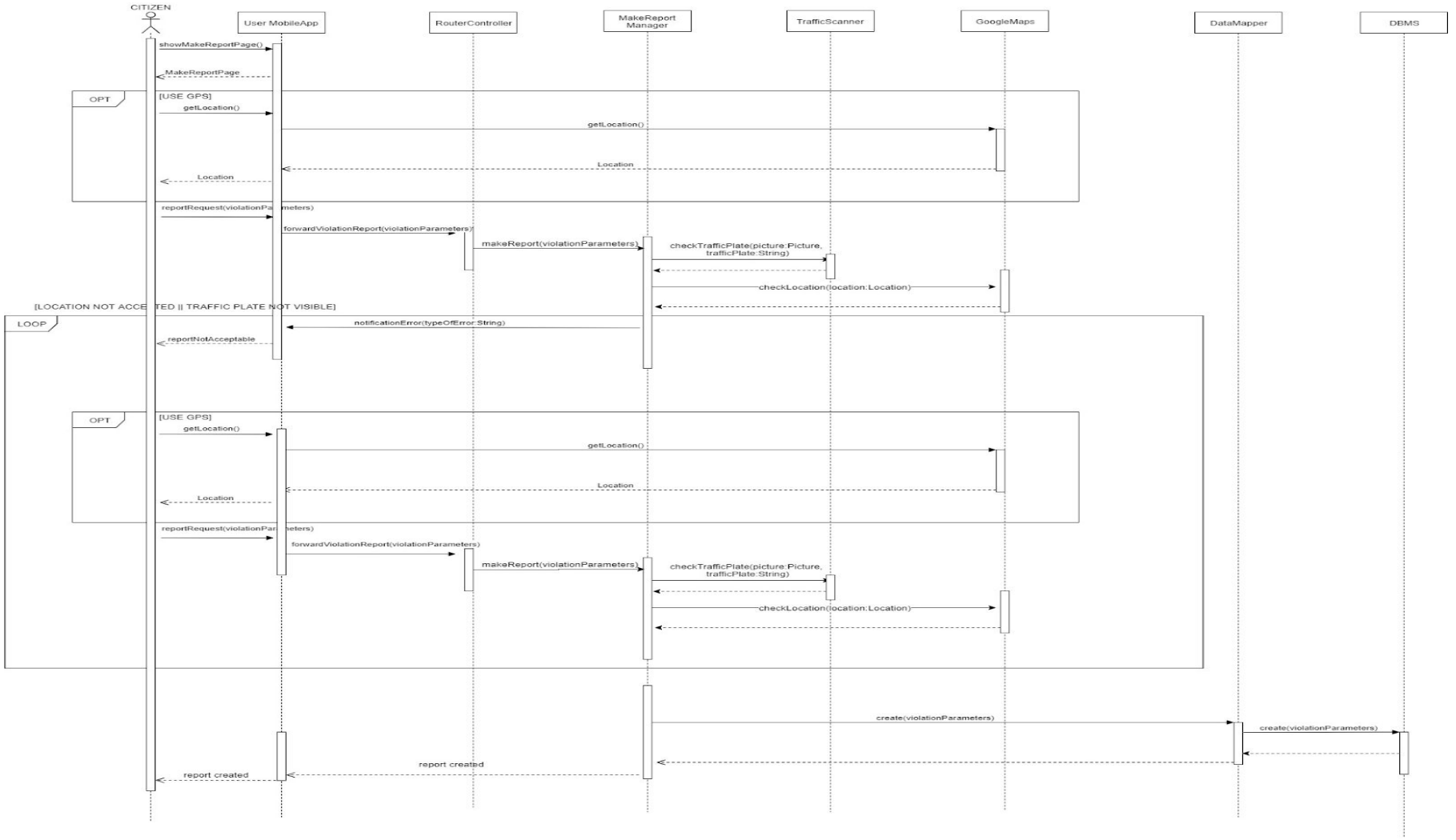


Figure 7: Make a report Sequence diagram

In this diagram the process through a citizen can send a violation report to SafeStreets is shown: at first the user must ask to the application the form to fill to send a new report; then he will fill this form with all necessary parameters (the vehicle involved, the kind of violation, the traffic plate hand written, a picture where the traffic plate is well visible and the location) in order to put a location the user has two option: using the Google Maps API or writing the exact address.
At this point the request will be forwarded at first to the special router and then to the MakeReport Manager that will check the correctness of the report exploiting two APIs: Google Maps and Traffic Scanner, there will be two different scenarios:

- something goes wrong: in this case user will be notified, telling him which parameters are wrong, and will be invited to fill the form again.
- Otherwise, if all is correct, the data will be forwarded by the MakeReport Manager to the DataMapper and then stored in the correct format into the DB.

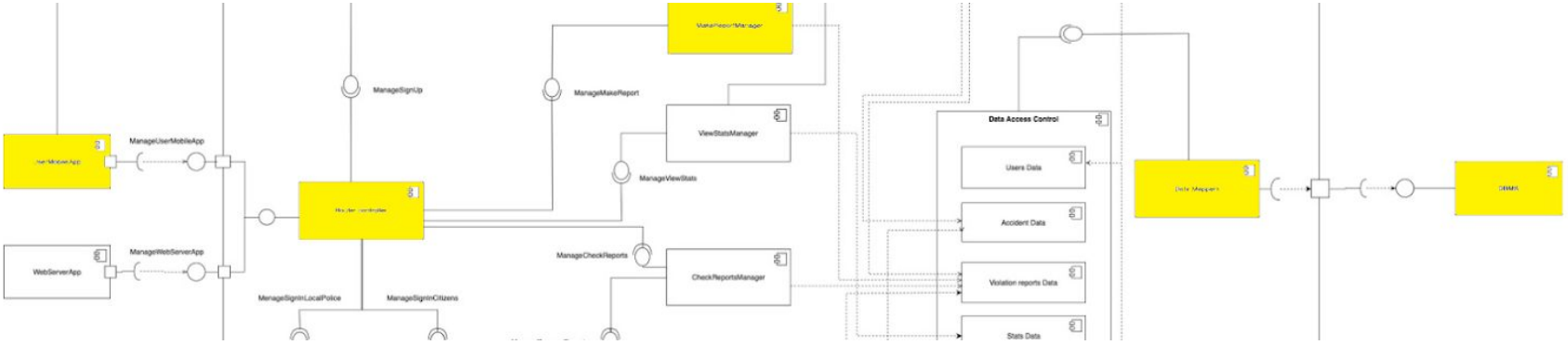


Figure 8: Components involved in the sequence Diagram “Make a report”

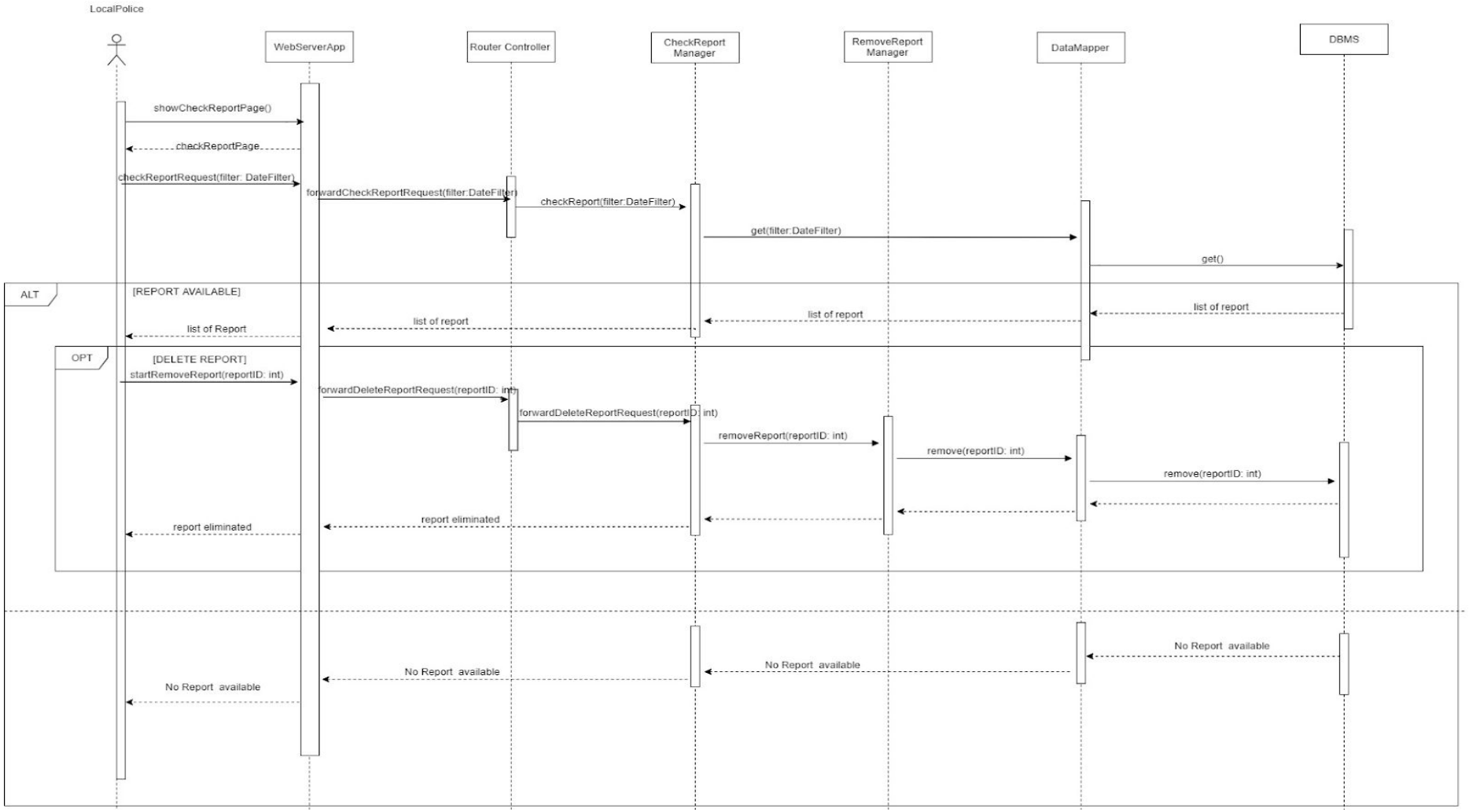


Figure 9 : Check report Sequence diagram

In this diagram the process through Local Police can check and decide to delete a report is shown. At first Local Police ask to the WebApp the page to do the request, then it ask also using a particular filter a list of reports sent by user: to do this the request is forwarded to the router and so to the CheckReport Manager, finally to the Data Mapper that will try to take the reports requested, so there will be two options:

- reports are not available: following the reverse path Local Police will be notified;
- reports are available and so data will be reported to Local Police

Now Local Police could decide to delete the report: in order to do that this request will be forwarded up(Local Police -> WebServerApp -> Router Controller -> CheckReportManager) to the DeleteReport Manager that will ask Data Mapper to delete the chosen report, identified by an ID, from the DataBase.

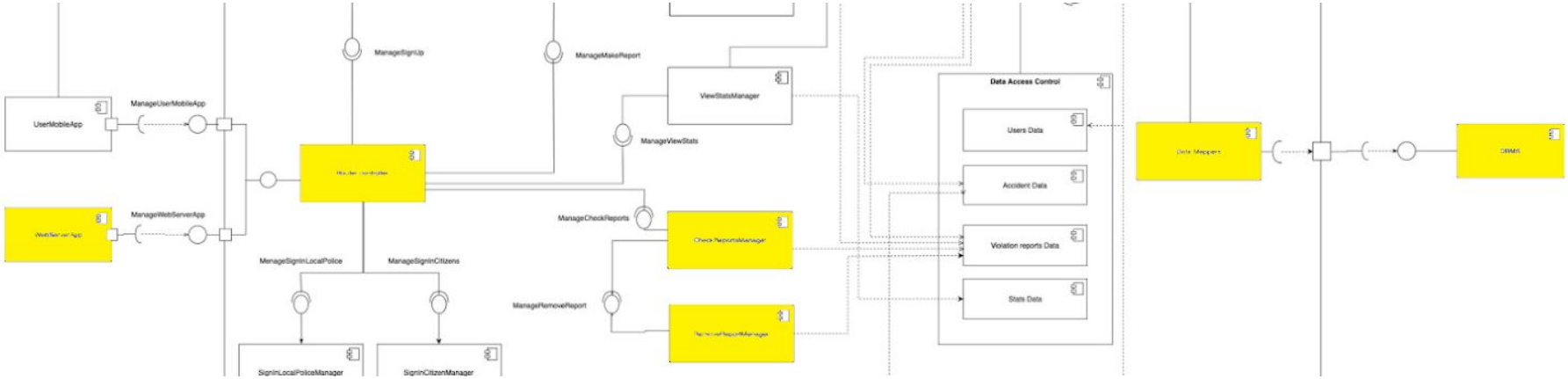


Figure 10 : Components involved in the sequence Diagram “Check report”

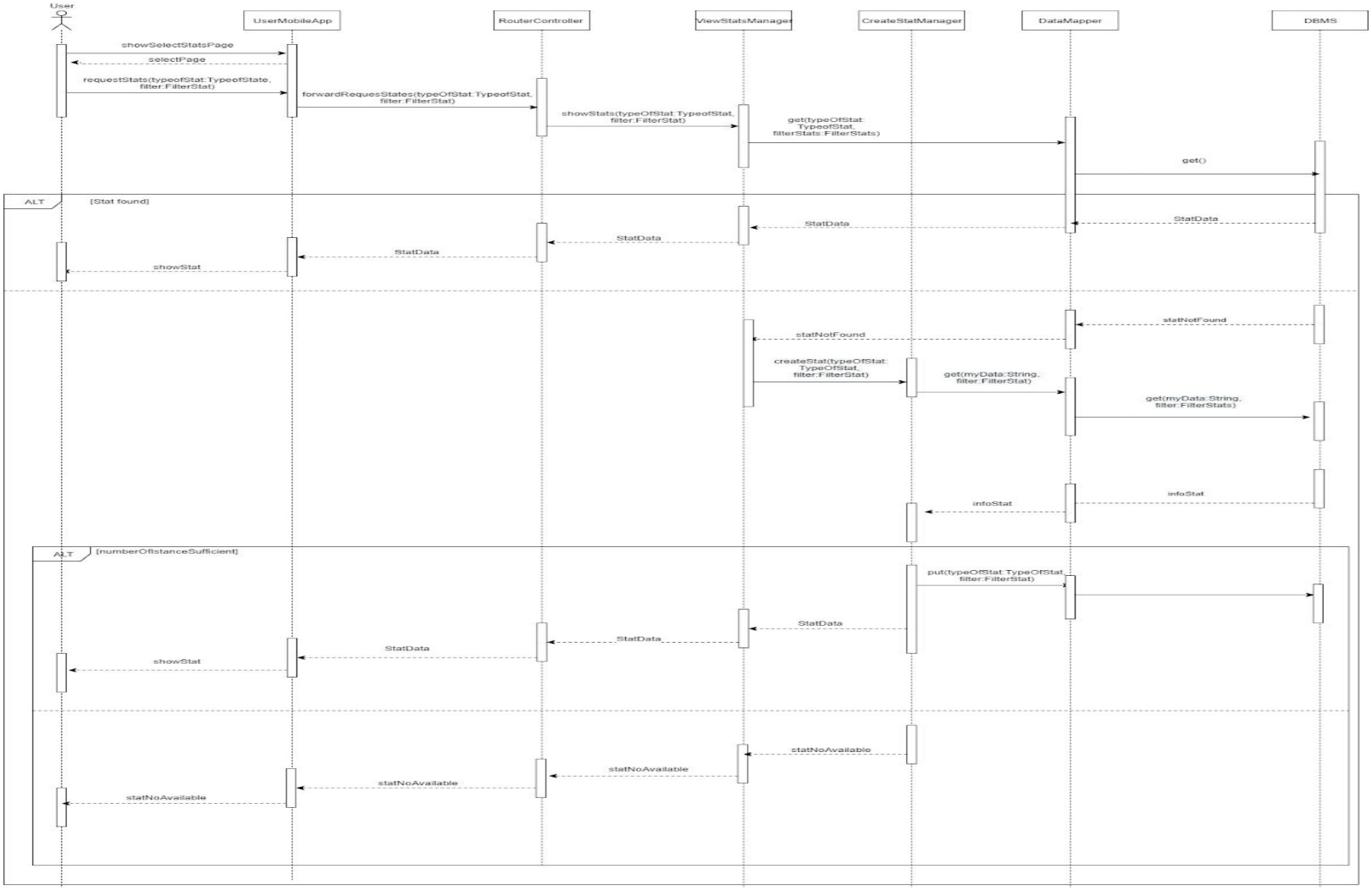


Figure 11: Request stats sequence diagram

In this sequence diagram is shown how an user (Local Police or Citizen) can access to different stats: at first, after have asked for the menù page, the application will forward to the router the request for the stat (it includes also a filter), then the request will be forwarded to the ViewStats Manager and finally to the DataMapper that will ask for data to the DB. At this point there are two different options:

- Stat is available: it will follow the reverse path in order to reach the user.
- Stat is not available: so the CreateStat Manager will be notified and will try to build the stat, also the task of update stats is manage by the CreateStat Manager. The CreatStat Manager will forward a request to the Data Mapper and so to the DB for the needed data;

  again two possibilities:

  - There are enough instances to build the requested stat: stat will be created and then, through the different components, forwarded up to the user.
  - Not enough instances: user will be notified that the request is not satisfiable.

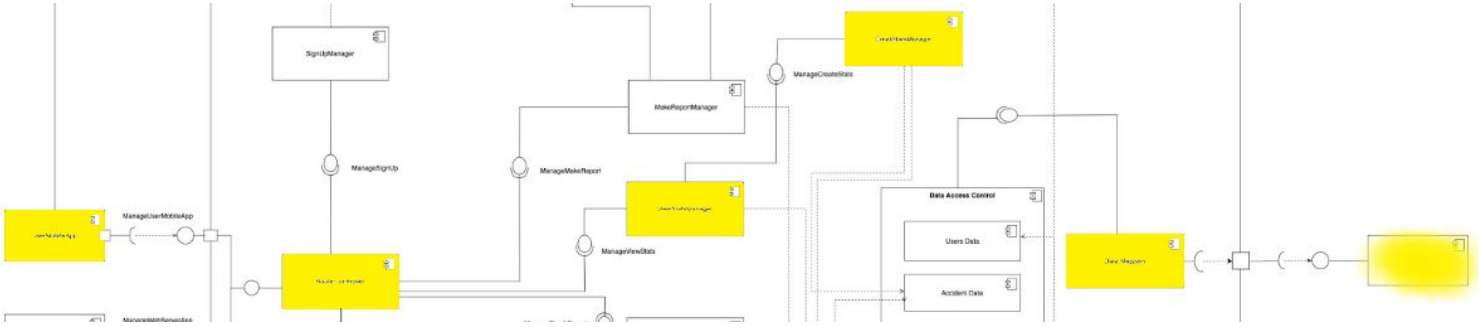


Figure 11: Components involved in the sequence Diagram “Request stats”

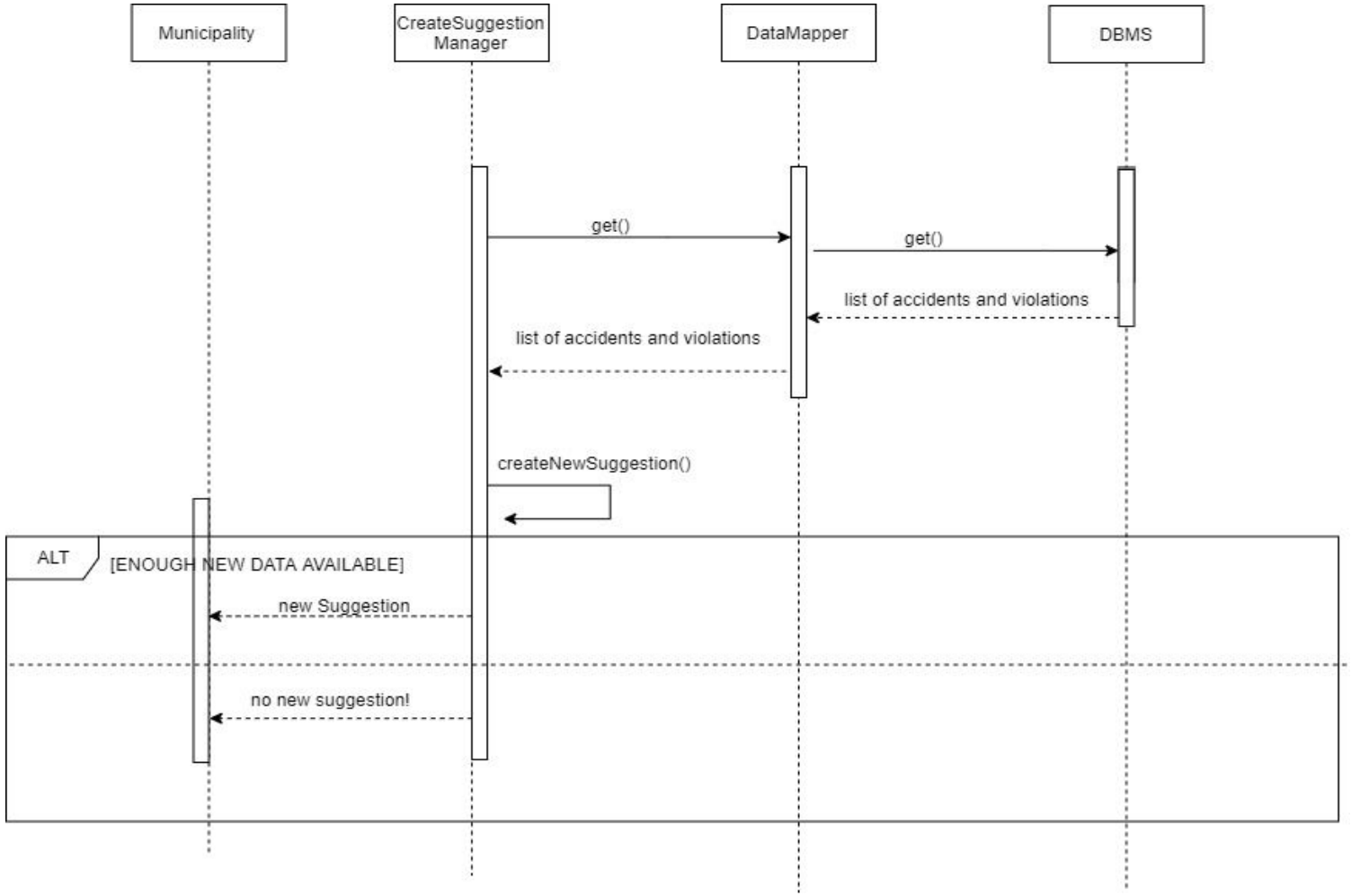


Figure 12: Create suggestion sequence diagram

This simple routine is activated when there is a timeout. The routine is activated by the CreateSuggestion Manager that will ask for data to the DataMapper and so to the DB, at this point the CreateSuggestion Manager, if checking the data received will verify that there are sensitive new data, will simply provide new suggestions calculated to the Municipality, else will notify that there are not new suggestion.

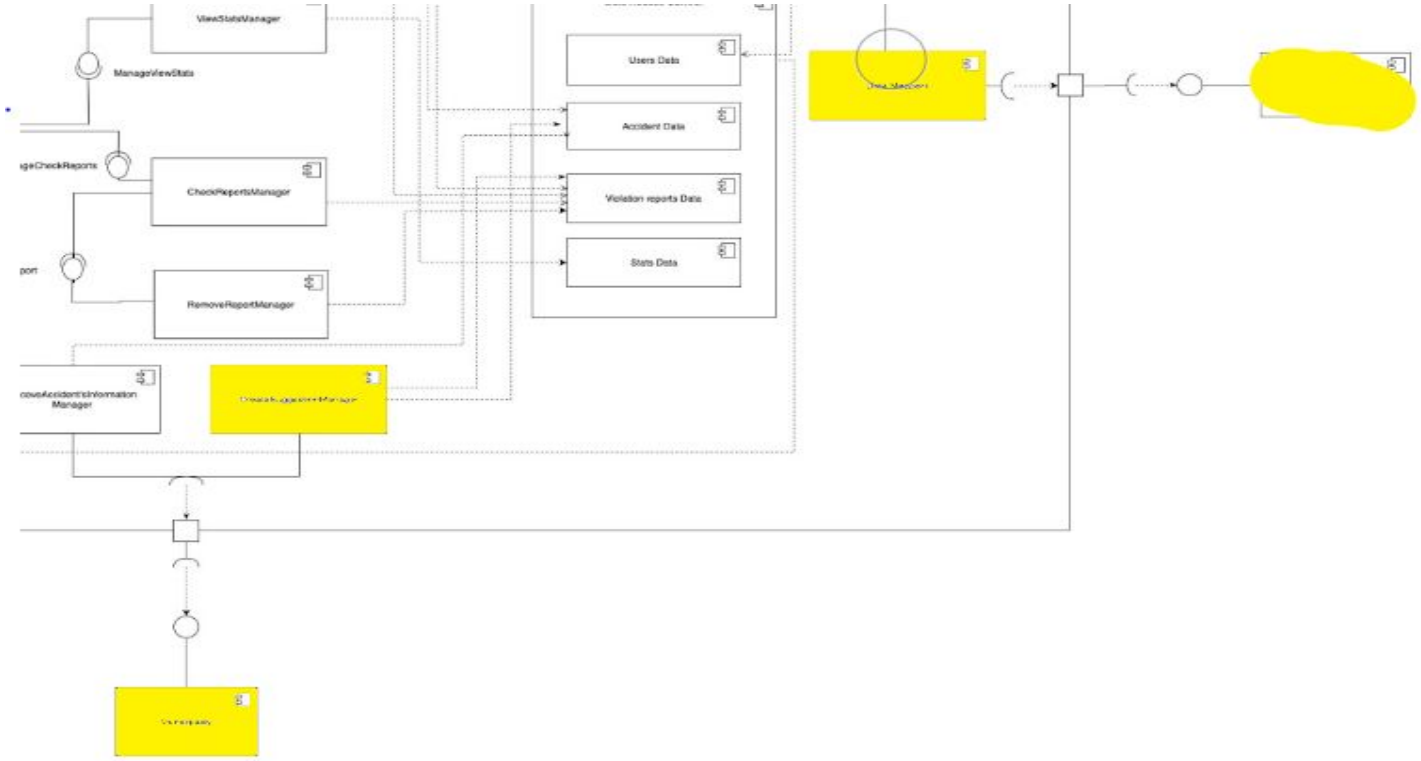


Figure 13: Components involved in the sequence Diagram “Create suggestion”

2.6 SELECTED ARCHITECTURAL STYLES & PATTERNS

REST

We have decided to develop the system following the Rest architectural style in order to build a loosely coupled application over HTTP. The goals in the deployment of our system (scalability, reliability, robustness ecc) fits perfectly with this style.

REST defines 6 architectural constraints:

1. Uniform Interface: It simplifies and decouples the architecture, which enables each part to evolve independently, moreover the goal is to have a common approach to access to resources in order that being familiar with a particular API means being familiar with all the others.
2. Client-Server: client application and server application MUST be able to evolve separately without any dependency on each other.
3. Stateless: each request will be treated as new, the client is responsible for managing the state of the application, so there will be avoided some problem like the synchronization of the user session.
4. Cacheable: in this way some interaction Client-Server will be avoided, so the latency will be reduced.
5. Layered System: with this structure both flexibility and security will increase.

DISTRIBUTED DATABASE

We have chosen to use a distributed database running on different machines for many reasons: first of all we will have a greater amount of memory so that the system will be scalable and it will be available for much more users (it will also possible adding new

machines obviously), moreover having multiples node will make the application more reliable indeed there will not be a single point of failure.

DATA MAPPER

The Data Mapper is a layer of software that separates the in-memory objects from the database. Its responsibility is to transfer data between the two and also to isolate them from each other. With Data Mapper the in-memory objects needn't know even that there's a database present; they need no SQL interface code, and certainly no knowledge of the database schema.

FACADE PATTERN

Using some routers to access to the main functions of the system we have applied a sort of facade pattern: a facade is an object that provides a simplified interface to access to subsystems with more complex interfaces.

MVC

We decided to use this pattern to guarantee the reusability, the maintainability and a parallel development of the code. MVC stands for Model-View-Controller, this pattern is used to separate application’s concerns, the three main components so are:

- Model: it contains the pure application data, it contains no logic describing how to present data to a user.
- View: presents the model data to the user, it knows how to access to data but doesn’t know what they means and what user can do to manipulate them.

- Controller: it exists between view and model, It listens to events triggered by the view and executes the appropriate reaction to these events.

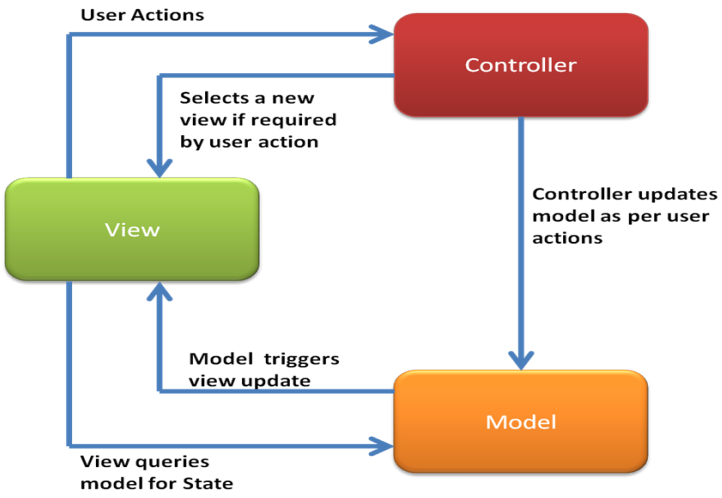


Figure 14 : MVC

2.7 OTHER DESIGN DECISIONS

- DBMS: the chosen one was MySQL since it's free and one of the most common, indeed a lots of documentation is available.
- Application Server: GlassFish 5.1 is the Open Source Java EE Reference Implementation;

3. User interface design

Here we present some UX diagrams to show the flow which the Customer will follow to navigate inside the application, according to the mockups contained in the RASD. About that, to avoid misunderstandings, it's important to underline that in the following diagrams a few windows are not shown because we want to give you a general idea of how the application works.

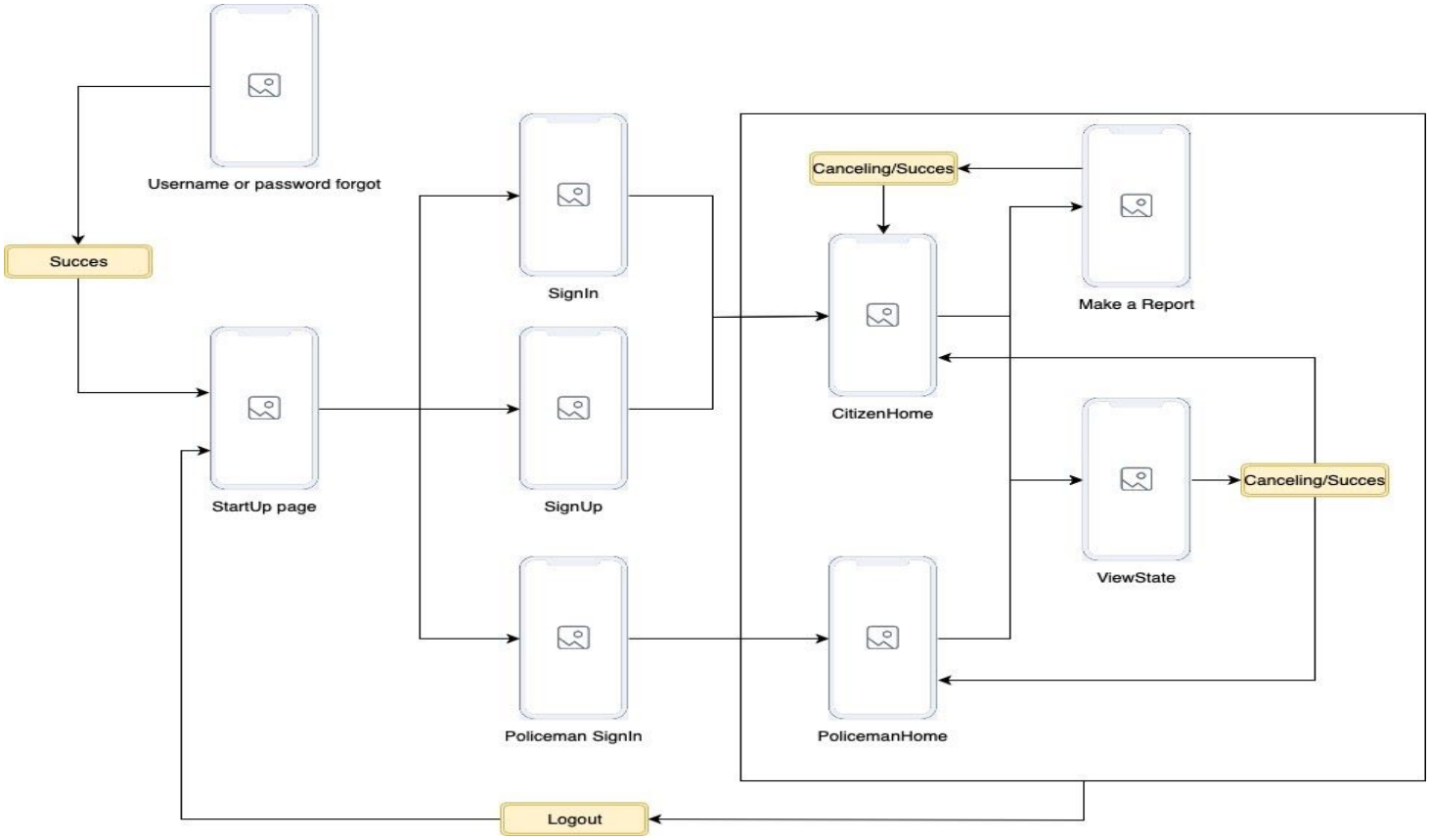


Figure 15: User interface

We felt it was better not to represent the UX diagram from WebPage because we have only startup page, sign In page, ViewStat page and CheckReport page that are not present in mobileApp.

4. Requirements Traceability

The design has been thought to guarantee that the system is able to enforce the requirements defined in the RASD and to achieve the goals. Here is described a mapping between these requirements and the design components in the application server.

1. **The system must provide to user the form to be filled:** this particular screenshot is provided directly by the userMobile App.

2. **The system must control the visibility of traffic plate:** this requirement is satisfied by the MakeReport Manager that must execute all controls on each report, in particular in this case, it will also use the TrafficScanner API.

3. **The system must accept a location both from GPS and through the manual insertion of address from the user:** this requirement is satisfied by the MakeReport Manager that is able to accept both a handwritten address and the one inserted using the Google Maps API.

4. **The system must recognize the correctness of user's information:** this requirement is still satisfied by the MakeReport Manager, because its task is to execute all controls, in the case of the location-control it will also use the Google Maps API.

5. **The system must allow to user's telephone to take a photo or load that one from the gallery:** this requirement is satisfied by the MakeReport Manager.

6. **The system must store the information taken from users' report:** this task is responsibility of the DataMapper that must transfer data, in a bidirectional way, between the in-memory and the DB.

7. **The system must process each report and classify each one depending on the vehicles which made the violation and the kind of that:** this requirement is satisfied by the MakeReport Manager that at first must to check the correctness of each report, then it has to send this report to the DataMapper that has the task to store data in the correct table of the DB.

8. **The system must store the information about accidents(occurred in the city) taken from municipality:** this task is responsibility of the DataMapper that must transfer data, in a bidirectional way, between the in-memory and the DB.

9. **The system must build security's stats crossing information taken from the municipality and the ones taken from users:** this requirement is managed by the CreateStats Manager that taking data through DataMapper can build its Stats, also allowing user to choose particular stats through a filter.

10. **The system, analyzing its stats, must provide possible suggestion to the municipality:** to satisfy this requirement is needed a routine that has as main actor the CreateSuggestion Manager, in case this latter will receive new sensitive data from the DataMapper, will be able to build a new suggestion.

11. **The system has to ask user which parameters use to filter data in order to see the areas in which he is interested in:** this particular screenshot is provided directly by the userMobile App.

12. **The system must allow the User to access to all registered data and stats highlighting the areas with the highest frequency of violations or "listing the class of vehicles "with the most violations:** this requirement is satisfied by the ViewStats Manager that will allow user to get these stats, in the case that these stats does not exist yet, they will be created by the CreateStats Manager.

13. **The system must recognize the user as a policeman:** this requirement is satisfied at the login time by the Signin Manager that for each access will ask to DataMapper user's data and then will compare these data with the signin data sent by user.

14. **If the user is a policeman the system must provide him a list of each reports with its information:** this requirement is satisfied by the CheckReport Manager, that will provide a user a list of reports that belongs to a specified period inserted in the request by the Local Police.

15. **The System must allow policeman to delete a report from the DB if it's not truthful:** In order to delete a selected report, user will provide the ID of the report to be eliminated to the DeleteReport Manager that through the DataMapper will remove that report from the DB.

16. **The system must provide citizen a list of his reports with their information:** to retrieve past report citizen will ask to the PastReport Manager, that through the DataMapper will retrieve data from the DB.

5. Implementation, integration and test plan

The system is divided in:

- UserMobileApp
- WebServerApp
- Application Server
- DBMS
- External system: Google Maps API, Traffic scanner and Municipality

In this section we are going to describe how the system will be implement, tested and integrated following a bottom-up approach. It's important to point out that this flow will be followed only for the components of the Application Server, just because both DB and External System are out from the core of the application, so they are considered reliable. Now we will try to provide at each step of the plan a visible application feature.

At first we are going to list all the core functionalities of SafeStreets highlighting their complexity, their importance in order to establish their priority in the implementation and testing.

FEATURE	IMPORTANCE FOR THE CUSTOMER	DIFFICULTY OF IMPLEMENTATION
SignUp and SignIn	Low	Low
Make Report	High	Medium
View Stats	High	Low
Create Stats	High	High

View Past Report	Low	Low
Check Report	High	Low
Remove Report	High	Medium
Create Suggestion	Medium	High

To achieve integration and testing we have chosen an incremental approach that facilitates bug tracking so as soon as a first version of component A and B are released, we integrate them and test the integration; then, if tests pass, we integrate also the first version of C that has been released in the meanwhile.

It’s important highlighting that the verification and validation phases will start as soon as the development of the system beings in order to find errors as quickly as possible.

So considering the priority of each features the components has to be implemented and tested with the following order:

- **Make Report:** this is the core function of SafeStreets and all the other ones are based on this, for this feature it is necessary to implement and produce unit test on the “MakeReport Manager” and the needed part of “DataMapper”, so at this point “MakeReport Manager” and “DataMapper” has to be integrated in order to have a correct mapping between the objects in the in-memory and the data into the DataBase; the integration has to be tested.

- **Remove Report:** this is one of the most important functions, because only the “Remove Report” can guarantee enough accurate stats. To implement this function we have to implement and produce unit test for the “RemoveReport Manager” and the needed part of “DataMapper”, then the two components has to been integrated and the integration has to be tested.

- **Check Report:** the Remove Report is based on this function, so for the reasons listed before also Check Report is very important. To implement this function we have to implement and produce unit test for the “CheckReport Manager” then the component must be integrated with the “DataMapper” and the integration must be tested. So now we have to integrate the component also with the “Remove Report”, obviously also this integration must be tested.

- **Create Stats:** this is one of the other reasons of being of the application, to make this function available is necessary to fully implement and produce unit test for the “CreateStats Manager” and the needed part of “DataMapper”, then the component must be integrated and the integration must be tested.

- **View Stats:** to let user able to access to stats the “ViewStats Manager” must be implemented and unit test must be produced, then this component must be integrated at first with the “DataMapper” and, after have tested the integration, with the “CreateStats Manager”, also this integration must be tested.

- **SignIn & SignUp:** these are the easiest functionalities of the System, they are not so important and so their components (“SignIn Manager” and “SignUp Manager”) can be implemented and tested in any order, then they must be integrated with

the “DataMapper” (after have implemented and tested the needed part) so that the login credentials can be stored into the DB and can be retrieved for each user access. The integration must be tested.

- **View Past Report:** this is not a primary-importance function, it just helps user to keep track of his past reports. In order to that the “PastReport Manager” must be implemented and unit test must be provided. Afterwards the component must be integrated with the “DataMapper” and this integration must be tested.

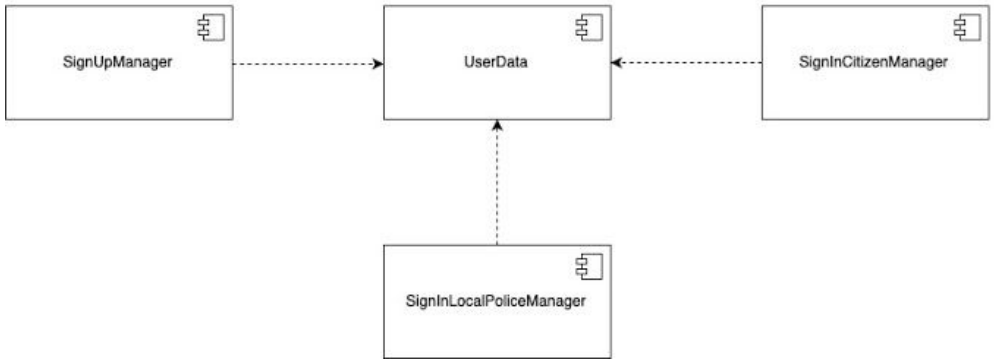
- **Create Suggestion:** this function involves the Municipality, in order to make this function available the “CreateSuggestion Manager” must be full implemented and tested, then it must be integrated with the DataMapper that has to provide to “CreateSuggestion Manager” the updated data so that it can produce new suggestions to the Municipality. The integration must be tested.

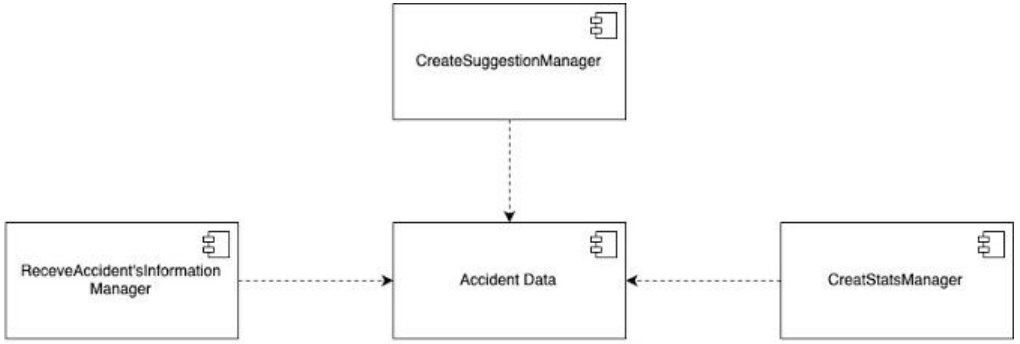
Finally the “Router” component must be implemented and tested, then integrated with most of the other components, because even if it doesn’t contain business logic, it is fundamental for the right behavior of the application (it is used to forward all requests to the right component).

5.1 Component integration

In the following diagrams we show all integrations between components of our system. Naturally, the arrows start from the component which ‘uses’ the other one.

Integration of the internal components of the Application Server


SignUpManager
UserData
SignInCitizenManager
SignInLocalPoliceManager

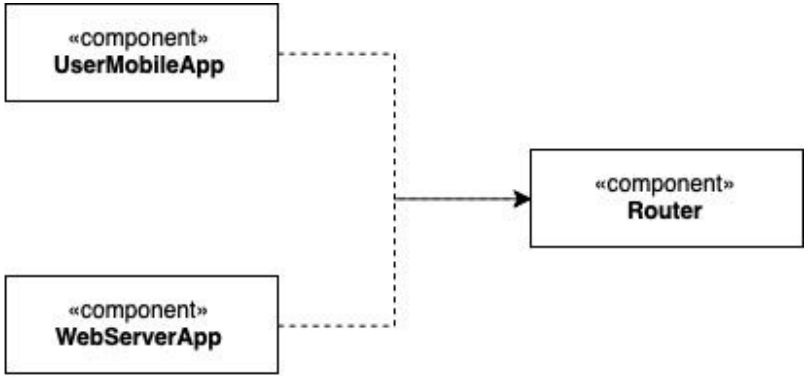

CreateSuggestionManager
ReceveAccident'sInformation Manager
Accident Data
CreatStatsManager


ViewStatsManager
Stats Data
CreatStatsManager

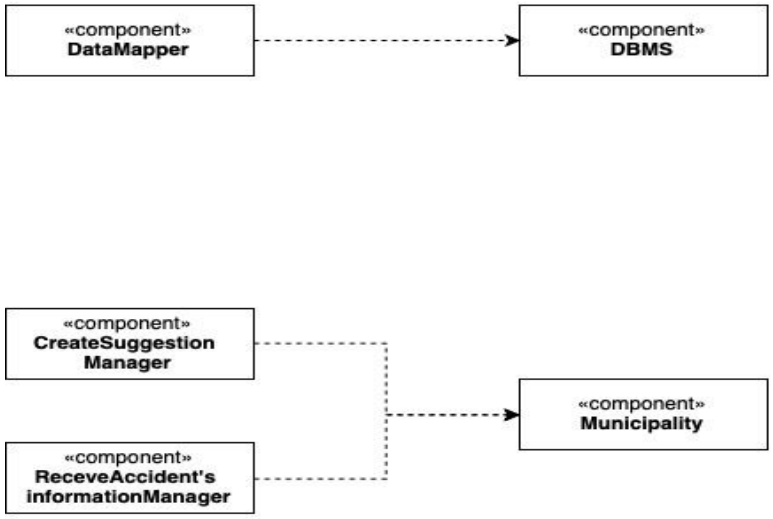

CheckReportManager
RemoveReportManager

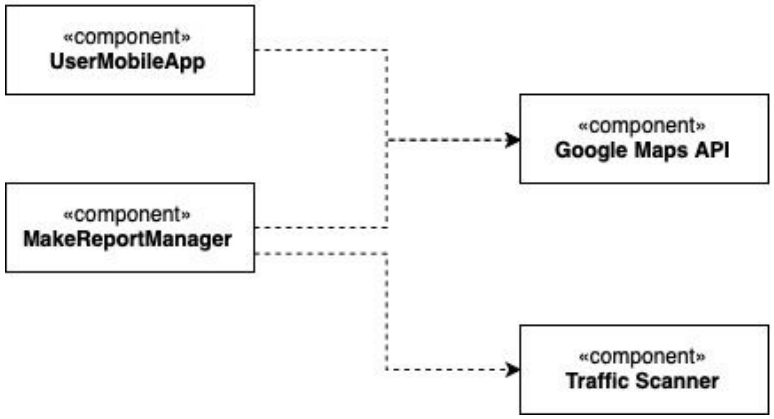

ViewStatsManager
CreateStatsManager

Integration of the frontend with the backend



Integration with the external services


«component»
DataMapper
«component»
DBMS
«component»
CreateSuggestion
Manager
«component»
ReceveAccident's
informationManager
«component»
Municipality


«component»
UserMobileApp
«component»
Google Maps API
«component»
MakeReportManager
«component»
Traffic Scanner

6. Effort spent

Giuseppe Italia

Description of the task	Hours
Introduction	0.5
Architectural design	30
Requirements traceability	0
Implementation, integration and test plan	3

Giuseppe Asaro

Description of the task	Hours
Introduction	0.5
Architectural design	30
Requirements traceability	0.5
Implementation, integration and test plan	3

7. Reference Documents

- Specification document: “Mandatory Project assignment AY 2019-2020”
- UML: https://www.uml.org/
- IEEE Standard for Information Technology-Systems Design-Software Design Descriptions