



POLITECNICO
MILANO 1863

SAFESTREETS

DESIGN DOCUMENT



SAFESTREETS

edoardo putti
9 December 2019

Table of Contents

1. Introduction	4
1.1 Purpose	4
1.2 Scope	5
1.3 Definitions, Acronyms, Abbreviations	5
1.3.1 Definitions	5
1.3.2 Acronyms	5
1.4 Revision history	5
1.5 Document Structure	5
2. Architectural Design	6
2.1 Overview	7
2.2 Component view	9
2.2.1 ER diagram	9
2.2.2 Report Manager and Notification Manager	10
2.2.3 Traffic Modules	12
2.3 Runtime view	15
2.3.1 Sequence diagrams – Sign Up	15
2.3.2 Sequence diagrams – Send Report	17
2.3.3 Sequence diagrams – Resolve Report	18
2.3.4 Sequence diagrams – Select TravelPlan	19
2.3.5 Sequence diagrams – Show Statistics	20
2.3.6 Object Diagram – Traffic Module	21
2.4 Component interfaces	22
2.5 Selected architectural styles and patterns	24

2.5.1	Client-server	24
2.5.2	Service-Oriented Architecture (SOA)	24
2.5.3	Model View-Controller (MVC)	24
2.6	Other design decisions	25
2.6.1	Password storage	25
3.	Algorithm Design	25
3.1	Car-plate Extractor – How it works	25
3.2	Traffic Modules – Dynamic configuration	26
3.3	Solution Calculator – How it works	27
4.	User Interface Design	29
4.1	App Mockups	29
5.	Requirements Traceability	33
6.	Implementation, Integration and Test Plan	34
7.	Effort Spent	37
8.	References	37
8.1	Software and Tools	37
8.2	Reference Documents	37

Figure 1: High level architecture.....	8
Figure 2: ER diagram	10
Figure 3: Component Diagram - Report manager & Notification manager.....	12
Figure 4: Component Diagram - Traffic module	14
Figure 5: success Frame - Authority Sign Up	15
Figure 6: Sequence Diagram- User Sign Up.....	16
Figure 7: Sequence Diagrams - New Report.....	17
Figure 8: Sequence Diagram - Resolve Report	18
Figure 9: Sequence Diagrams - Select Travel Plan.....	19
Figure 10: Sequence Diagrams - Statistics.....	20
Figure 11: Object Diagram -Traffic manager original state.....	21
Figure 12: Object Diagram - Traffic module after balancing and reconfiguration	22
Figure 13:User-login page	29
Figure 14: Authority-login page.....	29
Figure 15:User-Home page.....	30
Figure 16:Authority-Home page	30
Figure 17: Travel Plan visualization.....	31
Figure 18: New Report page	31
Figure 19: Authority-Statistics page.....	32

1. Introduction

1.1 Purpose

This document is the Design Document for the Safestreets application. Its aim is to provide a description of the system in terms of architectural components. DD contains a description of the Architectural design using component diagrams and sequence diagrams. It shows how each component is built, how it interacts with other components and with the external actors involved. This document's aim is also to provide a technical explanation of the behavior of some component using algorithms. It also shows interfaces through graphical screen representation.

1.2 Scope

Safestreets is an application in which the User can register and send report to the Authorities. The app allows, for the user, visualizing the traffic information, some information regarding relevant incidents or reports and see the status of the reports sent; for the Authority the app allows to see all the reports and also the statistics derived from them. The application contains a notification system and its role is to notify users about the closing of one of their reports.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Here is provided a list of definitions of words and expression used in the document

- **Closing notification:** a notification that the S2B sends when an Authority sets the status of a report to close
- **Dynamically Configuration:** with this term is intended a reconfiguration that can be done without powering off the server or any physical component.

1.3.2 Acronyms

- **S2B:** System to Be
- **API:** Application Programming Interface
- **RASD:** Requirements Analysis and Specification Document

1.4 Revision history

This is the first version of this document

1.5 Document Structure

- 1) **Introduction:** this serves as an introduction to the document to illustrate its purpose, scope, the conventions that will be used and its structure.
- 2) **Architectural Design:** After providing an overview of the system, in this section are included all the details of the architecture and the related design decision. Starting from the data model arriving to the description of the components and their role. After a static description, a runtime analysis of the interesting components is provided. Finally we describe the main component interfaces.
- 3) **Algorithm Design:** In this section are described the most interesting algorithms identified in the system, how they works and their context.
- 4) **User Interfaces Design:** After the technical description of the previous section, here are provided the indications on the User interactions with the app and mockups of the screens related to the main functionalities.
- 5) **Requirements Traceability:** This section explains the rationale behind our design decision in terms of mapping between the goal/requirements defined in the RASD and the components illustrated in this component.
- 6) **Implementation, Integration and Test Plan:** In this last section is provided a plan for the whole development process, giving indications on the general approach, the priorities and the details of the process.
- 7) **Effort Spent:** here is included a summary of the effort spent.
- 8) **References:** this is the section in which are included details on the Software and tools used and the references Documents on which the work is based.

2. Architectural Design

2.1 Overview

Here is provided a high-level representation of client-server interaction and of the submodule of the servers. Moreover is included a brief indication of the deployment. The orchestrator is needed only on the client request: his role it to dispatch the request to the appropriate component based on the type of the request. After that, the component can communicate directly with the client. In the server, the orchestrator and all the other submodules are states. I thought to use the elastic component architectural pattern for the components. Moreover, the orchestrator can eventually be duplicated using a fixed dimension pool whose size is configurable by the system administrator. The diagram only shows the overall interaction between client and server and the role of the orchestrator; interactions between the submodules are not shown. Further details on the submodules and the overall interactions will be provided in the following sections.

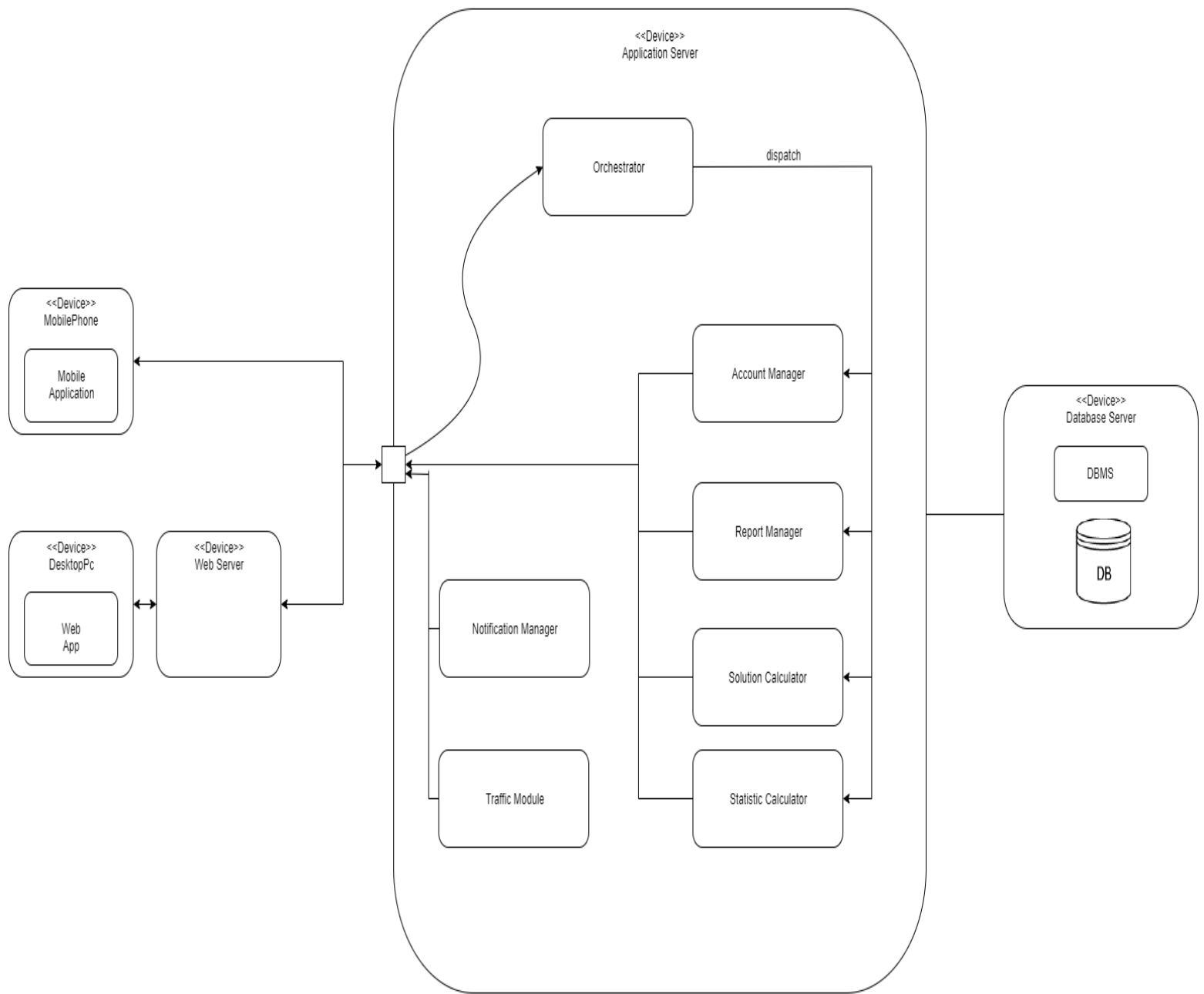


Figure 1: High level architecture

2.2 Component view

This section starts with the presentation of the Entity-Relationship diagram. In some of the diagrams following the ER, is included a fictitious component, App, that will be highlighted in green and serves the purpose of representing both the mobile app and the web app (through the web server), without adding complexity to the diagrams.

2.2.1 ER diagram

The following ER diagram represents the conceptual schema of the system database. The violation subclass presented in the schema are only two but could be more.

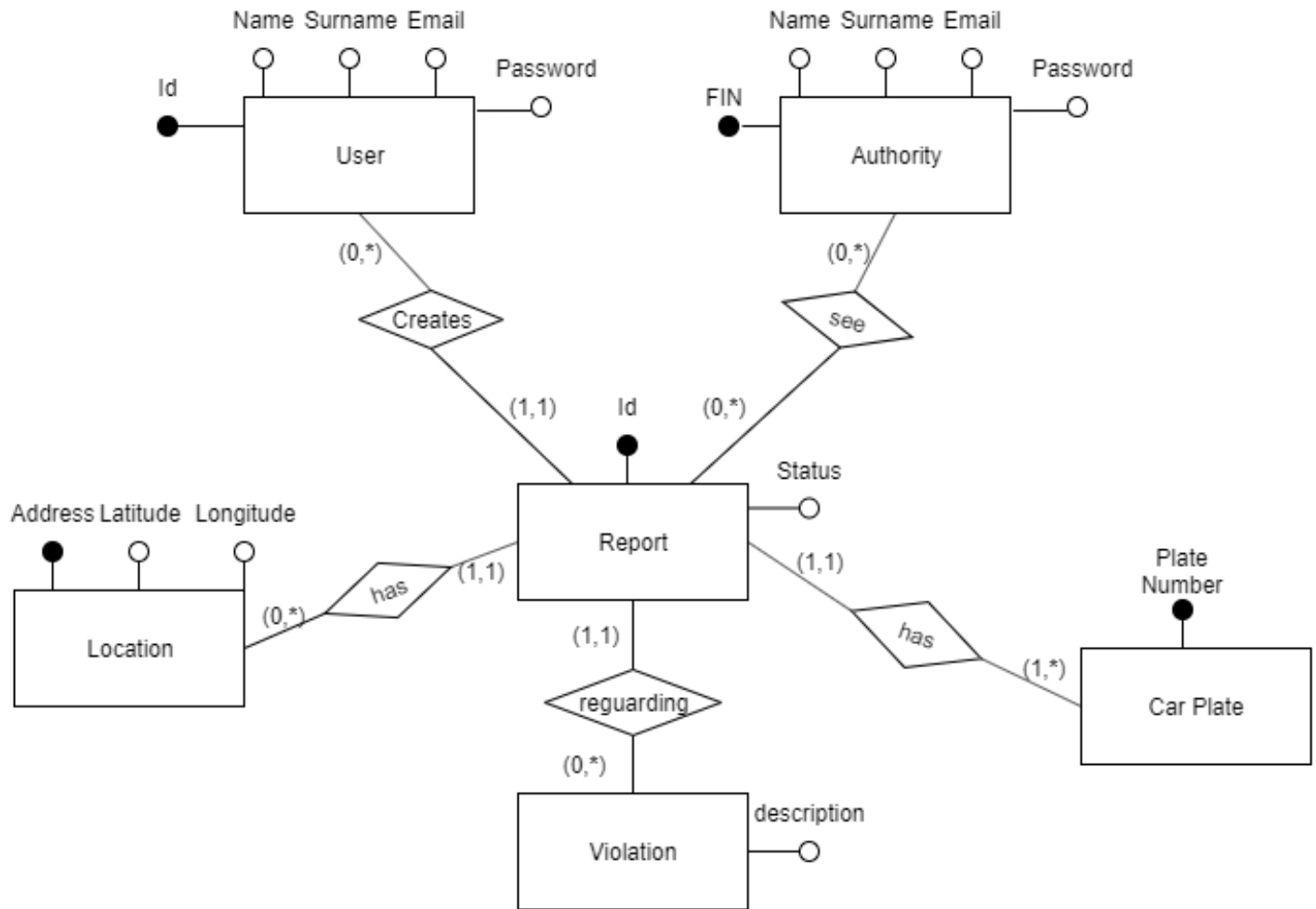


Figure 2: ER diagram

2.2.2 Report Manager and Notification Manager

Here is defined the Report Manager and the Notification Manager.

Appointment Manager has three subcomponents:

- *Report Handler*: is the only subcomponent that communicates with the database; it manages all the read and the write operations related to the reports. It is responsible for the final consistency check of the report before writing it to the database. It exports an interface to let external components read reports details and an internal interface to let the subcomponents of Report manager read and write report's details.
- *Report Editor*: this provides an interface to easy change report's details and store it to database through the interface of Report Handler. This module interacts with an external module "solution Calculator" to compute path solutions (in this diagrams it is displayed in gray and dashed line because it will not described here). This module also interact with the internal module "Carplate Extractor".
- *Carplate Extractor*: this module run an algorithm that extract the car plate from the picture that the user sent. This module is also responsible to check if the car plate inserted by the user is equal to the one of the picture.

Notification Manager provide an interface for the creation of new notification and it is composed of:

- *Notification Dispatcher*: this module dispatch the notification to the app when a Report's status is set to "closed".

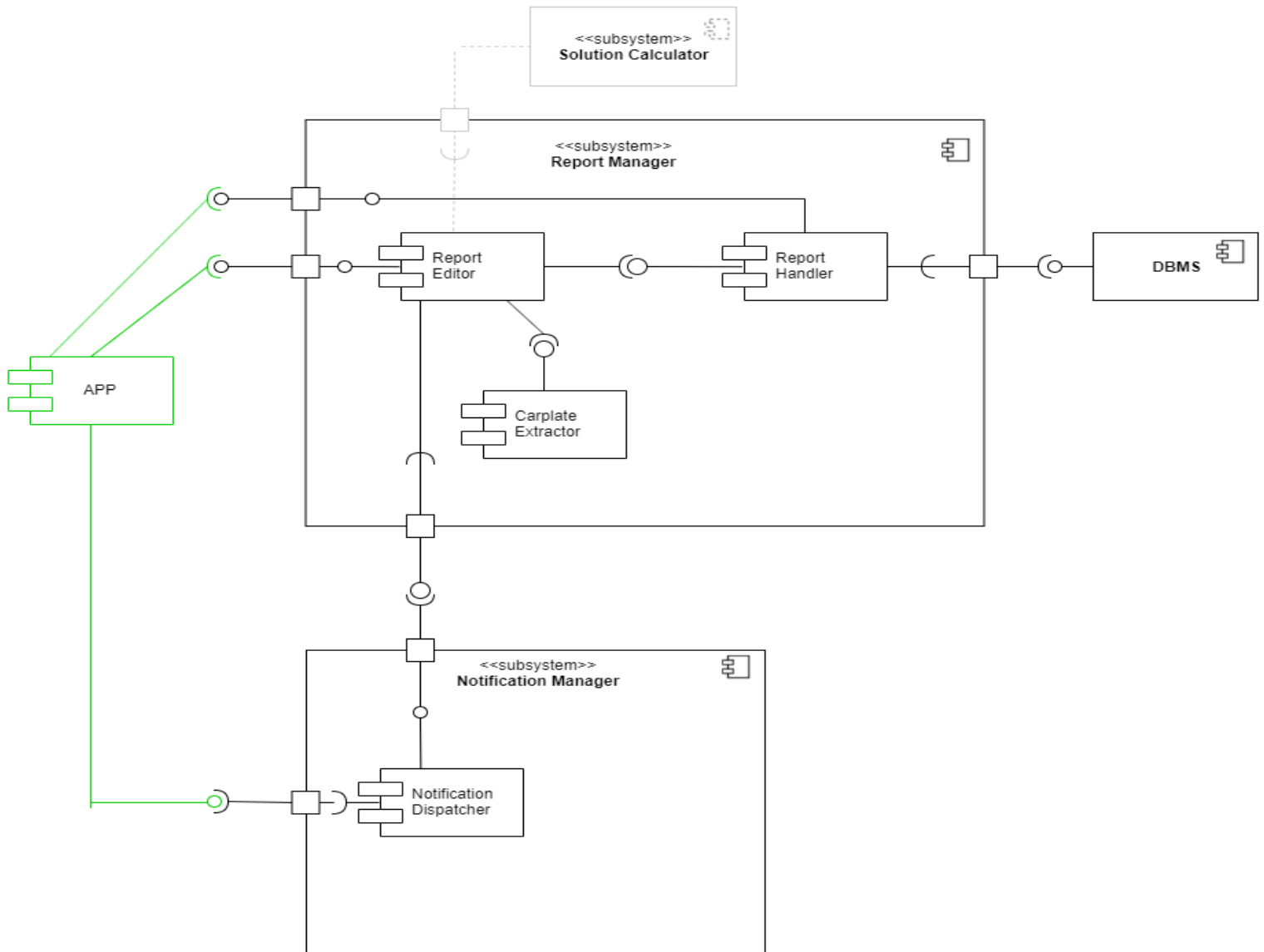


Figure 3: Component Diagram - Report manager & Notification manager

2.2.3 Traffic Modules

In the diagram, it is provided the representation of the Traffic Module. This module is needed to notify the User or the Authority of changes in traffic that can influence his/her travel or patrol. Moreover it is used by the solution calculator to avoid bad solution like, for instance, take a street where there was an accident to go to work. With the same convention as before, is represented here Solution calculator but it will not be detailed here.

The Traffic module has a quite articulated architecture and it will be explained here and in section 2.3 with an object diagram.

It is composed by:

- *Address Solver*: this component has a really basic functionality and it is exploited by the Traffic Manager to “understand” (see below) addresses. Provided an address as input, it decomposes it in its components and returns a hierarchy.
Example: “Italy, Rome, via Roma 67” -> transformed to:
“State:Italy”->“City:Rome”->“CityZone:NorthEast”->“Street:Roma”
->“Number:67”
- *Traffic Manager*: this serves as a registry to support the publish-subscribe architecture between Traffic Querier and Traffic Notifier. Moreover, this component provides an external interface to let other components have information about traffic. When it receives a request of traffic information for a specific address, it asks address solver to interpret the address and then it asks the appropriate Traffic Querier for the information.
- *Traffic Querier*: it is the module responsible for retrieving the traffic information and send them to the Traffic Notifier.
- *Traffic Notifier*: this is the component that decides if and when a User/Authority has to be notified. It subscribes to a specific Traffic Querier and receives the information from it.

Both the Querier and the Notifier are instantiated by zone. The zones of the Querier are meant to be very wide, like Italy, Spain, France and the zone of the Notifier are meant to be more specific, for instance: West Milan, Rome, East Venice.

The Notifier has to execute an algorithm for each request to decide if and when notify the User, so its load depends on the number of reports in that

specific zone and on the day of the week and the time of the day. For this reason, a load balancing mechanism is required. When a 'node' is under load it can be split into two or more parts.

The querier has a more stable load and, for this reason, no load balancing mechanism has to be implemented. By the way, the zones of the Querier can be configured by the System Administrator, even dynamically.

The load balancing mechanism of the Notifier and the dynamically configurable Querier has to be supported by the Manager: it is not a simple registry but it registers all the active Queries and Notifiers and manages the subscriptions when are modified.

The process is described in more details in section 3.

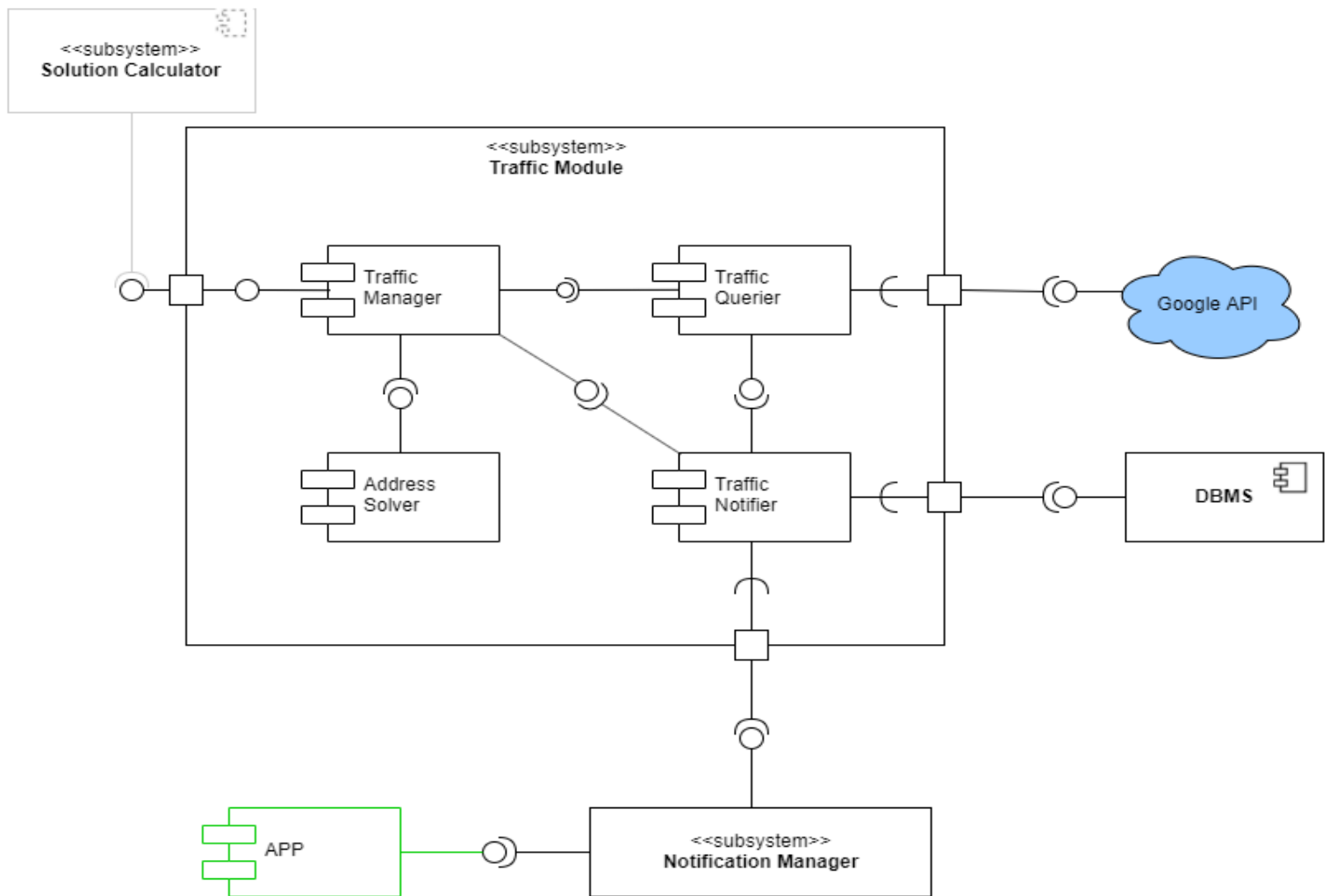


Figure 4: Component Diagram - Traffic module

2.3 Runtime view

In this section is provided a dynamic view of the system, illustrating the main interactions among components.

Five sequence diagrams are provided: Sign Up, Send Report, Resolve Report, Select TravelPlan, Show Statistics. These diagrams cover all the interesting interaction. Moreover, after that it is provided an Object Diagram of Traffic Modules, as an explicative example of how instances of their components are related.

2.3.1 Sequence diagrams – Sign Up

The sign Up operation is similar for both User and Authority so in this section is reported the complete sequence diagram for the user and only the success frame for the authority because the first parte and the failure frame are the same.

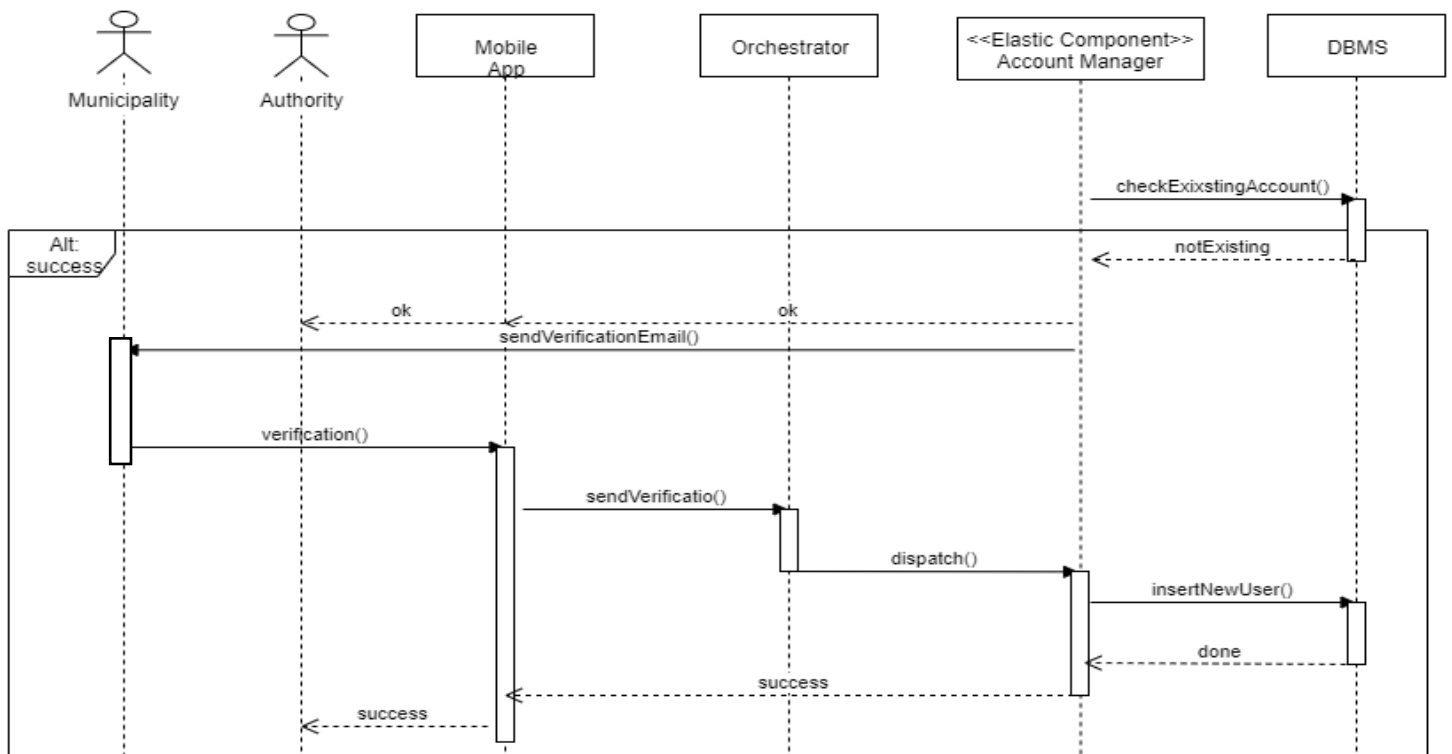


Figure 5: success Frame - Authority Sign Up

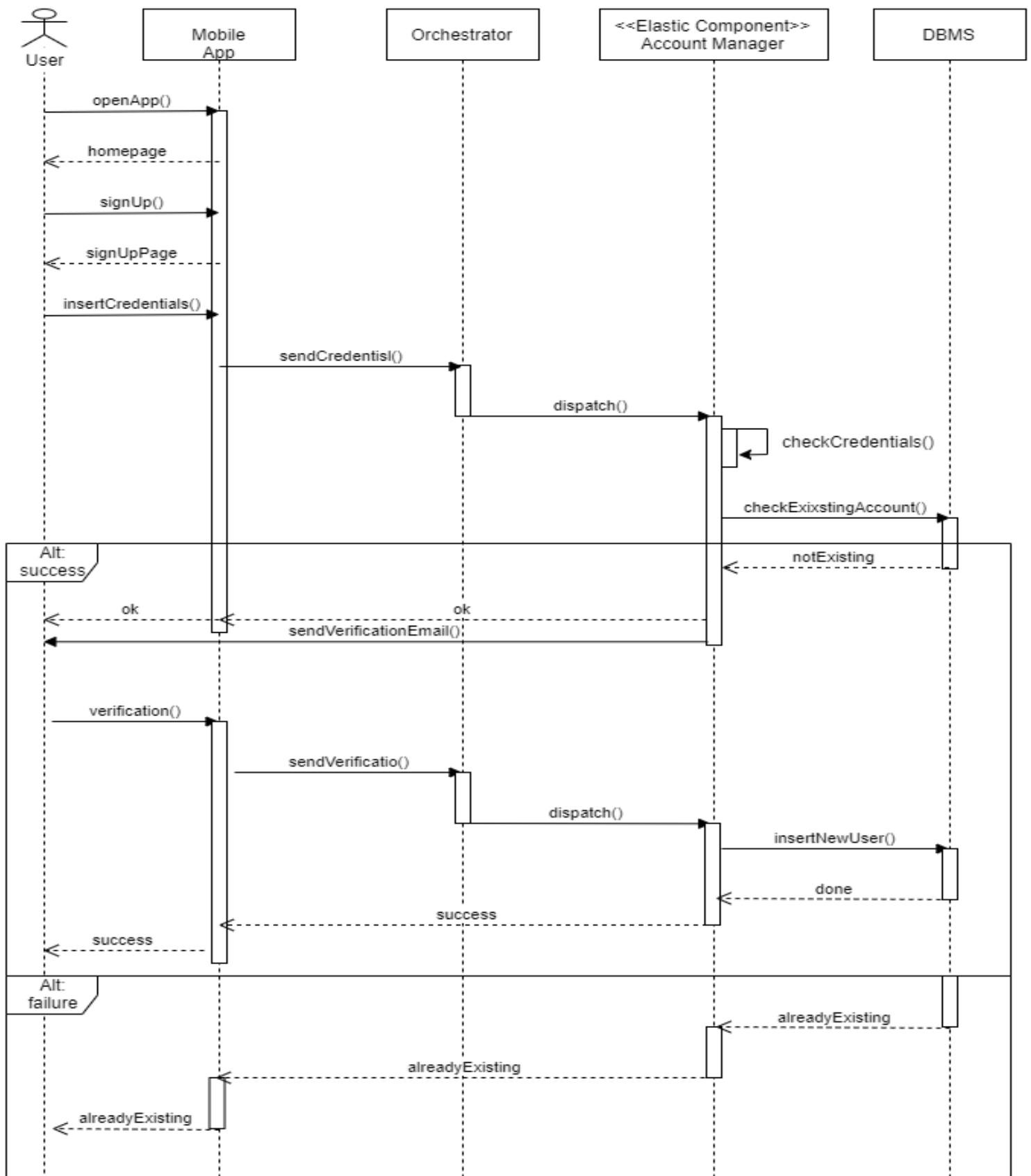


Figure 6: Sequence Diagram- User Sign Up

2.3.2 Sequence diagrams – Send Report

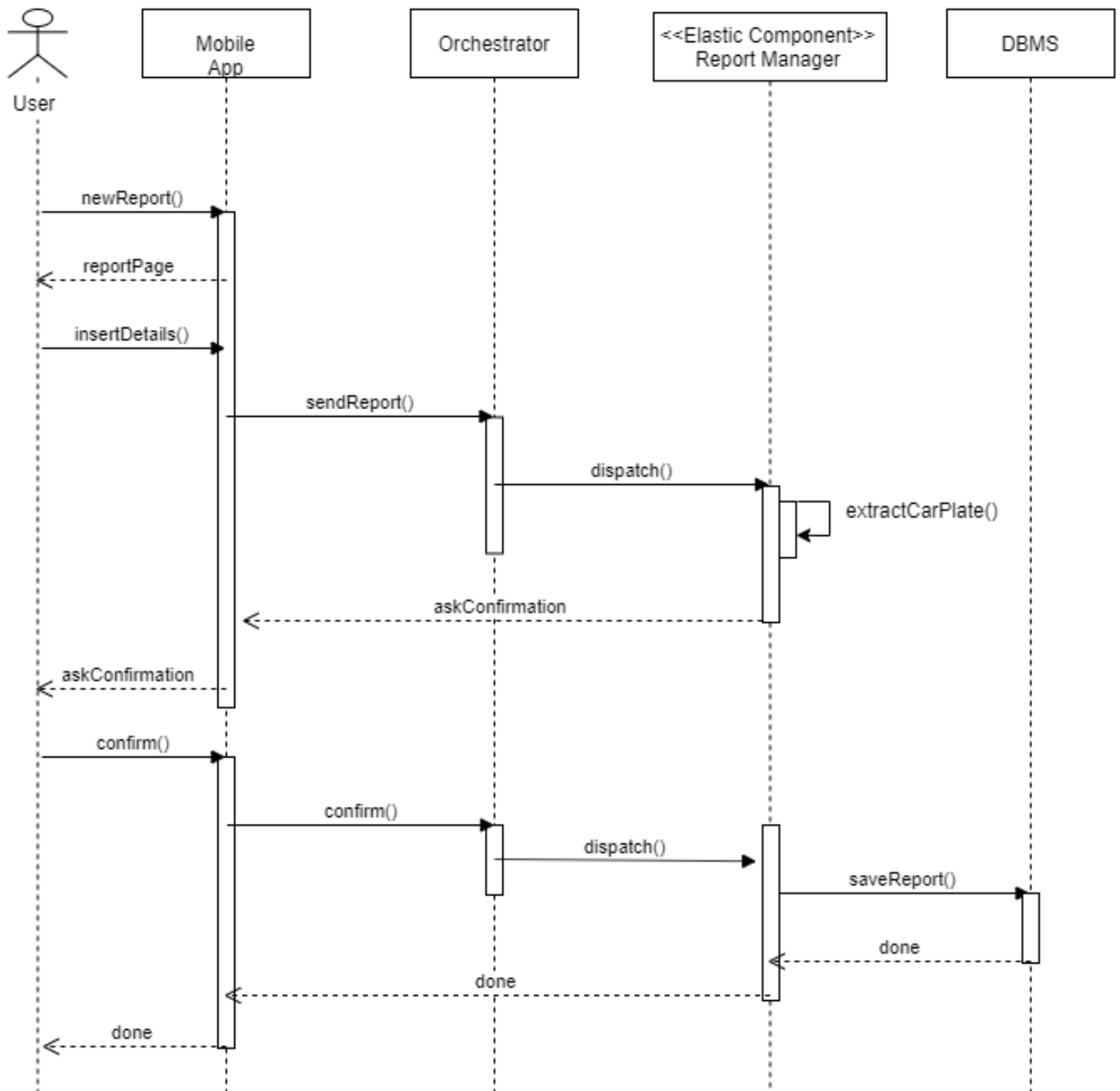


Figure 7: Sequence Diagrams - New Report

2.3.3 Sequence diagrams – Resolve Report

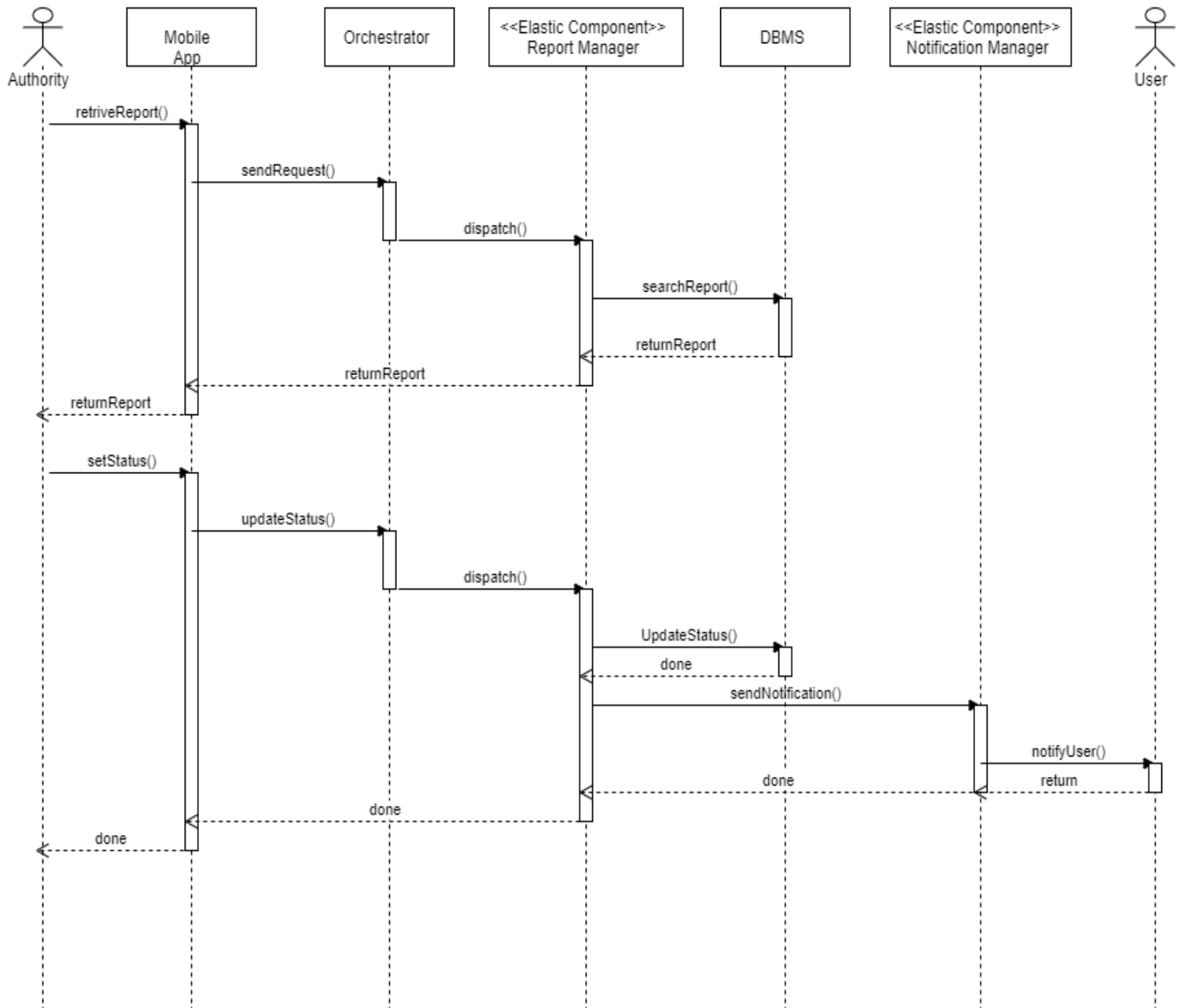


Figure 8: Sequence Diagram - Resolve Report

2.3.4 Sequence diagrams – Select TravelPlan

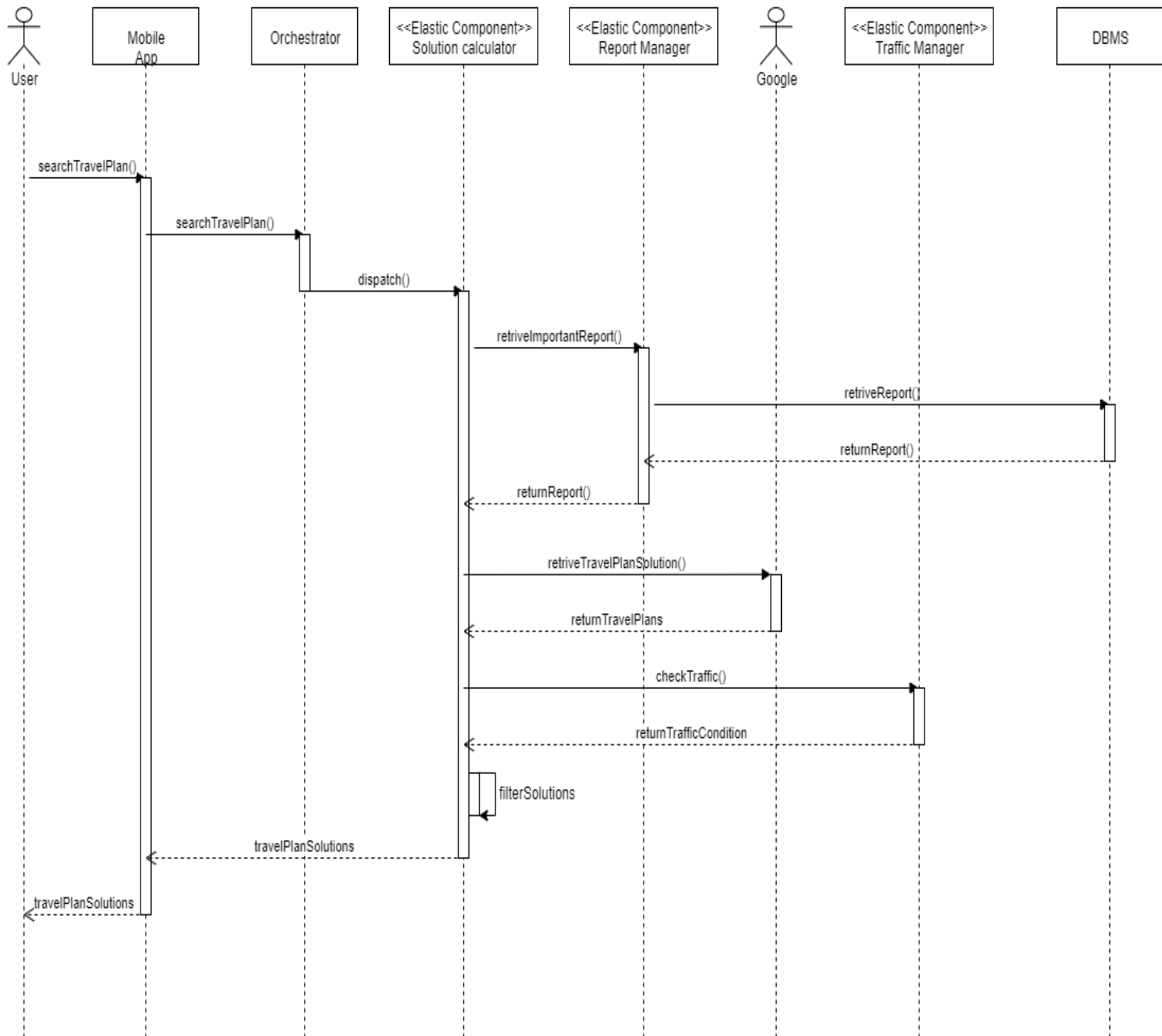


Figure 9: Sequence Diagrams - Select Travel Plan

2.3.5 Sequence diagrams – Show Statistics

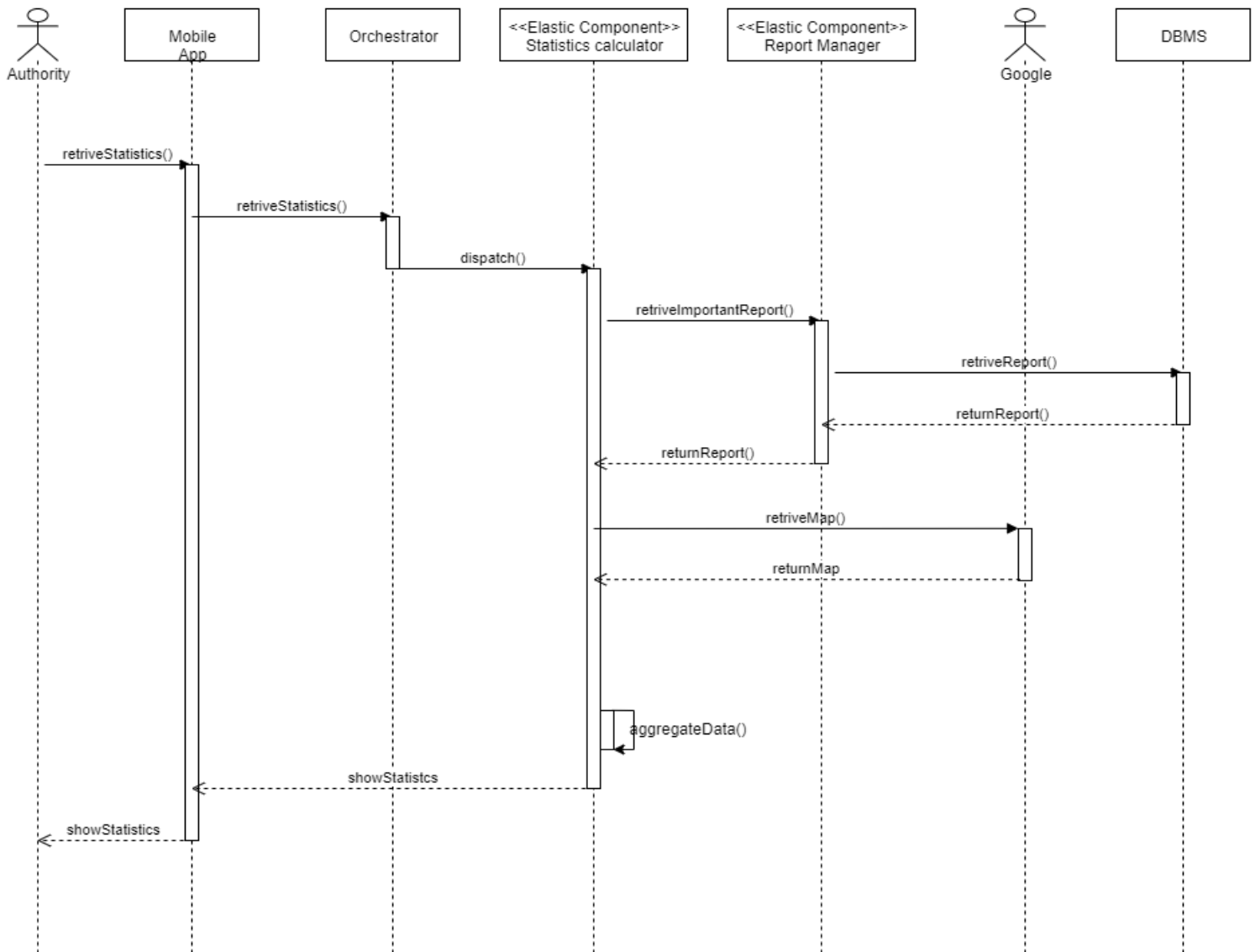


Figure 10: Sequence Diagrams - Statistics

2.3.6 Object Diagram – Traffic Module

Here is provided the object diagram of the traffic module.

The following two diagrams illustrate an evolution in the instances caused by a load balancing operation and an unexpected crash. Both diagrams are not complete of all instances and it is used a dashed line to represent the fact that some instances have been cut, to have a simpler and easier understanding of the diagrams.

In the first diagram, are shown two Querier, one for Italy and one for France. This partition of the region has been chosen by system administrator. The queriers are linked to the related Notifiers and to the Manager. The Manager keeps track of the Queriers, the active Notifiers and the Notifiers in the standby list.

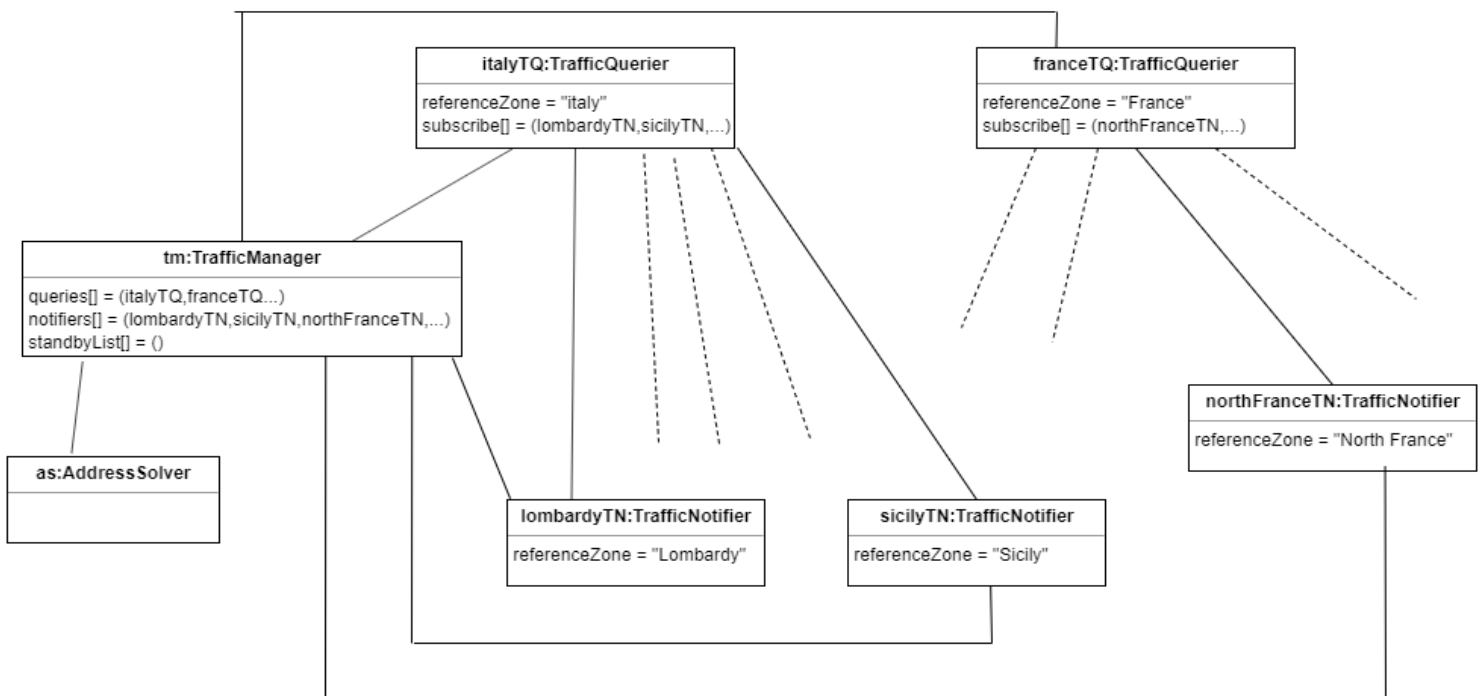


Figure 11: Object Diagram -Traffic manager original state

At some point, the Notifier related to Lombardy is under load and the load balancer split it into two Notifiers, respectively related to North Lombardy and South Lombardy. Their subscription to the Querier italyTQ is managed directly by the Manager as described in algorithm section (3.1).

The franceTQ Querier unexpectedly crashes and the Notifier that were subscribed to it are put in standby list by the manager.

The next diagram represent the final situation

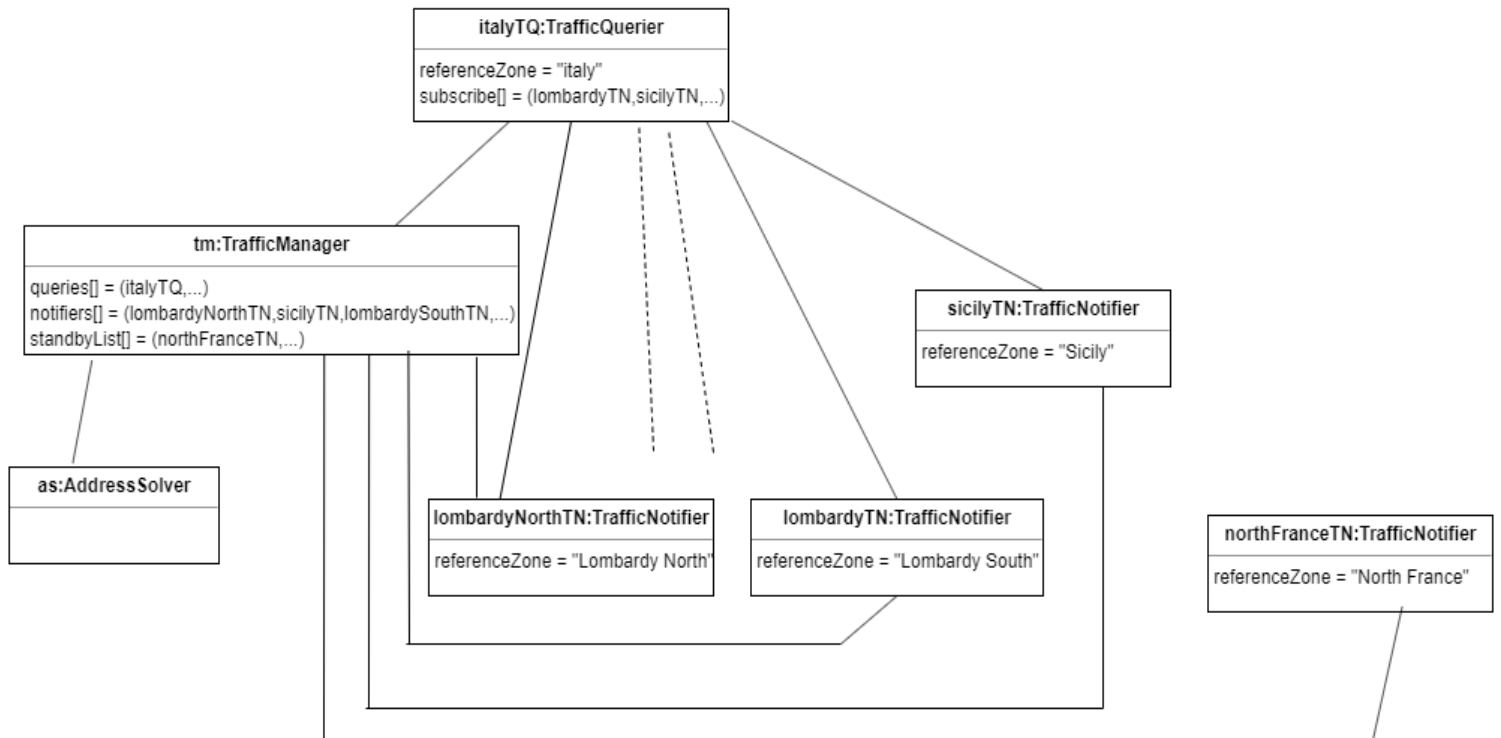


Figure 12: Object Diagram - Traffic module after balancing and reconfiguration

2.4 Component interfaces

Here are described most of the interfaces used and provided by the components defined in this section. The interfaces are not described here as are considered as trivial and as so, assumed to not need any description. An example of interfaces not described is the one used in Traffic Module, where they are mainly the standard interfaces of any publish subscribe system.

The User or the Authority that asks for the services are never passed as parameter because it is meant to be retrieved as a session parameter.

Report Manager

ReportEditorInterface

- Void newReport(String description, Date, Time, Location, String carModel)
- Void deleteReport(reportId)
- Void updateDescription(reportId, String description)
- Void updateDate(reportId, Date)
- Void updateTime(reportId, Time)
- Void updateLocation(reportId, Location)
- Void setStatusToClosed(reportId)

CarPlateExtractoreInterface

- String extractCarPlate(Image picture)

ReportViewerInterface

- Report[] getListFromDate(Date from, Date to)
- Report[] getListFromLocation(Location)
- Report[] getAll()
- Report getReport(reportId)

ReportStorerInterface

- Void update(reportId, Report)
- Void deleteReport(reportId)
- Void newReport(ReportId)

Notification Manager

This module make use of Data Interface, Notifiable, that is an interface extended by all the objects that can be sent as Notification.

NotificationDispatcherInterface

- Void notify(User, Notifiable)

2.5 Selected architectural styles and patterns

The following architecture styles have been used:

2.5.1 Client-server

The mobile is a client communicating directly with the application server. The browser supporting the web app is a client communicating with the web server. The application server behaves as a client querying the database server.

2.5.2 Service-Oriented Architecture (SOA)

The way clients interact with the application server is thought to be service-oriented. The single components are analyzed from a high-level point of view depending on the service they offer. SOA allows to easily extend the system by building and adding independent modules to the core.

2.5.3 Model View-Controller (MVC)

MVC pattern is followed throughout the whole system design. The clients are front-end components (views) interacting with logic component (controllers) which drive the information flow and the information retrieval from the database (model).

2.6 Other design decisions

2.6.1 Password storage

User's password and Authority's password are not stored in plain text, but they are hashed and salted with strong cryptographic hash functions.

3. Algorithm Design

3.1 Car-plate Extractor – How it works

The car-plate extractor is a submodule of the report manager that has inside a system of detection and recognition of the number plates of vehicles.

In particular, this system is has three different sub task that are:

License plate detection:

This task used a convolutional neural networks called YOLO (You Only Look One) and in particular YOLO v3. This network is trained to performed a regression task predicting the object bounding box and the object class.

The advantages in using this network is that process images in real-time.

License plate segmentation:

Once extracted the image of the plate, this starts a pipeline that have the following step:

- Conversion from BGR to GRAY
- Binarization
- Blur
- Segmentation

After the final step a histogram of pixel projection is performed both in horizontal and Vertical (see below).



License plate recognition:

Once the segmentation is finished each character is cut individually and resized in a square. As a result we have a 28x28 image. This data is given to a MLP (MultiLayer Perceptron) to extract the String corresponding to the car plate.

3.2 Traffic Modules – Dynamic configuration

As described before the Traffic module is subject to load balancing and dynamic configuration by the System Administrator. These two mechanisms cause four events to happen. How they are managed is described below:

- A Notifier is deleted: this is the only activity that is not managed by the Manager because there is no reason for doing it. If the Notifier is simply being closed, it detaches itself from the Querier. If the Notifier crashes or it suddenly closed, the Querier will notice this when trying to notify it and it will detach the dead Notifier. No other actions are needed.
- A Notifier is created: the new Notifier communicates its zone to the Manager. The Manager will return to the notifier a reference to the appropriate Querier using the Address Solver to interpret the zone of the Notifier. Thus, the Notifier can subscribe itself to the Querier.
- A Querier is deleted: all the Notifier previously attached to the Querier have to be analyzed. If a less specific Querier exists, they are attached to it, otherwise they are put in a standby list.
- A new Querier is instantiated: the standby list is scanned searching for Notifier that can be attached to the Querier (matching the two zone

through the Address Solver). If a less specific Querier exists, all the Notifier subscribed to it are analyzed and are eventually moved to the new Querier.

Meaning of specificity of a Querier

Let's assumed, for instance, that there are four Querier: *ItalyQ*, *MilanQ*, *LazioQ*, *ParisQ* respectively related zones: Italy, Milan, Lazio, Paris. *MilanQ* and *LazioQ* are more specific respect to *ItalyQ* because Milan and Lazio are inside the region Italy. On the other hand, *ItalyQ* is less specific with respect to *MilanQ* and *LazioQ*. *ParisQ* has no relation of specificity with all the others.

Hence:

- *LazioQ* crashes -> all the Notifiers subscribed to *LazioQ* are now moved to *ItalyQ*
- *RomeQ* Querier is created -> *ItalyQ* is less specific than *RomeQ* so *ItalyQ* is scanned searching for Notifiers associated to region Rome: if there are such Notifier, they are moved to *RomeQ*. Notice: the Notifiers that were associated to *LazioQ* but are actually outside the region Rome would remain associated to *ItalyQ*.
- *ParisQ* crashed -> all the Notifiers associated to *ParisQ* are moved to the standby list, because no 'less specific' Queriers are available.

Notice that Queriers are meant to be wide regions and not single cities, this was just an example.

3.3 Solution Calculator – How it works

Here is illustrated the procedure by which the S2B provides the User with the travel solutions for the selected destination. As described in the sequence diagrams previously the Solution Calculator takes care of handling the two-step articulated process: recovery of feasible travelPlanSolutions and filtering.

Recovery of feasible travelPlanSolutions:

- Solution Calculator retrieves the departure point and the arrival point from the user's input ;
- Based on this information, Solution Calculator interfaces with google, which provides it with a list of possible travel solutions.

Filtering

- Solution Calculator, through Report Manager, retrieves report information (Location and type);
- Solution Calculator, through Traffic Manager, retrieves information about traffic conditions;
- For each TravelPlan in TravelPlanSolutions, Solution Calculator checks if there is a TravelPlan that has:
 - o More than 50 minutes in addition to the normal travel time due to traffic
 - o More than 4 reports that can cause a relevant delay along the path

In case that a TravelPlan have this characteristics then it is removed by the TravelPlanSolution.

Once that the filtering process is finished the Solution Calculator sends TravelPlanSolution to the User.

4. User Interface Design

4.1 App Mockups

Here are provided several mockups to highlight how the user interface should be for the different type of users.



Figure 14: Authority-login page



Figure 13: User-login page

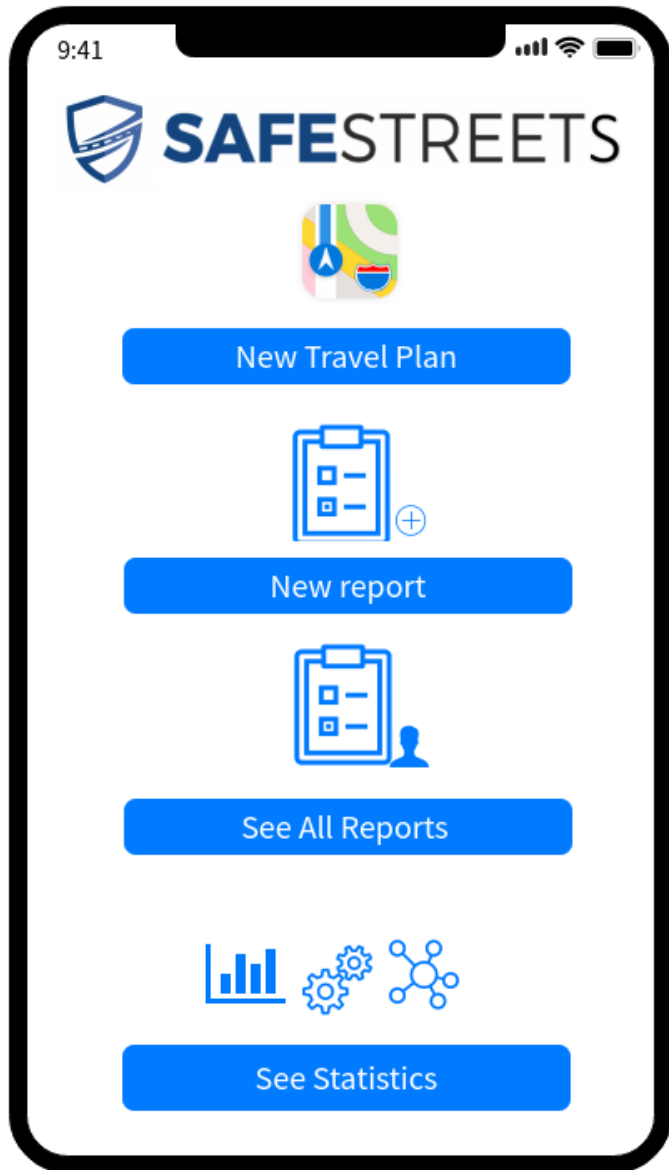


Figure 16:Authority-Home page

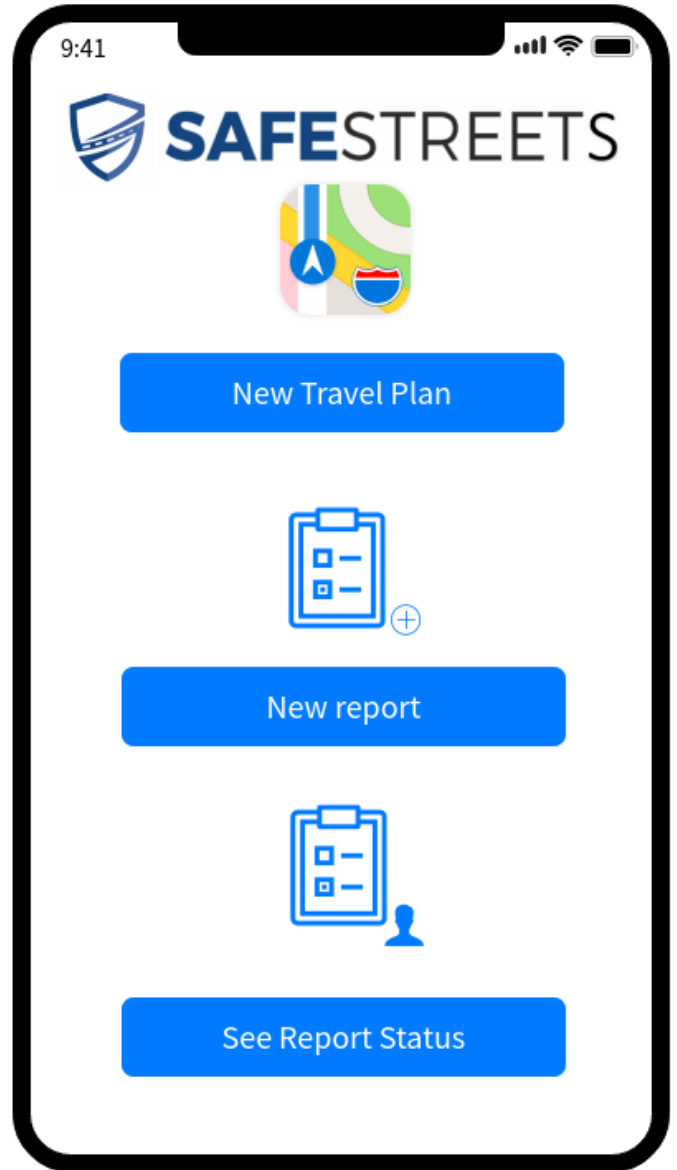



Figure 15:User-Home page



SAFEStreETS


Please insert the violation's type

Q insert type of violation


Please insert the car's model

Q insert car model

Please insert car plate or take a picture of it

Insert car plate or 

Please insert the location
or
acquired your current position

Insert location or 

Send

Figure 18: New Report page

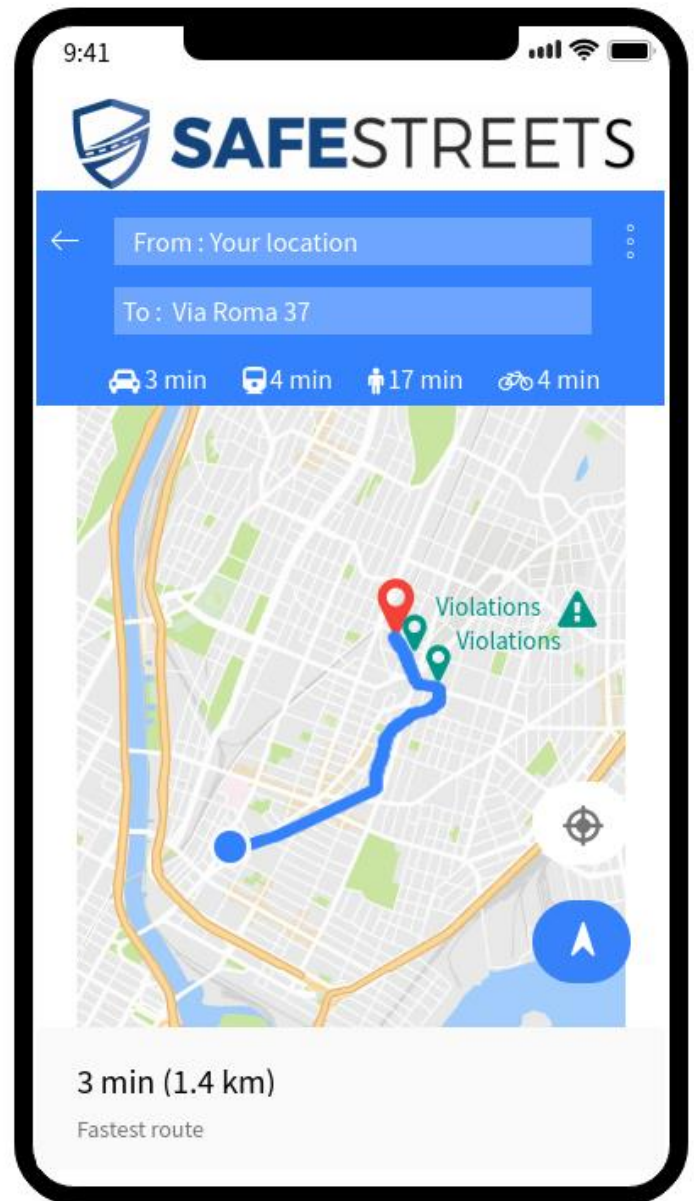


Figure 17: Travel Plan visualization



Figure 19: Authority-Statistics page

5. Requirements Traceability

Here is shown the mapping between goals and requirements identified in section 3.3 of the RASD and the modules of the business logic described in the previous sections of this document. It is to notice that there are listed only the components of the actual logic, omitting the DBMS, the Orchestrator, The Mobile App and the Web App because they are omnipresent.

Goal G1: a Person should be able to have a profile on Safestreets

➔ Requirements R1 to R6

Components:

- Account Manager: as it can be seen in the Sign Up sequence diagram, it is the component in charge of the sign up process.

Goal G2: a User should be able to send reports regarding traffic violations

➔ Requirements R7 and R8

Components:

- Report manager: allows the user and the authority to create and send new report regarding different kind of violations

Goal G3: Authorities should be able to access to all the reports

➔ Requirements R9 to R11

Components:

- Report manager: allows the Authorities to access all the Reports

Goal G4: Costumers should be able to access different kind of information depending on their role

➔ **Requirements R12 to R14**

Component:

- Based on the class of the user the Report Manager permit the uses of certain methods and prevent the use of others.

6. Implementation, Integration and Test Plan

Whit the goal of identifying a proper developing plan, for each feature it was analyzed:

- The importance for the Users and the Authorities: in terms of how much the feature are perceived fundamental from the users (from now on the term *user* will identify both Users and Authorities) point of view. Core features have been labelled with a high or medium-high importance. They have to be implemented completely during the first release. Features with importance low or medium-low are the ones that could be implemented in successive releases as additional features that add value to Safestreets but are not part of the core functionalities.
- Difficulty of implementation of the Back End: it defines an estimate of the effort that will be spent in the implementation of the communication issues, the logic and the data models on the server required by the feature.
- Difficulty of implementation of the Front End: this indicates an estimate of the effort required to implement in the App (Mobile app or Web app) the graphic and the user interactions.

The following table summarizes the result of the analysis.

Feature	Importance for user	Back End	Front End
Sign Up	Medium-high	Low	Low
Report creation and management	High	Medium-low	Medium-low
Report visualization	High	low	Medium-high
Solution Calculation	Medium-low	Medium-high	Medium-high
Notification system	Low	Medium-low	Low
Traffic updates	Medium-low	high	low

Verification and validation process will begin as soon as the system starts to be implemented. For each component (and subcomponent belonging to it) there are two phases: the first one is an analysis concerning in a manual inspection by teams members, the second one is an automatized test using tools support. It was chosen to make a manual inspection of the project because, in this way, there is a major formal level compared to a Walkthroughs analysis; in fact, inspection is an expensive technique, but it guarantees a low level of error in the final product.

Below there is a list ordered (time increasing) by system features and not by components because, for the integration process, it has been chosen to adopt Thread strategy. In this way it can be realized to the costumer more than one version of the product, from the version that contains just the core functionalities to the complete version, adding step by step new features. Moreover, Thread strategy allows implementing also a part of a specific component and complete it adding new features in a later time.

Components will be implemented, integrated and tested in the following order:

1. **Report creation and management:** to satisfy this functionality, the Report Manager is implemented, analyzed and tested. In particular the carplate-extractor should have a high accuracy.
2. **Sign up:** to satisfy this functionality, part of the Account Manager is implemented, analyzed and tested. In particular, the part that allows users to sign up into the system in order to be recognized during future access.

3. **Report visualization:** to satisfy this functionality, the implementation, analysis, and test of Report Manager continue. In it there isn't any additional component but, in this phase, it is handled the graphics interface concerning the report visualization. From now it is possible to have a core release.

From now on each feature is a single module so it can be released step by step.

4. **Solution calculation:** to satisfy this functionality, part of the solution calculator is implemented, analyzed and tested. Solution calculator , at this point, interfaces with the Report Manager and with google API. It doesn't interface with the traffic module.
5. **Notification system:** in this phase Notification Manager is implemented, analyzed and tested. At this point the Users can receive a notification when an Authority set one of their report to closed.
6. **Traffic Update:** to satisfy this functionality, the Traffic Module is implemented, analyzed and tested. Now Solution calculator processes travel solution taking into account the traffic.

During integration test process (and before its termination), the following must subsist:

- DB must be activated, it must contain an adequate quantity of data in order to test system's performances by overloading it. In this way, it is possible to test a big part of the components interfaced with the DBMS.
- Interactions with external actors must be present, both on the contractual and on the communicative level. In particular the Municipality have to subsist before Authority Sign Up is implemented. In order to complete test the system. Google must also be interfaced with the system before completing the implementation of the Traffic module.

It was chosen to implement, analyze and test at first, in a parallel way, Server and the Mobile App. The Web App development will start as soon as the implementation and the test of all high and medium-high priority (for the users) features will be completed.

7. Effort Spent

I spent a week to complete the project. Of this week I spent two days for the second chapter and for all the others one day each.

8. References

8.1 Software and Tools

- Word: for typesetting document
- GitHub & git: as version control
- Draw.io: for UML diagrams and ER diagrams
- Wireframe: for the mockups
- MeisterTask: for task assignment

8.2 Reference Documents

- 1016-2019 – IEEE Standard for Information Technology -Systems Design – Software Design Descriptions:
<http://ieeexplore.ieee.org/document/5167255/>
- 42010-2011 – ISO/IEC/IEEE System and software engineering – Architecture description: <http://ieeexplore.ieee.org/document/6129467/>
- YOLOv3 paper: <https://arxiv.org/pdf/1804.02767.pdf>