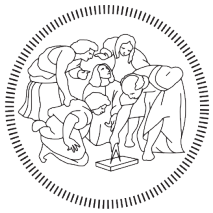

POLITECNICO
MILANO 1863

A.A. 2019/2020
Computer Science and Engineering
Software Engineering 2

SafeStreets

DD
Design Document

Bonatti Andrea Buttironi Monica
Caruso Marco Giuseppe

09/12/2019

Version: 1.0

Contents

1. Introduction

1.1 Purpose

This document is the Design Document (DD) of the SafeStreets system. Its purpose is to completely describe the logical and physical architecture of the system.

It's written for project managers, developers, testers and Quality Assurance people.

This document describes the components forming part of the system with the related run-time processes.

All the design choices are listed and motivated. In particular, this document contains:

- High-level architecture
- Main components, their interfaces and deployment
- Runtime behavior
- Chosen design patterns
- Details about the user interface and its flow
- Mapping of the requirements on the architecture's components
- Implementation, integration and testing plan

1.2 Scope

SafeStreets is a crowd-Âsourced application that intends to provide users with the possibility to notify authorities when traffic violations occur, and in particular parking violations.

The system addresses to:

- **Generic people**:Registered and Unregistered User
- **Authorities**: Local Officer and Municipal Employee

An unregistered user must be able to register to the service, becoming a registered user.

A registered user must be able to send reports about parking violation (which include date, time, position, type of violations and a picture of the vehicle), to see his/her past reports and to see valid violations occurred in a certain area (unsafe area).

An authority must be able to mine reports concerning his/her municipality and retrieve statistics about reports and issued tickets. In particular, a municipal employee must also be able to get informations about possible improvements to be realized on the municipal territory (starting from reports and accidents occurred) and to eventually set them or some of them as "done", while a local officer must be able also to validate reports.

SafeStreets data about valid reports are shared with the municipal ticket service in order to issue tickets, so the system must guarantee that the chain of custody of the information coming from the users is never broken.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **Report:** Collection of data that represents a violation, in particular
  - Picture: a photo of the vehicle that has been found committing a violation
  - Date: the date when the picture has been taken
  - Time: the hour when the picture has been taken
  - Position: the place, formatted using GPS location, of the vehicle that has been found committing a violation.
- **Improvement:** a possible road intervention finalized to the development of the road and to achieve a safer environment
- **Token:** string provided to the client as an answer to the login request, which contains information about the user and will always be part of the following requests
- **Pseudo-report:** report containing only violation type, position, date and time

1.3.2 Acronyms

- UU = Unregistered User;
- RU = Registered User;
- ME = Municipal Employee;
- LO = Local Officer;
- S2B = Software to Be;
- TS = Ticket Service;

- MS = Map Service;
- MAS = Municipal Accident Service;
- OCRS = OCR Service;
- VT = Violation type;
- DMZ = demilitarized zone;
- DBMS = database management system;
- I&T = implementation and testing;
- DMS = degrees minutes seconds;

1.3.3 Abbreviations

- [Rn] : n-th functional requirement

1.4 Revision history

- Version 1.0
  - Initial release

1.5 Reference documents

- *Specification document:* "Mandatory Project Assignment AY 2019/2020"

1.6 Document structure

The DD document is composed by seven chapters:

Chapter 1 is an introduction and describes the purpose of the document and defines the scope of the system. It also contains the description of definitions, acronyms and abbreviations used inside the document.

Chapter 2 describes the architectural design of the system. It's the core section of the document and contains a description of the high-level components and their interactions, the deployment of the system along with some dynamical views of the components, the selected design patterns and architectural styles and other design decisions.

Chapter 3 describes a UX model for the user interface (presented in RASD) for normal users and authorities

Chapter 4 provides the requirements traceability, specifying how the requirements identified in the RASD are linked to the design elements (components and architectural decisions) defined in this document

Chapter 5 includes the description of the implementation plan, the integration plan and the testing plan, specifying how all these phases are thought to be executed

Chapter 6 shows the effort which each member of the group has spent working on the project

Chapter 7 contains eventual references used during the writing of the document

2. Architectural description

2.1 Overview

The product is a distributed application based on the three logic layers of

Presentation manages the user's interaction with the system

Application handles the logic of the system

Data manages the information.

Those three layers are divided onto four different physical tiers. As shown in 2.1, Presentation and Data levels reside on a single tier, while Application level is split into two tiers (Web Server and Application Server). The Web Server is in charge of serving static assets and of forwarding API requests to the Application Server, which in turn implements the business logic of the application.

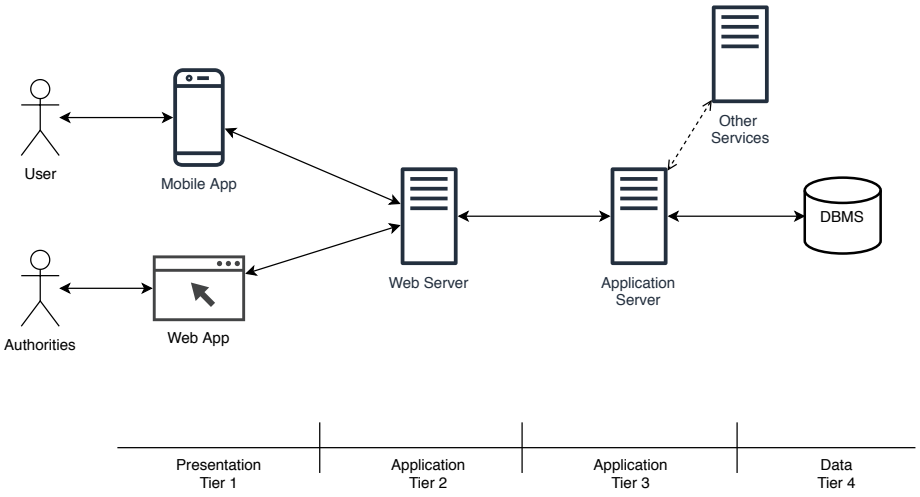

Figure 2.1: Layers and tiers diagram

In order to maximize the scalability of the system, both the Web and Application servers follow a scale-out approach: performances improvement is obtained through nodes replication. Because of this approach, load-balancing system are used in order to distribute the working load among the various nodes.

All the nodes of the Application Server use a "share everything" configuration, because there is only one shared database with one point of access.

Moreover, the Data layer is accomplished by exploiting an external DBMS service already available on the market. In this way we avoid devoting time on difficult problems about data replication and consistency, which are already solved by the existing and well tested database systems.

Every communication channel is secured by using firewalls. In this way, the Web Server is secured in a DMZ while Application Server protected as part of the LAN, so attacks and intrusions from malicious clients will be prevented. It is important to note that, because the DBMS is located on a different tier with respect to the Application Server, a firewall between them can improve the security of the system.

Finally, communication channels between the Application server and other services, like the Ticket Service, are secured for the same reasons explained above.

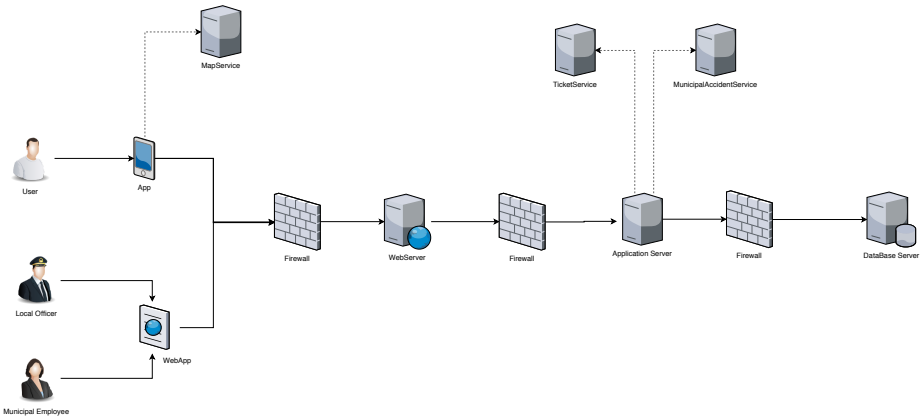

Figure 2.2: System architecture

2.2 Component view

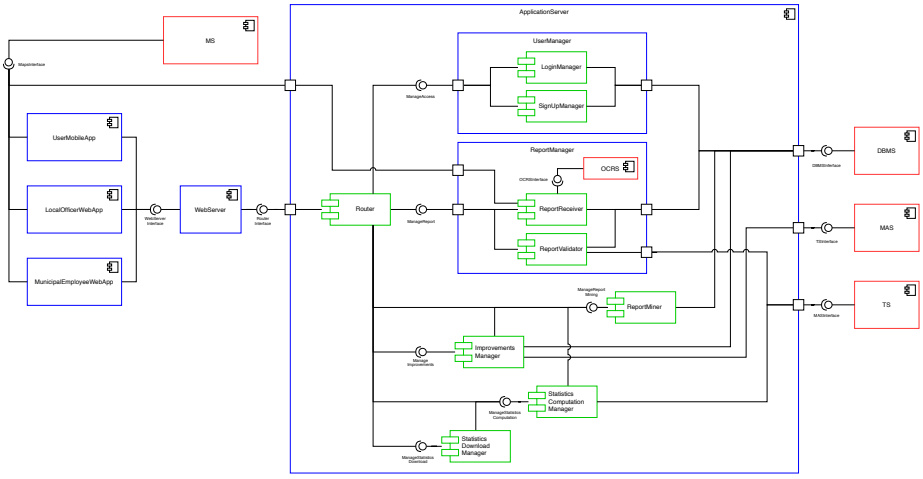

Figure 2.3: Component view diagram

The Component view diagram represents, explicitly, only the components of the Application server, as they depict the main section of the system. Below, is describe in depth the function of every internal server component made ad-hoc for the system (the ones in green).

Router : receives HTTP over SSL/TLS requests following the REST paradigm (see also **??**) and forwards them to the others internal components of the Application server. Then, it forwards the replies back to the clients. It is relevant to note that temporary tokens are adopted, in order to define which functionalities are accessible for each client. Every client receives its personal token after the login procedure.

UserManager :

- **LoginManager**: this component is responsible for granting access to registered users. In particular, the component receives the access credentials and, if they are correct, it returns a unique token used for further communication by the user. otherwise it returns an error message
- **SignUpManager**: this component is responsible for the registration of unregistered users. In particular it receives the new access credentials and, if they are acceptable, it saves them in the system's database, otherwise it returns an error message

ReportManager :

- **ReportReceiver**: this component is responsible for receiving the data of new reports and storing them in the database. In particular it takes care of the following tasks:

  - employ the MS APIs to retrieve the information about which municipality the report has been created from
  - recognizing the vehicle's plate in the picture, by employing the OCRS APIs. In case of an unrecognizable plate, the report status will be set to "NOT VALID" by default and a NULL vehicle will be set, otherwise it will be "NOT VERIFIED" (which means that a Local Officer has still to prove its validity) and the correct vehicle is associated to the report
  - saving the report in the system's database
- **ReportValidator**: this component is responsible for two main operations:
  - fetching, from the database, of reports with status set to "NOT VERIFIED"
  - modification of status of a set of report, by changing it from "NOT VERIFIED" to "VALID" or "NOT VALID" according to the request sent by the local officer. Then it saves them in the database and sends them to the TS (if set to "VALID"), responsible for issuing traffic tickets

ReportMiner : this component is responsible for obtaining reports by querying the database. It is crucial to note that the request can come directly from the Router or from other components like the ImprovementsManager and StatisticsComputationManager. In both cases, the municipality (obtained by the authority requiring or from the position required) is used for filtering reports. Only reports with status set to "VALID" are fetched. Various form of mining can be performed, in particular it's is possible to mine:

- All: all the reports produced in the specific municipality, are returned
- By Type: between all the reports produced in the specific municipality only those which have the given violation's type are returned
- By Date: between all the reports produced in the specific municipality only those which were composed on the given date are returned
- By Time: between all the reports produced in the specific municipality only those which were composed on the given time, regardless the date, are returned
- By Area: between all the reports produced in the specific municipality only those which were composed in the given area, defined by a center and a radius, are returned

In each case, if there are no reports satisfying the request, an error message is returned.

ImprovementManager : this component is responsible for getting the possible improvements belonging to the requesting authority's municipality and for setting their status. In order to determine the possible improvements, it crosses the data coming from the external MAS and from the MineReports component. When a new improvement is determined, it's status is set by default to "NOT DONE" and it is saved in the database. When no

further improvements can be discovered, all those which have the status set to "NOT DONE" are returned to the requester. Moreover, through this component, an authority can set the status of a specific improvement to "DONE", after fetching all the possible one as described before.

StatisticsComputationManager : this component is responsible for building statistics, belonging to the requesting authority's municipality, about the violations and the perpetrators who cause them, by crossing the information coming from reports received by the ReportMiner and from issued tickets coming from the TS. It is important to note that statistics are always crunched on request and never saved on the server, because they can change at any time.

StatisticsDownloadManager :this component is responsible for creating a non-materialized document (which means it is not saved on server side, but only generated and sent) about the statistics, belonging to the requesting authority's municipality, by fetching them from the StatisticsComputationManager. The resulting document is returned to the caller

2.3 Deployment view

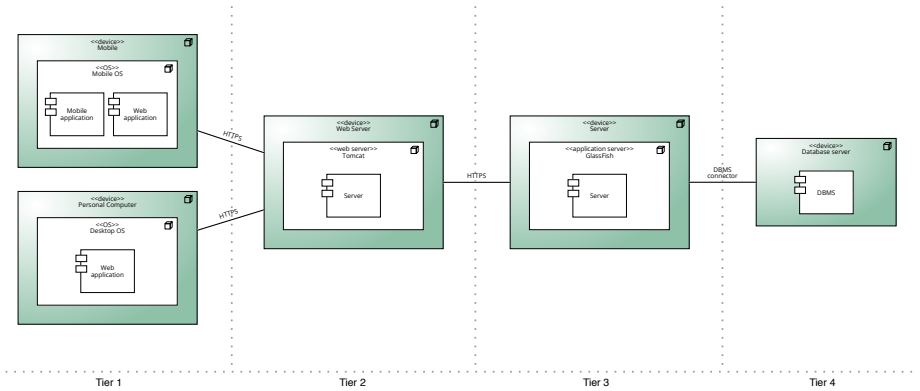

Figure 2.4: Deployment view diagram

This picture shows how the system should be deployed (ignoring external services):

- It is available for end users as a Mobile Application, while, for authorities, as a Web Application, accessible both from Mobile and PC (using a browser)
- The Web Server, deployed on its physical node, has the purpose of serving static assets without intervention by the Application Server. Any other request received from the clients (both the Web and Mobile application) is automatically forwarded to the Application Server

- The Application Server and the Database Server are deployed on two different physical nodes, in order to have more security for data and to achieve a decoupled architecture

2.4 Runtime view

This section contains the most important RuntimeView, organized by the previous depicted UseCases (see RASD document).

In order to simplify the complexity of those diagrams, we decided to describe in separate diagrams cases of access errors (which happen when the provided token does not authorize the requested operation), database's access errors and direct reply from the WebServer (the WebServer forwards every request, but in case of static contents request, it can reply directly without contacting the router if it has cached the content).

2.4.1 Registered User

2.4.1.1 Add report

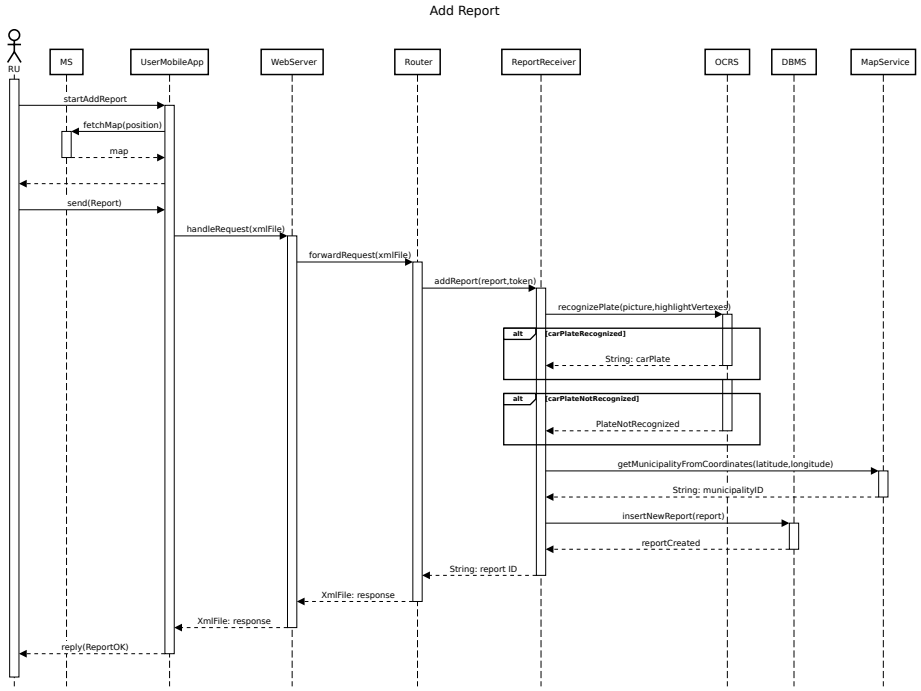

Figure 2.5: Add report runtime view diagram

In this sequence diagram the process through which a RU adds a report to the system is shown.

At first the RU chooses the "Add report" functionality on the UserMobileApp and composes the report. The app fetches a map from the MS and forwards the request to the Web Server, which contacts the Router. The Router

forwards the request to the ReportReceiver, which tries to recognize the plate with the help of the OCRS, gets the municipality where the report has been issued through the MS and, finally, adds the report to the database through the DBMS.

2.4.1.2 Get my reports

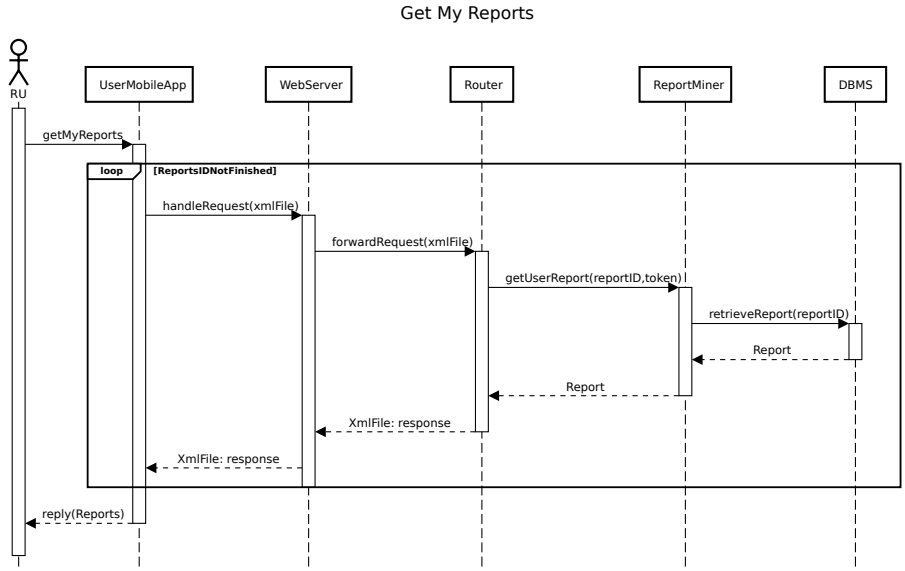

Figure 2.6: Get my reports runtime view diagram

In this sequence diagram the process through which a RU gets all the reports he/she has issued is shown.

At first the RU chooses the "Get my reports" functionality on the UserMobileApp. Then the app starts fetching all the reports issued by the user. In particular, every request passes through the WebServer, Router, ReportMiner and finally the DBMS, then it goes back to the UserMobileApp. Once all the reports have been fetched, they are presented to the RU.

2.4.1.3 Get unsafe areas

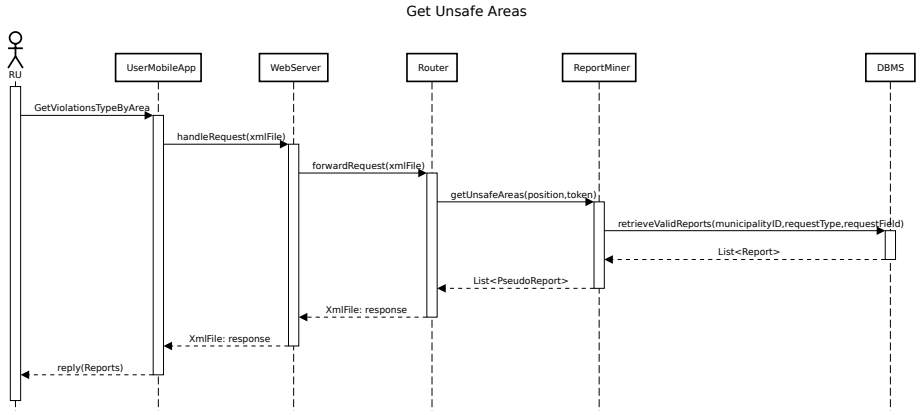

Figure 2.7: Get unsafe areas runtime view diagram

In this sequence diagram is displayed the process which permits a RU to discover all the types, dates and times of reports issued in a selected area.

This starts with the request, by the RU, that gets sent to the Web Server which will forward it to the Router. The Router will call "getUnsafeAreas" on the ReportMiner component that will query the database. When the response is ready it will go back through the same path.

2.4.2 Authority

2.4.2.1 Get statistics

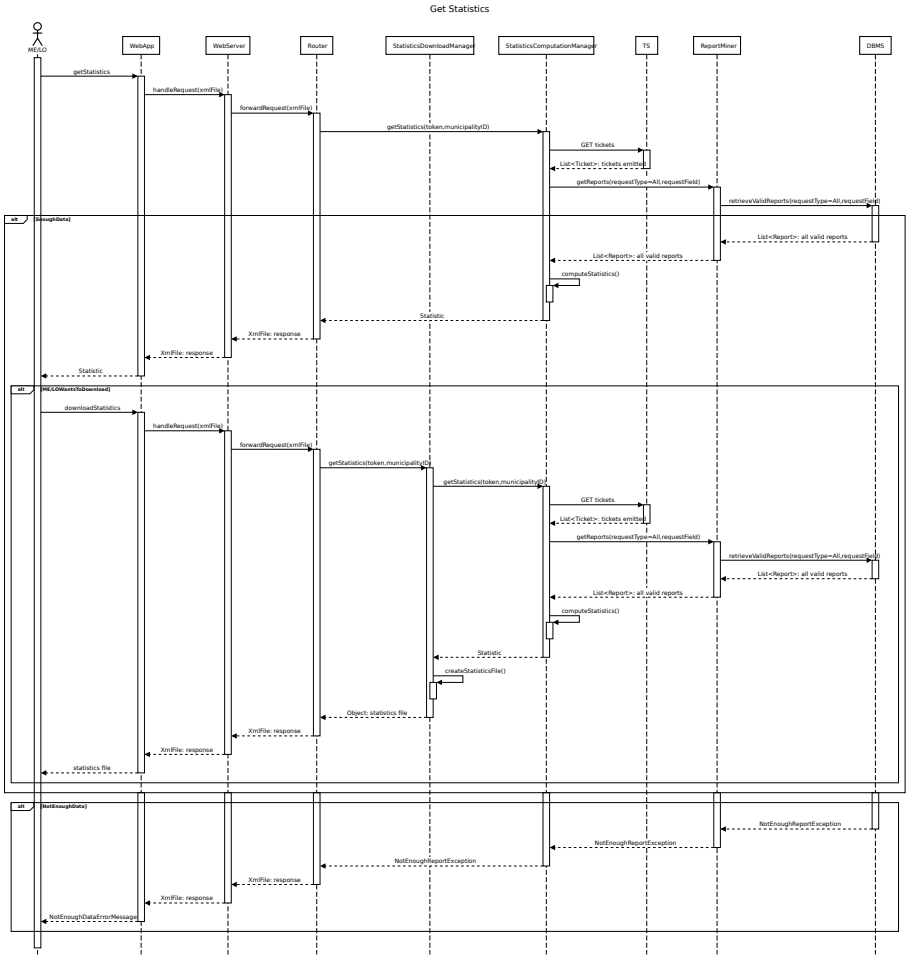

Figure 2.8: Get statistics runtime view diagram

In this sequence diagram the request gets sent to the WebServer and forwarded to the Router. Here, the StatisticsComputationManager is called. This component requests tickets data from the TS and then uses the ReportMiner component to access the database. If there are enough data, the StatisticsComputationManager generates new statistics. If the request from the authority was to only visualize the statistics, the StaticsComputationManager will send them back to the Router and to the authority, otherwise, by starting a new request, the StatisticsDownloadManager asks the StaticsComputationManager for new statistics, creates a non-materalized document and sends it back through the Router.

2.4.2.2 Mine reports

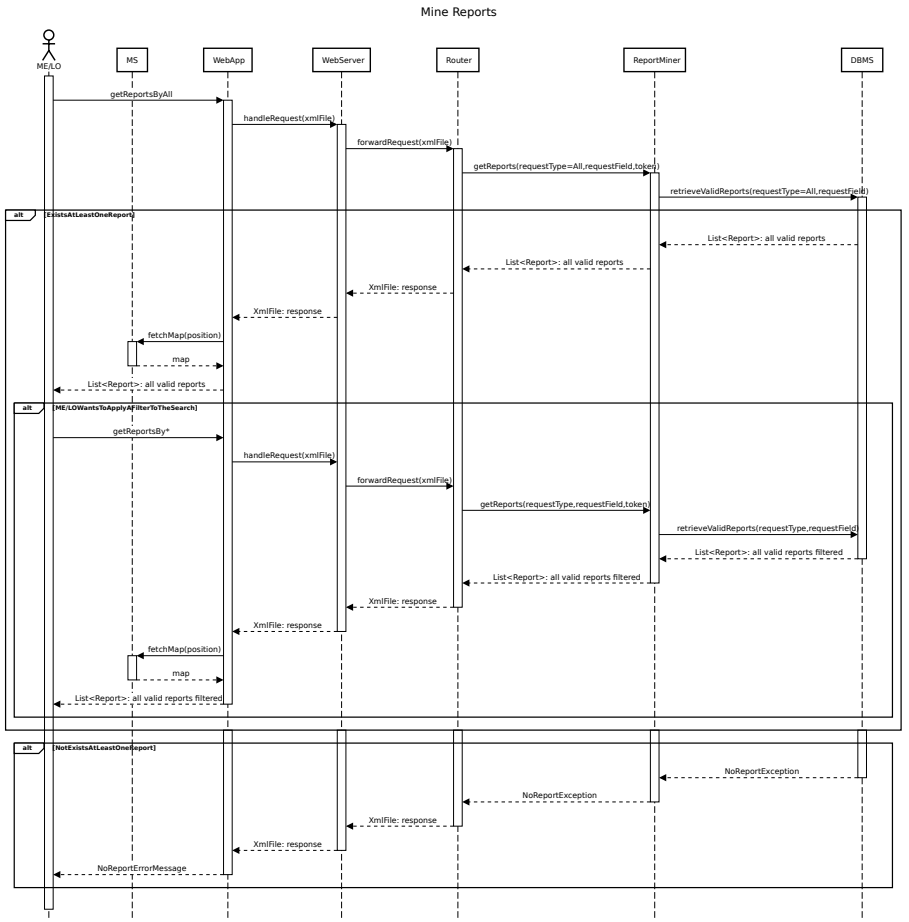

Figure 2.9: Mine reports runtime view diagram

This sequence diagram shows how the authorities can mine the reports.

At first all the reports in the authority's municipality are fetched through the following procedure. The mine request gets sent from the Web App to the Web Server which sends it forward to the Router. The ReportMiner gets called, it queries the database and, if enough reports are found, the list containing them is created and sent back to Web App. When there are less reports than needed, an error message will reach the authority. When the Web App receives the list of reports, it uses the MS to display them.

Now the requester is able to decide a filter to the search, choosing between the ones described in the ReportMiner component description. In order to fetch the filtered reports, the same procedure described before is used, but the requestType will be different from "ALL".

2.4.3 Local Officer

2.4.3.1 Validate reports

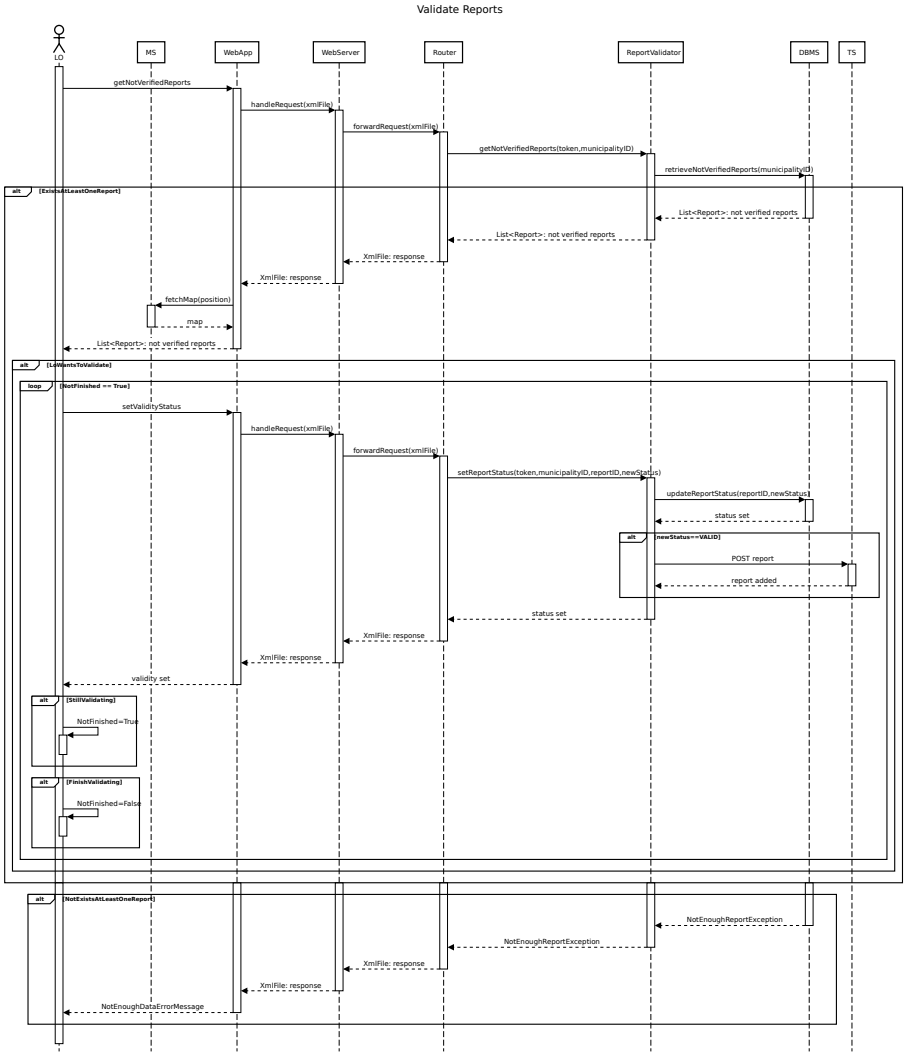

Figure 2.10: Validate reports runtime view diagram

The request gets sent with the standard route until the ReportValidator gets reached. Report validator queries the DBMS for reports to be validated, if there are none an error message is shown back to the Web App, otherwise the reports are sent back to the Web App, where they are displayed also thanks to of the MS. The LO can at this point start to validate the reports. When a report gets validated, with any result of validation, it gets sent back to the ReportValidator component which updates the database and, if the report is set as "VALID", it is also sent to the TS with a POST request. The process continues until the LO stops to validate reports.

2.4.4 Municipal Employee

2.4.4.1 Get improvements

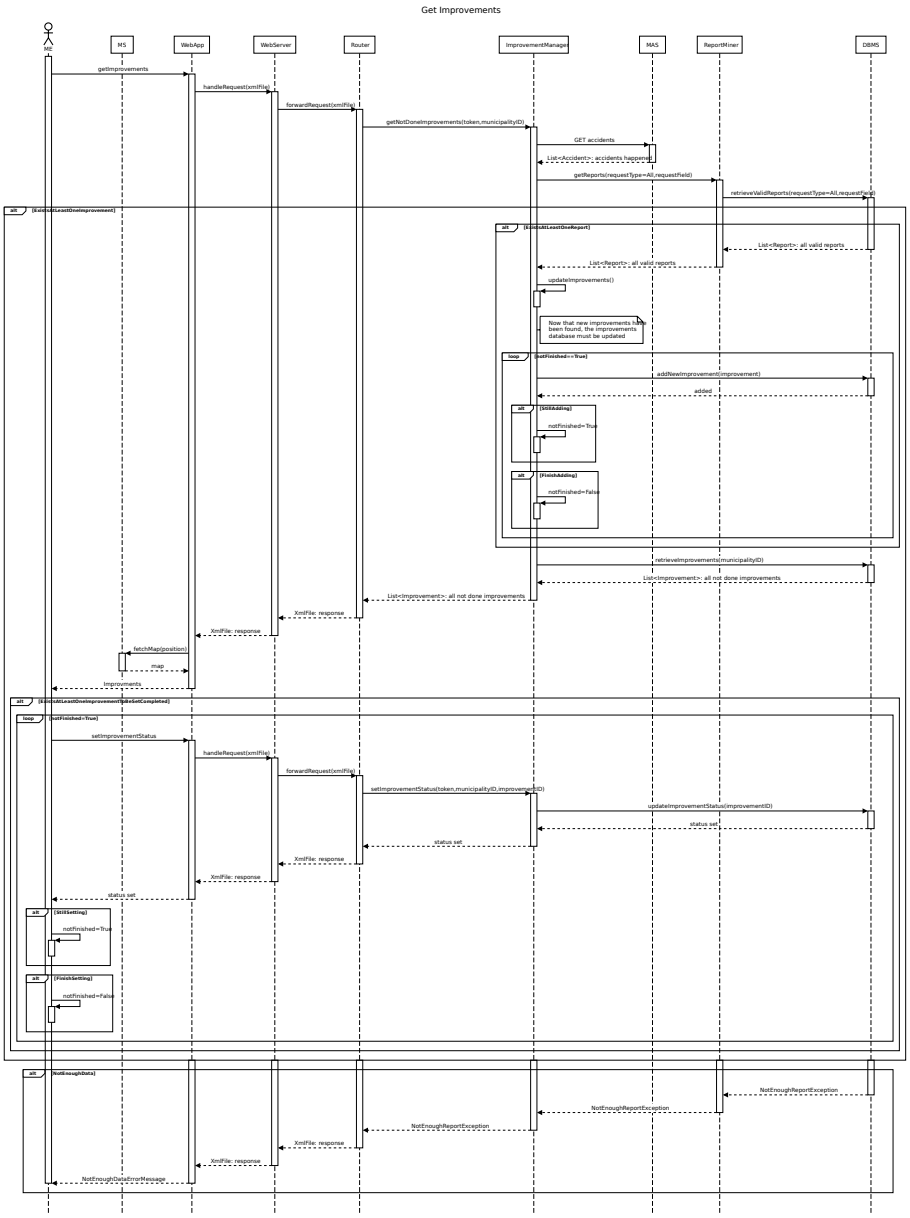

Figure 2.11: Get improvements runtime view diagram

This sequence diagram shows the request of improvements from a ME.

Using the standard route, the request reaches the ImprovementManager, which gets from the MAS, through a GET request, the list of accidents that took place in the ME's municipality. The ReportMiner then is tasked to obtain,

from the DBMS, the list of reports of the Municipality, retuning them, in case of success or returning an error to the Web App, in case of failure.

With the data received from the MAS and the ReportMiner, the ImprovementManager computes all possible improvements that, in a loop, are sent and memorized by the DBMS only if they were not already present. Finished the update, the ImprovementManager retrieves from the database, using the "retrieveImprovements" function, all possible "NOT DONE" improvements for the ME's municipality. The list of improvements is then sent back through the Router to the Web App where the ME can visualize and eventually set them as "DONE".

When at least one improvement gets its status changed, the Web App sends a request back to the Web Server and the Router. The ImprovementManager is then called by the Router to update the status of the improvement on the database.

2.4.5 Extra cases

2.4.5.1 Database access error

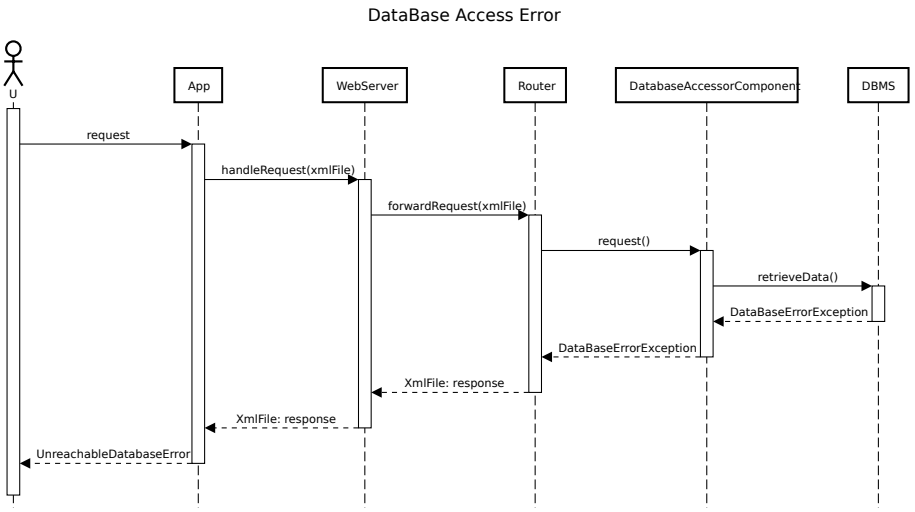

Figure 2.12: Database access error runtime view diagram

In this case a user U (which can be a RU, ME or LO) tries to use a functionality that accesses the database, but the database is not accessible.

2.4.5.2 Invalid token

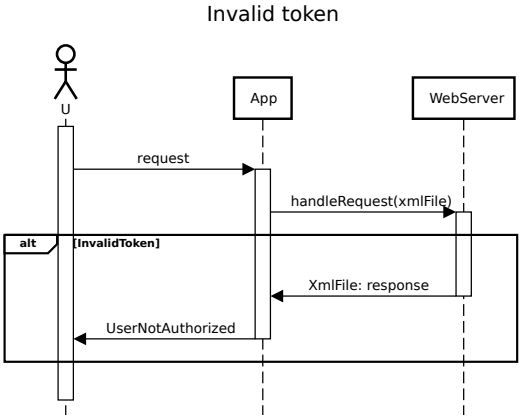

Figure 2.13: Invalid token runtime view diagram

In this case a user U (which can be a RU, ME or LO) tries to use a functionality that is not accessible with the given token (for example a RU that tries to MineReport).

2.5 Component interfaces

The following picture contains all the used interface in the system.

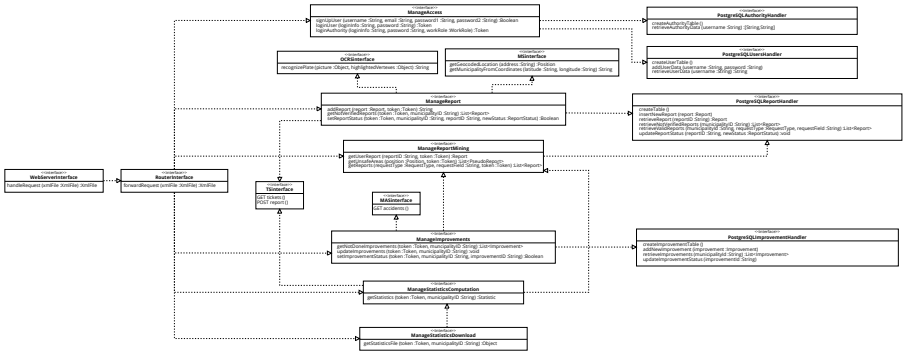

Figure 2.14: Component interfaces

As the development continued, the class diagram introduced in the RASD document has been updated. So we decided to report here the new version, with classes used by the interfaces described below.

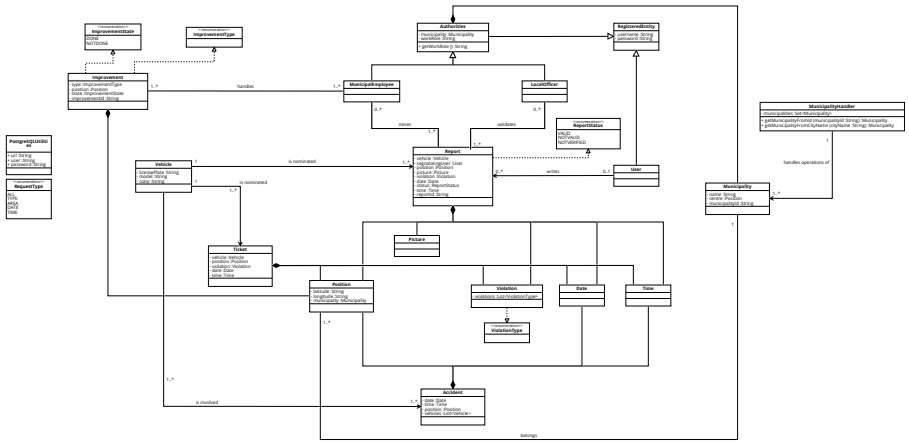

Figure 2.15: Class diagram

2.5.1 Web Server interface and Report interface

For the Web server interface and the Report interface a RESTful API has been chosen. Since both interfaces are a vital part of the system and the components that they connect belongs to the part that will be implemented, they will be discussed in depth.

2.5.1.1 REST table of resources

The following table represents the logic structures of the resources of the system and the operation that can be done on them.

URI	POST	GET	PUT	DELETE
/users/registration/?id=xxx	X	-	-	-
/users/login/?id=xxx	-	X	-	-
/users/authorities/login/?id=xxx	-	X	-	-
/reports/default	X	-	-	-
/reports/default/?id=xxx	-	X	-	-
/reports/default/unsafearea	-	X	-	-
/reports/notverified/?id=xxx	-	X	X	-
/reports/valid/?id=xxx	-	X	-	-
/improvements/?id=xxx	-	X	X	-
/statistics/visualize/?id=xxx	-	X	-	-
/statistics/download/?id=xxx	-	X	-	-

Table 2.1: REST table of resources

"X" : the operation is applicable on the resource
"-" : the operation is inapplicable on the resource

Here a quick description of the resources group:

- ***/users/**** represents the information related to the users, in particular their account information
- ***/reports/default/**** represents the resources accessible by the RU, in particular reports and "pseudo report" for the unsafe areas. These resources will be greatly used by the mobile app.
- ***/reports/notverified/**** contains all the received reports that haven't been immediately discarded by the OCRS but are still on impending evaluation by a LO
- ***/reports/valid/**** contains all the received reports that have been judged as valid by a LO
- ***/improvements/**** contains the improvements suggested to a municipality that can be retrieved by a ME
- ***/statistics/**** contains the statistics that can be retrieved by an authority

2.5.1.2 General request description

The data that will be transmitted will be composed of XML files.

To recognize the user who sent a request to the server, the system will employ tokens. A token is a string that is provided to the user as an answer to the login, it contains information on the user and will always be part of the requests, except the login and sign up. The contained information will be:

- User type information: the different type users (RU, LO and ME) will be identified in different ways to avoid ambiguity. Moreover the identifier for LO and ME will contain an identifier for the Municipality they work for
- User identifier: The single user will be identified to have information on who is making the request and give the correct permission to access data.
- Creation time: The token is a "one time only" use. Its validity is fixed and will generally last at least for a session, this permits to recycle pieces of tokens and avoids the malicious use of old ones to get data.

2.5.1.3 Detailed requests

POST /users/registration/?id=id

This request is used to register a user

Parameters

Field	Type	Description
id	String	The username of the user who is trying to register

Fields

Field	Type	Description
email	String	The email of the user
passwordFirst	String	The password of the user
passwordSecond	String	The same password as before, used to confirm the first password

Success 201 (resource created)

Error 401

Field	Description
ExistingUsername	Someone with the same username is already registered
DifferentPassword	The second password is different from the first one
ExistingMail	This email is already associated with another account

GET /users/login/?id=id
This request allows a RU to login

Parameters

Field	Type	Description
id	String	The username of the user who is trying to login

Fields

Field	Type	Description
loginInformation	String	The email or username of the user
password	String	The password of the user

Success 200 (request ok)

Field	Type	Description
token	String	A token that represents the user
reportIDs	String[]	The list of id associated with the reports uploaded by the user

Error 401 (Unauthorized)

Field	Description
WrongUsernameOrPassword	The written username and password does not correspond to any existing user

GET /users/authorities/login/?id=id
This request allows a ME or LO to login

Parameters

Field	Type	Description
id	String	The username of the user who is trying to login

Fields

Field	Type	Description
loginInformation	String	The username of the user
password	String	The password of the user
workRole	String	This will be 'ME' or 'LO'

Success 200 (request ok)

Field	Type	Description
token	String	A token that represents the user and the municipality he/she works in
municipalityID	String	The id of the municipality where the ME or LO works, this will be a parameter for the following requests

Error 401 (Unauthorized)

Field	Description
WrongUsernameOrPassword	The written username and password does not correspond to any existing user
NotCorrespondingRole	The selected work role does not correspond to the user which given login and password corresponds to

POST /reports/default
This request adds a report to the system

Fields

Field	Type	Description
vehicle	Object	The vehicle information
licensePlate	String	The license plate of the vehicle
position	Object	The position, expressed in DMS, of the vehicle when the report was submitted
latitude	String	The latitude where the vehicle was recorded to be
longitude	String	The longitude where the vehicle was recorded to be
picture	Object	Representation of the image of the vehicle
violation	Object[]	An array of the type of violation
violationType	String	The type of violation
date	String	The datetime in dd-MM-yyyyThh:mm:ss format
highlightVertexes	Object	The coordinates on the picture of where the license plate is located
vertexOne	Number[]	The coordinates (on the picure) of the top-left vertex
vertexTwo	Number[]	The coordinates (on the picture) of the bottom-right vertex

Success 201 (resource created)

Field	Type	Description
id	String	The id that the system has assigned to the sent report. This id will uniquely identify the report and will also contain information about the user which sent it

GET /reports/default/?id=id
This request retrieves a report form the system.

Parameters

Field	Type	Description
id	String	The id that uniquely identifies the report that the user wants to see

Success 200 (request ok)

Field	Type	Description
vehicle	Object	The vehicle information
licensePlate	String	The license plate of the vehicle
position	Object	The position, expressed in DMS, of the vehicle when the report was submitted
latitude	String	The latitude where the vehicle was recorded to be
longitude	String	The longitude where the vehicle was recorded to be
picture	Object	Representation of the image of the vehicle
violation	Object[]	An array of the type of violation
violationType	String	The type of violation
date	String	The datetime in dd-MM-yyyyThh:mm:ss format
highlightVertexes	Object	The coordinates on the picture of where the license plate is located
vertexOne	Number[]	The coordinates (on the picure) of the top-left vertex
vertexTwo	Number[]	The coordinates (on the picture) of the bottom-right vertex

Error 403 (Forbidden)

Field	Description
NoReportError	The requested resource caused an error on the database, this could both mean that the resource was not found on the database or that the database had internal error or an error on the connection

GET /reports/default/unsafearea
This request retrieves the type of violations in certain area

Fields

Field	Type	Description
position	Object	The position, expressed in DMS, of the center of the area which the RU wants to know about
latitude	String	The latitude where the vehicle was recorded to be
longitude	String	The longitude where the vehicle was recorded to be

Success 200 (request ok)

Field	Type	Description
pseudoReport	Object[]	The list of partial reports that can be seen bya a RU
position	Object	The position, expressed in DMS, of the vehicle when the report was submitted
latitude	String	The latitude where the vehicle was recorded to be
longitude	String	The longitude where the vehicle was recorded to be
violation	Object[]	An array of the type of violation
violationType	String	The type of violation

Error 404 (Resource not found)

Field	Description
NoReportError	The requested resource caused an error on the database, this could both mean that the resource was not found on the database or that the database had internal error or an error on the connection