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 2.1 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.

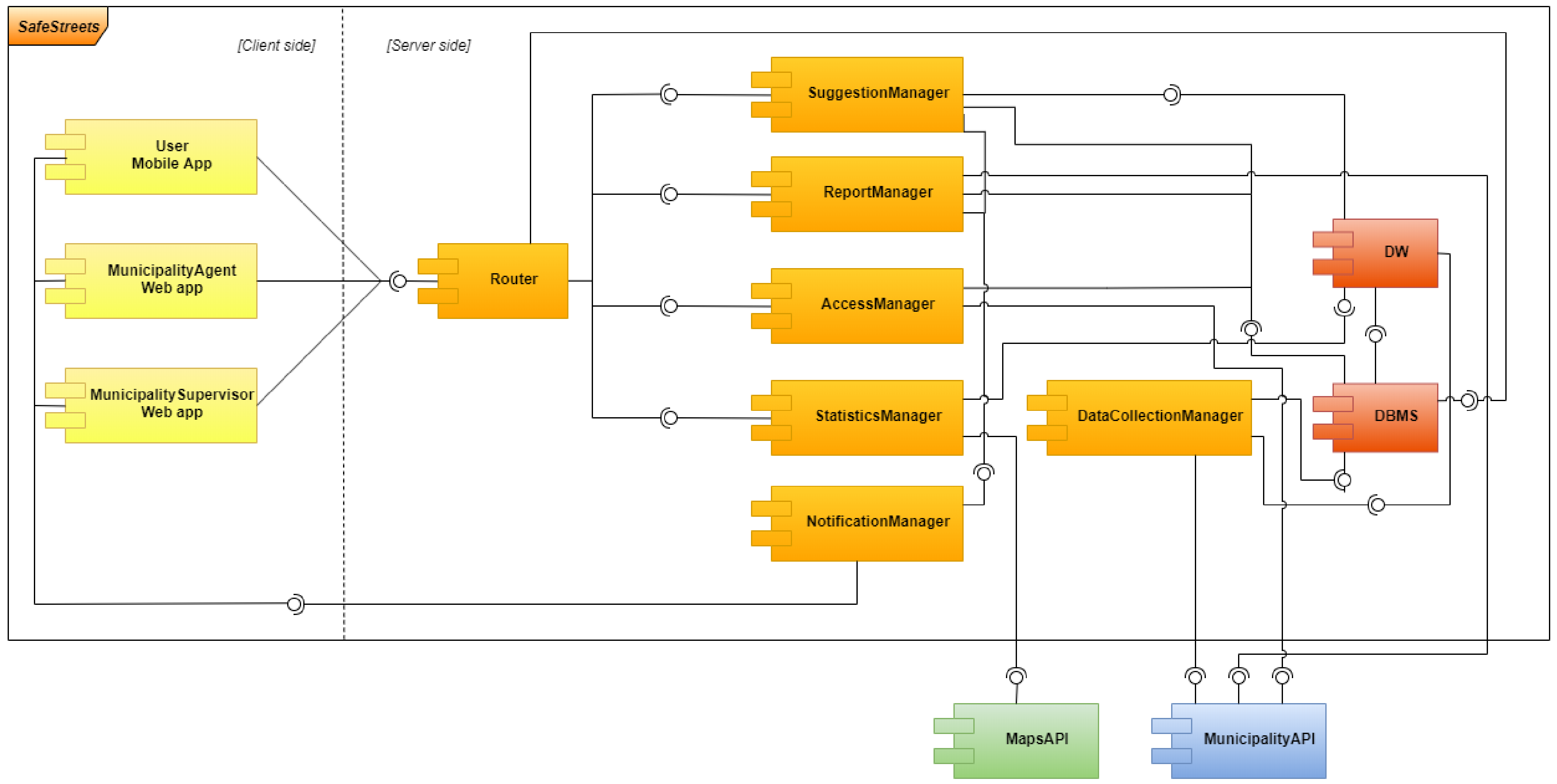

Figure 2.1: UML Component Diagram

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 2.2 a more detailed view is provided, putting in evidence the various interfaces for the mobile and for the web app.

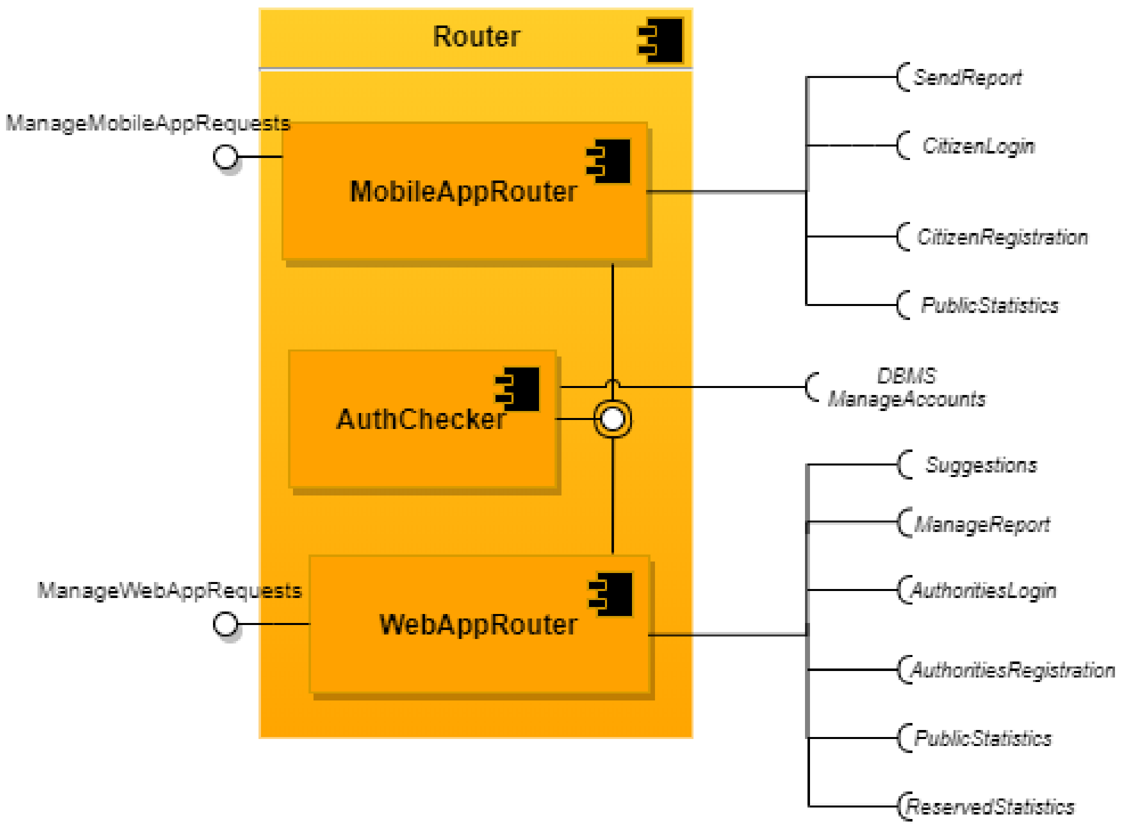

Figure 2.2: UML Component Diagram for *Router* component

- **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 apps* through the *NotificationManager* 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 2.3.

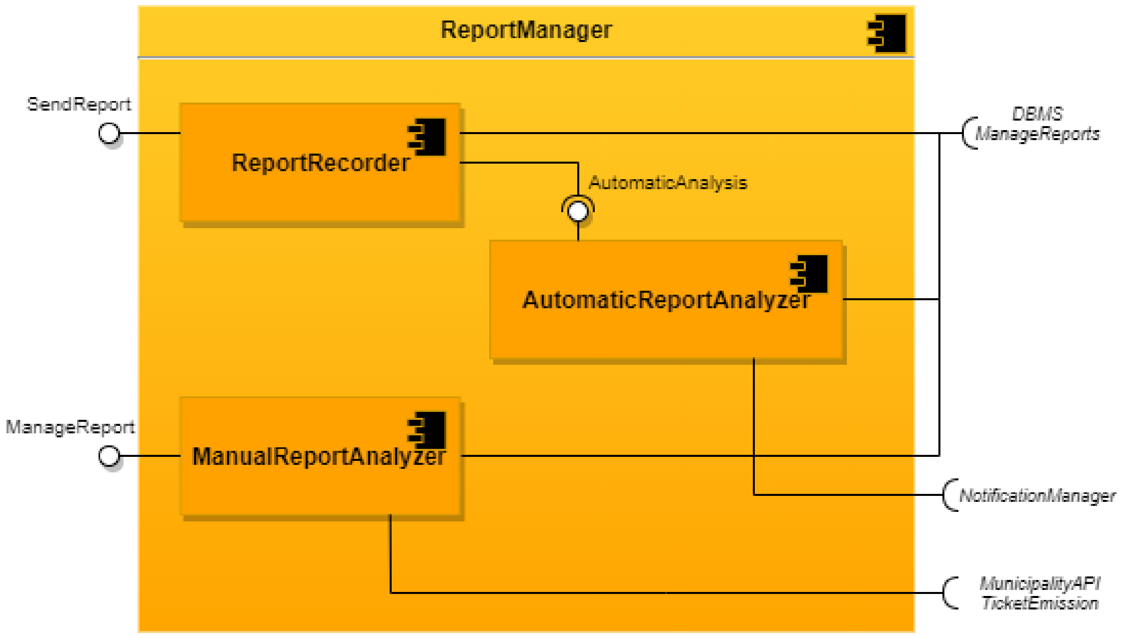

Figure 2.3: UML Component Diagram for *ReportManager* component

- **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 2.4.
- **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 2.6.
- **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.

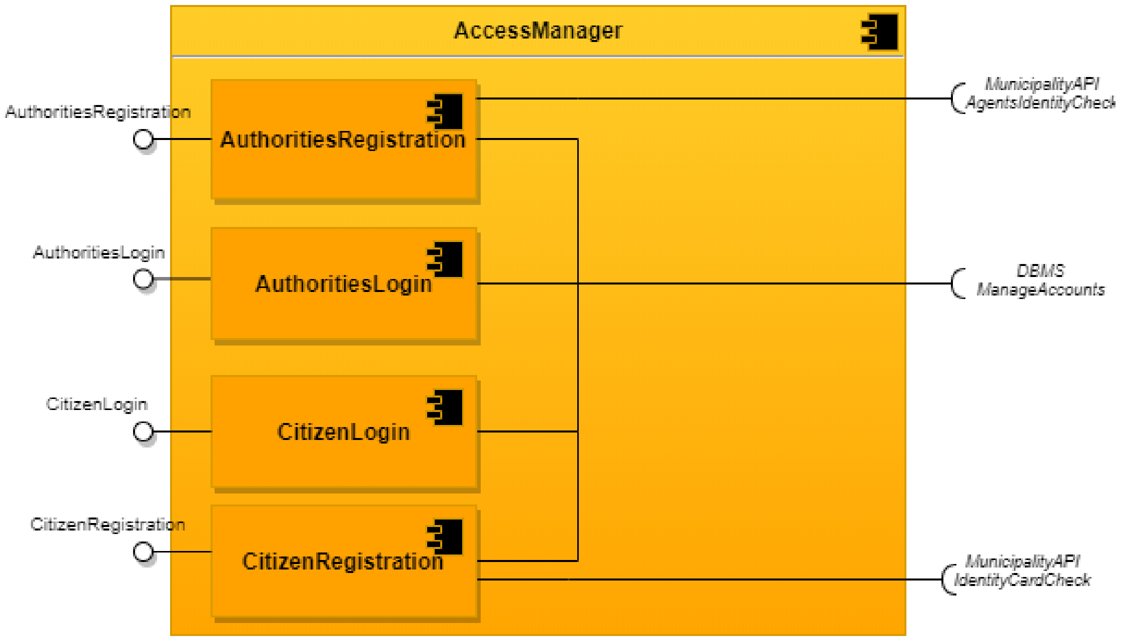

Figure 2.4: UML Component Diagram for *AccessManager* component

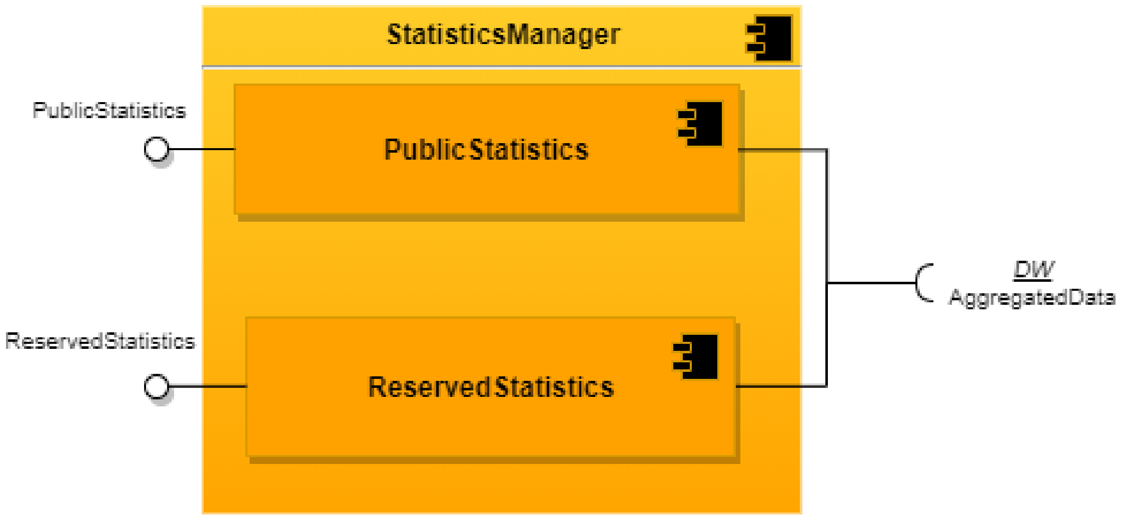

Figure 2.5: UML Component Diagram for *StatisticsManager* component

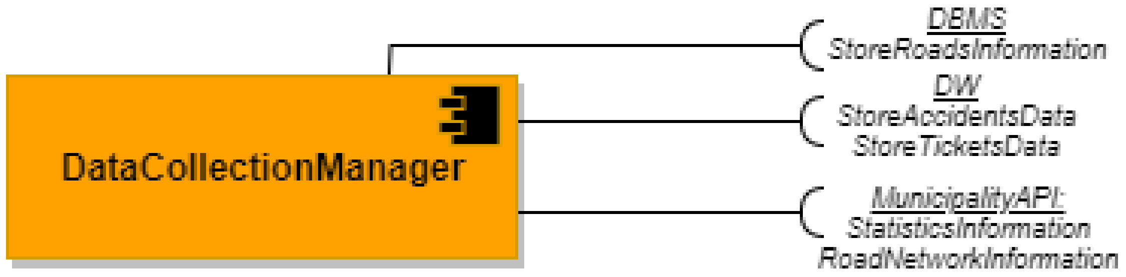

Figure 2.6: UML Component Diagram for *DataCollectionManager* component

- **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 2.7, 2.8, 2.9, 2.10 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.

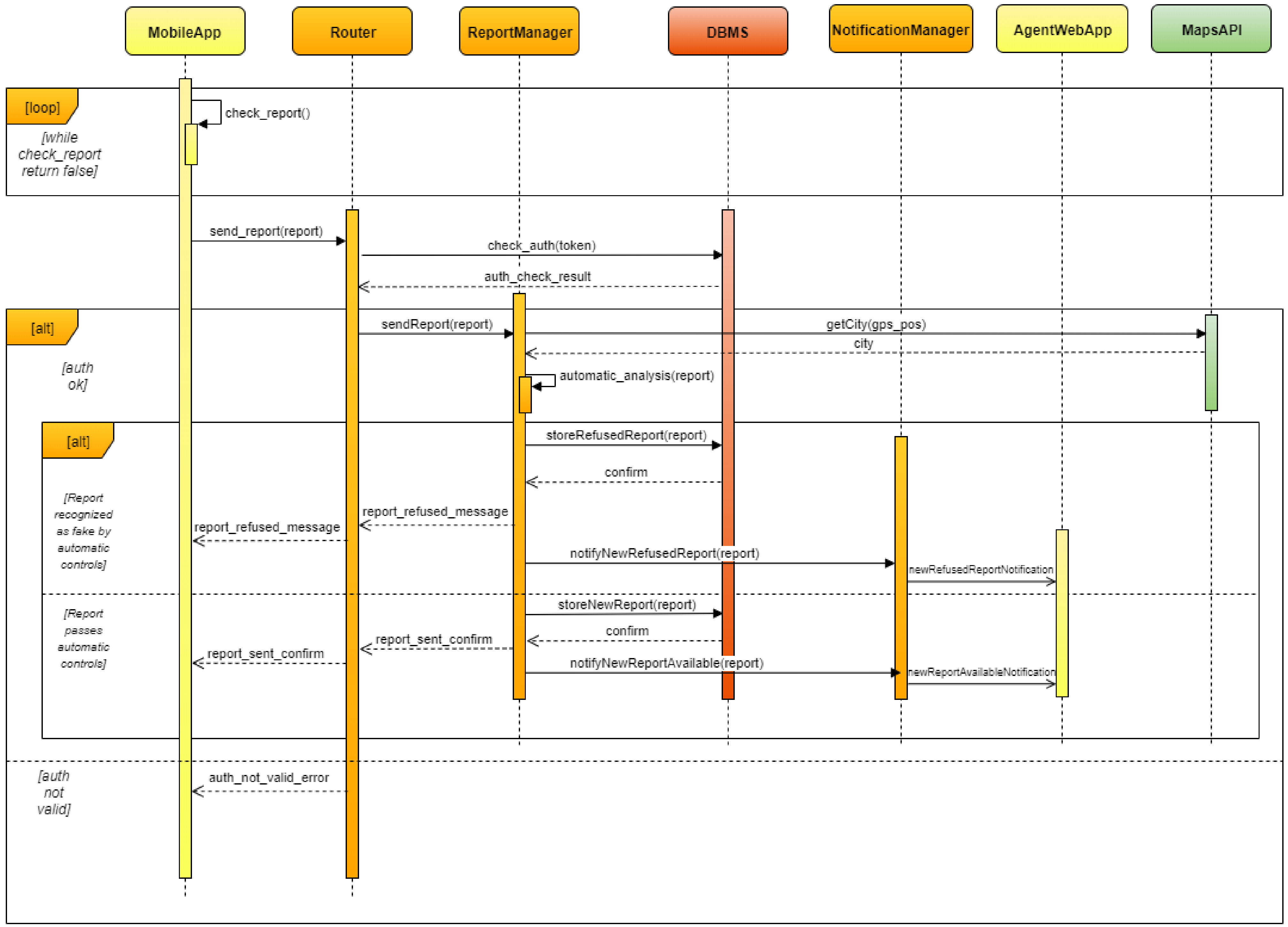

Figure 2.7: UML Architectural Sequence Diagram for *SendReport* use case

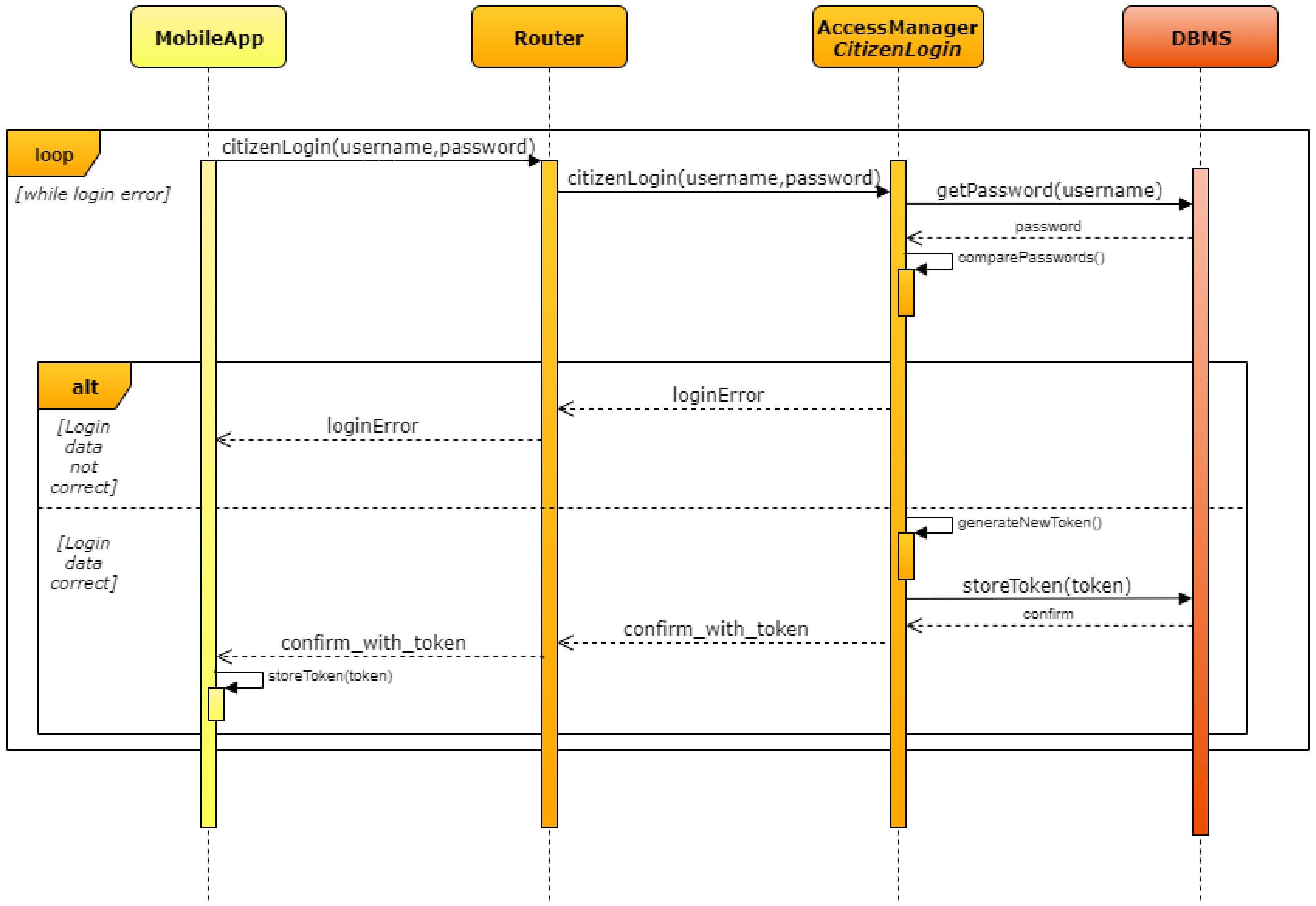

Figure 2.8: UML Architectural Sequence Diagram for *UserLogin* use case

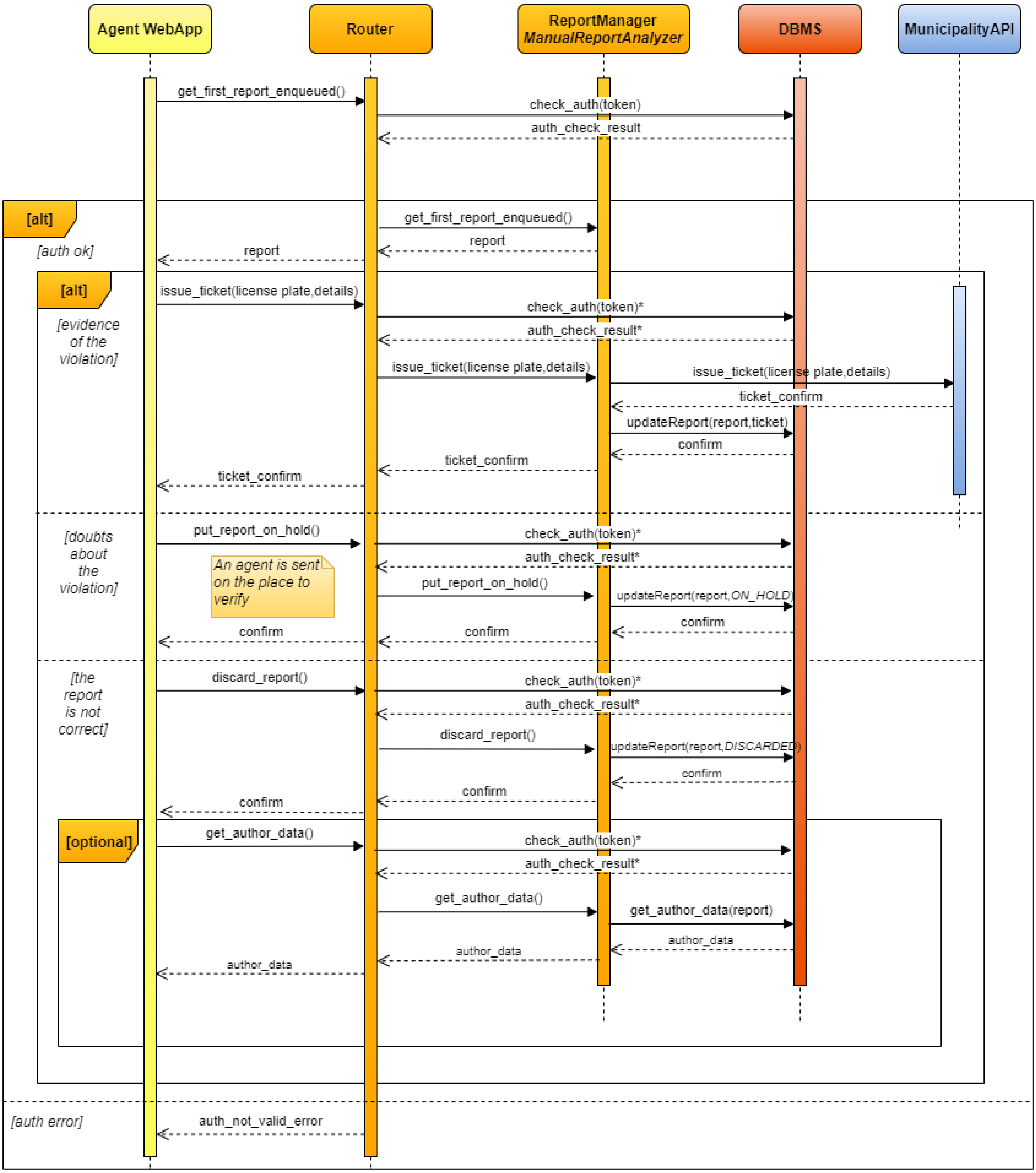

Figure 2.9: UML Architectural Sequence Diagram for *AgentCheckReport* use case

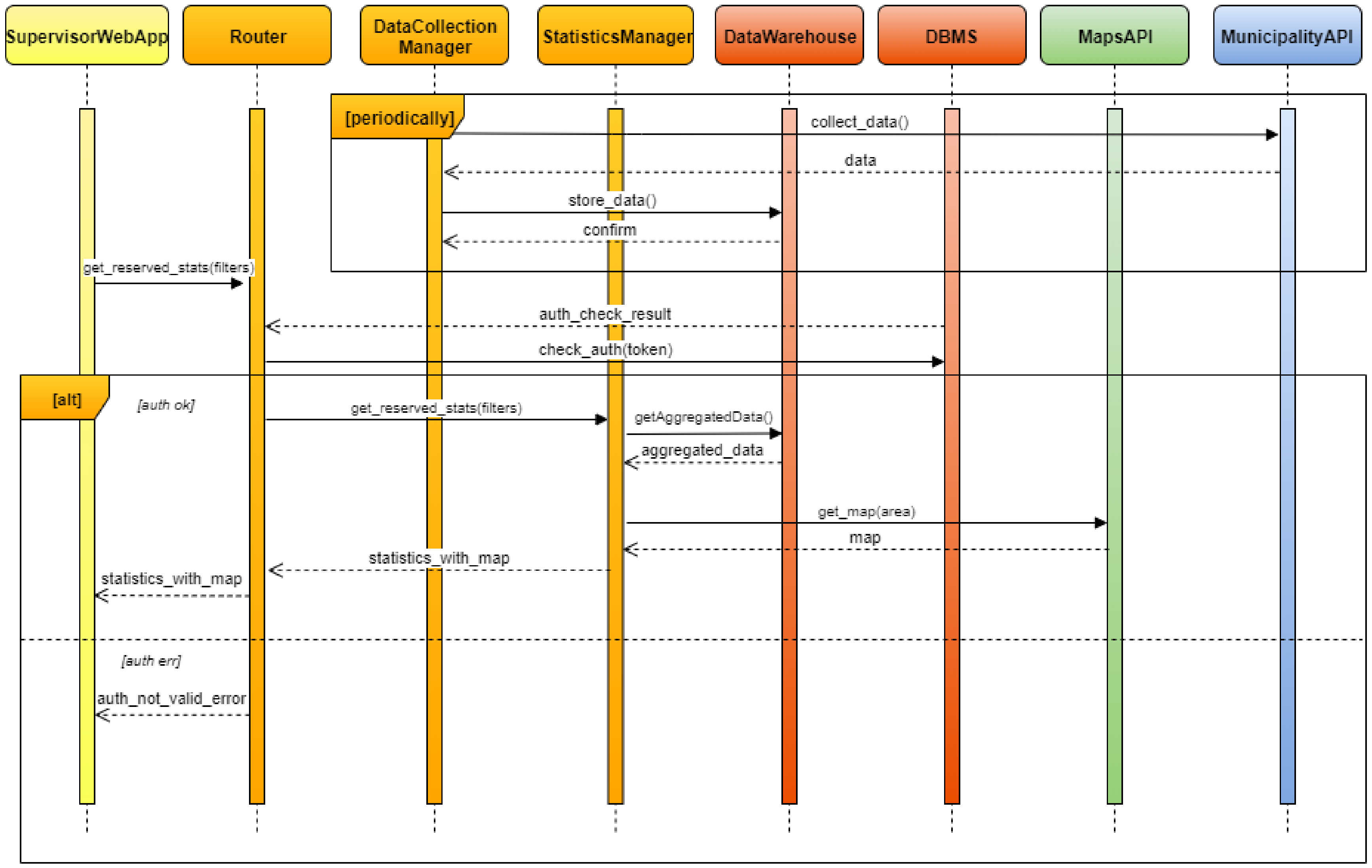

Figure 2.10: UML Architectural Sequence Diagram for the *CheckStatistics* use case

In the sequence diagram of figure 2.7 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 2.9 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 *, is 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

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