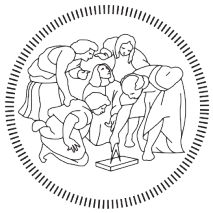


POLITECNICO
MILANO 1863

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.

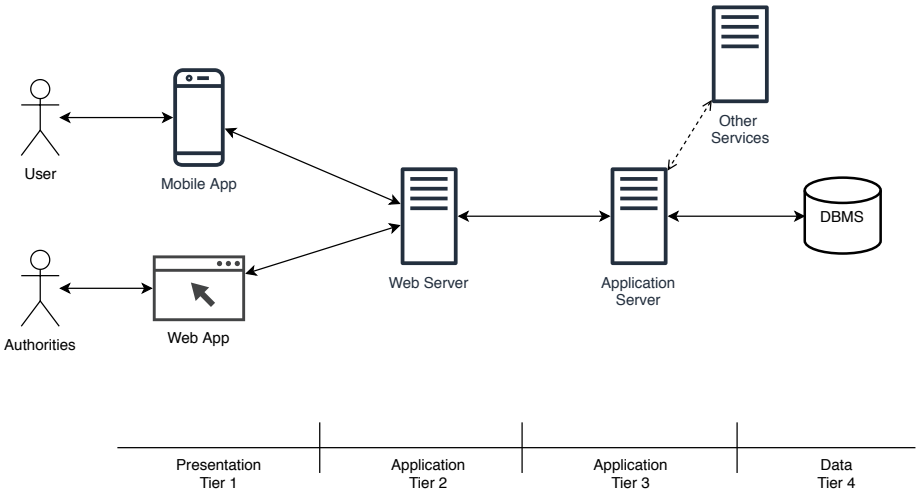


Figure 2.1: Layers and tiers diagram

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.

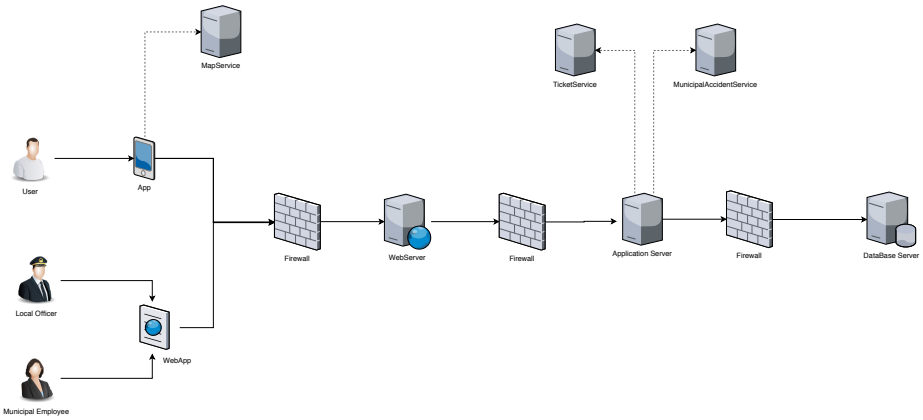


Figure 2.2: System architecture

2.2 Component view

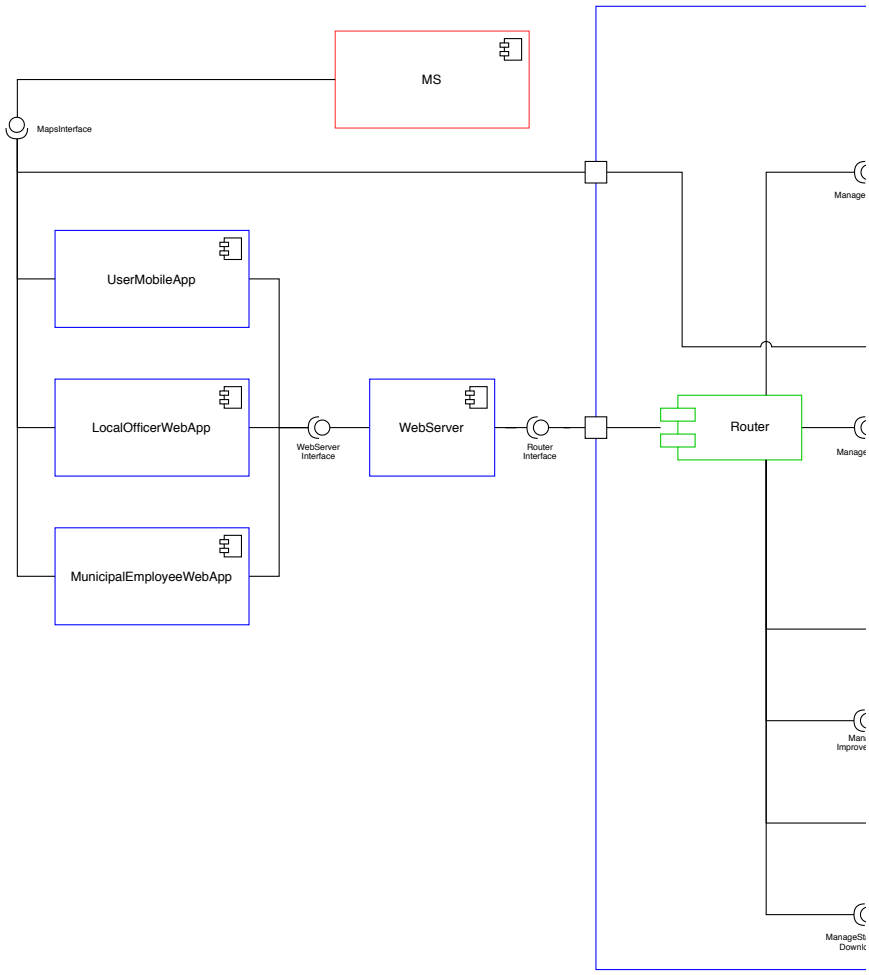


Figure 2.3: Component view diagram

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

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)

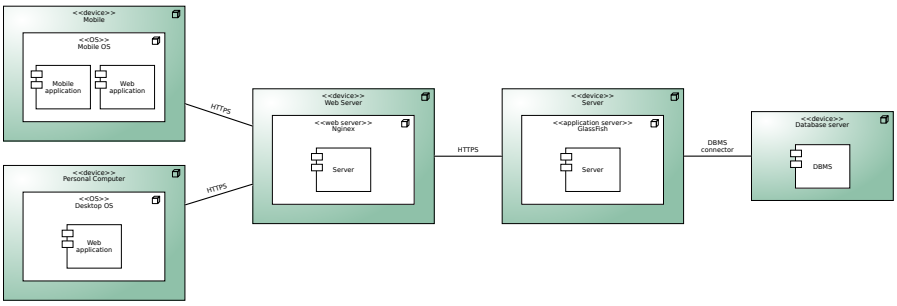


Figure 2.4: Deployment view diagram

- 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

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.

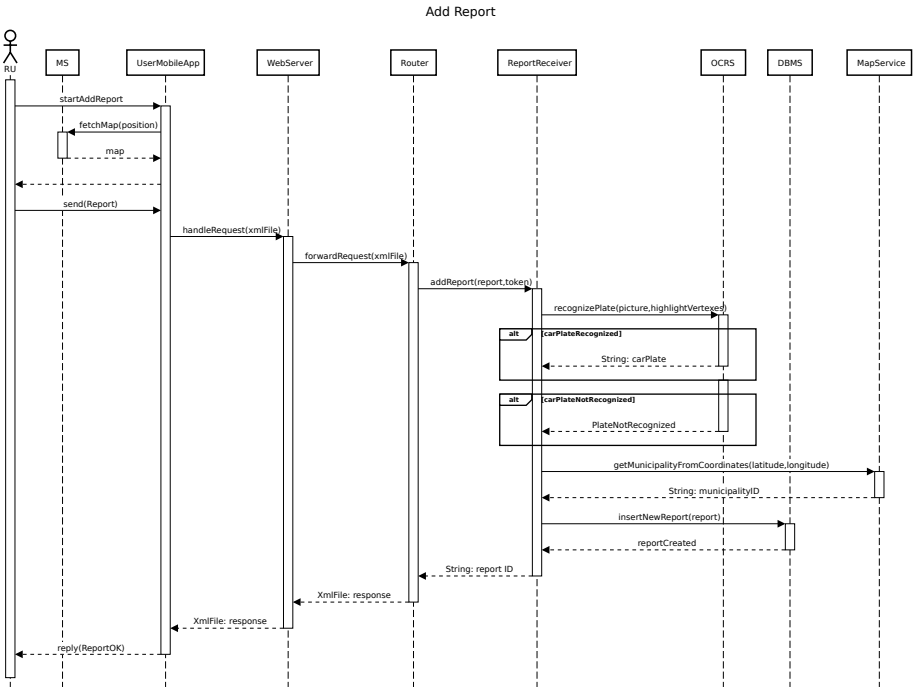


Figure 2.5: Add report runtime view diagram

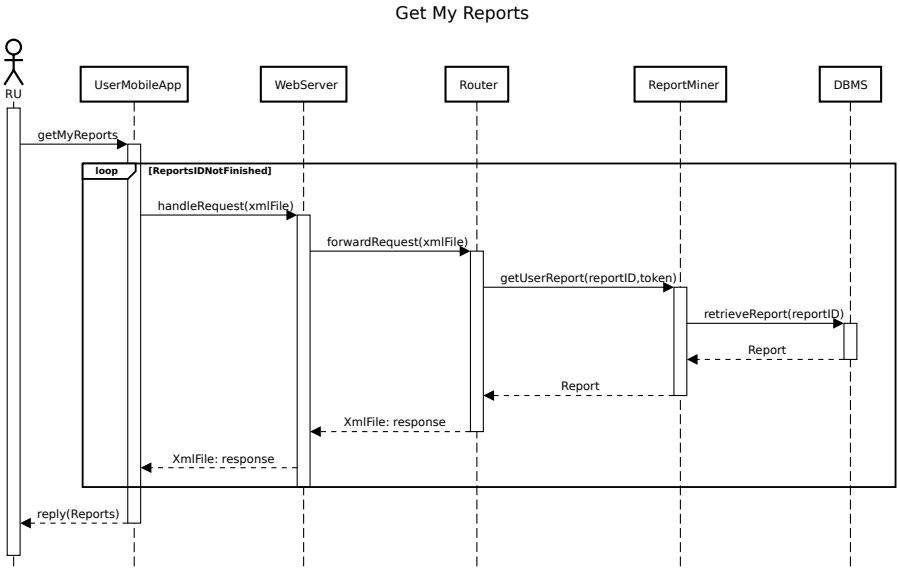


Figure 2.6: Get my reports runtime view diagram

2.4.1.2 Get my reports

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 UserMo-

bileApp. 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

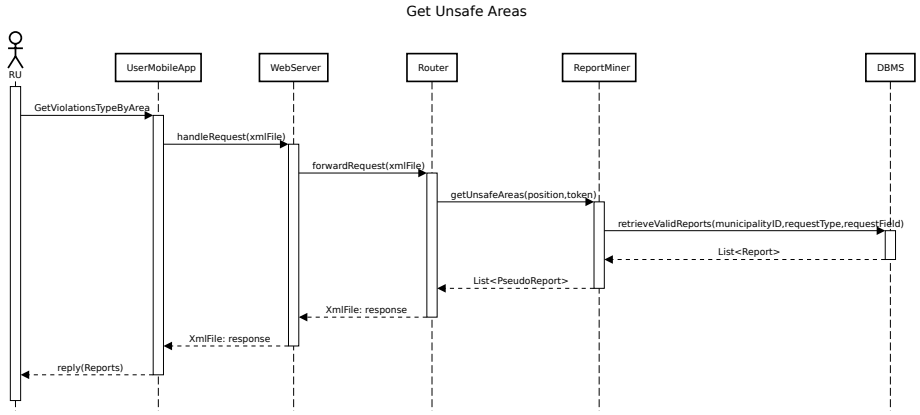


Figure 2.7: Get unsafe areas runtime view diagram

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

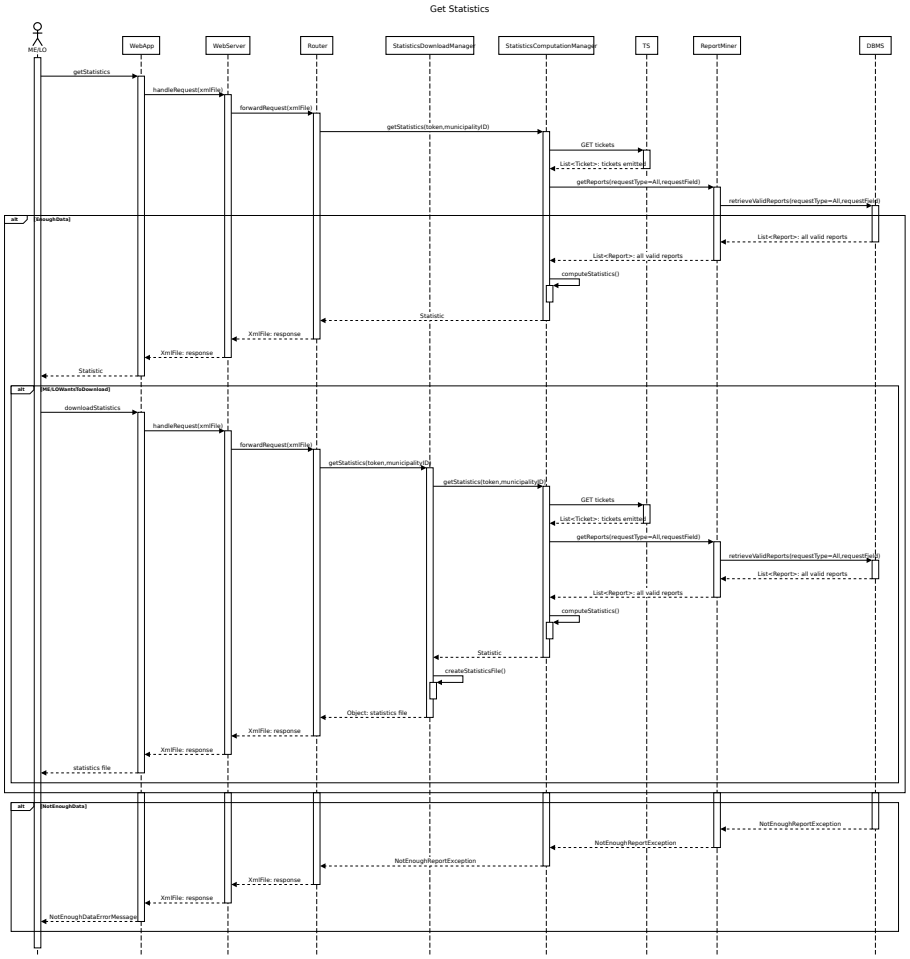


Figure 2.8: Get statistics runtime view diagram

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

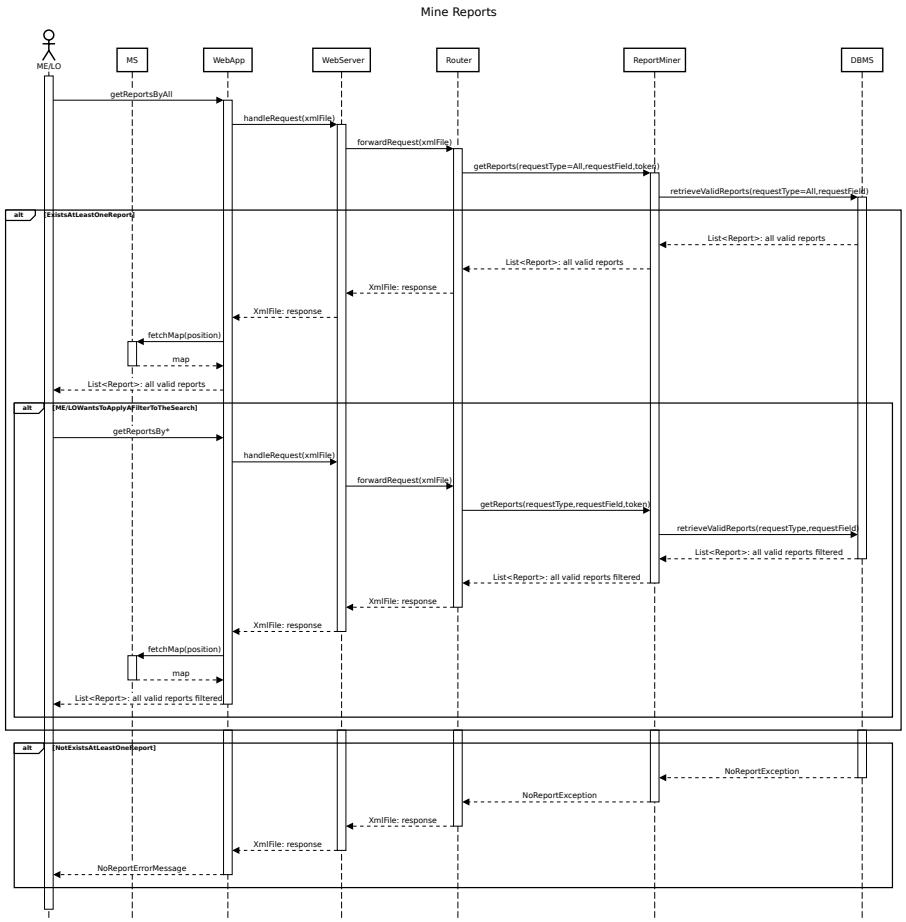


Figure 2.9: Mine reports runtime view diagram

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

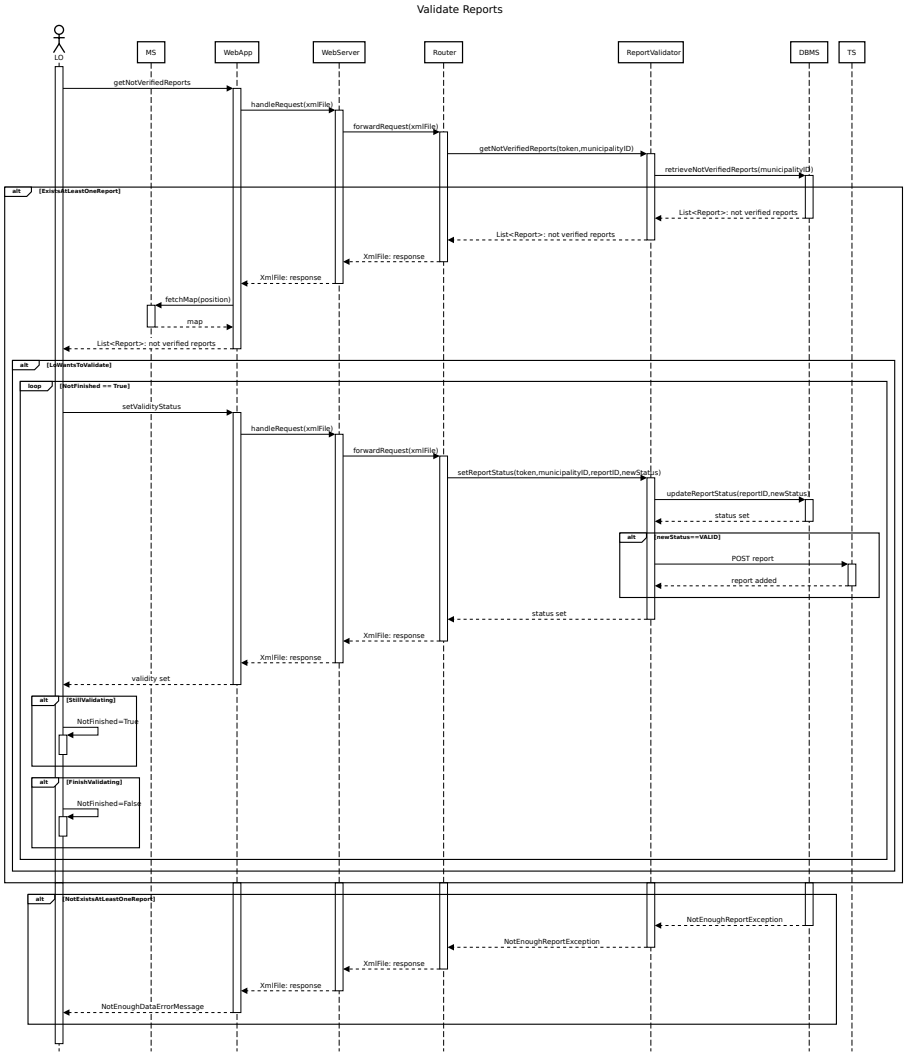


Figure 2.10: Validate reports runtime view diagram

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

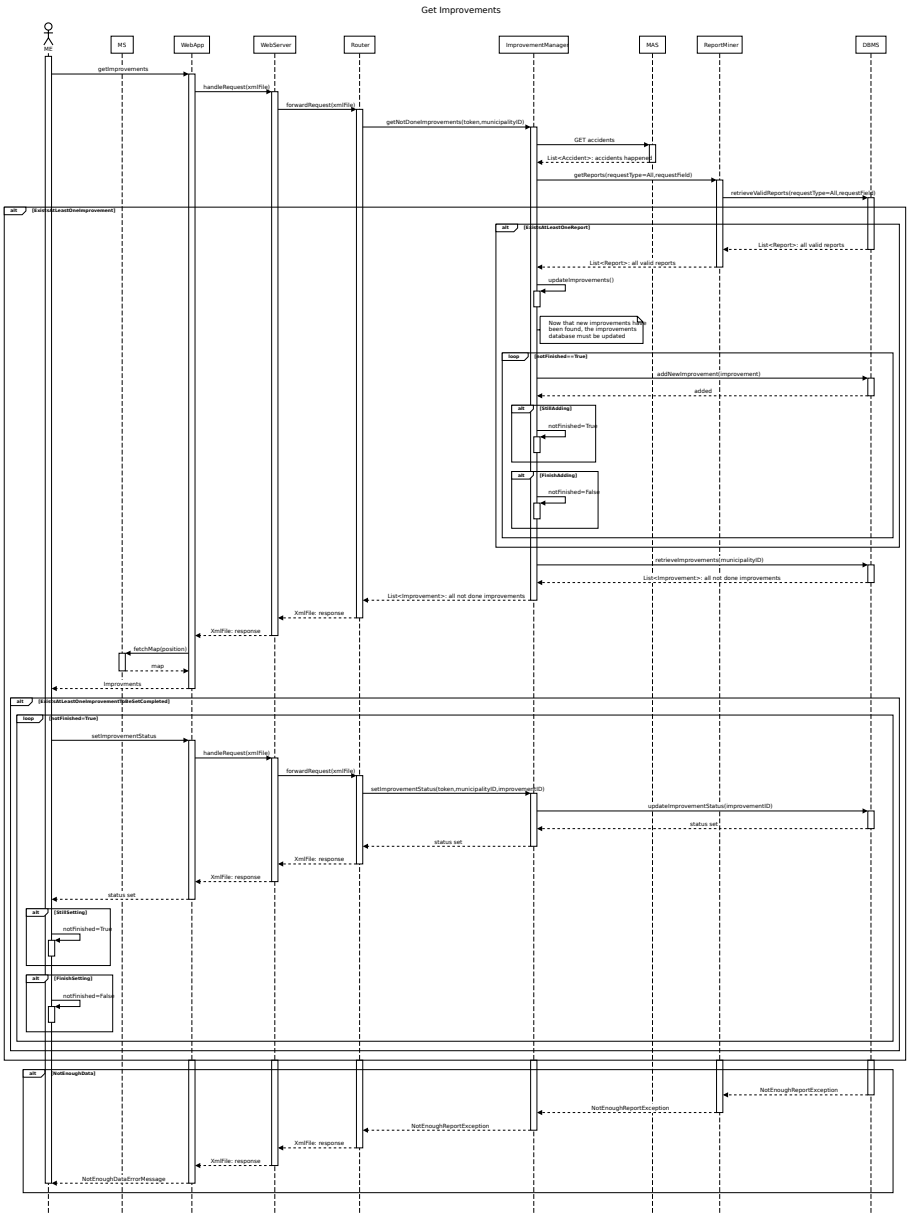


Figure 2.11: Get improvements runtime view diagram

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

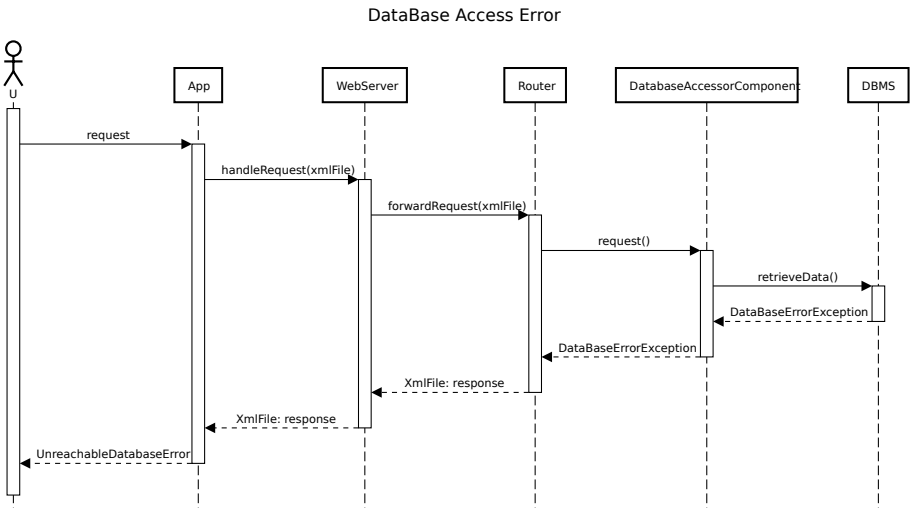


Figure 2.12: Database access error runtime view diagram

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

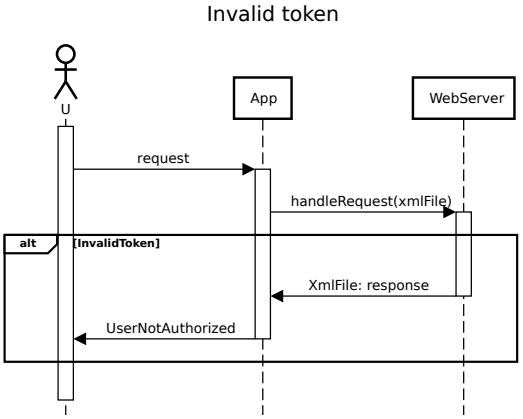


Figure 2.13: Invalid token runtime view diagram

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.

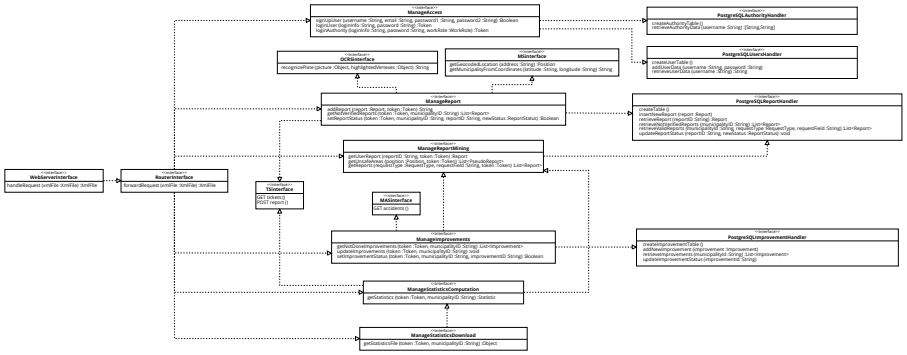


Figure 2.14: Component interfaces

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.

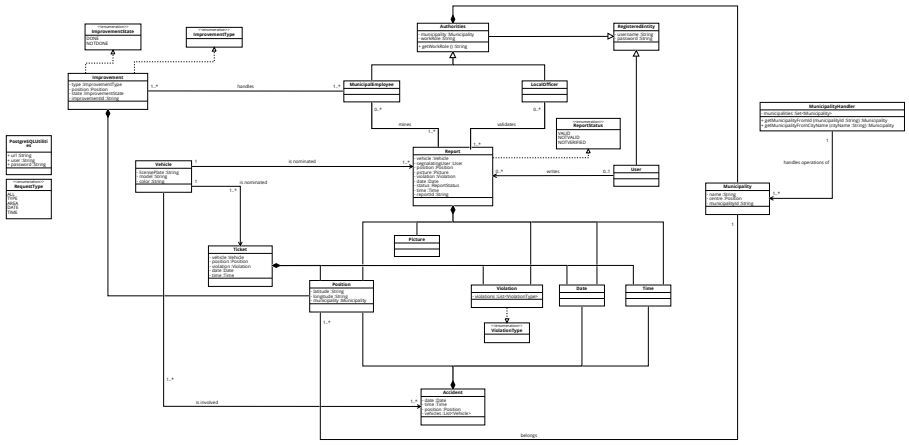


Figure 2.15: Class diagram

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 (none of them admits DELETE).

URI	POST	GET	PUT
`/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

`GET` `/reports/notverified/?id=id`

This request retrieves the reports that are waiting for validation in a certain municipality.

Parameters

Field	Type	Description
id	String	The id that uniquely identifies the municipality which the LO works for

Success 200 (request ok)

Field	Type	Description
reports	Object[]	A list of the valid reports of a certain municipality
↪reportId	String	The string that uniquely identifies a report
↪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

Error 403 (Forbidden)

Field	Description
UserNotAuthorized	The id of the municipality and the token of the user have been analyzed. It was found that the user was not an LO or the LO's municipality was not the one of the reports requested

`PUT` `/reports/notverified/?id=id`

This request modifies the status of a report

Parameters

Field	Type	Description
id	String	The id that uniquely identifies the municipality which the LO works for

Fields

Field	Type	Description
id	String	The id of the report
newStatus	String	The result of the validation performed by the LO

Error 403 (Forbidden)

Field	Description
UserNotAuthorized	The id of the municipality and the token of the user have been analyzed. It was found that the user was not an LO or the LO's municipality was not the one of the reports requested

GET `/reports/valid/?id=id`

This request gets all the valid reports in a certain municipality.

Parameters

Field	Type	Description
id	String	The id that uniquely identifies the municipality which the LO works for

Fields

Field	Type	Description
requestType	String	The type of request issued (i.e. "by area")
requestField	String	The field that contains precise information on the request

Success 200 (request ok)

Field	Type	Description
reports	Object[]	A list of the valid reports of a certain municipality
↪reportId	String	The string that uniquely identifies a report
↪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

Error 403 (Forbidden)

Field	Description
UserNotAuthorized	The id of the municipality and the token of the user have been analyzed. It was found that the user was not an LO or the LO's municipality was not the one of the reports requested

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

`GET` `/improvements/?id=id`

This request retrieves all the suggested improvements in a certain municipality

Parameters

Field	Type	Description
id	String	The id that uniquely identifies the municipality which the ME works for

Success 200 (request ok)

Field	Type	Description
improvements	Object[]	The list of suggested improvements
↪type	String	The type of the improvement
↪position	Object	The position of the improvement expresses in DMS
↪latitude	String	The latitude where the suggested improvement will be expected to be
↪longitude	String	The longitude where the suggested improvement will be expected to be
↪state	String	The status of the improvement, "DONE" or "NOT DONE"
↪improvementId	String	The id that uniquely identifies the improvement on the database

Error 403 (Forbidden)

Field	Description
UserNotAuthorized	The id of the municipality and the token of the user have been analyzed. It was found that the user was not an ME or the ME's municipality was not the one of the reports requested

Error 404 (Resource not found)

Field	Description
NotEnoughReportError	The requested improvements could not be found on the database and the available information on the Municipality is not enough to compute correct suggestions

PUT `/improvements/?id=id`

This request modifies the status of an improvement from "not done" to "done"

Parameters

Field	Type	Description
id	String	The id that uniquely identifies the municipality which the LO works for

Fields

Field	Type	Description
improvementId	String	The id that uniquely identifies the improvement on the database

Error 403 (Forbidden)

Field	Description
UserNotAuthorized	The id of the municipality and the token of the user have been analyzed. It was found that the user was not an ME or the ME's municipality was not the one of the reports requested

`GET` `/statistics/visualize/?id=id`

This request gets the available statistics on a certain municipality and lets the ME visualize them.

Parameters

Field	Type	Description
id	String	The id that uniquely identifies the municipality which the LO works for

Success 200 (request ok)

Field	Type	Description
statistics	Object[]	The various statistics
↪firstFieldName	String	The name of the first field of the graph
↪secondFieldName	String	The name of the second field of the graph
↪firstFieldValues	Number[]	The values of the first field
↪secondFieldValues	Number[]	The value of the second field

Error 403 (Forbidden)

Field	Description
UserNotAuthorized	The id of the municipality and the token of the user have been analyzed. It was found that the user was not an ME or the ME's municipality was not the one of the reports requested

Error 404 (Resource not found)

Field	Description
NotEnoughReportError	The requested statistics could not be found on the database and the available information on the Municipality is not enough to compute accurate statistics

`GET` `/statistics/download/?id=id`

This request gets the url of the pdf file where the visualized statistics are written

Parameters

Field	Type	Description
id	String	The id that uniquely identifies the municipality which the LO works for

Success 200 (request ok)

Field	Type	Description
url	String	The url where the ME can download the pdf file

Error 403 (Forbidden)

Field	Description
UserNotAuthorized	The id of the municipality and the token of the user have been analyzed. It was found that the user was not an ME or the ME's municipality was not the one of the reports requested

Error 404 (Resource not found)

Field	Description
NotEnoughReportError	The requested statistics could not be found on the database and the available information on the Municipality is not enough to compute accurate statistics

2.5.2 TS interface and MAS interface

Since both the TS and the MAS are optional external services which are not completely specified it will be assumed that their communication interfaces will be provided trough RESTful API given the fact that a stateless communication will ease the load on both our system and the Municipality's system. There will be assumptions only on the expected response for the GET requests and for the POST only the field of the message will be described. Both the URIs, controls on accesses and security will be overlooked.

2.5.2.1 Request of data about accidents from the MAS

This request will be structured as a GET and the expected success message will be structured as follows.

Field	Type	Description
accidents	Object[]	The list of accidents that a Municipality can provide
date	String	The datetime in dd-MM-yyyyThh:mm:ss format
position	Object	The position, expressed in DMS, where the accident happened
latitude	String	The latitude where the accident was recorded to have happened
longitude	String	The longitude where the vehicle was recorded to have happened
vehicles	Object[]	The vehicles that were involved in the accident
licensePlate	String	The license plate of the vehicle

2.5.2.2 Request of data about tickets from the TS

This request will be structured as a GET, the expected success message will be structured as follows.

2.5.2.3 Forwarding of data about valid reports to the TS

The request will structured as a POST, the content of the sent message will be as follows.

Field	Type	Description
tickets	Object[]	The tickets issued in a certain Municipality
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 ticket was issued
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
date	String	The datetime in dd-MM-yyyyThh:mm:ss format

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 ticket was issued
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
date	String	The datetime in dd-MM-yyyyThh:mm:ss format

2.5.3 DBMS interface

The communication between the database and our system will be handled by a set of interfaces, implemented by the components, organized in two families characterized by the table that they manage.

In the database there will be four different tables: the table of the reports, the table of the users, the table of the authorities and the table of the improvements which will be respectively handled with the interfaces PostgreSQLReportHandler, PostgreSQLUsersHandler, PostgreSQLAuthorityHandler and PostgreSQLImprovementsHandler.

2.5.3.1 PostgreSQLReportHandler

This interface will expose six methods that will all establish a connection with the server where the database is hosted. The parameters for the connection (URL of the server, user and password) will be available in a class called PostgreSQLUtilities. The methods will then try to send a query to the database, in case of success the connection will be closed and the method will return, otherwise the exception launched by the connection will be handled launching an exception on the upper level.

The methods are:

createTable: this method will create a new report table on the database, if one is already existing this function won't have any effect

Exceptions

Name	Description
DataBaseErrorException	An error has occurred in database

insertNewReport: this method will add a new report on the database

Parameters

Name	Type	Description
report	Report	The report that will have to be uploaded to the database

Exceptions

Name	Description
DataBaseErrorException	An error has occurred in database

retrieveReport: this method will get a report from the database

Parameters

Name	**Type**	Description
reportId	String	The id of the report that will be retrieved from the database

Return

Type	Description
Report	The report taken from the database

Exceptions

Name	**Description**
DataBaseErrorException	An error has occurred in the database

retrieveNotVerifiedReports: this method will get all the reports which state is set as "NOT VERIFIED" and the id of the municipality where the report has been composed is equal to the one in the parameters

Parameters

Name	**Type**	Description
municipalityId	String	The id of the municipality in charge of the report

Return

Type	**Description**
List<Report>	The list of reports taken from the database which satisfy the query

Exceptions

Name	**Description**
DataBaseErrorException	An error has occurred in the database

updateReportStatus: this method will update, changing the status of the report to the one in the parameters, the tuple where the reportId is the same as the one in the parameters

Parameters

Name	Type	Description
reportId	String	The id of report that will have its status changed
newReportStatus	ReportStatus	The new status that the report will have

Exceptions

Name	Description
DataBaseErrorException	An error has occurred in the database

retrieveValidReports: this method will get all the reports which state is set as "VALID" and where the id in the parameters is equal to the one of the municipality where the report has been composed.

There will be a different query for each type of RequestType and requestField will contain the requirements of the search.

Parameters

Name	Type	Description
municipalityId	String	The id of the municipality in charge of the report
requestType	RequestType	The type of request, it may be "ALL", "TYPE", "AREA", "DATE" and "TIME"
requestField	String	The requirements of the research

Return

Type	Description
List<Report>	The list of reports which satisfy the condition of the query

Exceptions

Name	Description
DataBaseErrorException	An error has occurred in the database

2.5.3.2 PostgreSQLUsersHandler

This interface will expose three methods dedicated to the management of the users's data. These methods will try to establish the connection to the database with the same procedure as the methods of PostgreSQLReportHandler, the handling of the exceptions will also be the same. The methods are:

createUserTable: this method will create a new table of the users on the database, if one is already existing this function won't have any effect.

Exceptions

Name	Description
DataBaseErrorException	An error has occurred in the database

addUserData: this method will register on the database the data of a new user

Parameters

Name	Type	Description
username	String	The username of the new user
password	String	The password of the new user

Exceptions

Name	Description
DataBaseErrorException	An error has occurred in the database

retrieveUserData: This method will be used during the login phase. The username, that i assured to be unique, will be used to search for the password of the user

Parameters

Name	Type	Description
username	String	The username of the user that is trying to login

Return

Type	Description
String	The password of the user

Exceptions

Name	Description
DataBaseErrorException	An error has occurred in the database

2.5.3.3 PostgreSQLAuthorityHandler

This interface will expose two methods dedicated to the management of the authorities' data. These methods will try to establish the connection to the database with the same procedure as the methods of PostgreSQLReportHandler, the handling of the exceptions will also be the same. The methods are:

createAuthorityTable: This method will create a new table of the authorities on the database, if one is already existing this function won't have any effect

Exceptions

Name	Description
DataBaseErrorException	An error has occurred in the database

retrieveAuthorityData: this methods works in the same way as retrieveUsersData, sending a username and receiving the password and workRole of the associated authority

Parameters

Name	Type	Description
username	String	The username of the user that is trying to login

Return

Type	Description
String	The password of the authority
String	The workRole of the authority

Exceptions

Name	Description
DataBaseErrorException	An error has occurred in the database

2.5.3.4 PostgreSQLImprovementHandler

This interface will expose four methods dedicated to the retrieval and insertion of improvements into the database. These methods will try to establish the connection to the database with the same procedure as the methods of PostgreSQLReportHandler, the handling of the exceptions will also be the same.

The methods are:

createImprovementTable: this method will create a new table of the authorities on the database, if one is already existing this function won't have any effect

Exceptions

Name	Description
DataBaseErrorException	An error has occurred in the database

addNewImprovement: this method will add a new improvement to the database

Parameters

Name	Type	Description
improvement	Improvement	The new improvement that has been calculated and was not already on the database

Exceptions

Name	Description
DataBaseErrorException	An error has occurred in the database

retrieveImprovements: this methods retrieves the list of improvements that are suggested for a municipality which id is the same of the one in the parameters and the state is set as "NOT DONE"

Parameters

Name	Type	Description
municipalityId	String	The id of the municipality in charge of the report

Return

Type	Description
List<Improvements>	The list of improvements that are suggested for a municipality

Exceptions

Name	Description
DataBaseErrorException	An error has occurred in the database

updateImprovementStatus: this method will update to "DONE" the state of the improvement that has its id equal to the one in the parameter

Parameters

Name	Type	Description
improvementId	String	The id of the improvement which completion state has been set to "DONE"

Exceptions

Name	Description
DataBaseErrorException	An error has occurred in the database

2.5.4 Map Service interface

The map service that will be adopted is the one provided by Google, in particular the Geocoding api and the SDK maps api will be used. The interface will possess methods for both APIs.

getGeocodedLocation: this method converts the address given from the user, this happens in case the ME wants to mine reports inserting the address where to execute the search

Parameters

Name	Type	Description
address	String	The address inserted by the ME

Return

Type	Description
Position	The corresponding position to the address in the parameters

getMunicipalityFromCoordinates

Parameters

Name	Type	Description
latitude	String	The latitude of the position which the belonging municipality is needed
longitude	String	The longitude of the position which the belonging municipality is needed

Return

Type	Description
String	The municipalityId of the municipality which the position belongs to

2.5.5 OCRS interface

The OCRS is a bought component imported in our system. It exposes only one method, used for recognizing car plates from an image and an highlighted area on it, added by the user.

recognizePlate: This method is used for recognizing a vehicle's plate given its picture and an highlighted area on it. The highlighted area is defined by a rectangle, which is represented by its vertexes' coordinates.

Parameters

Name	Type	Description
picture	Object	Representation of the image of the vehicle
highlightVertexes	Object	The coordinates on the picture of where the license plate is located
vertexOne	Number[]	The coordinates of the top-left vertex
vertexTwo	Number[]	The coordinates of the bottom-right vertex

Return

Type	Description
String	The string containing the vehicle's plate

Exceptions

Name	Description
PlateNotRecognizedException	This exception is thrown when the plate is not recognizable from the given picture

2.5.6 Web and Application Server interfaces

The Web server has one interface, responsible for receiving REST requests and handle them by responding directly, if the request comes from a WebApplication and the response is cached, otherwise by forwarding to the router through the RouterInterface.

The Application server has one external interface, the RouterInterface, and has more internal interface, for each one of its components, as described below.

2.5.6.1 WebServerInterface

handleRequest: This method is used for handling the REST requests coming from the clients. In particular, if the WebServer has the response, returns directly it, otherwise forward the request to the RouterInterface.

Parameters

Name	Type	Description
xmlFile	XmlFile	The xml file containing the data of the request

Return

Type	Description
XmlFile	The xml file containing the response of the request

Exceptions

Name	Description
Exception	When an error occurs on Server's side, an xml file containing the description of the exception is returned

2.5.6.2 RouterInterface

forwardRequest: this method is used for handling the REST requests coming from the WebServer. In particular it forward the requests to the correct component.

Parameters

Name	Type	Description
xmlFile	XmlFile	The xml file containing the data of the request

Return

Type	Description
XmlFile	The xml file containing the response of the request

Exceptions

Name	Description
Exception	When an error occurs on Server's side, an xml file containing the description of the exception is returned

2.5.6.3 ManageAccess

signUpUser: this method is used for registering a user in the system

Parameters

Name	Type	Description
username	String	The user's username
email	String	The user's email
password1	String	The first password typed by the user
password2	String	The second password typed by the user

Return

Type	Description
Boolean	A boolean value which is true when the signUp goes well

Exceptions

Name	Description
ExistingUsernameException	Someone with the same username is already registered
DifferentPasswordException	The second password is different from the first one
ExistingMailException	This email is already associated with another account

loginUser: this method is used for logging a user in the system, by providing it a token used for further validation

Parameters

Name	Type	Description
loginInfo	String	The user's username or email
password	String	The user's password

Return

Type	Description
Token	The token that identify the user in the system

Exceptions

Name	Description
WrongUsernameOrPasswordException	The written username and password does not correspond to any existing user

loginAuthority: this method is used for logging an authority in the system, by providing it a token used for further validation

Parameters

Name	Type	Description
loginInfo	String	The authority's username
password	String	The authority's password
workRole	WorkRole	The authority's work role ("ME" or "LO")

Return

Type	Description
Token	The token that identify the authority in the system

Exceptions

Name	Description
WrongUsernameOrPasswordException	The written username and password does not correspond to any existing user
NotCorrespondingRoleException	The selected work role does not correspond to the user which given login and password corresponds to

2.5.6.4 ManageReport

addReport: this method is used for adding a new Report. In particular it tries to recognize the plate with the help of the OCRS and, if the plate is not recognized, the status of the report is set to "NOT VALID", otherwise is set to "NOT VERIFIED". Then saves the report in the database. The returned string is the identifier of the report, used for further requests made by the user.

Parameters

Name	Type	Description
report	Report	The report received from the user
token	Token	The token that identifies the user in the system

Return

Type	Description
String	The ID of the report added by the user

getNotVerifiedReports: this method is used for fetching all the reports in the database, issued in the municipality specified by the municipalityID, which have still to be verified

Parameters

Name	Type	Description
token	Token	The token that identifies the authority in the system
municipalityID	String	The municipality of the requesting local officer

Return

Type	Description
List<Report>	The list containing the reports still to be verified

Exceptions

Name	Description
UserNotAuthorizedException	The id of the municipality and the token of the user have been analyzed. It was found that the user was not an LO or the LO's municipality was not the one of the reports requested

setReportStatus: this method is used in order to set the status of a report, given its ID, to "VALID" or "NOT VALID"

Parameters

Name	Type	Description
token	Token	The token that identifies the authority in the system
municipalityID	String	The municipality of the requesting local officer
reportID	String	The ID of the report of which the state is changed
newStatus	ReportStatus	The new status of the report ("VALID" or "NOT VALID")

Return

Type	Description
List<Report>	The list containing the reports still to be verified

Exceptions

Name	Description
UserNotAuthorizedException	The id of the municipality and the token of the user have been analyzed. It was found that the user was not an LO or the LO's municipality was not the one of the reports requested

2.5.6.5 ManageReportMining

getUserReport: this method is used for fetching, from the database, the report with the given reportID, issued by the requesting user.

Parameters

Name	Type	Description
token	Token	The token that identifies the user in the system
reportID	String	The ID of the report which has to be retrieved

Return

Type	Description
Report	The report the user has required

Exceptions

Name	Description
UserNotAuthorizedException	The id of the report and the token of the user have been analyzed. It was found that the user was not the one who submitted the report and as such the RU was not permitted to see the report

getUnsafeAreas: this method is used for fetching from the database all the pseudoreports issued within 300 meters from the given position. A pseudoreport is like a report, but containing only info about violation's type, date and time of issuing.

Parameters

Name	Type	Description
position	Position	The position indicated by the user from which find violations
token	Token	The token that identifies the user in the system

Return

Type	Description
List<PseudoReport>	The list containing the violations occurred within 300 meters from the given position, with their date and time

getReports: This method is used for retrieving all valid reports issued in the municipality specified by the municipalityID. Eventually some condition of mining can be applied. In particular is possible to mine a subset of the previous reports, by defining one of the following property:

- AREA: only the reports within a given radius from a given position are fetched.
- VIOLATION: only the reports with the given violation are fetched
- DATE: only the reports issued in the specified date are fetched
- TIME: only the reports issued in the specified time are fetched

The necessary parameters are contained in the "requestField" field.

Parameters

Name	Type	Description
requestType	RequestType	The type of mining, can be "ALL", "AREA", "VIOLATION", "DATE", "TIME"
requestField	String	The necessary parameters for the request
token	Token	The token that identifies the authority in the system

Return

Type	Description
List<Report>	List of valid reports retrieved from the database

Exceptions

Name	Description
UserNotAuthorizedException	The id of the municipality and the token of the user have been analyzed. It was found that the user was not an LO or the LO's municipality was not the one of the reports requested

2.5.6.6 ManageImprovements

getNotDoneImprovements: this method is used for retrieving all the improvements with the status set to "NOT DONE" from the database

Parameters

Name	Type	Description
token	Token	The token that identifies the authority in the system
municipalityID	String	The municipality of the requesting local officer

Return

Type	Description
List<Improvement>	List of not done improvements

Exceptions

Name	Description
UserNotAuthorizedException	The id of the municipality and the token of the user have been analyzed. It was found that the user was not an LO or the LO's municipality was not the one of the reports requested

updateImprovements: this method is used for updating the improvements saved in the database. In particular data coming from the MAS and the reports fetched via the ReportMiner are used. Every new improvement found has the status set to "NOT DONE" by default and is saved in the database.

Parameters

Name	Type	Description
token	Token	The token that identifies the authority in the system
municipalityID	String	The municipality of the requesting local officer

setImprovementStatus: this method is used for setting the status of an improvement, given its ID, from "NOT DONE" to "DONE".

Parameters

Name	Type	Description
token	Token	The token that identifies the authority in the system
municipalityID	String	The municipality of the requesting local officer
improvementID	String	The ID of the improvement which status has to be changed to done

Return

Type	Description
Boolean	A boolean which is true if the operation of changing status goes correctly

Exceptions

Name	Description
UserNotAuthorizedException	The id of the municipality and the token of the user have been analyzed. It was found that the user was not an ME or the ME's municipality was not the one of the reports requested

2.5.6.7 ManageStatisticsComputation

getStatistics: this method is used for calculating the current statistics, based on the data coming the TS and the reports fetched via the ReportMiner.

Parameters

Name	Type	Description
token	Token	The token that identifies the authority in the system
municipalityID	String	The municipality of the requesting local officer

Return

Type	Description
Statistic	A statistic object containing the new statistics created

Exceptions

Name	Description
UserNotAuthorizedException	The id of the municipality and the token of the user have been analyzed. It was found that the user was not an ME or the ME's municipality was not the one of the reports requested

2.5.6.8 ManageStatisticsDownload

getStatisticsFile: this method is used for getting a file containing the current statistics, based on the data coming the TS and the reports fetched via the ReportMiner

Parameters

Name	Type	Description
token	Token	The token that identifies the authority in the system
municipalityID	String	The municipality of the requesting local officer

Return

Type	Description
Object	An object containing the new file containing the statistics created

Exceptions

Name	Description
UserNotAuthorizedException	The id of the municipality and the token of the user have been analyzed. It was found that the user was not an ME or the ME's municipality was not the one of the reports requested

2.6 Selected architectural styles and patterns

To build the system a number of different architectural style choices have been made. Between all possible alternatives the following are the one that was decided to use:

Client-Server : The client server style will be the one chosen in the system for the interaction between the user and the server components. This architectural style supports the request-response pattern, that is the one that mostly fits the way actors interact with the system: they make requests invoking some services provided by the server, and the server provides a response according to the received request.

Four tier architecture : The architecture of the whole system has been organized in four tiers such as Presentation, Application 1, Application 2 and Data access. In the first tier, the presentation one, there will be both the user's app and the authority's web app. The presentation tier won't be responsible for almost any computations (only the logic behind the request and the display of the UI will be handled) forwarding all data request onto the two following tiers. The first application tier, Application 1, will be responsible of receiving the request from all users (RU, ME and LO) and will eventually forward the requests to the Application 2 layer which will be the one really in charge of computation. The Application 1 tier will also protect the following layer from malicious attacks limiting the requests that really reach Application 2, in fact Application 1 will also work as a proxy, directly answering the clients request if it already possess all necessary data in its cache. The Data access tier will be the one responsible to hold all information, its role it's a vital one since it will allow to reach a stateless architecture.

RESTful APIs : Both the communication form and to both users, authorities and municipal services will be numerous and will need immediate response. For that purpose REST was adopted to work as the mean to communicate between parties. The RESTful APIs will be implemented over the HTTPS protocol, both because its use is standard in these cases, for its ease of understanding and maintenance, and because it will grant added security onto the data transfer. The content of the messages will be formatted using XML, this format was preferred because of its ease of use when transmitting images. The choice of REST over SOAP was done

to fully utilize the database to create a stateless architecture, its advantages can be seen in the reduction of workload on the server will permit to handle multiple simultaneous communication sessions with both users and services with a little number of threads. This way the cost of the server will remain contained while ensuring good performances.

2.7 Other design decisions

2.7.1 Framework selection

Java Enterprise Edition was selected for the implementation of the server components, because it can be used to build reliable and scalable application, modeling the components as Enterprise Java Beans, and using Java Server Pages for building dynamical user interfaces. Moreover, Java Persistence APIs can be used for the interaction with the DBMS.

2.7.2 Security

Some security measures have been taken to guarantee legitimacy of data and requests:

- Passwords, personal email addresses and username sent to and from the server won't be in plain text. Such information will be hashed and salted with strong cryptographic functions.
- The mechanism of the token used during communication between client and server will guarantee secrecy over the identity of the user who's making a request, it won't be possible to extract any information from it.

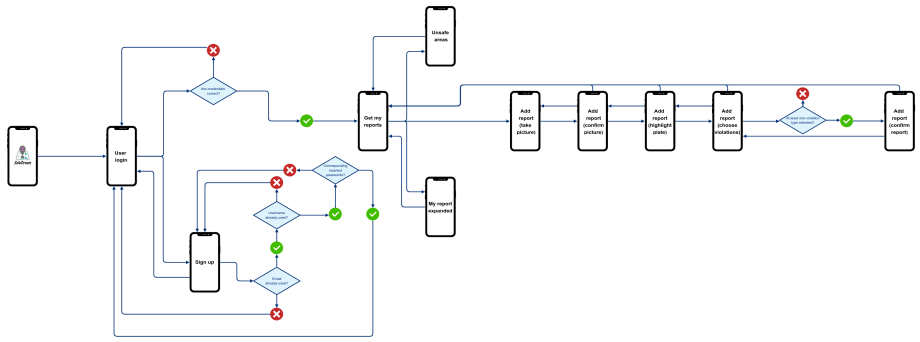
2.7.3 Service providers

For the MS, Google Maps has been chosen. This was decided thanks to its reliability and ease of use.

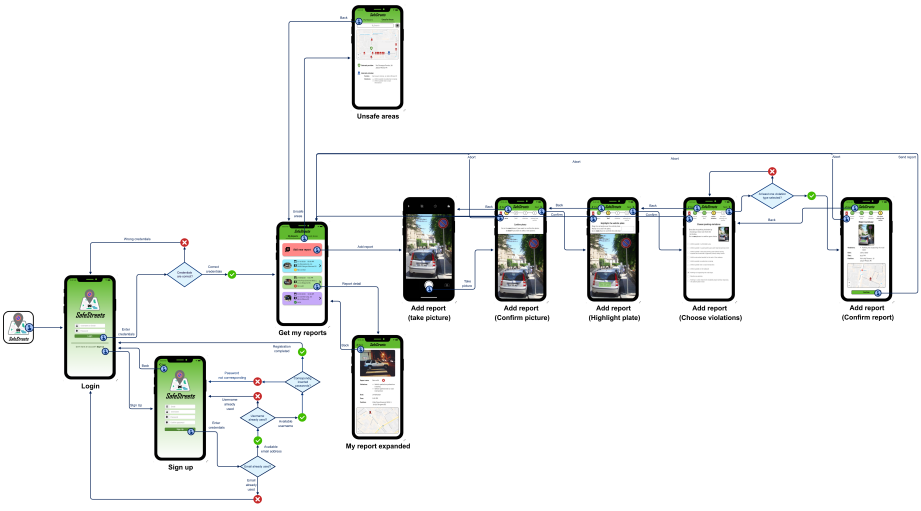
3. User interface design

3.1 User flow

3.1.1 User simple flow

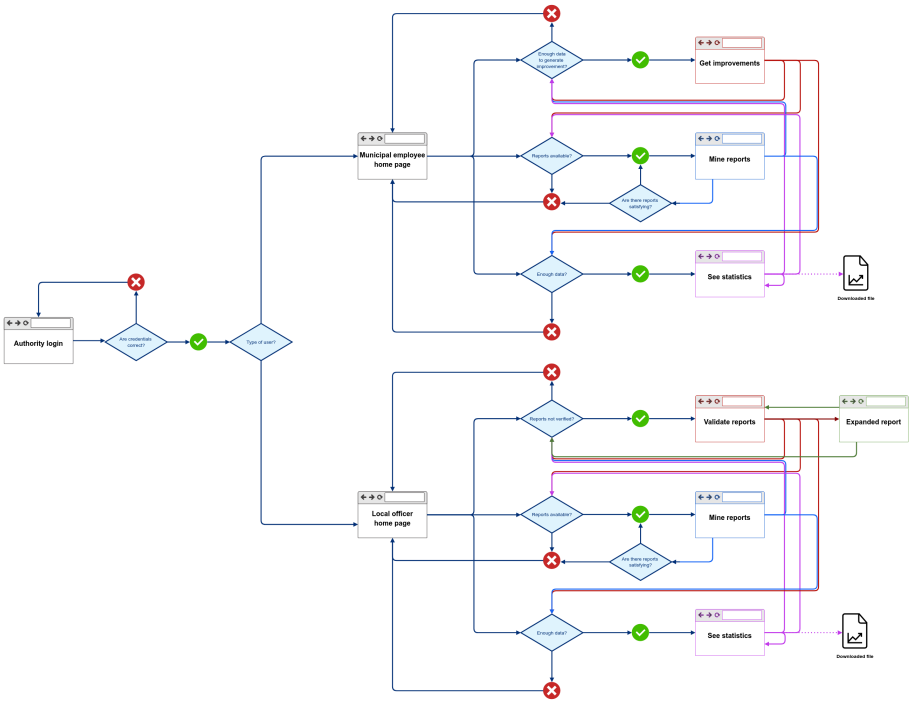


3.1.2 User flow with mockups

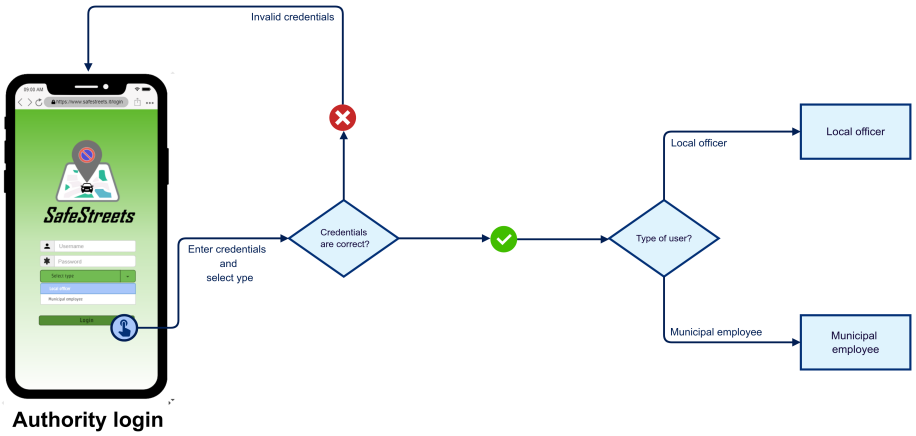


3.2 Authority flow

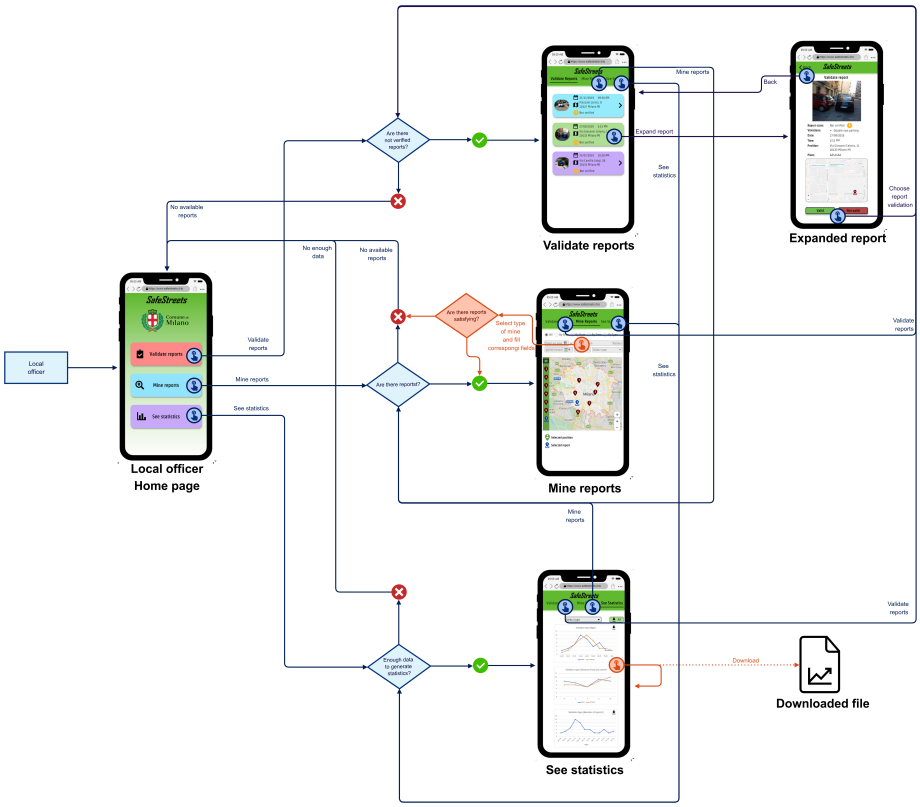
3.2.1 Authority simple flow



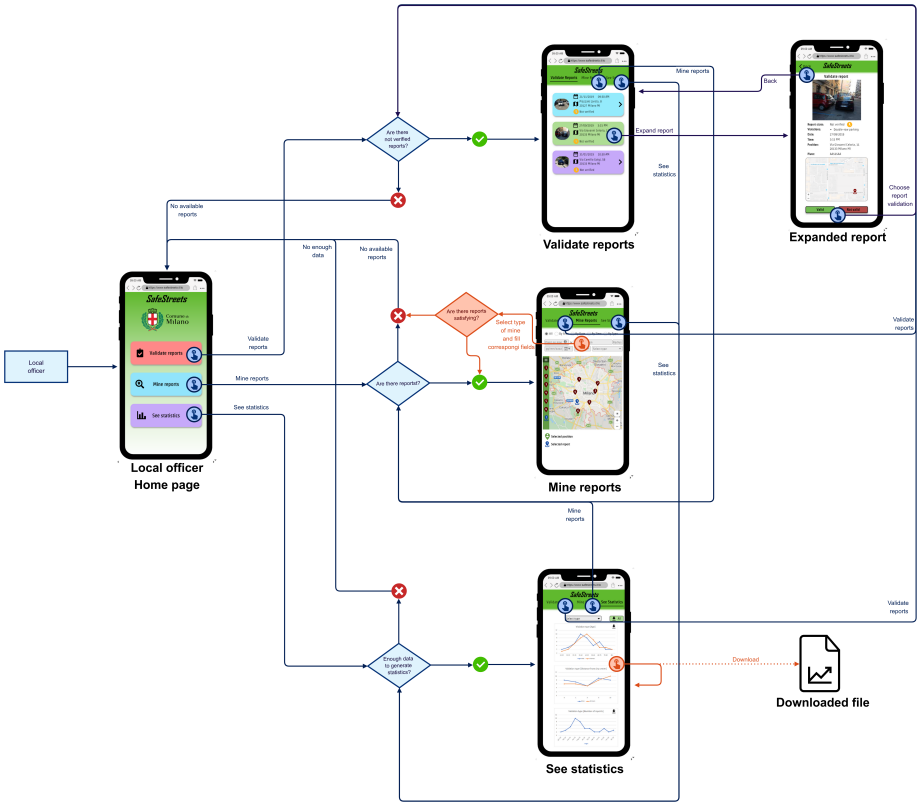
3.2.2 Authority flow with mockups



3.2.3 Municipal Employee flow



3.2.4 Local Officer flow



4. Requirement traceability

4.1 Sign up

[R1] A user must be able to sign up to the system with a unique personal username and password

- *SignUpManager:* this component receives email address, username and password of a new user, registers them in the database and notifies the user of the success, in case of no errors. Otherwise, the registration fails and the user is notified by an error message.

4.2 Login

[R2] The system must allow only registered users to login with their username and password

- *LoginManager*: this component receives user's username and password and, in case there is a match in the database, it returns a token used for further identified communication.

[R3] The system must allow only municipal authorities to login with their username and password

- *LoginManager*: this component receives authority's username and password and, in case there is a match in the database, returns a token used for further identified communication.

[R4] A users, municipal employee or local officer must be uniquely identified by his/her username

- *SignUpManager*: this component prevents that, during registration, no two users have the same username, while, for authorities, this requirement holds a priori.

4.3 Add report

[R5] When composing the report, the system must be able to access the user's device camera and GPS sensor

- *UserMobileApp*: the application is able to access user's device camera

[R6] When composing the report, a user must take a picture from the device's camera and highlight the license plate

- *UserMobileApp*: the application let user take a picture only from the device's camera and highlight the license plate

[R7] When composing the report, time and date get automatically retrieved from the internet

- *UserMobileApp*: the application automatically retrieves time and date from the internet as soon the picture for the report has been confirmed

[R8] When composing the report, position gets automatically retrieved from the device's GPS

- *UserMobileApp*: the application automatically retrieves position from the device's GPS as soon the picture for the report has been confirmed.

[R9] When composing the report, a user can choose at least one type of violation

- *UserMobileApp*: the application let the user choosing at least one type of violation

[R10] When composing the report, a user can't add the same violation type two times in the same report

- *UserMobileApp*: the application does not let the user add twice the same type of violation in the same report

[R11] When composing the report, a user can revert each phase of the creation of the report at any time, before sending it

- *UserMobileApp*: the application let the user revert each phase of the creation of the report at any time, before sending it

[R12] When composing the report, a user can abort the creation of the report at any time, before sending it

- *UserMobileApp*:the application let the user abort the creation of the report at any time, before sending it

[R13] Once a report has been sent, it can't be aborted or reverted

- *UserMobileApp*: the application does not let the user abort or revert the report, after sending it

[R14] When receiving a report, the system must store it, recognize the car plate, if possible, and marked as unchecked

- *ReportReceiver*: this component receives the report and does the following actions:
  - retrieving of the municipality where the report has been created, by employing the MS APIs
  - recognizing the vehicle's plate in the picture, by employing the OCRS apis. In case of an unrecognizable plate, the report status is set to "NOT VALID" by default, otherwise is set to "NOT VERIFIED" by default(which means that a local officer has still to prove its validity)
  - saving the report in the system's database

4.4 Get my reports

[R15] When a user asks for his/her reports, the system must provide the saved reports sent by that user

- *ReportMiner*: this component can fetch, from the database, all the reports sent by the requesting user and return them

4.5 Get unsafe areas

[R16] When getting the valid reports by area, a user can choose a position, or automatically get his/her from the GPS

- *UserMobileApp*: the application, when getting the valid reports by area, let the user the possibility to choose between setting manually the position or fetching it via GPS

[R17] When getting the valid reports by area, the system must provide all the valid reports near the position given by the user and display their violation type through the MS

- *ReportMiner*: this component can fetch, from the database, all the reports issued within 300 meters from the position received from the user and returns them

4.6 Mine reports

[R18] When mining the information, a municipal employee or a local officer can access only to violations type and date and time of reports occurred in his/her municipality

- *ReportMiner*: this component fetches, from the database, only the reports issued in the same municipality of the requesting authority

[R19] When mining the information, a municipal employee or a local officer can filter reports by area, date, time or type of violation

- *ReportMiner*: this component let the requesting authority fetching reports. If no filter is applied, then all the reports issued in the same municipality of the requesting authority are returned. Otherwise, a subset of them is chosen, according to one of the following chosen filter:
  - by area: only reports issued within the given radius from the given position are returned
  - by date: only reports issued in the given date are returned
  - by time: only reports issued in the given time, regardless the date, are returned
  - by type of violation: only reports having the given type of violation are returned.

4.7 Validate reports

[R20] A municipal officer must be able to mark a report as valid or not valid

- *ReportValidator*: this component can fetch "NOT VALID" reports, from the database, and let a local officer setting their status to "VALID" or "NOT VALID".

4.8 Retrieve statistics

[R21] When retrieving statistics, a municipal employee or a local officer can access only to reports of violations that occurred in his/her municipality

- *StatisticsComputationManager* : this component, in order to compute statistics, makes a request to the component ReportMiner, with no filter applied (the type of the request is "all"), passing the token of the requesting authority. Thanks to the token, the ReportMiner is able to get the municipality with which filtering the reports

[R22] The system must be able to calculate statistics from the reports of violations and issued tickets of the municipal employee or local officer's municipality

- *StatisticsComputationManager*: this component, once received the reports from the ReportMiner, cross them with the data coming form the TS and produce statistic, which are returned to the requesting authority, or component, in form of data
- *StatisticsDownloadManager*: this component let the requesting authority download a non materialized document of the statistics. The necessary data are obtained from the StatisticsComputationManager, by passing to it the token of the authority, with the approach described in the previous component description

4.9 Get improvements

[R23] When getting improvements, a municipal employee can access only to data of reports occurred in his/her municipality

- *ImprovementsManager*: this component, in order to compute improvements, makes a request to the component ReportMiner, with no filter applied (the type of the request is "all"), passing the token of the requesting authority. Thanks to the token, the ReportMiner is able to get the municipality with which filtering the reports

[R24] The system must be able to retrieve information about accidents from the MAS (municipal accident system)

- *ImprovementsManager*: this component is able to asks the MAS for fetching data about happened accidents

[R25] The system must be able to identify the possible unsafe areas of the municipal employee or local officer's municipality from the reports of violations and data coming the MAS

- ImprovementsManager: this component, by crossing the data coming from the MAS and from the ReportMiner, is able to compute new improvements, set as "NOT DONE" by default, and propose them, in addition to already computed "NOT DONE" improvements, obtained after previous requests and saved in the database.

5. Implementation, integration and test plan

The system is composed by various subsystems, some of which need to be implemented and tested

To be implemented and tested:

- UserMobileApp
- LocalOfficerWebApp
- MunicipalEmployeeWebApp
- WebServer
- ApplicationServer

Already implemented and tested:

- DBMS
- OCRS
- MS
- TS
- MAS

The approach followed is a "bottom-up" one, where all the interfaces are defined a priori, in order to develop non-implemented components in parallel and test them. Moreover, implementation and integration follow the same order, such a way that the system will be ready and functional as soon as possible.

It is worth noting that external components already implemented do not need to be tested.

The following table specifies the main features of the system, their importance for the customer and their difficulty of implementation.

Feature	Importance for the customer	Difficulty of implementation
SignUp and Login	Low	Low
Add a report for RU	High	Medium
Retrieve reports for RU	High	Low
Get unsafe areas for RU	Medium	Low
Mine reports for ME/LO	High	Medium
See statistics for ME/LO	Medium	Medium
Download statistics for ME/LO	Medium	Low
Validate reports for LO	High	Low
Get improvements for ME	High	High
Functional mobile app for RU	High	Medium
Functional web app for LO	High	Medium
Functional web app for ME	High	Medium

Table 5.1: Importance and difficulty of features

The order of implementation and testing of missing components is based on the previous table, starting from components associated to more difficult and crucial features.

The following table specifies the order of implementation and testing, associating components to relative features. It is important to note that, components enlisted in the same "Phase of I&T" have to be developed in parallel.

Phase	Component	Features
1	ReportReceiver	Add a report for RU
1	ReportMiner	Retrieve reports for RU, Mine reports for ME/LO, Get unsafe areas for RU
2	ReportValidator	Validate reports for LO
2	ImprovementsManager	Get improvements for ME
2	StatisticsComputationManager	See statistics for ME/LO
3	StatisticsDownloadManager	Download statistics for ME/LO
4	Router	No specific user feature
4	SignUpManager	SignUp and Login
4	LoginManager	SignUp and Login
5	WebServer	No specific user feature
5	UserMobileApp	Functional mobile app for RU
5	LocalOfficerWebApp	Functional web app for LO
5	MunicipalEmployeeWebApp	Functional web app for ME

Table 5.2: Implementation phases

In particular here is described how the previous features are obtained through the system's components:

SignUp and login : this feature is realized through the following components:

- SignUpManager: responsible for registering new users

- LoginManager: responsible for recognizing registered users and authorities

Add a report for RU : this feature is realized through the component *ReportReceiver*, which receives the report, submits it to the *OCRS*, in order to get its validity status and the eventual vehicle plate, and saves it in the database through the *DBMS*.

Retrieve reports for RU : this feature is realized through the component *ReportMiner*, which receives the request, coming from the *Router*, asks the database through the *DBMS* and returns the all the reports composed by the requesting user.

Get unsafe area for RU : this feature is realized through the component *ReportMiner*, which receives the request, coming from the *Router*, asks the database through the *DBMS* and returns violations type, date and time of valid reports issued in the requested area

Mine reports for ME/LO : this feature is realized through the component *ReportMiner*, which receives the request, coming from the *Router*, asks the database through the *DBMS* and returns the valid reports issued in the authority's municipality that satisfy the requested condition (All, Date, Time, Violation Type, Area)

See statistics for ME/LO : this feature is realized through the component *StatisticsComputationManager*, which receives the request coming from the *Router*, asks the database through the *DBMS* for valid reports and retrieves data about issued tickets through the *TS*, crosses them, produces and returns statistics to the requesting authority

Download statistics for ME/LO : this feature is realized through component *StatisticsDownloadManager*, which asks *StatisticsComputationManager* for statistics, generate a non materialized document and returns it to the requesting authority

Validate reports for LO : this feature is realized through the component *ReportValidator*, which is able to retrieve not verified reports belonging to the LO's municipality from the database through the *DBMS* and let the LO setting their validity

Get improvements for ME : this feature is realized through the component *ImprovementsManager*, which is able to get valid reports, issued in the ME's municipality, from the database through the *DBMS* and cross their data with the information coming from the *MAS*, in order to generate new possible improvements. Then, they are presented to the ME, along with previously generated improvements, saved in the database, still not completed. Finally the ME can set the status of the improvements received

Functional mobile app for RU : this feature is realized through the implementation, testing and integration of the *UserMobileApp*

Functional web app for LO : this feature is realized through the implementation, testing and integration of the *LocalOfficerWebApp*

Functional web app for ME : this feature is realized through the implementation, testing and integration of the *MunicipalEmployeeWebApp*;

It is crucial to note that, for each feature, all the described components have to be integrated and unit tested.

The integration is driven by the following diagrams, which describes the dependencies between all the components of the system.

The arrow relation from a component A to a component B indicate that A "uses" B. So, in order to integrate and unit test component A, component B must be already integrated and unit tested. Finally, components that are not used by no other components do not need to be integrated and unit tested (in our case the components highlighted in red, which are the already implemented ones, are be the starting points for the integration and unit testing).

ReportReceiverIntegration

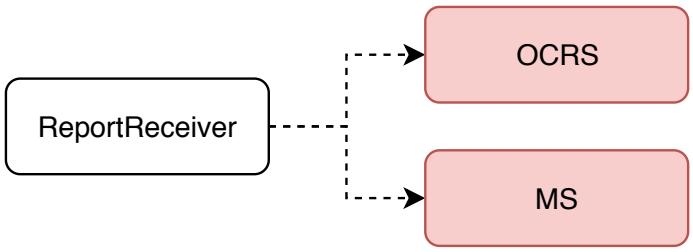


Figure 5.1: Report receiver integration

DatabaseAccessorsIntegration

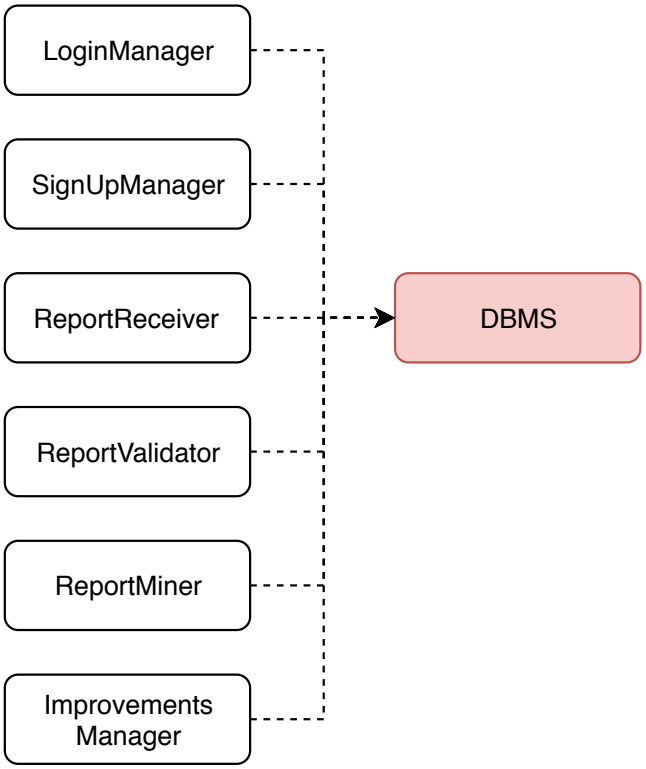


Figure 5.2: Database accessors integration

TicketServiceAccessorsIntegration

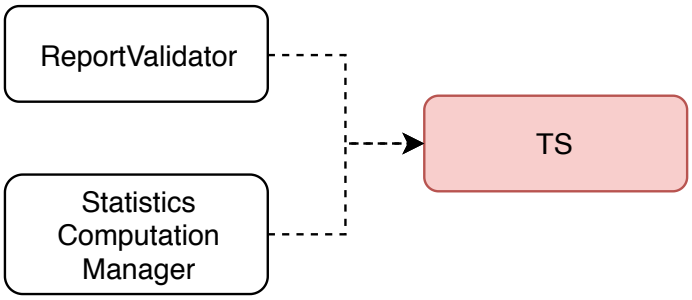


Figure 5.3: Ticket Service accessors integration

ImprovementManagerIntegration

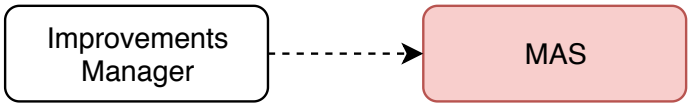


Figure 5.4: Improvement manager integration

ReportMinerAccessorsIntegration

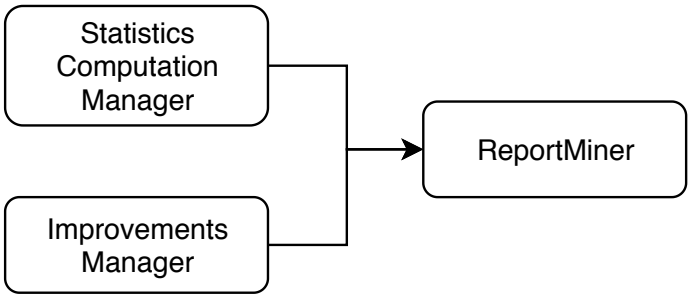


Figure 5.5: Report miner accessors integration

StatisticsDownloadManagerIntegration

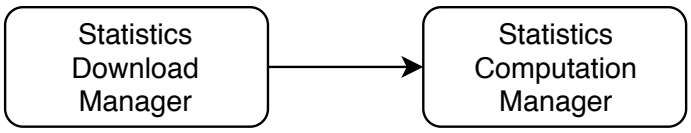


Figure 5.6: Statistics download manager integration

RouterIntegration

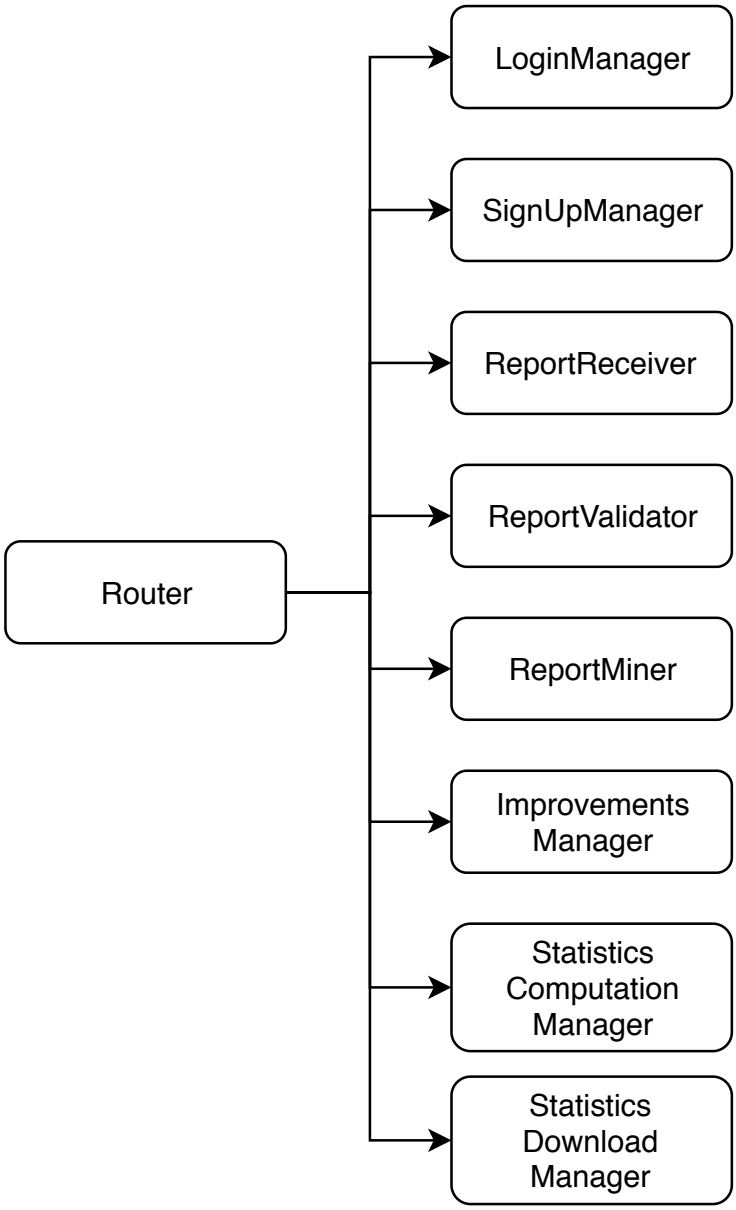


Figure 5.7: Router integration

WebServerIntegration

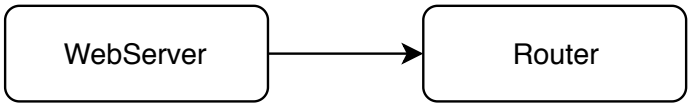


Figure 5.8: WebServer integration

AppsIntegration

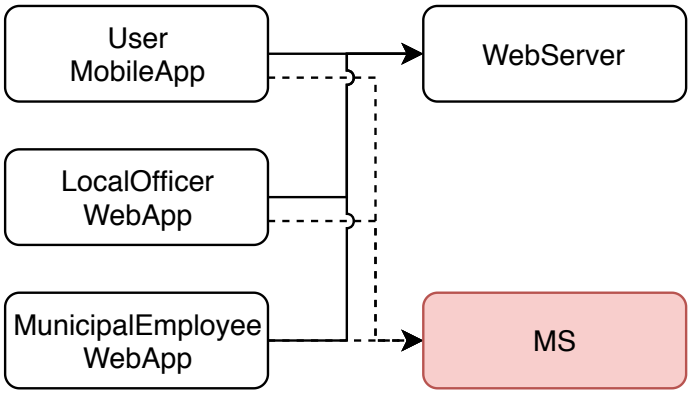


Figure 5.9: Apps integration

6. Effort spent

Student	Time spent	Total (h)
Bonatti Andrea	Part 1: 1 h Part 2: 20 h Part 3: 2 h Part 4: 1 h Part 5: 1 h	25
Buttironi Monica	Part 1: 4 h Part 2: 15 h Part 3: 5 h Part 4: 2 h Part 5: 2 h	28
Caruso Marco Giuseppe	Part 1: 1 h Part 2: 17 h Part 3: 1 h Part 4: 3 h Part 5: 3 h	25
Total	Part 1: 6 h Part 2: 52 h Part 3: 8 h Part 4: 6 h Part 5: 6 h	78

Table 6.1: Effort spent

7. References

- LaTeX for document typesetting
- Github (https://github.com) for version control
- Signavio (http://www.signavio.com/)
- Moqups (https://moqups.com/) for flows
- UML Sequence Diagram Online Tool (https://sequencediagram.org) for runtime view diagrams