Politecnico di Milano
School of Industrial and Information Engineering
Master of Science in Computer Science and Engineering

Software Engineering 2 Mandatory Project

Safe Streets.
Design Document

Authors:
Rosetti Nicola
Sartoni Simone
Torri Vittorio

Academic Year 2019/2020

Milano, 09/12/2019
Version 1.0

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

1.2 Scope

1.3 Definitions,Acronyms,Abbreviations

1.4 Revision History

1.5 Reference Documents

1.6 Document Structure

2 Architectural Design

2.1 Overview: High-level components and their interaction

The application will be developed using the client-server paradigm on a three-tiered architecture. The three layers of the application (Presentation, Application and Data) are divided into clusters of machines (i.e. tiers) that actually cooperate to provide a specific functionality. In this case we have three tiers and each tier is responsible for one of the three layers. The client side is responsible (only) for the presentation layer; therefore, in this architecture, the thin-client has been adopted considering the fact that the required functionalities client-side are limited. The UIs provided are just meant to show results and to allow clients to choose what they want. In the App case, the client contains all the presentation layer while in the WebApp case the layer is splitted between the client and WebServer; the WebServer is responsible for contacting the application server and forward the client requests to it. The Application tier takes care of the application layer encapsulating all is needed concerning the application logic. It receives the requests from the clients and handles them. It's also responsible for sending asynchronous notifications to the presentation layer when certain conditions are met. Here we have multiple Servers cooperating together to improve performance, scalability, fault tolerance and availability. An elastic component (i.e. load balancer) is used to rule the accesses to different Application Servers, dinamically balancing the load among all the Servers. The Application tier communicates with the Data tier, responsible for the Data Access layer. This tier is composed by several DataServers: each one is associated with a single replica of the data and exploits the DBMS technology to access the DataBase. The Database is fully replicated in different nodes. Techniques and protocols are used to ensure consistency among replicas: they will be full explained in "Other design decisions" section. Again a load balancer is used to dinamically share the load among different machines. To ensure and improve security firewalls are installed before and after the application servers to filter accesses from external and unsafe networks. By the creation of a DMZ (demilitarized zone) external entities can only have access to the exposed services. Security is crucial because the application works mainly with sensible information. To provide the required functionalities the system exploits datawarehousing. The datawarehouse is a component in the Data tier able to deal with historical data and aggregate data taken from the Databases exploiting data mining technologies to answer complex queries: used techniques are clustering, associative rules and classification. This component periodically queries one dataServer to retrieve new information and updates on the data since its last update.

2.2 Component view

In the figure **??** is reported the *Component Diagram* for the SafeStreet system. This is a high level view in which the main components are shown, then some components will be better detailed.

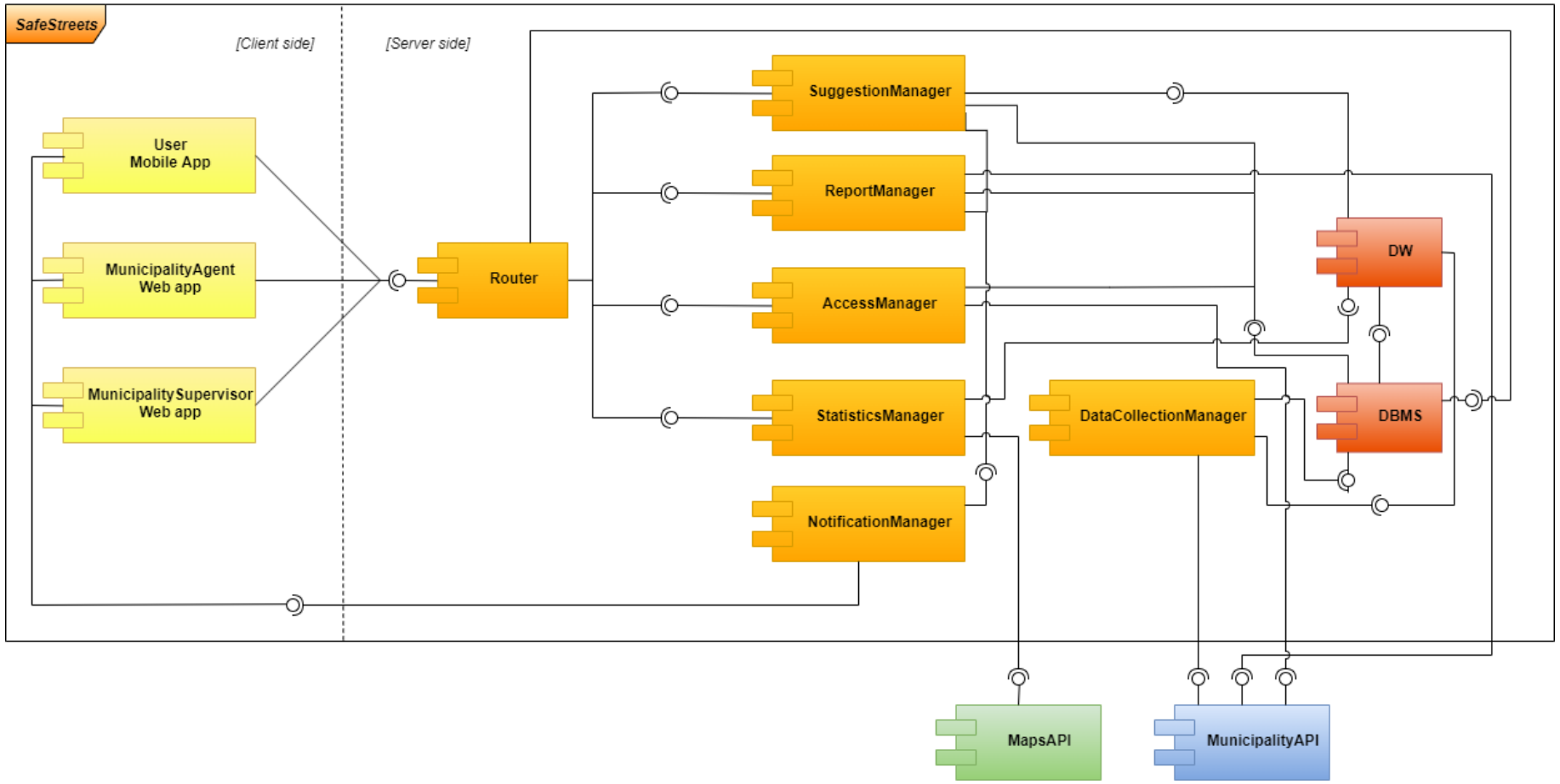

Figure 2.1: UML Component Diagram

In this diagram when two components can communicate using different interfaces a single interface link is reported, for the sake of readability. The various components are now described and detailed:

- **Router**: it has the role of dispatching the requests coming from the users applications. Before doing this it has also the important role of verifying the user authentication, checking the token which is sent with all requests performed by an authenticated user. In figure **??** a more detailed view is provided, putting in evidence the various interfaces for the mobile and for the web app.

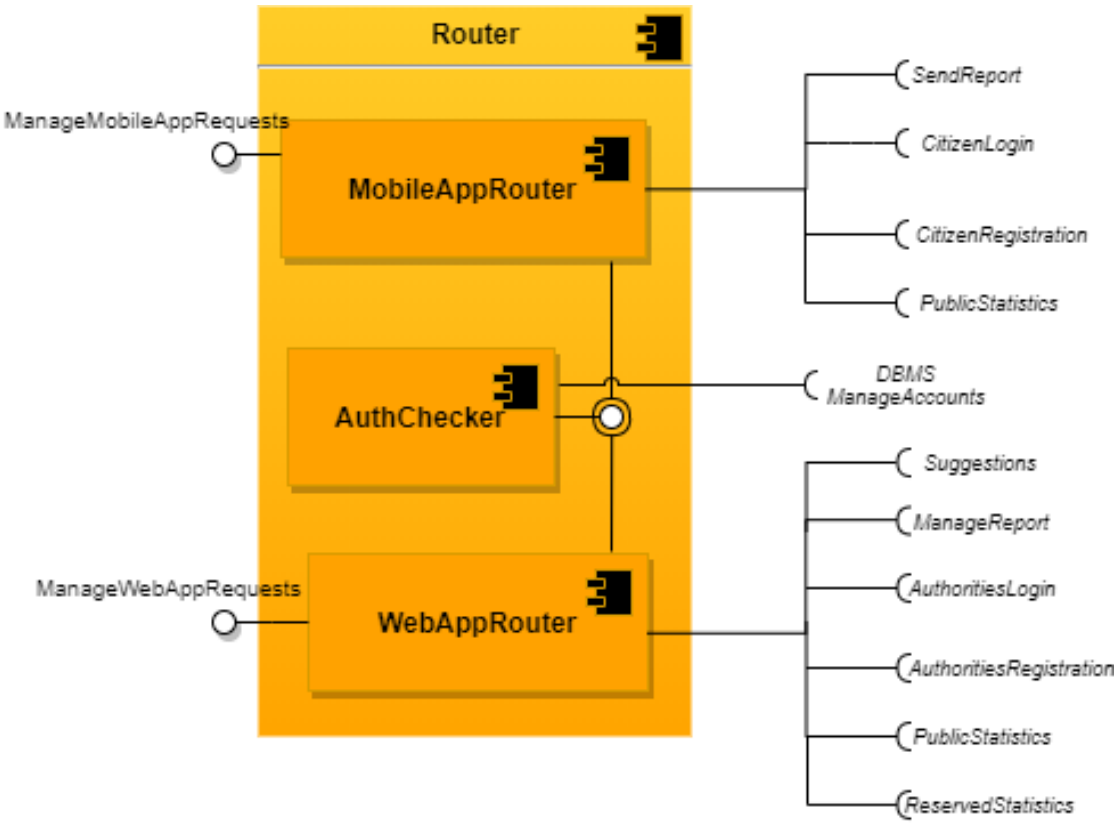

Figure 2.2: UML Component Diagram for *Router* component

- **SuggestionsManager** this components is the one which elaborates and provides the *Suggestions* for the municipalities. It analyzes the data exploiting the *DataWarehouse* component for the aggregated queries about reports, accidents and tickets and the *DBMS Component* to retrieve non-aggregated information about the streets. It can provide the actual available suggestions for a municipality as an answer to a request coming from the *Router*, but it also notify the *Municipality Supervisor web app* through the *Notification-Manager* component when it discovers a new suggestion (the suggestion discover task is periodically executed, once a month).
- **ReportManager**: this component is in charge of all concern the management of the reports sent by the users. It receives them by the mobile application and execute the

automatic analysis to identify possible fake reports, it stores them through the *DBMS Component* and it provides the reports to the the *Municipality Agents web app*, answering to their requests of confirm/enqueue/discard and allowing them to emit a ticket, exploiting the *Municipality APIs* which provide an interface for this service. Its detailed diagram is shown in figure **??**.

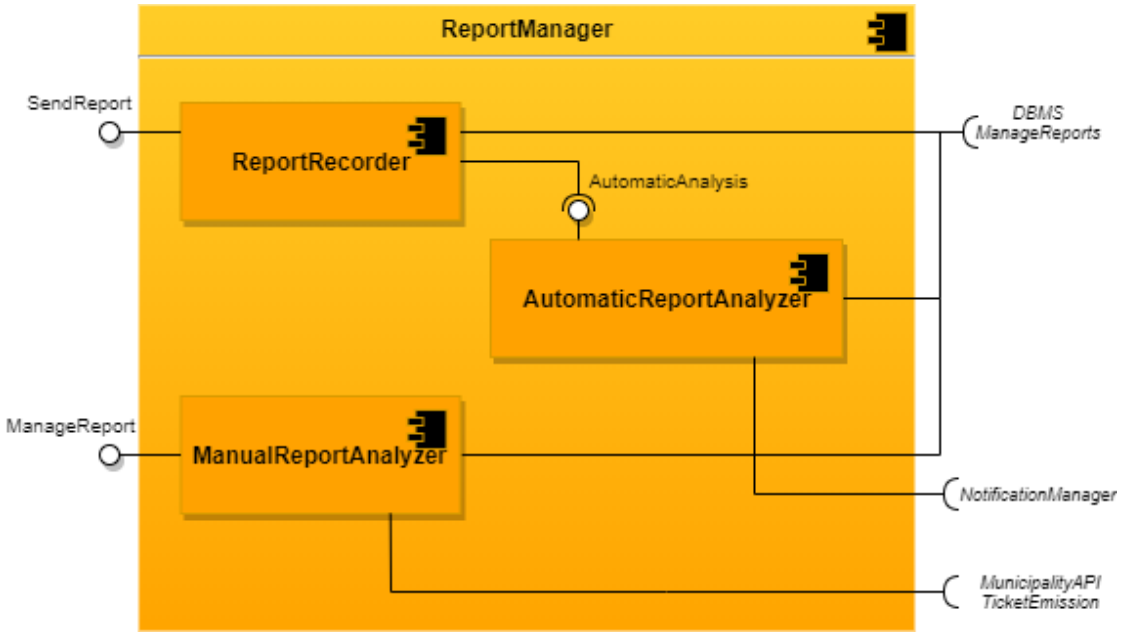

Figure 2.3: UML Component Diagram for *ReportManager* component

- **AccessManager**: it manages everything about registration and login of the users. For the citizens registration it calls the municipality service for the verification of identity cards, while for the agents and supervisors registration it calls the municipality service for their identity verification. He stores the accounts data through the *DBMS* component and it calls it also to verify the correctness of the login credentials. Once a login has been correctly performed it generates a token, stored in the database and sent back to the user, necessary to perform authenticated requests. This component is detailed in figure **??**.
- **StatisticsManager**: it answers to the statistics requests, both public and reserved, and in doing this it calls the *DataWarehouse* component. It is shown in the diagram in figure **??**.
- **NotificationManager**: this component is the one able to send push notifications to the user clients, both mobile apps and web apps. It is exploited by the *SuggestionManager* and by the *ReportManager*.
- **DataCollectionManager**: this component periodically retrieves the data made available by the municipality about accidents, tickets and street characteristics. These latter are store in the operational database, while the others are retrieved as aggregated data and so are directly stored in the datawarehouse.
- **DBMS**: this component manages the operational database, the main base for the functions of SafeStreets, being in charge of reports, users and streets data.

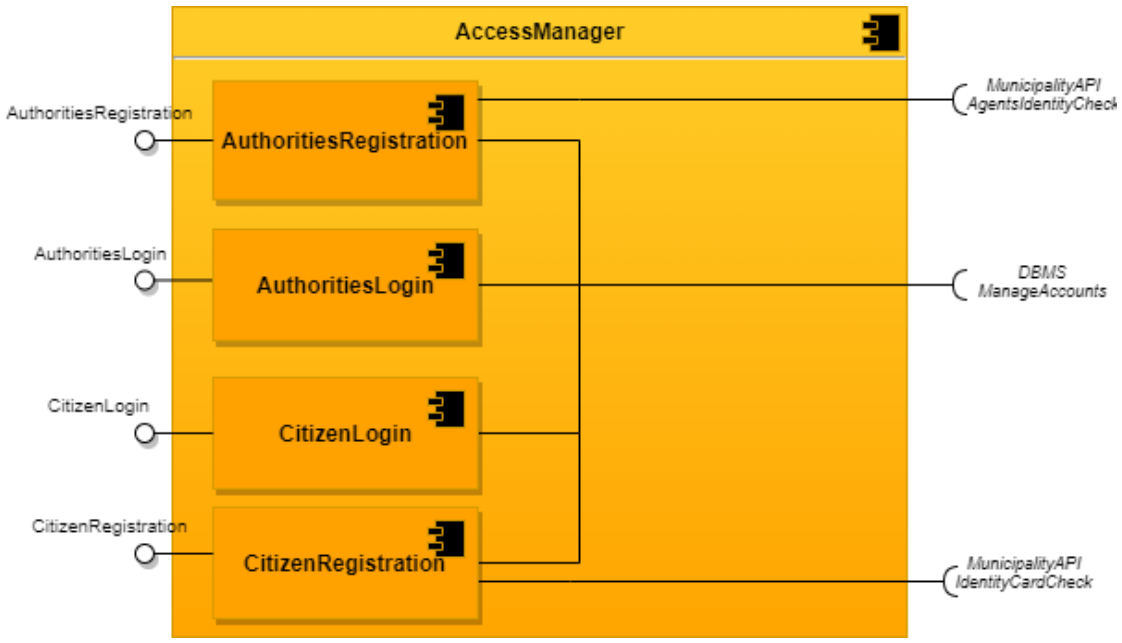

Figure 2.4: UML Component Diagram for *AccessManager* component

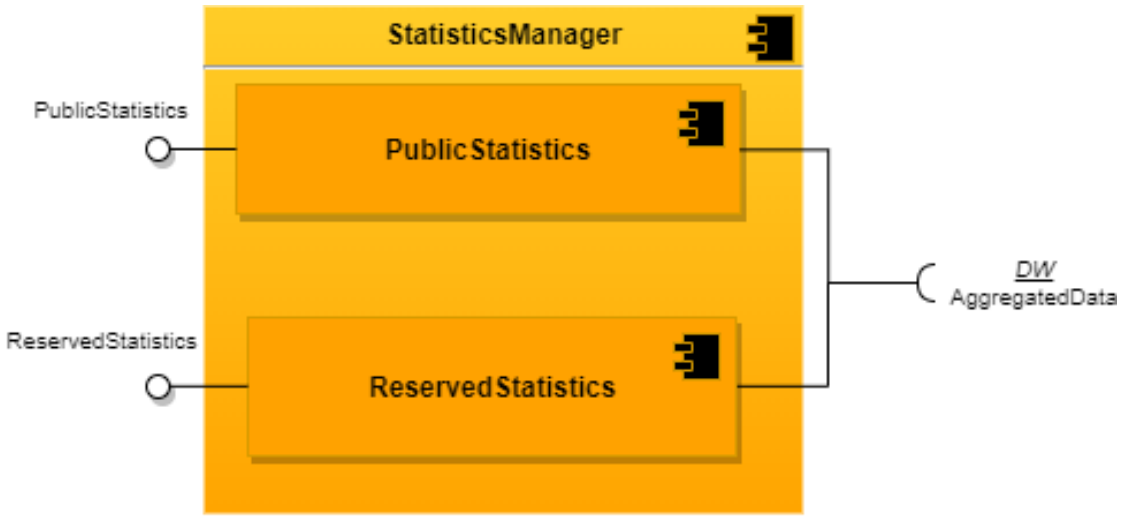

Figure 2.5: UML Component Diagram for *StatisticsManager* component

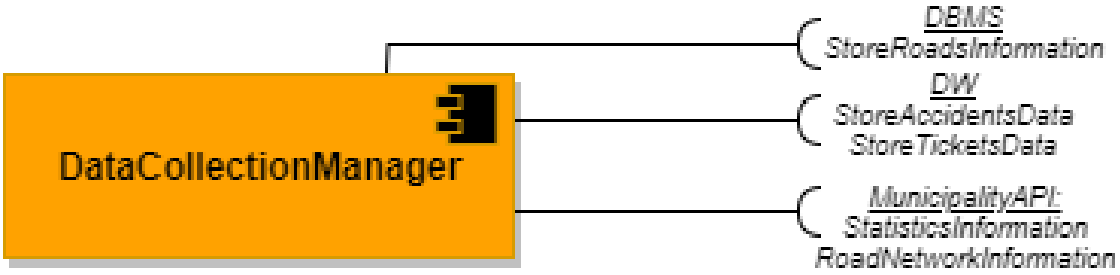

Figure 2.6: UML Component Diagram for *DataCollectionManager* component

- **DataWarehouse**: this component manages the datawarehouse, the secondary database for the system, used to answer aggregate queries for statistics and suggestions. It is periodically alimented with the dbms data and with the already aggregated data coming from the municipality through the *DataCollectionManager* component.
- **MunicipalityAPI**: this is an external component, managed by the municipalities (an instance for each of them which supports the system), whose interface is standardized as described in the *Components Interfaces* section. It is necessary to retrieve the data for building statistics and suggestions and to have the possibility of traffic ticket emission through SafeStreets.
- **MapsAPI**: this is an external component which is exploited to show maps with the statistics data.
- **UserMobileApp**: this component entirely resides in the *Presentation layer* and it's just an interface to allow the users to use SafeStreets on their mobile devices, sending reports and retrieving statistics information. It performs only minimal controls on the forms before sending them (ex: they are not missing mandatory field).
- **AgentWebApp**: this component is the *Presentation layer* of the web app which allows agents to see the reports, analyze them, emit tickets and retrieve users data.
- **SupervisorWebApp**: this component is the *Presentation layer* of the web app which allows supervisors to see the reports, possibly analyze them, retrieve all type of statistics and receive the suggestions elaborated by the system.

2.3 Deployment view

2.4 Runtime view

In the figures ??, ??, ??, ?? some architectural sequence diagrams are provided, showing the interactions among the different components in the execution of various functions of the systems. The only methods present here and not mentioned in the *Component Intefaces* section are internal method of a single component.

2.4.1 User sends a Report

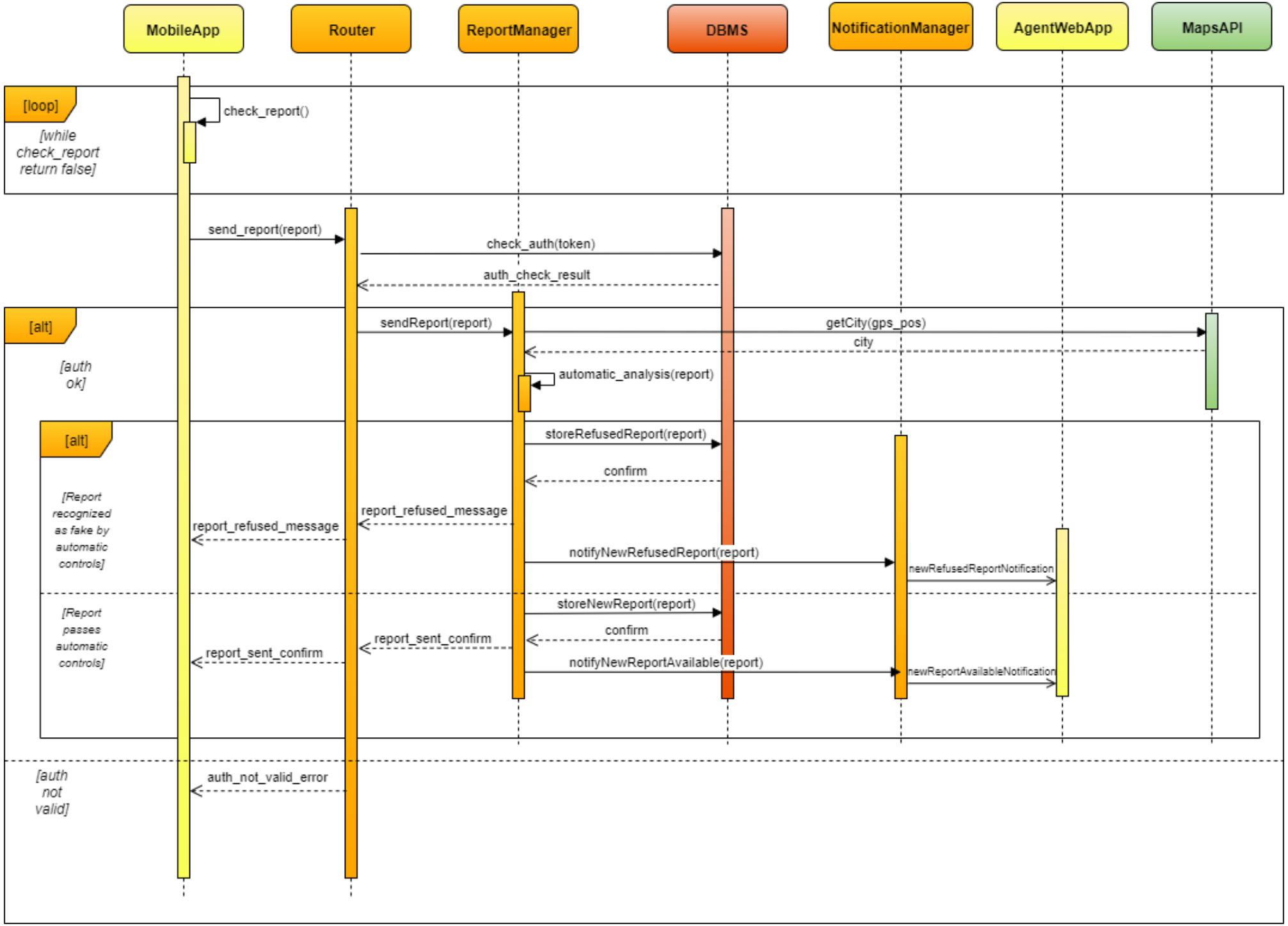

Figure 2.7: UML Architectural Sequence Diagram for *SendReport* use case

In this sequence diagram is shown what messages are actually exchanged among components to process correctly a report send by a User. The User, through the Mobile App, fills the report form and sends it to the server for a confirm. The report is handled at first by the Router that checks if the User can actually send reports (i.e. verifies if the corresponding token has the rights needed). if the check is negative, then the request is discarded. Otherwise the ReportManager contacts the MapsAPI to retrieve the municipality in charge for a specific report. After this, the ReportManager checks internally if the pictures in the report have been modified or not. If so, the report is stored as "Refused" and sent to the Notification Manager, that will send a notification about the refused report to the involved MunicipalityAgent Web App. If the pictures are not fake then the report is stored as New report and the NotificationManager will be in charge of notifying the Municipality Web App about the new pending report.

2.4.2 User Logs in

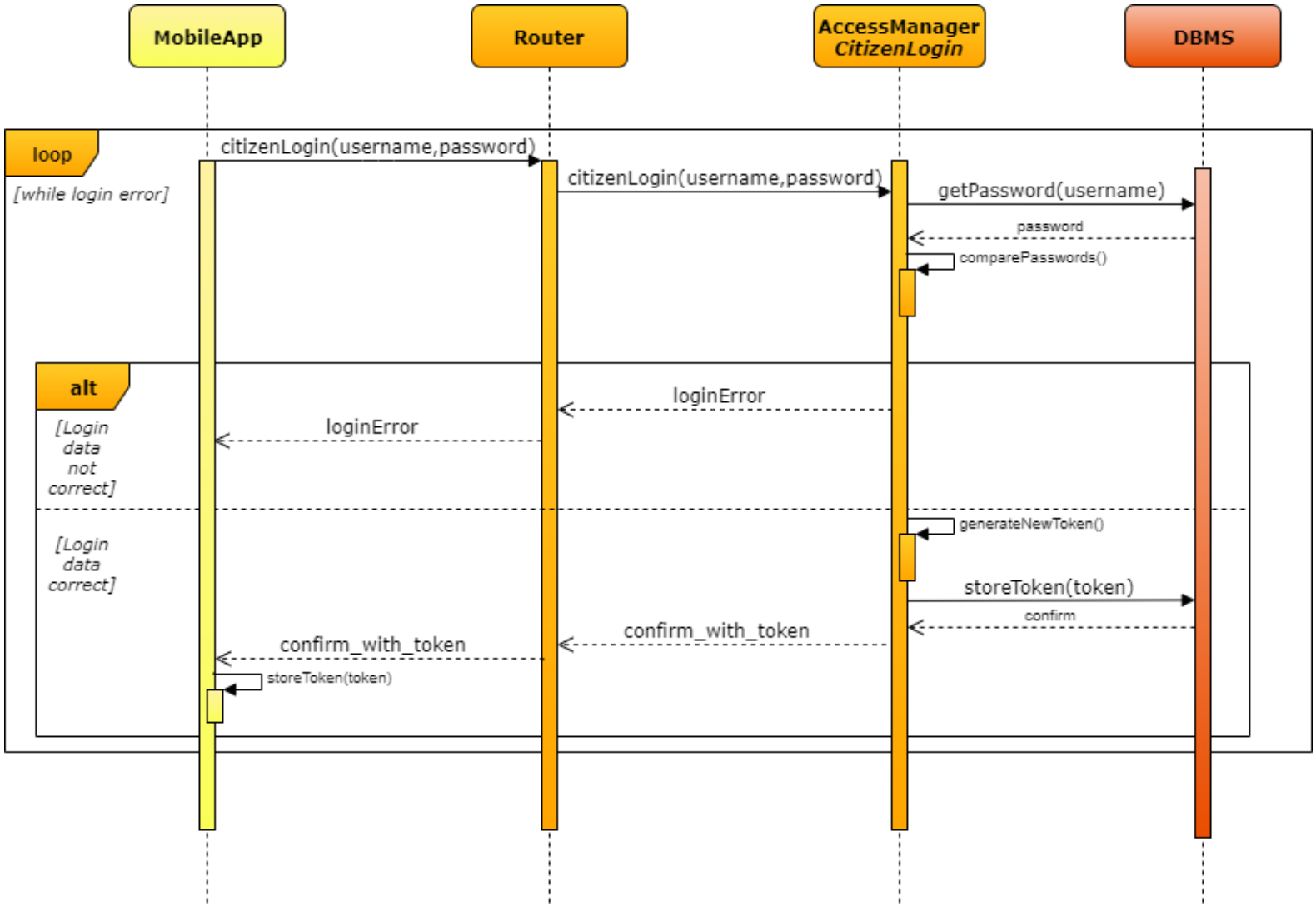

Figure 2.8: UML Architectural Sequence Diagram for *UserLogin* use case

2.4.3 Agent Checks Report

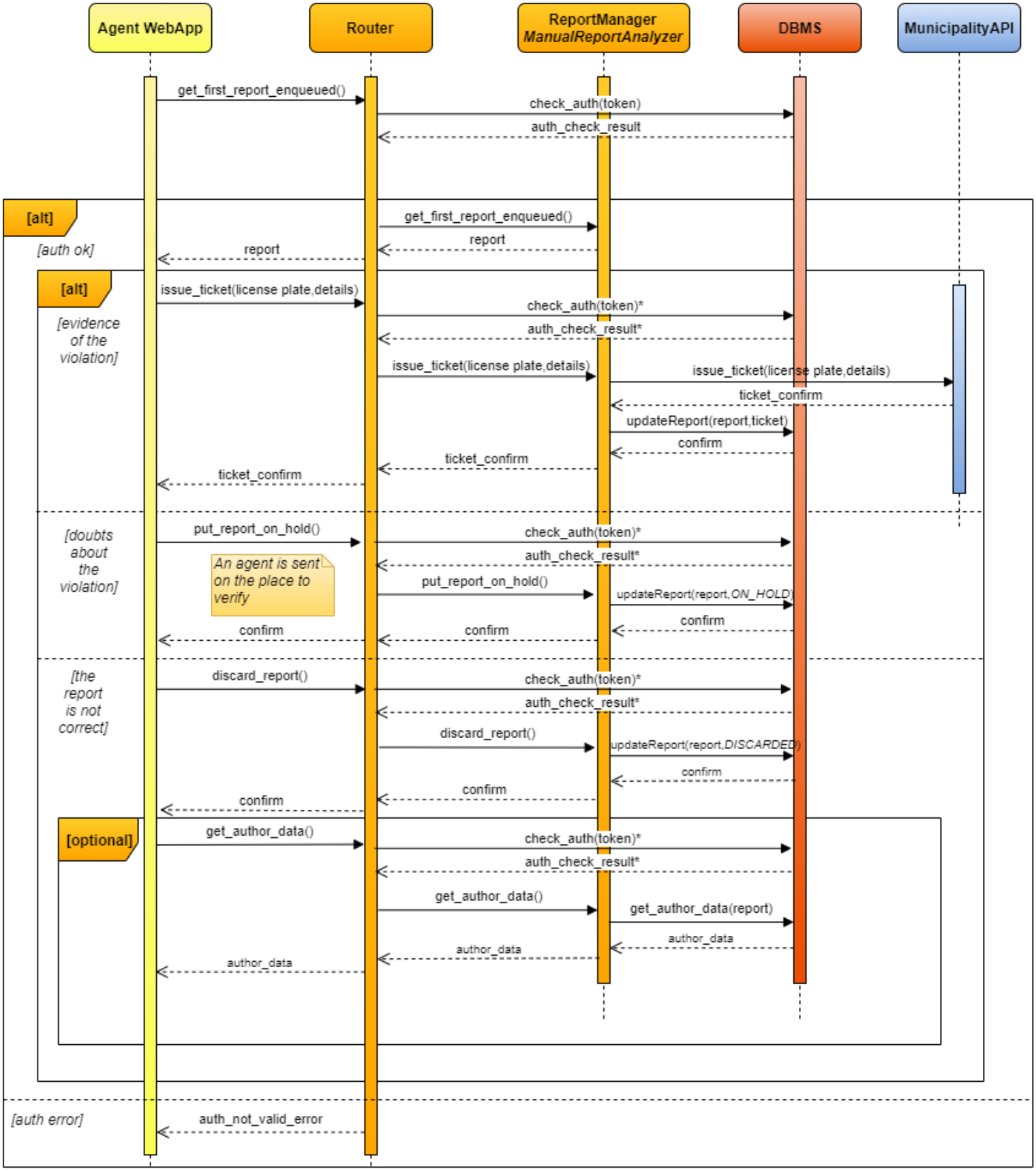

Figure 2.9: UML Architectural Sequence Diagram for *AgentCheckReport* use case

In this sequence diagram are described the components involved i nthe user login and their interaction. The User fills the spaces in the Mobile App inserting his username and password. The login request is then sent to the Router that forwards the message to the AccessManager. In particular the subcomponent **CitizenLogin** handles the login, asking the DB the password corresponding to the specified username. After having received the password, it internally checks if the two passwords (the one coming from the user and the one stored)coincide: if so, the user is successfully logged in and this is notified back to the client, showing him the main page of the App; if not, a notification is sent back to the client, showing him an error message and asking him to insert again username and password. Notice that everytime a new login is performed a new token is generated associated to the specific logged client.

2.4.4 Supervisor Checks Statistics

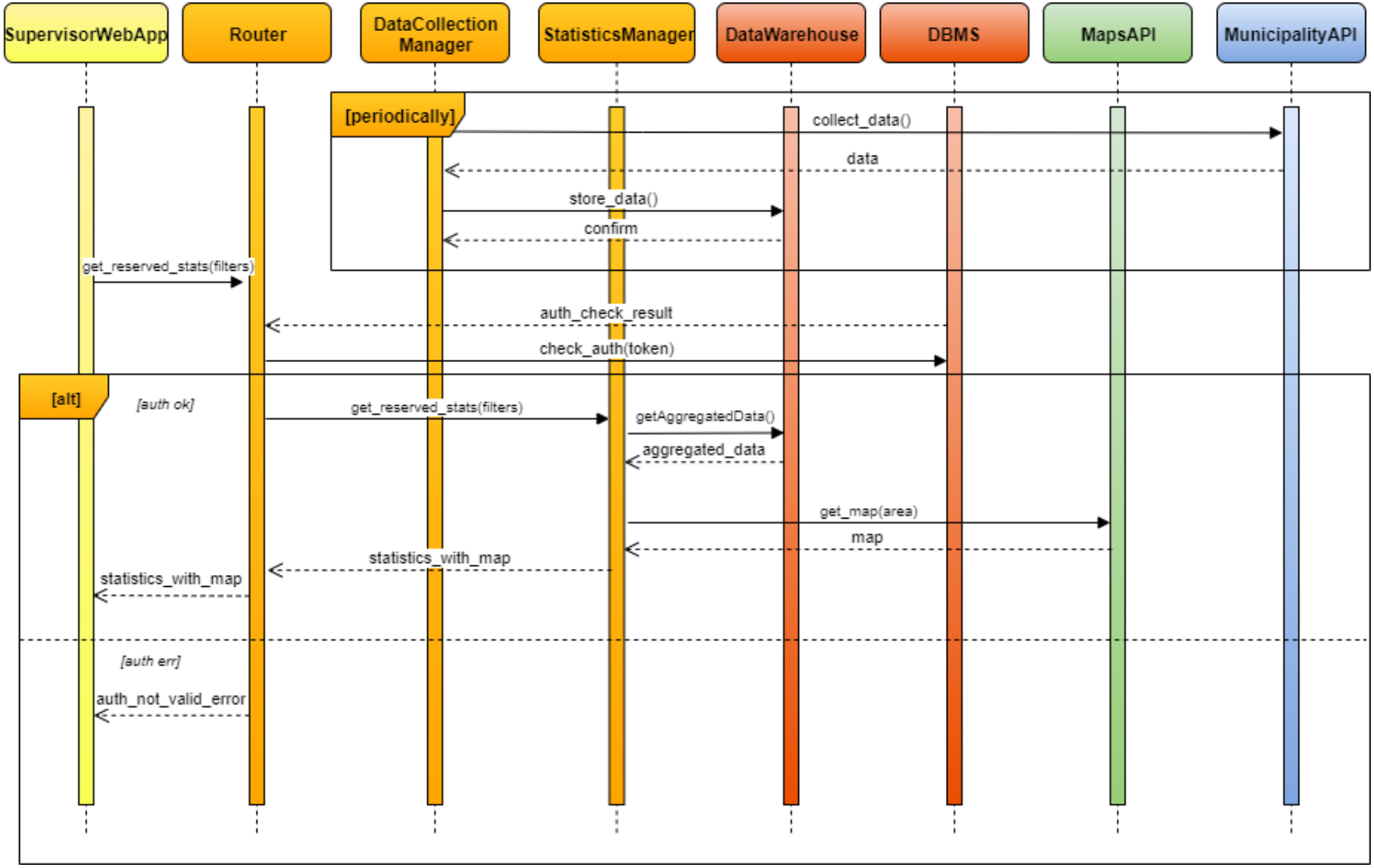

Figure 2.10: UML Architectural Sequence Diagram for the *CheckStatistics* use case

In the sequence diagram of figure ?? it has assumed that the *getCity* request to the municipality API returns a city for which the *SafeStreets service* is active. In effect the mobile app should not allow the user to send a report for a city in which the system is not active.
In the sequence diagram of figure ?? the control for the user authentication is shown for every request, but only the first one presents the *Alternative* choice, for the others, marked with *, it has been assumed a positive answer, for the sake of readability, avoiding to repeat the *Alt* with the error message returned in case of authentication failure. A similar assumption has been made for the municipality answer to the ticket emission request, it has been avoided to report the negative answer, for which the agent would be required to retry the request.

2.5 Component interfaces

2.6 Selected architectural styles and patterns

2.7 Other design decisions

2.7.1 Consistency and update strategies among replicas

In this application, we need a client-centric consistency among replicas because end users (and therefore application servers that act as clients towards the DataServers) don't always connect to the same DataServer, due to the presence of the load balancer. Every DataServer can respond to a request to read or write data so we use an active replication protocol. In particular we exploit leaderless replication in which the decision on the value to read and the write to perform is decided by all the replicas or at least a quorum of them. The type of chosen consistency model is the "read your writes": the effect of a write operation by a process on a data item x will always be seen by a successive read operation on x by the same process. Concerning the update propagation we opted for propagating a notification of the operation, assuming that there will be more writes than reads. The propagation strategy chosen is the Gossiping strategy: when a replica is updated then it just propagates that update to all the nodes that it knows; if a replica receives an update that it has already received then the probability of propagating that information is decreased on that replica.

3 User interface design

4 Requirements Traceability

In this section is shown how the requirements are actually ensured and what components actually ensure them. It's worth to notice that for the sake of simplicity the Router has been ignored but it's always involved when dealing with a request coming from one of the clients.

[R1] The system must allow people to register to it providing personal data (name, surname, birthdate, identity card number, fiscal code) and selecting a username and a password: This requirement is provided by the **AccessManager** and the **User Mobile App** components. The **User Mobile App** allows the user to fill blank spaces with username and password and the **AccessManager** handles registration requests from the client.

[R2] The system must verify the correctness of the provided personal data of a registering user checking them from the identity card number, blocking the registration if they are not correct: the **AccessManager** checks the received data from the Client, calling the **munipality API services** that actually possesses sensible data of the users.

[R3] The system must allow registered users to login through their username and password: this requirement is provided by the **AccessManager** (**AuthoritiesLogin** and **CitizenLogin** components) that handles the requests of login from the user.

[R4] The system must allow logged user to fill a report violation form: this requirement is fulfilled by the **User Mobile App** comonent that allows user to choose the "Report a violation" feature.

[R5] The system must let the user select the type of violation detected: the requirement is fulfilled by the **User Mobile App** component, showing an empty space to fill with the type of violation detected in the report a violation process.

[R6] The system must allow the user to insert the license plate in a violation report: the requirement is fulfilled by the **User Mobile App** component, showing an empty space to fill with the license plate of the car that committed the violation in the report a violation process.

[R7] While reporting the violation, the system must allow users to take one or more pictures of the potential violation: the requirement is fulfilled by the **User Mobile App** component, allowing user to click on the "Take a picture" button.

[R8] The system must not allow users to choose pictures not taken in the moment of the report: this requirement is guaranteed by the **User Mobile App** by not allowing client to just pick some random picture from his local storage.

[R9] The system must collect the current position of the user, using GPS: the **User Mobile App** component ensures this requirement by taking the current position of the User using GPS location.

[R10] The system must allow user to confirm or delete the current report: the **User Mobile App** component ensures this requirement by either sending the report for a check through the "confirm" button or deleting the current report clicking on the back button.

[R11] After confirmation, the system must add the current date and time to the report: the **ReportManager** actually takes the requests from the client and attaches to them the current time and date taken from the application server internal clock.

[R12] The system must store confirmed report: the **ReportManager** is responsible for this requirement, calling the DataServer to actually store the report.

[R13] The system must check reports to try to find if the pictures of the violations have been modified: the **ReportManager** manages to check if the pictures are fake or not or have been modified.

[R14] The system must try to find, according to the GPS position of the user and the pictures sent, if the position is fake or not: the **ReportManager** ensures this requirement by running an algorithm to define if the position is correct or not.

[R15] The system must discard the report if it has been recognized as fake according to the previous requirements (R13-R14): the **ReportManager** is in charge of this, deleting the report if recognized as fake.

[R16] The system must try to automatically recognize the license plate in the photo, possibly with the help of the value inserted by the user: the **ReportManager** is in charge of this , running an algorithm for text recognition in the picture(s).

[R17] ????? The system must store into stable memory the reported violation if correct (i.e. not recognized as fake). ??????

[R18] The involved municipality must be calculated considering in which city the reported violation has been found, based on the GPS position of the user that has sent the report: the **ReportManager** provides this requirement by asking the Maps Service the municipality associated to the position of the user.

[R19] The system must send the reported violations to the involved municipality: the **ReportManager** and the the NotificationManager are responsible for this. After having saved it, the **ReportManager** sends the report to the NotificationManager that forwards it to the involved municipality.

[R20] The system must allow an agent to see the reports for its municipality, checking them in order of arrival: The Municipality Web app component is responsible for this, showing on screen the reports in order of arrival.

[R21] The system must allow an agent to issue a traffic ticket to a certain person (i.e. license plate) through the correspondent municipality service:

[R22] The system must allow an agent to put on hold a violation report if it needs to be checked in person: The MunicipalityAgent Web App ensures this by shwoing the possibility of putting a report on hold while checking it.

[R23] The system must allow an agent to discard a violation if it has been verified as fake or it cannot be verified (the vehicle is not there anymore) or it is a duplicated report.??

[R24] The system must allow an agent to retrieve the data of the author of a violation report: the MunicipalityAgent Web App asks to the Municipality API Data about the author of the report.

[R25] The system must allow an agent to create an account, asking the municipality services to verify its identity: the **AccessManager** is responsible for this, asking the Municipality API to check the identity of the agent.

[R26] The system must allow an agent to login, inserting its username and password: Municipality Web App and **AccessManager** are responsible for this. The Municipality Web App allows user to insert data, while **AccessManager** checks if they are correct.

[R27] The system must mine this information from the reported violations: the DataWareHouse is respnsible for this, performing datamining on reports stored in the DataBase of the application.

[R28] The system must allow the users (even if not authenticated) to select the see information for a city. The user can choose either the city where he is, using the GPS position, or an arbitrary selected location. The Mobile App component ensures this requirement by allowing user to choose the city he wants.

[R29] The system must allow the user to select information about streets or areas in the city selected and to specify if he wants information for a specific street or area or a classification of streets or areas. The Mobile Web App is responsible for this requirement by allowing user to choose the area or street he wants.

[R30] The system must show the the data corresponding to the selection of [R29]: the mobile Web App and Statisticsmanager components are responsible for this requirement; the StatisticsManager contacts the MapAPI to provide an image for the city/area/street selected. This image is then shown by the Mobile App component to the user.

[R31] The system must take information about accidents and tickets from the municipality: this requirement is ensured by the DataCollectionManager that periodically queries the Municipality API to retrieve new informations and saves them on the DataWareHouse.

[R32] The system must use this information to build statistics, crossing them with reported violations: the StatisticsManager and the DataWareHouse component ensure this requirement; the DataWareHouse components build statistics exploiting dataMining techniques

[R33] The system must not allow common users to see confidential data about other people: The Mobile App component is in charge for this requirement by showing only data that does not violate privacy.This requirement is also guaranteed by a subcomponent in the Router (i.e. AuthChecker) to check if a request made by a specific client is valid or not.

[R34] The system must allow the user to choose a topic: areas or streets with most accidents, areas or streets with the highest number of traffic tickets issued, areas or streets where there have been the best improvements, information for a specific area or street. The Mobile App iprovides this requirement by allowing user to choose the topic is interested into.

[R35] The system must show to the user the information about the topic selected according to [R34]: The Mobile App component ensures this requirement by showing as an image the topic requested to the user.

[R36] The system must allow authenticated supervisors to retrieve information about the vehicles with the highest number of violations in a selected area or street: the MunicipalitySupervisor Web App allows supervisors to select the statistics they want; the Statistics;anager asks the the DataWareHouse the type of Data requested by the User.

[R37] The system must allow supervisors to access only information about their own municipality: the MunicipalitySupervisor Web App is responsible for this, allowing supervisor to only check Statistics and suggestions about his city. This requirement is also guaranteed by a subcomponent in the Router (i.e. AuthChecker) to check if a request made by a specific client is valid or not.

[R38] The system must allow a supervisor to create an account, asking the municipality services to verify its identity: the **AccessManager** and the MunicipalitySupervisor Web App are responsible for this requirement. the **AccessManager** receives the request and ask the MunicipalityAPI to check if the inserted identity is correct.

[R39] The system must allow a supervisor to login, inserting its username and password: The MunicipalitySupervisor Web App and the **AccessManager** are responsible for the requirement. The MunicipalitySupervisor WebApp allows supervisor to enter his username and password and forwards the request to the **AccessManager** that handles it asking the DataServer if the data inserted is correct.

[R40] The system must take information about accidents, tickets and street networks (bike lanes, sidewalks, parking areas,...) from the municipality, exploiting the municipality services: the DataWareHouse component ??'

[R41] The system must elaborate this information, combined with reports information, and try to find possible solutions for problems: the DataWareHouse component is responsible for this requirement trying to aggregate data coming from the Databases concerning reports, issued traffic tickets and accidents.

[R42] The system must notify the municipality about new possible interventions: the NotificationManager the DataWareHouse and SuggestionManager components are responsible for this requirement: the DW component forwards the new intervention to the SuggestionManager that forwards it to NotificationManager that sends it to the involved municipality.

5 Implementation, integration and test plan

6 Effort spent

Nicola Rosetti

Date	*Hour*	*Section*
17-10-2019	1.5 h*	Component and high-level architecture analysis

Table 6.1

Simone Sartoni

Date	*Hour*	*Section*
17-10-2019	1.5 h*	Component and high-level architecture analysis

Table 6.2

Vittorio Torri

Date	*Hour*	*Section*
18-11-2019	1 h*	Componenst and high-level architecture analysis
21-11-2019	1 h	Components diagram
22-11-2019	0.5 h	Components diagrams
23-11-2019	1.5 h	Components diagrams and components view
24-11-2019	2 h	Sequence diagrams and components view

Table 6.3

** Group work*

7 References