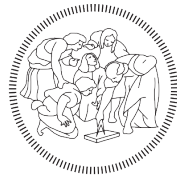


Politecnico di Milano

SAFESTREETS - DD



POLITECNICO
MILANO 1863

Samuele Meta - Stiven Metaj

Supervisor: Matteo Rossi

Department of Computer Science and
Engineering

December 9, 2019

Contents

1	Introduction	3
1.1	Purpose	3
1.2	Scope	3
1.3	Definitions, Acronyms, Abbreviations	4
1.3.1	Definitions	4
1.3.2	Acronyms	4
1.3.3	Abbreviations	5
1.4	Revision History	5
1.5	Reference Documents	5
1.6	Document Structure	5
2	Architectural Design	7
2.1	Overview	7
2.2	Component View	8
2.3	Deployment View	11
2.4	Runtime View	13
2.4.1	Registration Runtime View	13
2.4.2	Login Runtime View	15
2.4.3	Create Violation Report Runtime View	17
2.4.4	Accept Violation Report Runtime View	19
2.4.5	Statistics Runtime View	21
2.4.6	Notification Runtime View	23
2.5	Component Interfaces	24
2.5.1	REST API	24
2.6	Selected Architectural Styles and Patterns	43
2.6.1	Multi-tier Architecture	43
2.6.2	Use of RESTful guidelines	44
2.6.3	DBMS	45
2.7	Other Design Decisions	47

2.7.1	Thin Client	47
2.7.2	MVC	47
2.7.3	Firewalls	47
3	User Interface Design	48
3.1	User eXperience Diagrams	48
3.2	Web Interface	50
4	Requirements Traceability	52
5	Implementation, Integration and Test Plan	55
5.1	Overview	55
5.2	Unit Testing	56
5.3	Integration Testing	56
5.4	Component Integration	56
6	Effort Spent	61

1 Introduction

1.1 Purpose

The following Design Document (DD) is aimed to provide an overview of the *SafeStreets* application, explaining how to satisfy the project requirements declared in the RASD and stating the successive refinements made together with the Stakeholders according to their needs. The document is mainly intended to be used by developers teams as a guidance in the development process, by testing teams to write automated testing and to avoid structural degradation of the system in case of maintenance or future extension. Indeed, its purpose is to provide a functional description of the main architectural components, their interfaces and their interactions, along with the design patterns.

1.2 Scope

As explained in the RASD document, *SafeStreets* is a crowd-sourced mobile application that allows Citizens to notify Authorities about traffic violations, with particular emphasis on parking contraventions. Citizens are able to manage their reports, visualizing the whole history and removing them. On the other hand, Authorities can access a complete overview of the incoming reports and choose if to accept or reject them. The System can also provide useful statistics to all kind of Users, even if only Guests, according to their role. *SafeStreets* is structured in a multitier architecture. More specifically, the Business Logic layer has the task of computing the previously described actions and interacting with external third-party services, through the use of interfaces. This layer is connected with the Data layer, in which are stored all the Users' data. Finally, the Presentation layer is built through the *thin Client* paradigm, in which the Client needs to perform almost no computation, allowing a more portable system.

1.3 Definitions, Acronyms, Abbreviations

In this section follow definitions, acronyms (including their meaning) and abbreviations used in this document.

1.3.1 Definitions

- *Client*: a desktop computer or mobile device that is capable of obtaining information and applications from a Server.
- *Server*: a computer or computer program which manages access to a centralized resource or service in a network.
- *Firewall*: a part of a computer system or network which is designed to block unauthorized access while permitting outward communication.
- *Port*: an endpoint of communication in an operating system.
- *Design Pattern*: reusable software solution to a commonly occurring problem within a given context of software design.

1.3.2 Acronyms

DBMS	DataBase Management System
HTTPS	Hyper Text Transfer Protocol over SSL
API	Application Programming Interface
REST	REpresentational State Transfer
MVC	Model View Controller
OS	Operative System
UI	User Interface
UX	User Experience
URL	Uniform Resource Locator
RASD	Requirements Analysis and Specification Document
SPA	Single Page Application
ERD	Entity-Relationship Diagram
SSL	Secure Sockets Layer

Table 1.1: *Acronyms*

1.3.3 Abbreviations

- [R.n]: n-th Requirement in the RASD document

1.4 Revision History

[TBD]

1.5 Reference Documents

- Specifications document “SafeStreets. Mandatory project assignment”
- SafeStreets RASD Document
- IEEE Standard 1016-2009: IEEE Standard on Software Design Descriptions
- UML documentation: <https://www.uml-diagrams.org>

1.6 Document Structure

The rest of the document is organized as follows:

- **Architectural Design:** details the System’s architecture by defining the main components and the relationships between them, as well as specifying the hardware needed for the System deployment. It will also be focused on design choices and architectural styles, patterns and paradigms.
- **User Interface Design:** provides further details on the UI defined in the RASD document through the use of UX modeling.
- **Requirements Traceability:** shows the relations between the requirements from the RASD and the design choices of the DD and how they are satisfied by the latter.
- **Implementation, Integration and Test Plan:** provides a roadmapping of the implementation and integration process of all components and explains how the integration will be tested.

- **Effort Spent:** describes how the work has been split between the members of the team and how long did the DD take to be completed.

2 Architectural Design

2.1 Overview

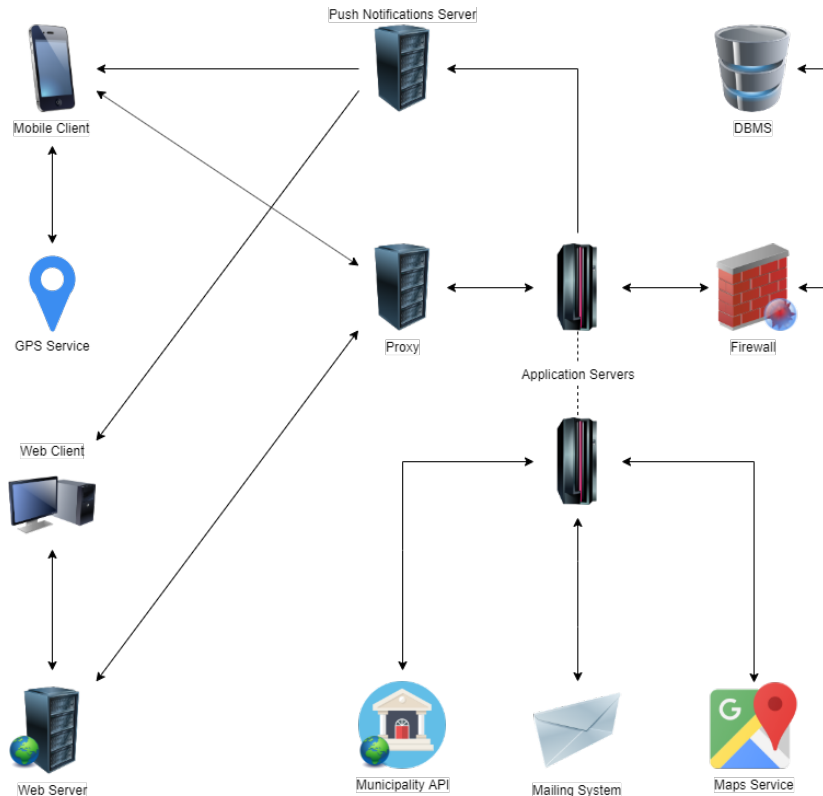


Figure 2.1: *Overview of the System.*

The above image provides a general overview of the architecture of the *System*. We can recognize 2 distinct main parts of it: the first is about granting access to the application through a mobile or a web client; the second is about services offered or used by the *System* (i.e. the Mailing System or the Maps Service).

2.2 Component View

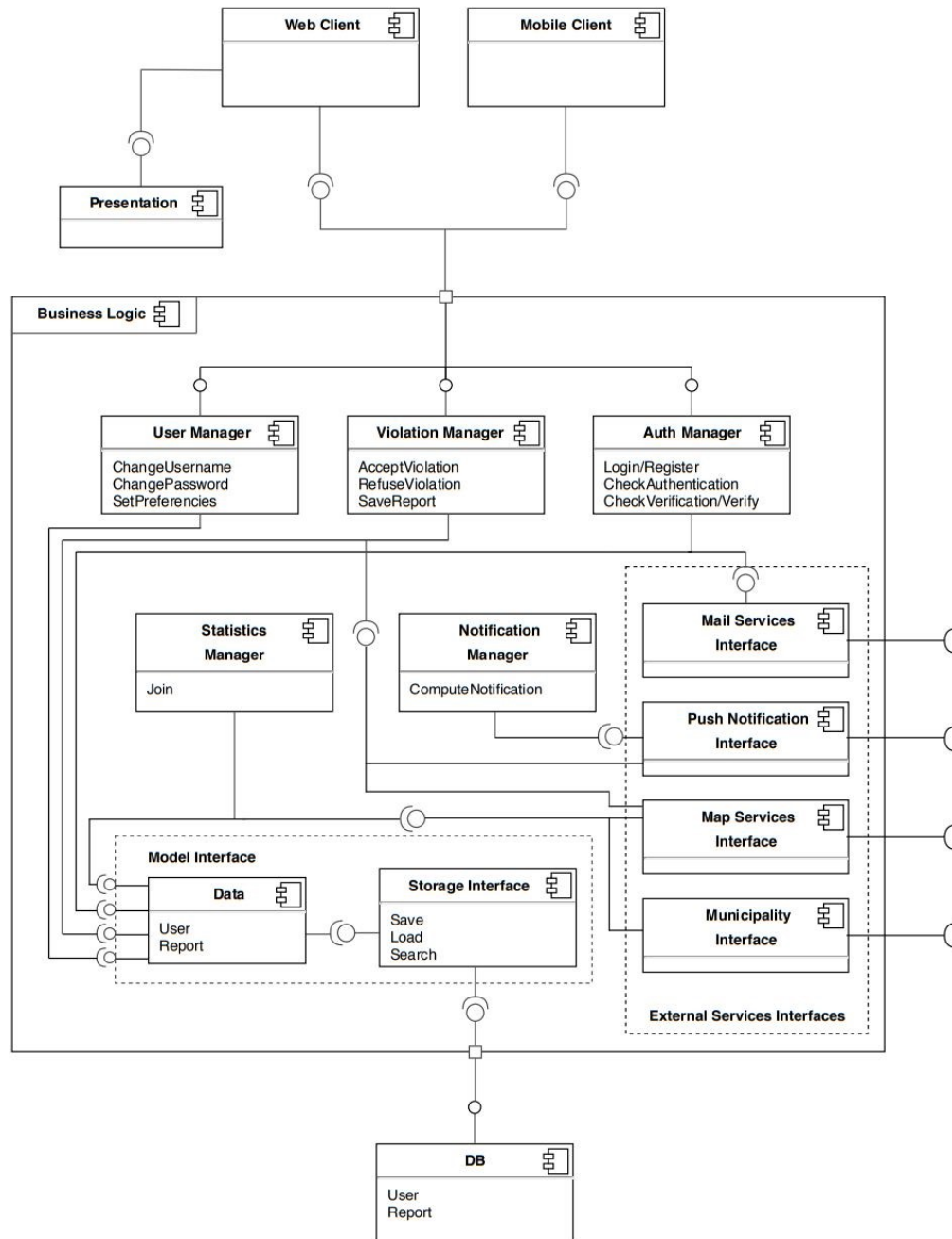


Figure 2.2: Component Diagram

The UML component diagram aims at capturing the internal modular structure of components, showing how they are connected together in order to form larger components. Components are wired together by using an assembly connector to connect the required interface of one component with the provided interface of another component. Let's have a closer look at each component:

Web Client, Mobile Client

This component represents the Client machines that access to the API of the Business Logic of the System. They do not have any notable functionality to be outlined, due to the fact that they are implemented as thin clients as explained below. The Web Clients, accessing through a browser, need the Presentation component in order to display the web pages of the application. This layer provides to it the structure of the User Interface without accessing data and application logic. On the other hand, the Mobile Client embeds the Network Manager, the Presentation Component and the Data Manager.

User Manager

This Manager includes all the operations that affect the user-related data. It exposes methods to change account credentials and preferences. Furthermore, it manages the data stored in the DB through the interaction with the interface of the Model Interface.

Authentication Manager

This Manager handles all the methods related to the authentication task and access to the System. Specifically, it deals with both the User registration and login, making use of the Model Interface to interact with the DB. It handles credentials verification and constraints, guaranteeing the creation of consistent accounts. Moreover, it also uses the Mail Service Interface to communicate with the Mailing Service and send registration emails.

Report Manager

This Manager allows the Citizen to create reports and send them to the platform. It interacts with the Model Interface in order to store them permanently and to retrieve the list of the previous reports. On the other hand, it allows to the Authorities to have a complete overview of the incoming reports and gives the possibility to accept or refuse them.

Statistics Manager

This Manager handles the requests of Guests, Citizens and Authorities to obtain insights about data through statistics. It is responsible to aggregate the

information, eventually interacting with the Municipality Interface, and return it to the User. It also invokes the Authentication Manager to check the role of the User, in order to return only the allowed statistics.

External Services Interfaces

Some of the components in the System are also dedicated to communicating with external services through specific interfaces. These interactions are bilateral and essential to guarantee the application's functionalities. These components are both on the Client side and on the Server side. In particular the interfaces needed by the System are:

- *Mail Service Interface*: is responsible of the interaction with the mail service. It is used to send an email confirmation to the User, at the request of the Authentication Manager, during the registration phase.
- *Map Service Interface*: is responsible of providing, at the request of the Statistics Manager, a visual representation of raw data whenever these have a spatial dimension. It can also be invoked by the Report Manager in the eventuality that the Citizen is not able to remember the name of the street and needs a map as an helper.
- *Push Notification Interface*: interacts with the Push Notification Service and is responsible to notify to the Citizen messages of interest, such as the outcome of the profile verification or the acceptance of a report.
- *Municipality Interface*: is responsible of providing, at the request of the Statistics Manager, additional data in possession of the Municipality, in order to join them with local statistics and provide to the User a complete overview.

Model Interface

The Model Interface includes two sub-components. The Data component provides the set of classes corresponding to the tables contained in the Database. The Storage Interface provides the methods for querying the Database.

Data Base

This component represents the DBMS, which provides the interfaces to retrieve and store data. In the data base, for each user, credentials and application data are safely and securely stored.

2.3 Deployment View

Here the deployment diagram of the whole system is shown. Its main aim is to specify the distribution of components capturing the topology of the system's hardware.

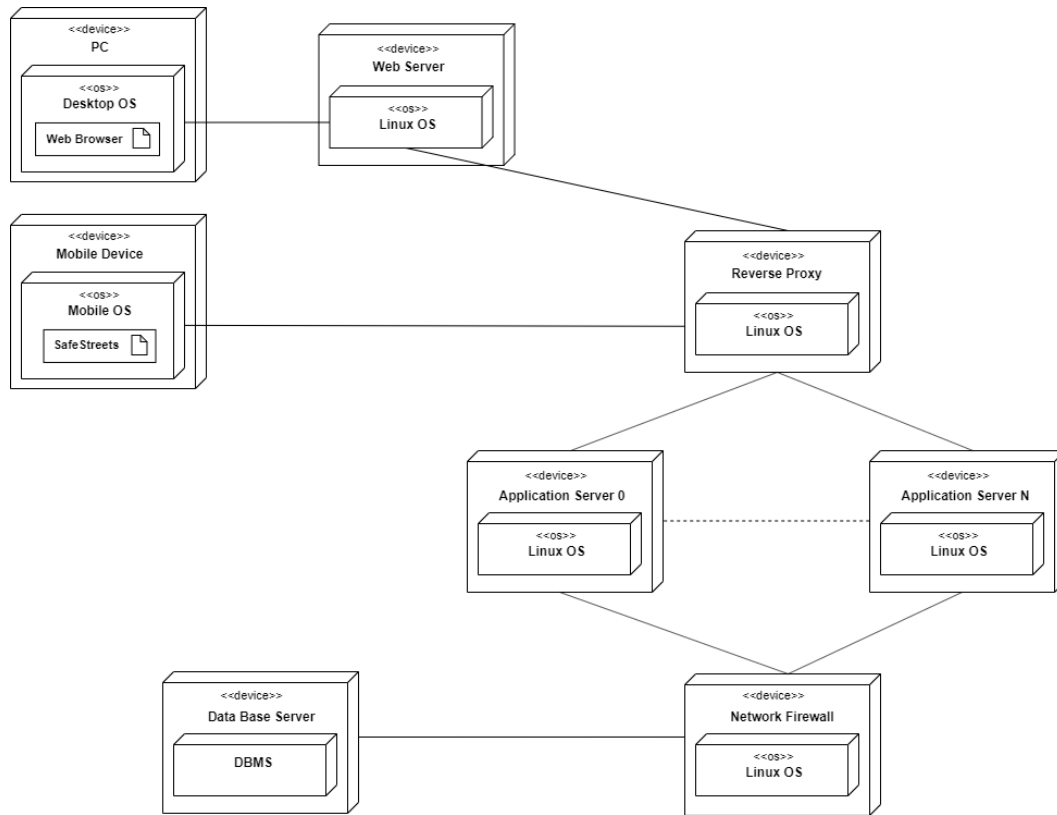


Figure 2.3: *Deployment Diagram*

The System presents a multitier architecture in which the role of each node will be specified in the next paragraphs.

Clients

The first tier is composed by clients machines, which can be either mobile or desktop. In the first case, the client will be able to access *SafeStreets*' functionalities through the dedicated native application, in the second case, by means of any web browser.

Web Server

A web server is used to store, process and deliver web pages to desktop clients. The communication between client and server takes place using the HTTPS protocol.

Reverse Proxy

This node helps to achieve increased parallelism and scalability, avoiding to lock the CPU and compressing the content. It is responsible for the load balancing as it distributes requests between all Application Servers. The Reverse Proxy also increases security and anonymity by protecting the identity of the backend servers and acting as an additional shield against security attacks.

Application Servers

This level of the architecture encloses all the Business Logic of the System. *SafeStreets*' Application Servers are fully replicated to balance the workload and provide an higher degree of fault tolerance.

Firewall

The access to the Database is mediated by a network firewall in order to avoid unauthorized access to the data and the credentials of the User or malicious attacks like DDoS.

Data Base Server

This is the last layer of the architecture: all the data are stored in a Data Base Server equipped with a relational DBMS. Furthermore, credentials are not stored in plain text, but hashed and salted: this method adds a new layer of security to safeguard Users' sensitive data.

2.4 Runtime View

In the following sections the most characterising runtime views will be detailed by making use of sequence diagrams. As stated before in the document, the Presentation, the Network Manager and the Data Manager will be omitted in order to provide a cleaner overview, focusing the attention on the most important high level interactions. Moreover, whenever a Client forwards a request, the Reverse Proxy receives it on behalf of the Server, checks whether such request conforms to its defined schema and, in case the User has to be logged in, checks if the provided authToken is associated to an actual logged in User. Also in this case, the Proxy's functionalities have been collapsed into the Authentication Manager in order to avoid an excessive complexity of the diagrams.

2.4.1 Registration Runtime View

The User has to register to the service before accessing the functionalities of the application, except for the publicly available statistics. The User fills a form containing the necessary information, which is sent to the Application Server through an HTTP POST request. The Authentication Controller handles the request, verifies if the information is correct and complete, then, through the Model Interface, creates a new entry in the User table on the Database. At this point the User Account has been created, but it has to be confirmed. An email with a confirmation URL is sent by the external mailing service. Once the User clicks on the provided link, the User entry on the Database is updated and the Account is set as active.

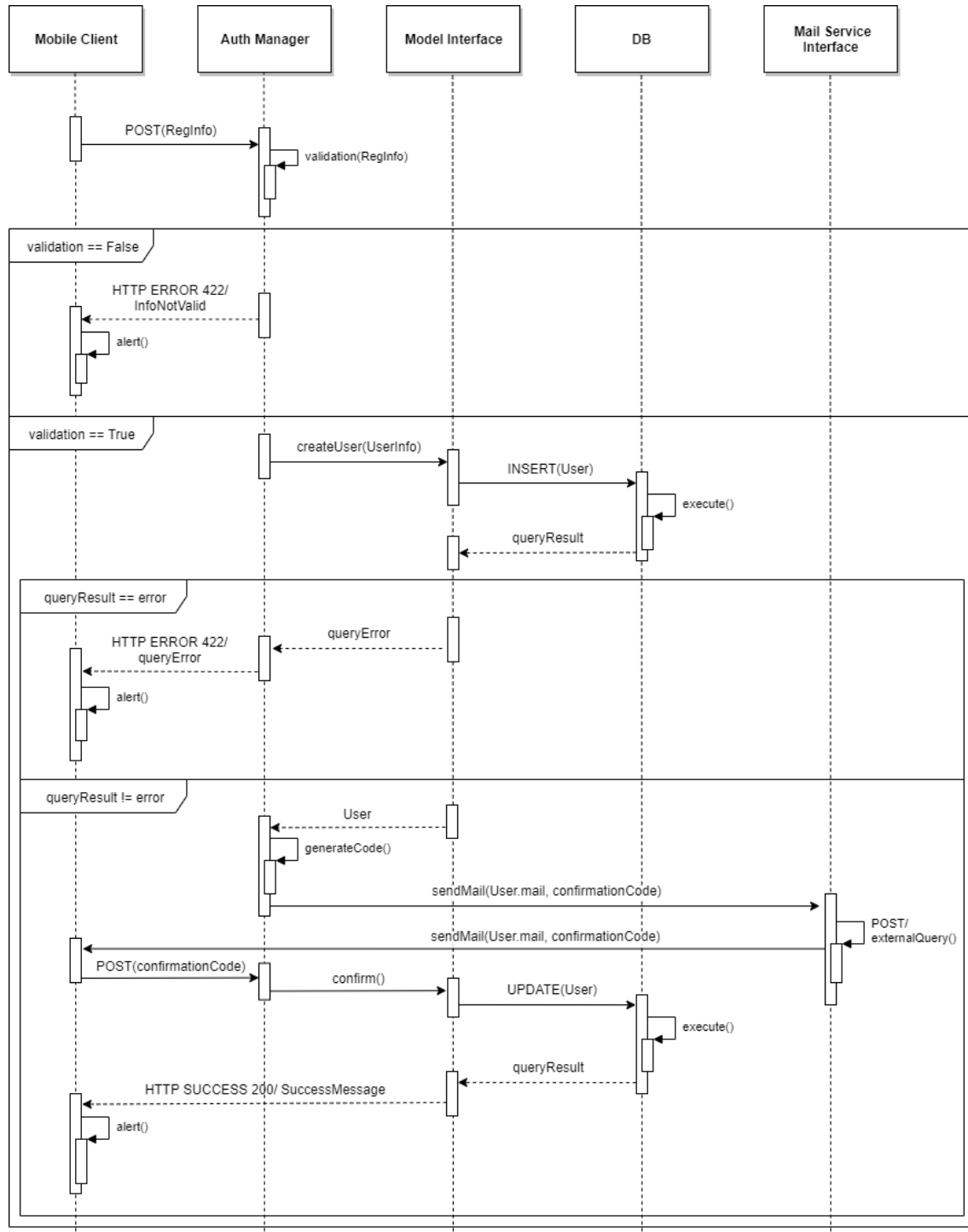


Figure 2.4: *Registration Runtime View*

2.4.2 Login Runtime View

The User has to login before accessing the functionalities of the application. The User submits the login information through an HTTP POST request. The request is handled by the Authentication Controller that validates the request and checks if the Database has an entry for the requested account. If the Account is present and the credentials are correct the Authentication Controller generates an Access Token, sets it as the current access Token for the specified Account and returns it to the Client.

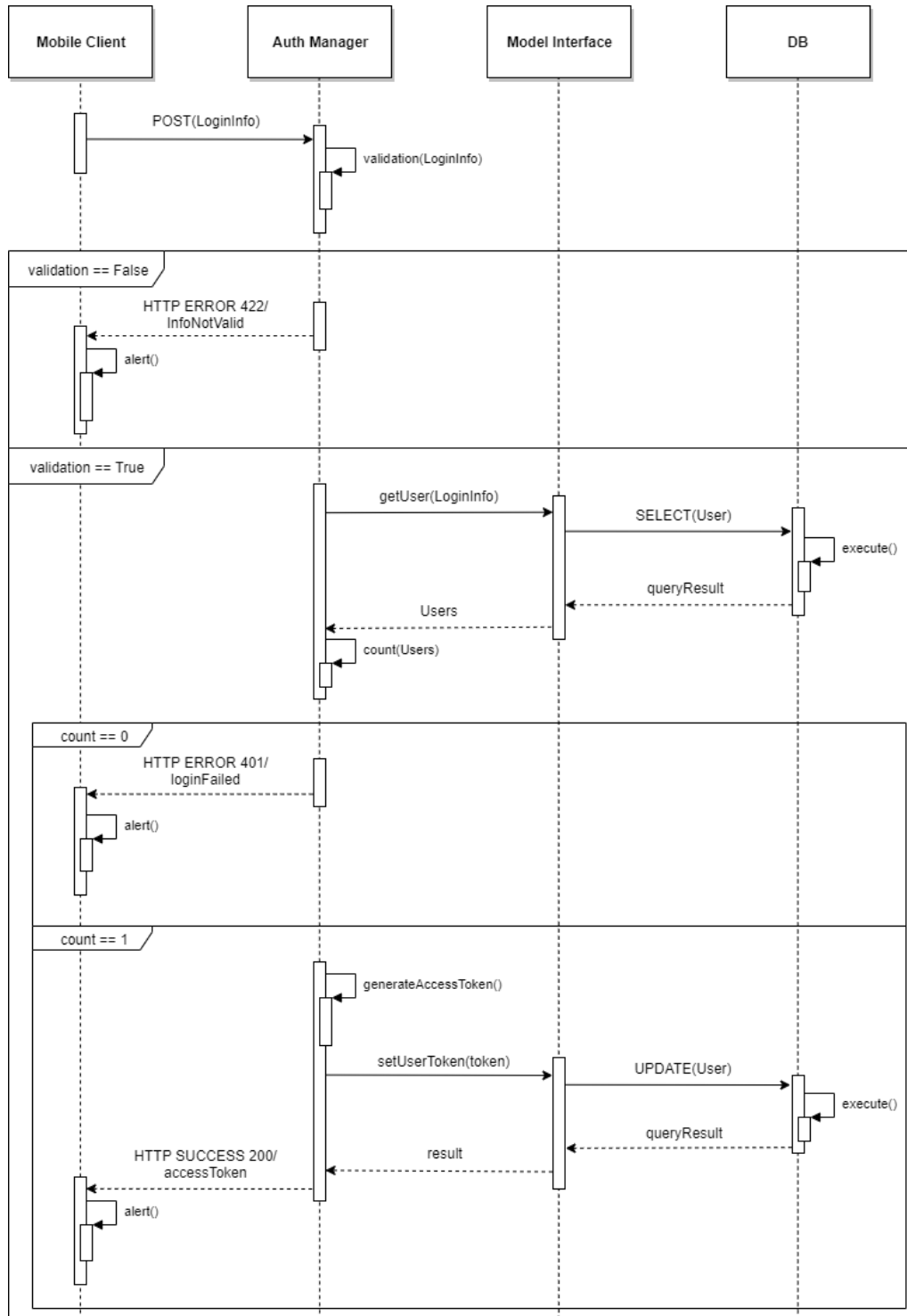


Figure 2.5: *Login Runtime View*

2.4.3 Create Violation Report Runtime View

The Citizen opens the violation report section of the application, fills the required fields and submits the request to create a new report, along with his/her access token. The Violation Manager delegates the Authentication Manager to check the validity of the token and if the Citizen has already verified its account. If this authentication step succeeds, the control goes back to the Violation Manager which checks the correctness of the fields and delegates the permanent saving of the report to the Model Interface. The end of the procedure is notified to the Mobile Client through an alert.

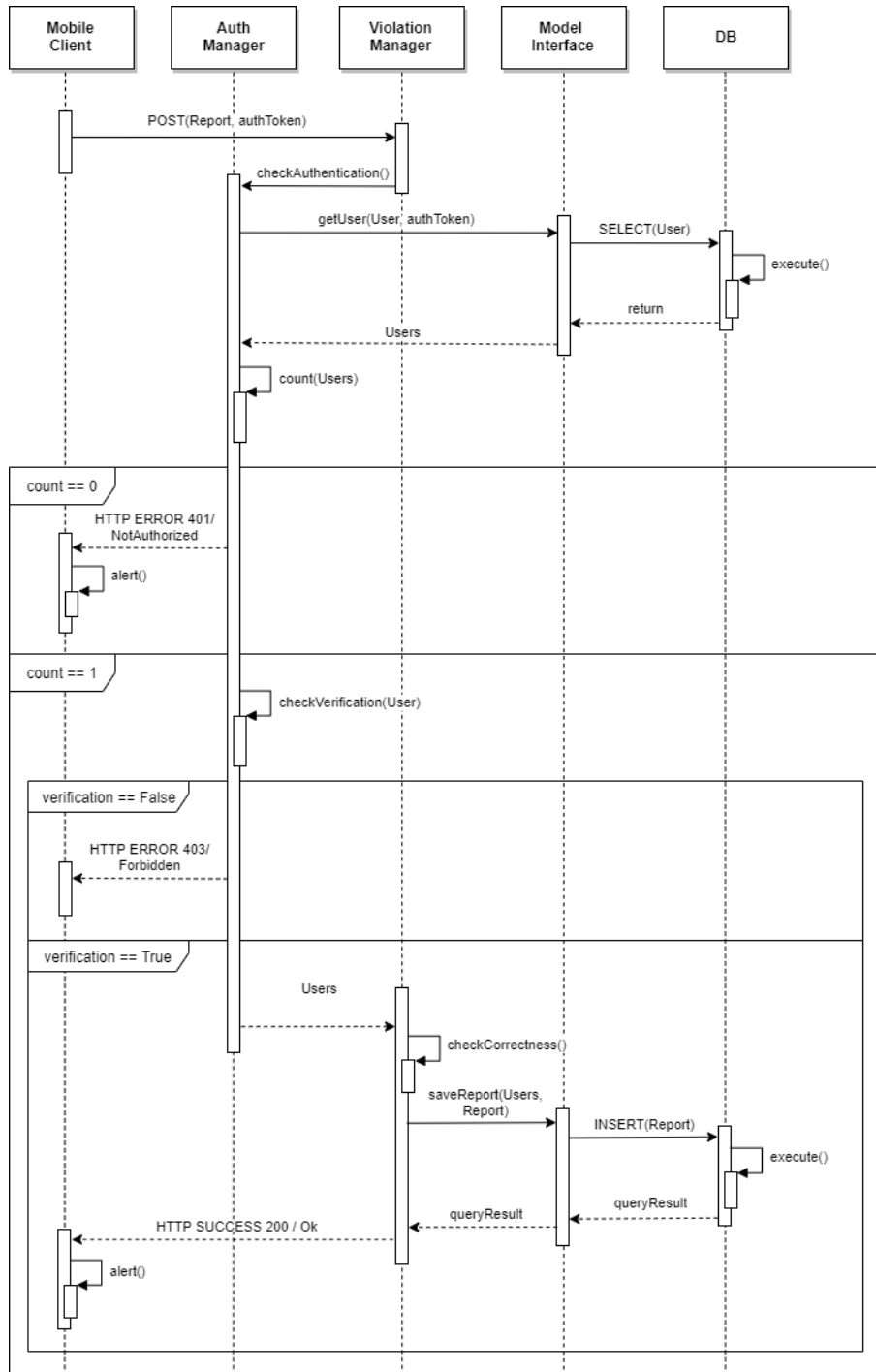


Figure 2.6: *Create Violation Report Runtime View*

2.4.4 Accept Violation Report Runtime View

The Authority, opening the all incoming violations section of the application, implicitly submits a HTTP GET request to the Model Interface, along with his/her access token. Once the Authentication Manager has checked the correctness of this parameter and if the Authority has already verified his/her account, the list of the pending reports are returned to the Client. At this point, through a HTTP POST, the violation is send back to the Violation Manager, including also the outcome of the evaluation (accepted/rejected). This information with be made persistent forwarding the data to the Model Interface. The end of the procedure is notified to the Mobile Client through an alert.

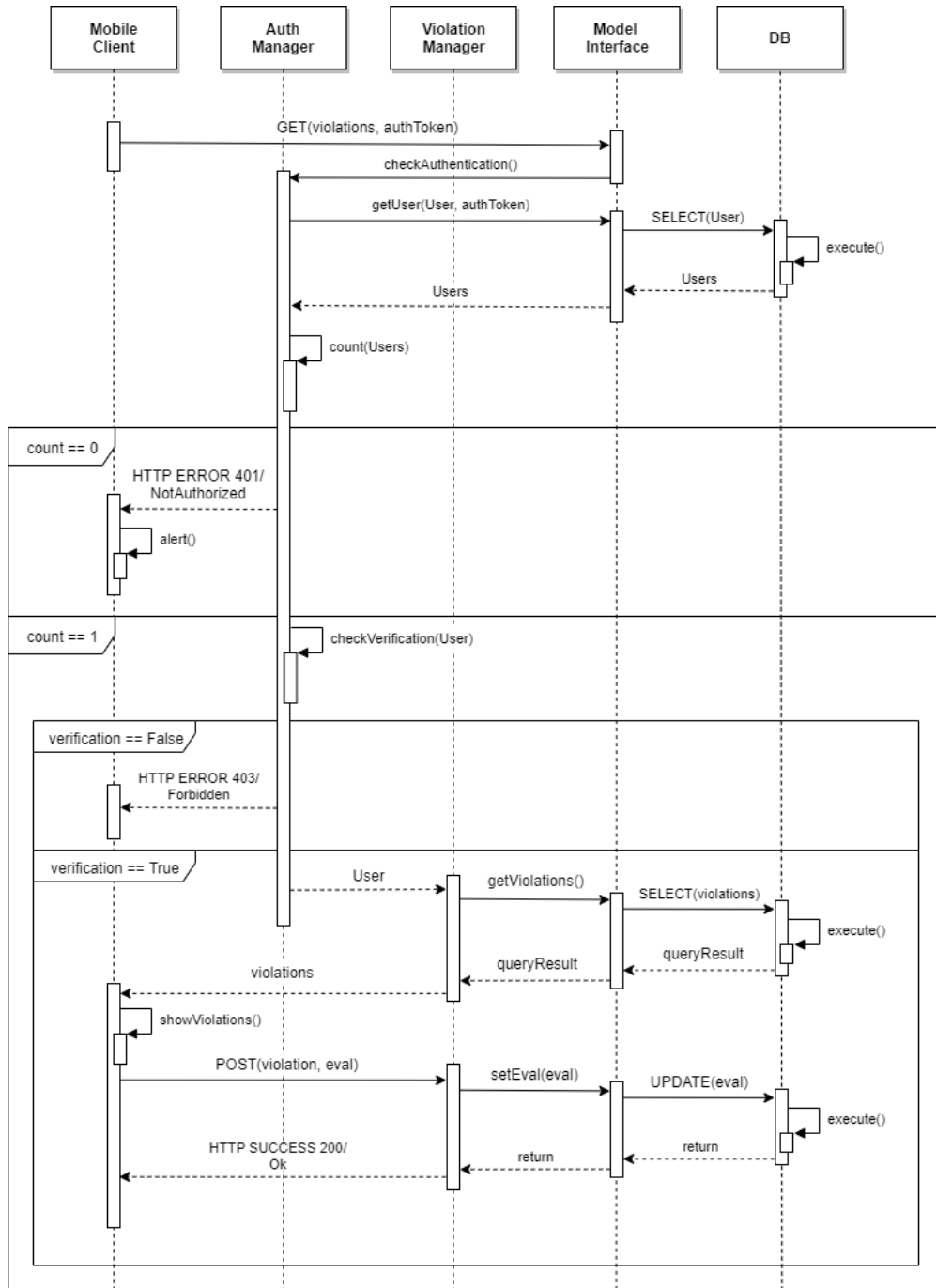


Figure 2.7: *Accept Violation Report Runtime View*

2.4.5 Statistics Runtime View

The Client, selecting the desired statistic, forwards the request to the Statistics Manager, along with the access token. Once the Authentication Manager has checked the correctness of this parameter and if the User has the rights to access the required information, gives back the control to the Statistics Manager. It collects from the Model Interface the local data, the Municipality data from the Municipality Interface and the map from the Map Service Interface. At this point, all this information are joined in the final result, which is returned to the Client.

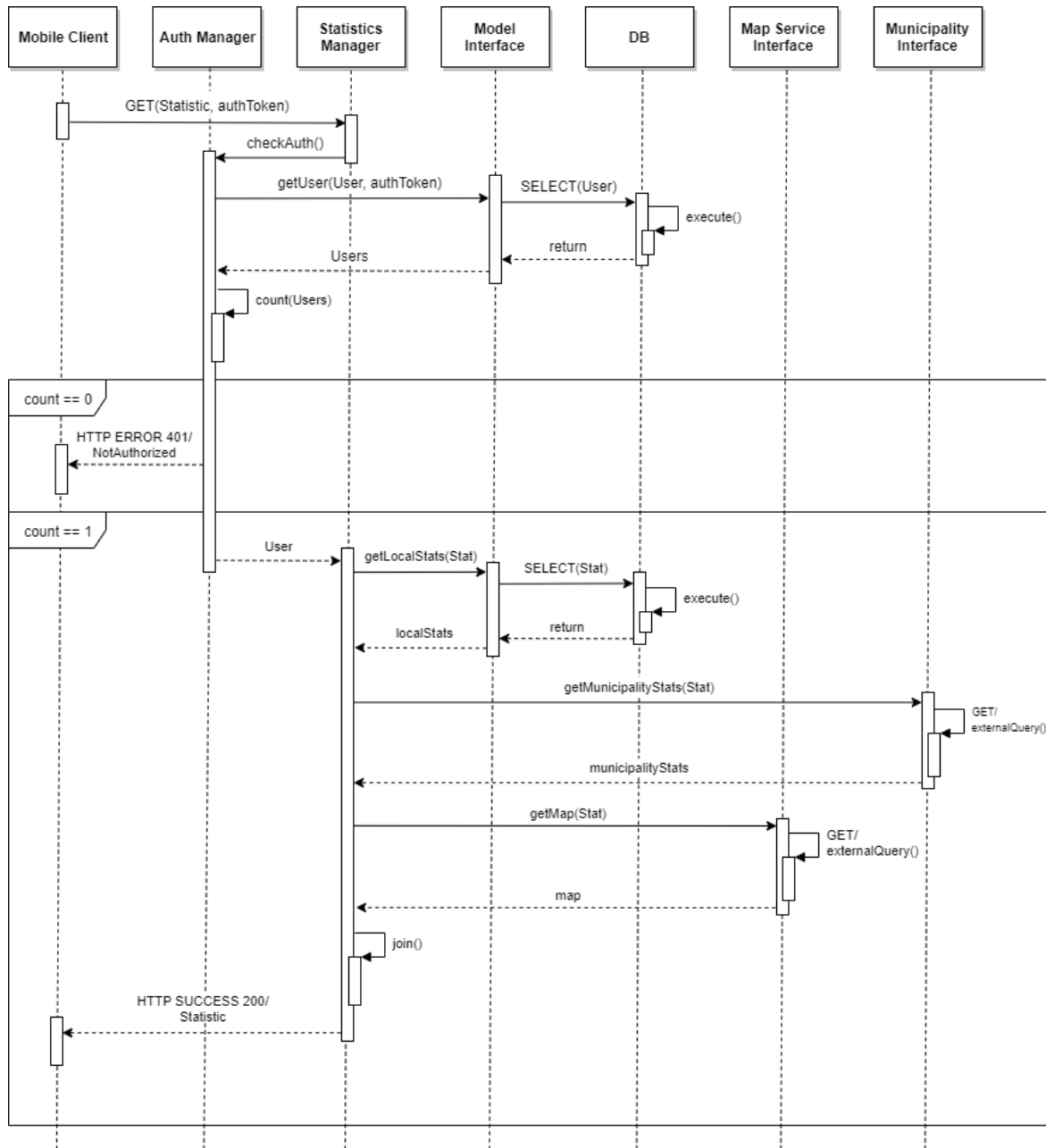


Figure 2.8: *Statistics Runtime View*

2.4.6 Notification Runtime View

Once there is a change in the violation report status, the Violation Manager notifies the Notification Controller of the changes occurred. The Notification Controller gathers the new notifications information and configures the Push Notification Service. The Push Notification Service will then notify the Client of the acceptance or refusal of his/her violation report.

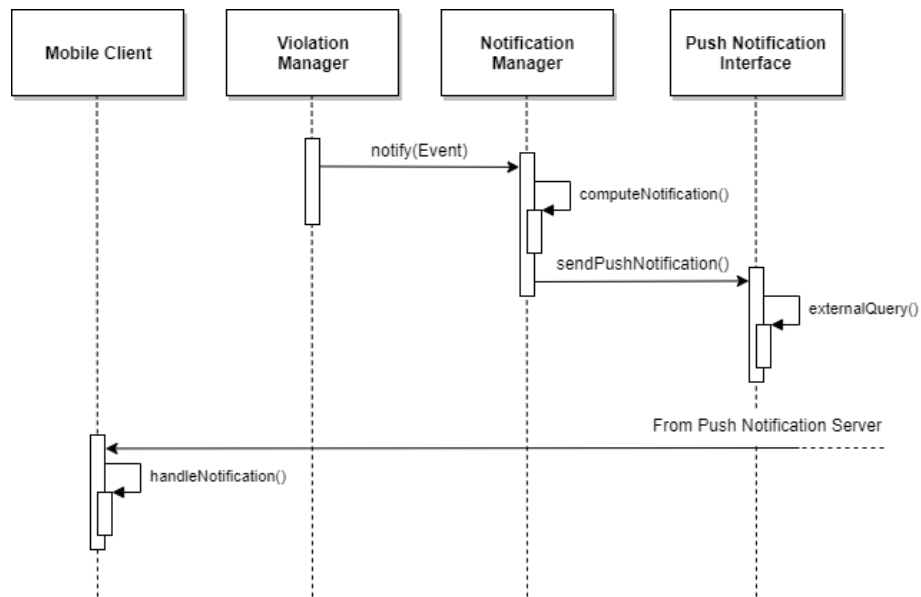


Figure 2.9: *Notification Runtime View*

2.5 Component Interfaces

Inside this section we provide the interfaces of our components; we explain in details how the different components of our *System* communicate to offer a readable and easy understandable guide to who will develop the communication system behind the possible operations of the application.

2.5.1 REST API

The application will use the REST paradigm (explained better later on). So we focus on the different REST API endpoints that the application uses, describing in details the following attributes for each of the requests:

- The operation name of the request
- The precise endpoint URL
- The method type (GET, POST, PUT or DELETE)
- The URL parameters (in case of GET or DELETE)
- The data parameters (in case of POST or PUT)
- The success and the error responses. Let's specify right now the possible responses in our application APIs:
 - Code: 200 OK
 - Code: 400 BAD REQUEST
 - Code: 401 UNAUTHORIZED
 - Code: 403 FORBIDDEN
 - Code: 404 NOT FOUND
 - Code: 422 UNPROCESSABLE ENTRY
- A briefly more precise description of what the request existence permits (what operation the *User* is allowed to do thanks to this?)

In particular we have different API interfaces for each *Manager* in the application. The managers names and their functionalities follow:

- **Authentication Manager:** The Authentication Manager provides what is needed for the registration and the login to the application; furthermore we want the possibility to verify an account, so we expect the Authentication Manager to manage this.

- **User Manager:** This is the manager related to the possibilities given to the *User* in order to obtain and update information such as preferences, settings or credentials.
- **Violation Manager:** The Violation Manager provides all the necessary operations regarding violations reported by *Citizens*. For example, it has to permit to add or delete (if not already accepted) the reports or to accept them by *Authorities*.
- **Statistics Manager:** This last manager is related to the feature of statistics visualization and process.

For reading purposes for each following page we will have only one endpoint (two at maximum if the sizes of the tables are such that they can stay without problems in only one page).

Authentication Manager - Registration of a *Citizen*

Endpoint	*/auth/register/citizen
Method	POST
URL Params	
Data Params	fiscalCode: [alphanumeric] name: [text] surname: [text] mail: [alphanumeric] username: [alphanumeric] password: [alphanumeric] birthDate: [Date]
Success Response	Code: 200 Content: {message: "Registration successful, check your email to complete the creation of your account"}
Error Response	Code: 403 Content: {error: "Already registered"} Code: 422 Content: {error: "Registration Data not correct"}
Notes	Allows a Client to request the registration of a new <i>Citizen</i>

Authentication Manager - Registration of an *Authority*

Endpoint	*/auth/register/authority
Method	POST
URL Params	
Data Params	authorityID: [alphanumeric] name: [text] surname: [text] mail: [alphanumeric] username: [alphanumeric] password: [alphanumeric]
Success Response	Code: 200 Content: {message: "Registration successful, check your email to complete the creation of your account"}
Error Response	Code: 403 Content: {error: "Already registered"} Code: 422 Content: {error: "Registration Data not correct"}
Notes	Allows a Client to request the registration of a new <i>Authority</i>

Authentication Manager - Login of a *User*

Endpoint	*/auth/login
Method	POST
URL Params	
Data Params	username: [alphanumeric] password: [alphanumeric]
Success Response	Code: 200 Content: { userType: [text] accessToken: [alphanumeric] }
Error Response	Code: 401 Content: {error: "Wrong mail or password"} Code: 422 Content: {error: "Login Data not correct"}
Notes	Allows a Client to obtain an authentication token and to login as <i>Citizen</i> or as <i>Authority</i>

Authentication Manager - Activation of an account

Endpoint	*/auth/activate
Method	GET
URL Params	activationCode: [alphanumeric]
Data Params	
Success Response	Code: 200 Content: {message: "Account activated"}
Error Response	Code: 401 Content: {error: "Invalid token"} Code: 403 Content : {error: "Account already activated"} Code: 404 Content: {error: "Incorrect activation Code"}
Notes	Allows a Client to activate the account

User Manager - Get a *Citizen* account information

Endpoint	*/citizen/{id}
Method	GET
URL Params	accessToken: [alphanumeric]
Data Params	
Success Response	<p>Code: 200</p> <p>Content: { fiscalCode: [alphanumeric] name: [text] surname: [text] email: [alphanumeric] birthDate: [Date] username: [alphanumeric] password: [alphanumeric] applicationSettings: [List <Setting>] }</p>
Error Response	<p>Code: 400 Content: {error: "Wrong request"}</p> <p>Code: 401 Content: {error: "Login not correct"}</p> <p>Code: 403 Content: {error: "User ID does not match with the provided authentication token"}</p> <p>Code: 404 Content: {error: "User not found"}</p>
Notes	Allows a Client to obtain the information related to the <i>Citizen</i> associated to the provided Token

User Manager - Modification of a *Citizen* account information

Endpoint	*/citizen/{id}
Method	PUT
URL Params	
Data Params	accessToken: [alphanumeric] oldPassword (optional): [alphanumeric] newPassword (optional): [alphanumeric] name (optional): [text] surname (optional): [text] birthDate (optional): [Date] email (optional): [alphanumeric] applicationSettings (optional): [List <Setting>]
Success Response	Code: 200 Content: {message: "Settings correctly updated"}
Error Response	Code: 400 Content: {error: "Wrong request"} Code: 401 Content: {error: "Login not correct"} Code: 403 Content: {error: "User ID does not match with the provided authentication token"} Code: 404 Content: {error: "User not found"} Code: 422 Content: {error: "New settings data not correct"}
Notes	Allows a Client to modify the information or the settings related to the <i>Citizen</i> associated to the provided Token

User Manager - Get an *Authority* account information

Endpoint	*/authority/{id}
Method	GET
URL Params	accessToken: [alphanumeric]
Data Params	
Success Response	Code: 200 Content: { authorityID: [alphanumeric] name: [text] surname: [text] email: [alphanumeric] username: [alphanumeric] password: [alphanumeric] applicationSettings: [List <Setting>] }
Error Response	Code: 400 Content: {error: "Wrong request"} Code: 401 Content: {error: "Login not correct"} Code: 403 Content: {error: "User ID does not match with the provided authentication token"} Code: 404 Content: {error: "User not found"}
Notes	Allows a Client to obtain the information related to the <i>Authority</i> associated to the provided Token

User Manager - Modification of an *Authority* account information

Endpoint	*/authority/{id}
Method	PUT
URL Params	
Data Params	accessToken: [alphanumeric] oldPassword (optional): [alphanumeric] newPassword (optional): [alphanumeric] name (optional): [text] surname (optional): [text] email (optional): [alphanumeric] applicationSettings (optional): [List <Setting>]
Success Response	Code: 200 Content: {message: "Settings correctly updated"}
Error Response	Code: 400 Content: {error: "Wrong request"} Code: 401 Content: {error: "Login not correct"} Code: 403 Content: {error: "User ID does not match with the provided authentication token"} Code: 404 Content: {error: "User not found"} Code: 422 Content: {error: "New settings data not correct"}
Notes	Allows a Client to modify the information or the settings related to the <i>Authority</i> associated to the provided Token

Violation Manager - Showing violations reports by a *Citizen*

Endpoint	*/citizen/home
Method	GET
URL Params	accessToken: [alphanumeric]
Data Params	
Success Response	Code: 200 Content: { violations: [violationID: [alphanumeric] reportDate: [Date] reportTime: [Time] plate: [text] latitude: [float] longitute: [float] streetName: [text] violationType: [int] photos: [List <Photo>]] }
Error Response	Code: 400 Content: {error: "Wrong request"} Code: 401 Content: {error: "Login not correct"}
Notes	Allows a Client to show violations reports made by the <i>Citizen</i> associated to the provided Token (at the start of application in the home page)

Violation Manager - Filtering violations reports by a *Citizen*

Endpoint	*/citizen/home
Method	GET
URL Params	accessToken: [alphanumeric] endDate: [Date] startDate (optional): [Date] distance (optional): [int] violationTypes (optional): [List <int>] plate (optional): [text]
Data Params	
Success Response	Code: 200 Content: { violations: [violationID: [alphanumeric] reportDate: [Date] reportTime: [Time] plate: [text] latitude: [float] longitute: [float] streetName: [text] violationType: [int] photos: [List <Photo>]] }
Error Response	Code: 400 Content: {error: "Wrong request"} Code: 401 Content: {error: "Login not correct"} Code: 404 Content: {error: "Too many filters, no violation reports found"}
Notes	Allows a Client to show violations reports made by the <i>Citizen</i> associated to the provided Token (at the start of application in the home page)

Violation Manager - Showing a *Citizen*'s violation report (by the *Citizen*)

Endpoint	*/citizen/report/{id}
Method	GET
URL Params	accessToken: [alphanumeric]
Data Params	
Success Response	<p>Code: 200</p> <p>Content: {</p> <p> violationID: [alphanumeric]</p> <p> reportDate: [Date]</p> <p> reportTime: [Time]</p> <p> plate: [text]</p> <p> latitude: [float]</p> <p> longitute: [float]</p> <p> streetName: [text]</p> <p> violationType: [int]</p> <p> photos: [List <Photo>]</p> <p>}</p>
Error Response	<p>Code: 400</p> <p>Content: {error: "Wrong request"}</p> <p>Code: 401</p> <p>Content: {error: "Login not correct"}</p> <p>Code: 403</p> <p>Content {error: "Report ID does not match with the provided authentication token"}</p> <p>Code: 404</p> <p>Content: {error: "Wrong request, report not found"}</p>
Notes	Allows a Client to show to the <i>Citizen</i> associated to the provided Token the violation report with id={id}

Violation Manager - Adding a violation report by a *Citizen*

Endpoint	*/citizen/report
Method	POST
URL Params	
Data Params	accessToken: [alphanumeric] reportDate: [Date] reportTime: [Time] plate: [text] latitude: [float] longitude: [float] streetName: [text] violationType: [int] photos: [List <Photo>]
Success Response	Code: 200 Content: {message: "Violation report correctly created"}
Error Response	Code: 400 Content: {error: "Wrong request"} Code: 401 Content: {error: "Login not correct"} Code: 422 Content: {error: "Violation report data not correct"}
Notes	Allows a Client to add a violation report related to the <i>Citizen</i> associated to the provided Token

Violation Manager - Deleting a violation report

Endpoint	*/citizen/report/{id}
Method	DELETE
URL Params	accessToken: [alphanumeric]
Data Params	
Success Response	Code: 200 Content: {message: "Violation report correctly deleted"}
Error Response	Code: 401 Content: {error: "Login not correct"} Code: 403 Content {error: "Report already accepted"} Code: 403 Content {error: "Report ID does not match with the provided authentication token"} Code: 404 Content: {error: "Wrong request, report not found"}
Notes	Allows a Client to delete a violation reported by the <i>Citizen</i> associated to the provided Token

Violation Manager - Showing to an *Authority* a *Citizen's* report

Endpoint	*/authority/report/{id}
Method	GET
URL Params	accessToken: [alphanumeric]
Data Params	
Success Response	<p>Code: 200</p> <p>Content: {</p> <ul style="list-style-type: none"> violationID: [alphanumeric] reportDate: [Date] reportTime: [Time] plate: [text] latitude: [float] longitute: [float] streetName: [text] violationType: [int] photos: [List <Photo>] citizenFiscalCode: [alphanumeric] citizenName: [text] citizenSurname: [text] citizenDateOfBirth: [Date] <p>}</p>
Error Response	<p>Code: 400</p> <p>Content: {error: "Wrong request"}</p> <p>Code: 401</p> <p>Content: {error: "Login not correct"}</p> <p>Code: 403</p> <p>Content {error: "Report already accepted"}</p> <p>Code: 404</p> <p>Content: {error: "Wrong request, report not found"}</p>
Notes	Allows a Client to show a violation report made by a <i>Citizen</i> to an <i>Authority</i> associated to the provided Token

Violation Manager - Accepting a violation report (by an *Authority*)

Endpoint	*/authority/report/{id}
Method	POST
URL Params	
Data Params	accessToken: [alphanumeric]
Success Response	Code: 200 Content: {message: "Violation report correctly accepted"}
Error Response	Code: 400 Content: {error: "Wrong request"} Code: 401 Content: {error: "Login not correct"} Code: 403 Content {error: "Report already accepted"} Code: 404 Content: {error: "Wrong request, report not found"}
Notes	Allows a Client to let the <i>Authority</i> associated to the provided Token to accept a violation report

Statistics Manager - Showing area safeness statistics

Endpoint	*/stats/area
Method	GET
URL Params	latitude: [float] latitude: [float] radius (optional): [int]
Data Params	
Success Response	Code: 200 Content: { areas: [latitude: [float] longitude: [float] radius: [int] safetyLevel: [int]] }
Error Response	Code: 400 Content: {error: "Wrong request"} Code: 404 Content: {error: "Wrong request, statistics not found"}
Notes	Allows a Client to showing area statistics related to latitude and longitude parameters

Statistics Manager - Showing intervention suggestions

Endpoint	*/stats/interventions
Method	GET
URL Params	latitude: [float] latitude: [float] radius (optional): [int]
Data Params	
Success Response	Code: 200 Content: { interventions: [latitude: [float] longitude: [float] intervention: [text]] }
Error Response	Code: 400 Content: {error: "Wrong request"} Code: 404 Content: {error: "Wrong request, statistics not found"}
Notes	Allows a Client to showing suggestions of interventions inside the area related to latitude and longitude parameters

2.6 Selected Architectural Styles and Patterns

In this section the most relevant architectural styles and patterns used in the application are presented. We will also provide explanations about their role and the advantages they bring.

2.6.1 Multi-tier Architecture

The chosen architecture for the *System* is the multi-tier one; this consists in the physical division of *presentation*, *application processing* and *management functions*. This division guarantees to each tier a specific task to manage in order to spread in a correct way the computation (avoiding also a central node in charge of every task necessary for the *System* functionalities). Further this type of architecture permits developers to create flexible Code and to modify in the future only the required specific layer instead of work on the entire application.

In particular we adopt a standard 4-tier architecture providing the 4 following layers (also described in details):

- **Presentation tier:** This is the only layer that the final *User* can access directly. The purpose of it is to offer an interface to the *User* in order to access in an easy and understandable way to all the tasks requested and all the results produced by the *System*.
- **Web tier:** This is the layer used by the web clients (in our case offered only to *Authorities* profiles). Its purpose is to retrieve the Code and the resources in order to run the web application. The latter follows the Single Page Application (SPA) paradigm; this means that the application doesn't load entire new pages from the server at any new request by the *User*, instead the application interacts with the *User* in a dynamic way, rewriting the page when requested by the input.
- **Domain/Business logic tier:** This tier is the core of our application. Separated from the presentation tier (and also from the web one) we need a tier that controls the logic and the functions offered by the *System*; for this reason we have that the *Domain/Business logic tier* includes the application servers, the proxy server and the push notification one.
- **Data Storage tier:** This last tier includes all the architecture behind storage and management of the application's data; in particular here we

here we find the DBMS server, the firewall protecting the accesses to data and all future additions about storage and data.

We can re-visualize the overview diagram indicating which components belongs to which tier.

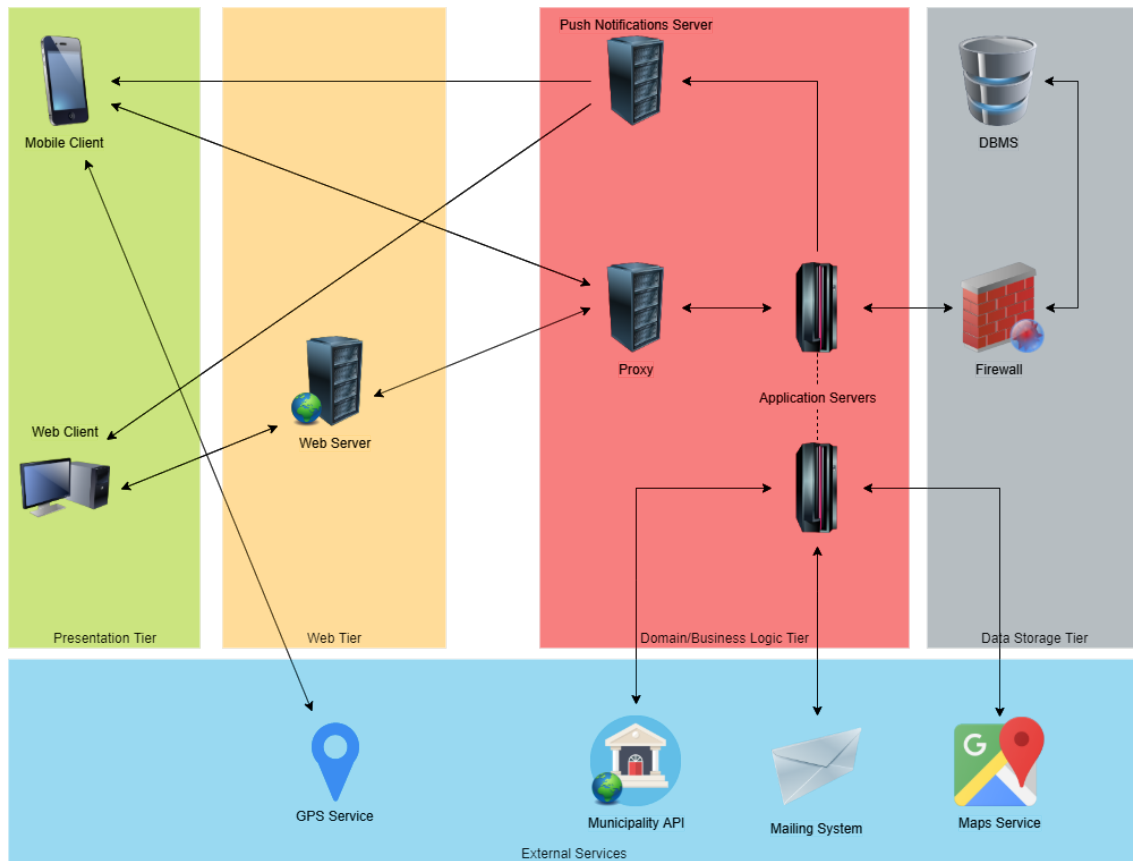


Figure 2.10: *Overview of the System visualizing different tiers.*


2.6.2 Use of RESTful guidelines

We need a paradigm that explain how the client exchanges information, makes requests or receives responses with respect to the application servers. We choose to build a communication system according to the **RE**presentational **S**tate **T**ransfer constraints; this permits to have a simpler component design and a simpler management of intermediate layers and components like proxies or firewalls thanks to the stateless nature of this paradigm. Data Exchange is made through the HTTP protocol (using SSL to encrypt sensible data for privacy and security reasons).

Furthermore a stateless way to communicate permits an uniform interface and a well-scalable system with future possibilities of upgrades of the *System*.

2.6.3 DBMS

The Data Storage tier consists in a *Relational* DBMS. It permits to represent the relationships between data that need to be stored; further make complex operations on the tables is easier with this design (they are not well handled with a non-relational model). Let's quickly explain some of the conventions used in the Entity-Relationship Diagram:

- **Tables Conventions:** Each table has its name and a list of attributes; each attribute is described by its name, its type and by special Codes (briefly explained in next points).
- **PK:** the Primary Key of the table.
- **FK:** a Foreign Key of the table.
- **U:** the attribute is Unique (i.e. different *Citizens* can't have the same Username).
- **N:** the attribute is Nullable (i.e. a Violation Report can have or not a Description).
- : this indicates a "One To Many" relation between entities A and B. In particular A can exist also without any B in the database.

The diagram follows in the next page.

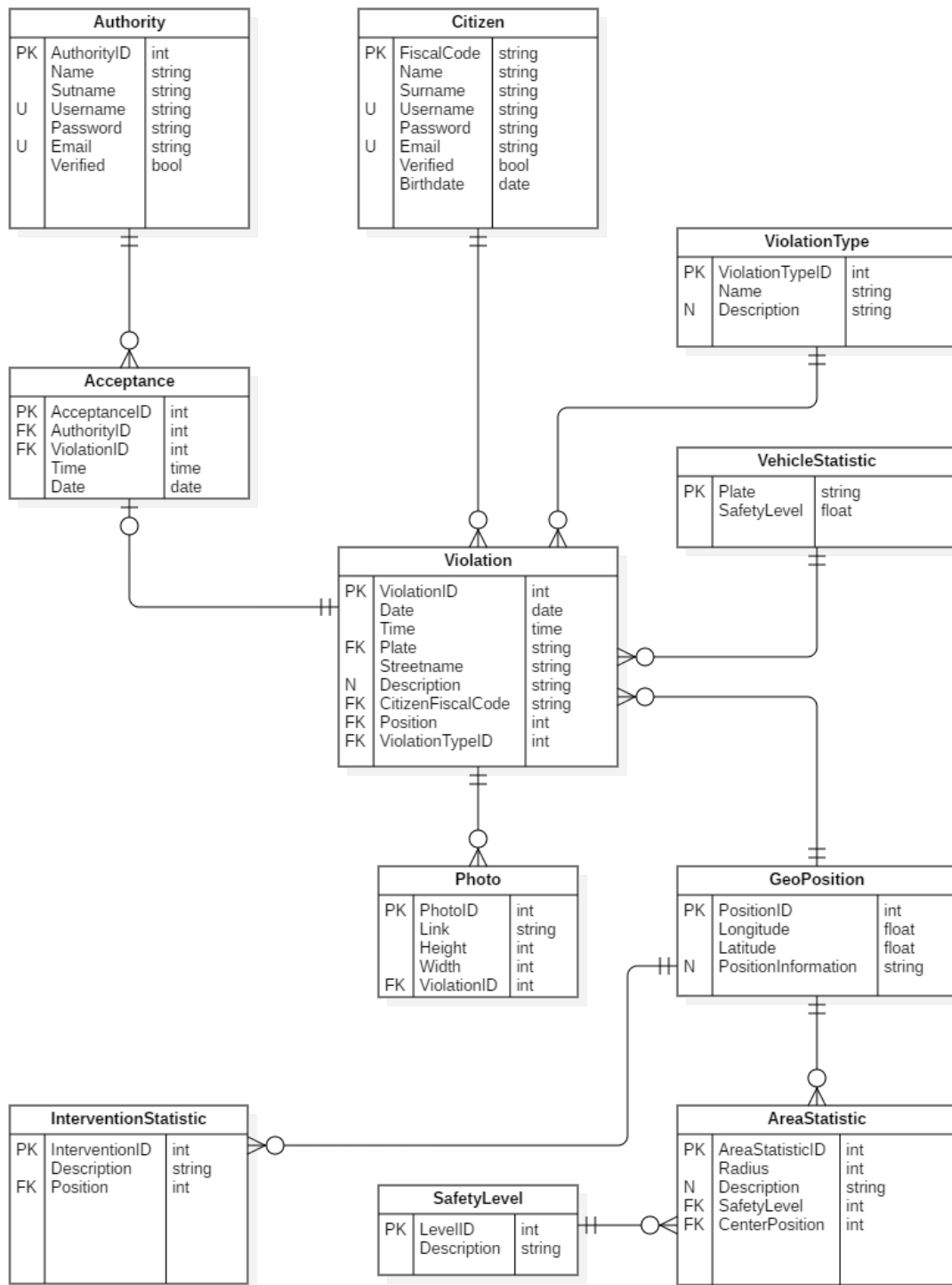


Figure 2.11: *Entity-Relationship Diagram present in the DBMS.*

2.7 Other Design Decisions

Here we explain other minor paradigms followed by the application.

2.7.1 Thin Client

In order to have a client as light as possible the application follows the "Thin Client" paradigm; in this way the client doesn't need to perform big computations but instead it must handle only minimal efforts in order to communicate with the application servers (in fact almost all of the computation performed by the *System* is positioned in the application servers).

Moreover, this paradigm permits the benefit of an easier synchronization of data across multiple entities and an easier way to update the application (the client doesn't need to change so much even if the change, with the respect to the previous behavior, is huge); also we can know that a such that client doesn't relies on local store or local CPU.

2.7.2 MVC

Carrying on the description of the client we follow the Model-View-Controller (MVC); This architectural pattern impose to separate the application in 3 different components permitting a full encapsulation with the consequence that each component can be modified in a independent way with the respect of other ones; furthermore this permits an easier way to perform testing operations.

2.7.3 Firewalls

Talking about security instead we make use of firewalls in our architecture. In particular we have an actual firewall protecting the access to our DBMS, for the application servers security we leave at the reverse proxy the task. These components permit to protect the internal network of the *System* from possible attacks by the untrusted external network; furthermore they operate by letting a possibility for filtering packets that pass through them (also for different reasons from that of secure the network).

3 User Interface Design

3.1 User eXperience Diagrams

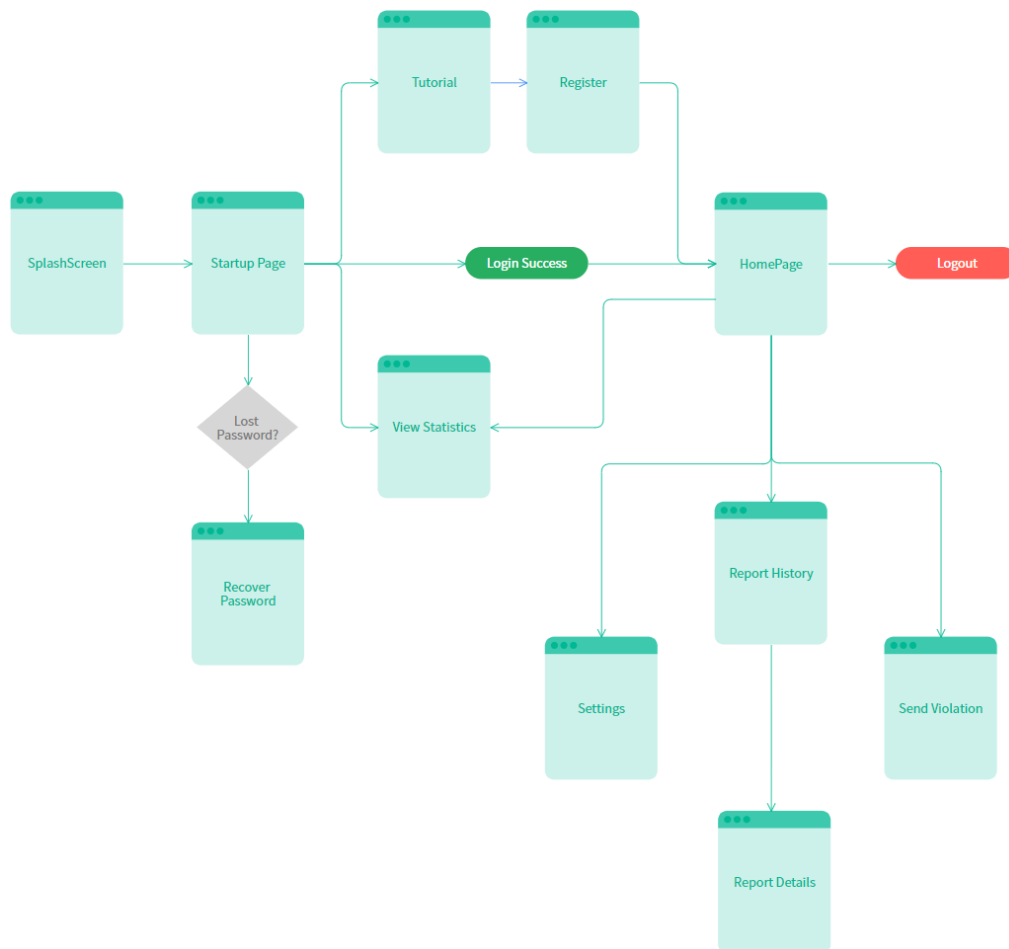


Figure 3.1: *Overview of the System.*

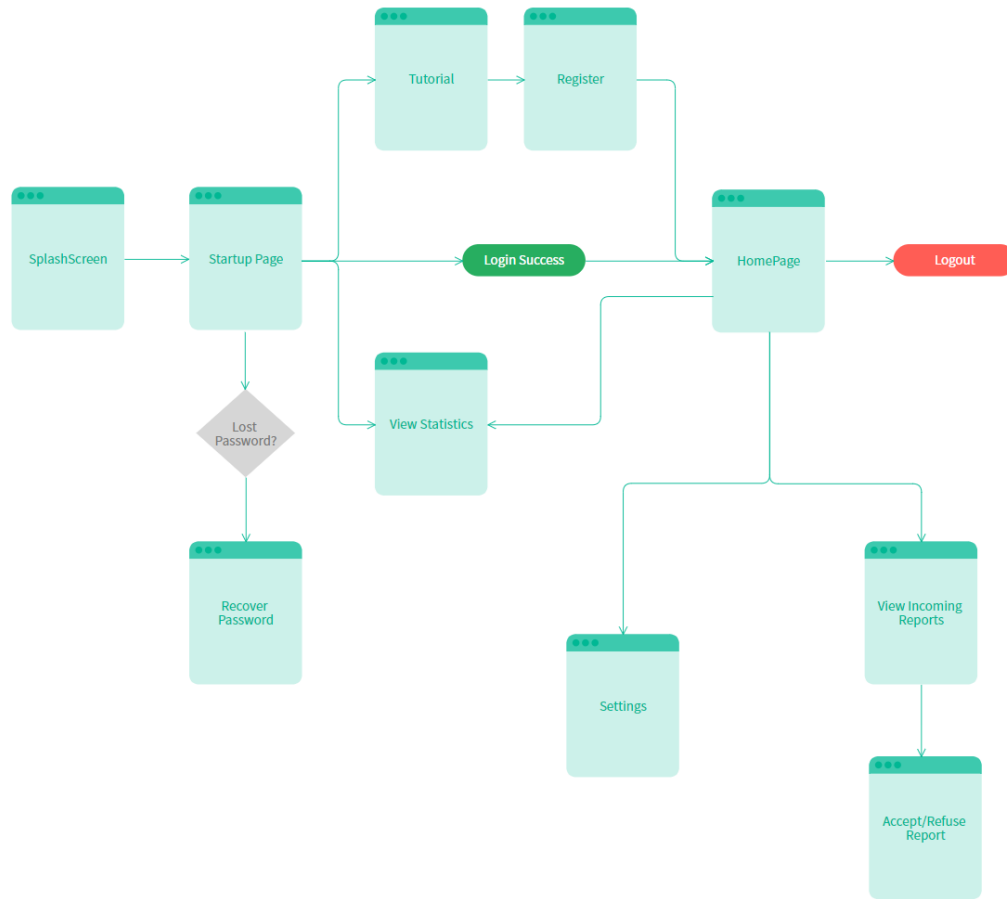


Figure 3.2: *Overview of the System.*

For the User Interface Design, we refer to section 3.1.1 of the RASD document. In this section, we enrich what was shown in the RASD by means of an User Experience diagram, which is principally thought to explain as clearly as possible the relationships between the various sections of the application. As we can see from the diagram, the leftmost part is merely devoted to the necessary operations to log into the application:

- Startup Page
- Login
- Tutorial and Register
- Password Recovery

The main window of the application is the HomePage, from which the core functionalities of *SafeStreets* are reachable. If the User is a Citizen, he/she can access the Settings to modify his/her preferences, the Send Violation page to send a new report and the Report History page. There he/she can access the history of all the previous reports and explore them in details. The flow is quite the same even for the Authority, except for the executive pages. In this case the View incoming reports is dedicated to show all the pending violation reports and from it is possible to accept or refuse them in Accept/Refuse Report page. Finally, as described in the RASD, even unregistered Users can access the View Statistics page, so it can be reached through different paths, according to the different roles.

3.2 Web Interface

The mockup shows the **SafeStreets** web interface. The top navigation bar includes the logo, links for **Violations Reports**, **Statistics**, **Interventions Suggestions**, and a user status bar indicating "Hi Marco, you are logged in" with user and settings icons.

The left sidebar contains **Filter Options** with the following sections:

- Start Date:** A calendar for April 22, 2012, with the 22nd selected.
- End Date:** A calendar for April 22, 2012, with the 22nd selected and a checkmark icon.
- Violation Type:** A dropdown menu.
- Specific Plate:** A text input field.
- Specific Fiscal Code:** A text input field.
- Position Filter:** A map showing the area around Triennale Milano and Castello Sforzesco.

The main content area displays a list of violation reports:

- Violation Report #4b11s8:** Date and Time: April 20 2012, 13:42. 3 Photos. Latitude: 45.500264, Longitude: 9.191120. Street Name: Via Taormina, 27, 20159 Milano. Plate: SB 342 BY. Reporter Full Name: Marco Rossi. Buttons: Details, Accept.
- Violation Report #4b11s9:** Date and Time: April 20 2012, 15:26. 4 Photos. Latitude: 45.58102, Longitude: 9.106381. Street Name: Cascina San Giuseppe, 20030 Senago. Plate: SB 342 BY. Reporter Full Name: Giuseppe Serna. Buttons: Details, Accepted.
- Violation Report #4b11t0:** Date and Time: April 21 2012, 10:07. 3 Photos. Latitude: 45.483193, Longitude: 9.163772. Street Name: Corso Sempione, 54, 20154 Milano. Plate: SB 342 BY. Reporter Full Name: Damiano Veraschi. Buttons: Details, Accept.
- Violation Report #4b12w3:** Date and Time: April 22 2012, 16:20. 1 Photo. Latitude: 45.49833.

Figure 3.3: Mock up of the Web Page interface offered to Authorities.

In order to give the possibility to use the application also without the mobile app we want to implement a web interface reserved only to the *Authorities*. In particular in this way we give the opportunity to those *Authorities* that are working in office to use *SafeStreet* to check statistics about areas, vehicles or interventions suggested or to consult reports before going out the office.

4 Requirements Traceability

In this section we provide a relation between Managers and requirements defined in the RASD document. In particular we want to show how the system architecture's components satisfy the requirements. The traceability table follows:

Component (DD)	Requirements (RASD)
Authentication Manager	<p>[R.1] A <i>Citizen</i> account can be created if and only if the User provides his/her fiscal code.</p> <p>[R.2] A <i>Citizen</i> account can be created if and only if the person is of legal age.</p> <p>[R.3] An <i>Authority</i> account can be created if and only if the User provides his/her badge number.</p> <p>[R.4] System shall check if the User's credentials are valid</p> <p>[R.6] To access the services, <i>Citizens</i> and <i>Authorities</i> need to login with their credentials.</p> <p>[R.7] The System shall ask the User to agree on the policy about data acquisition.</p> <p>[R.8] The System shall allow only verified <i>Citizens</i> to report a violation.</p> <p>[R.24] The System shall allow the <i>Authority</i> to view the details of a report.</p> <p>[R.25] The System shall allow the <i>Authority</i> to view the <i>Citizens</i> profile.</p> <p>[R.32] The System shall check the role of the User before providing the requested statistics.</p> <p>[R.33] The System shall allow <i>Guests</i> to navigate publicly available statistics.</p>
User Manager	<p>[R.5] System shall allow the User, under certain conditions, to change his email and password.</p>

	<p>[R.40] The System shall allow a User to consult and modify application settings and preferences.</p>
Violation Manager	<p>[R.8] The System shall allow only verified <i>Citizens</i> to report a violation.</p> <p>[R.9] The System shall provide an interface to report a violation.</p> <p>[R.10] The System shall allow the <i>Citizen</i> to upload up to 4 photos in the report.</p> <p>[R.11] The System shall precompile the <i>Street Name</i> field using the smartphone's GPS.</p> <p>[R.12] The System shall allow the <i>Citizen</i> to manually insert the <i>Street Name</i> field.</p> <p>[R.13] The System shall precompile the <i>Date</i> and <i>Time</i> using the smartphone's clock.</p> <p>[R.14] The System shall allow the <i>Citizen</i> to manually insert the <i>Date</i> and <i>Time</i> fields.</p> <p>[R.15] The System shall prevent the sending of reports with missing or incorrect fields.</p> <p>[R.17] The System shall check the correctness of the license plate through a Machine Learning algorithm.</p> <p>[R.18] The System shall check the latitude and longitude metadata attached to the photo and cross them with the <i>Street Name</i> provided by the <i>Citizen</i>.</p> <p>[R.19] The System shall check the <i>Time</i> field, if manually input by the <i>Citizen</i>, is not subsequent to the precompiled one.</p> <p>[R.20] The System shall perform signal processing in order to spot possible counterfeit images.</p> <p>[R.28] The System shall allow the <i>Citizen</i> to modify a pending report.</p>
Model Interface	<p>[R.15] The System shall prevent the sending of reports with missing or incorrect fields.</p> <p>[R.16] The System shall store the violations of every single <i>Citizen</i>.</p> <p>[R.21] The System shall allow the <i>Authority</i> to see the list of all the reported violations.</p> <p>[R.22] The System shall allow the <i>Authority</i> to filter the results according to the distance.</p> <p>[R.23] The System shall allow the <i>Authority</i> to filter the results according to the time.</p>

	<p>[R.24] The System shall allow the <i>Authority</i> to view the details of a report.</p> <p>[R.25] The System shall allow the <i>Authority</i> to view the <i>Citizens</i> profile.</p> <p>[R.26] The System shall allow <i>Citizens</i> to view the list of all the reported violations since they registered.</p> <p>[R.27] The System shall allow the <i>Citizen</i> to filter his own reported violations.</p> <p>[R.30] The System shall allow the User to filter the results.</p> <p>[R.35] The System shall be able to join its own knowledge base with the results returned by the Municipality API.</p> <p>[R.36] The System shall process the data in order to generate meaningful insights.</p>
Statistics Manager	<p>[R.29] The System shall provide the most recent statistics.</p> <p>[R.31] The System shall allow the User to search for a specific street in the map.</p> <p>[R.32] The System shall check the role of the User before providing the requested statistics.</p> <p>[R.38] The System shall apply machine learning algorithms on statistical data.</p> <p>[R.39] The System shall apply natural language processing on the results in order to express them in natural language.</p>
Notification Manager	<p>[R.41] The System shall notify the User in case of successful actions or in case of errors.</p> <p>[R.42] The System shall notify Authorities about new reports in a selected Area.</p>
Municipality API Interface	<p>[R.34] The System shall be able to request Municipality's statistics, if those are provided.</p> <p>[R.35] The System shall be able to join its own knowledge base with the results returned by the Municipality API.</p>
Map Interface	<p>[R.11] The System shall precompile the <i>Street Name</i> field using the smartphone's GPS.</p> <p>[R.31] The System shall allow the User to search for a specific street in the map.</p> <p>[R.37] The System shall be able to display the results in a graphical format.</p>

5 Implementation, Integration and Test Plan

5.1 Overview

As previously detailed, the System can be broken down in many components that can be divided in these three categories:

- **Frontend components:** constituted by the Client application and, in case of Web Clients, the Presentation, as motivated in Chapter 2.
- **Backend components:** constituted by all the components which are related with the Business Logic and interact with the Data Base. Are part of this category the User Manager, the Authorization Manager, the Violation Manager, the Statistics Manager, the Notification Manager, the Model Interface and all the interfaces related to the external components.
- **External components:** constituted by all the external services on which the System relies. Are part of this category the Mail, Map, Notification and Municipality components, provided by third-party services, and the DBMS.

In order to implement, integrate and test the System, a *bottom-up* strategy will be used, following an incremental approach. More specifically, firstly the components of the same subsystem will be implemented, integrated and tested and only then the whole subsystem will be integrated and tested, in order to comply to the specifications. It has to be noticed that the components of external services are supposed to be reliable, thus not object of further testing. Then, they will be used in place of stubs in order to test the interfaces.

5.2 Unit Testing

During the implementation phase, unit tests will be performed in order to spot as soon as possible bugs and undesired behaviours of the System's components. This is a good practice in software development since it allows to fix errors with the lowest cost possible both in terms of effort and time. Correctly performing unit tests can also be an useful compass during the designing of the System. In order to speed up this process for every new code version, an automated support tool can be used. This choice is led by two main motivations: it saves times for developers, who are no longer required to perform unit tests manually, and because it is easier to guarantee that tests valid for previous versions still hold correctly for new ones. Unit tests on components will focus on exercising interfaces, internal behaviour and modules' interaction. In order to accomplish this last aspect, the missing components will be simulated to allow working in isolation. Drivers will be used to simulate calls to a certain component, while stubs will simulate the call performed by a component under testing.

5.3 Integration Testing

Once unit tests will have assured that each component was correctly implemented, the focus will move on testing and validating the joint behaviour among them. This means analyzing groups of components which depend on one another, highlighting their interaction and spotting any defects. As previously mentioned, the procedure will follow an incremental approach: integration testing will be firstly performed on different components of the same subsystem, and only then these will be, in turn, integrated together forming the whole System. During this phase, the stubs and drivers used in the unit tests will be substituted by real component, except for the Client (driver) and the DBMS (stub).

5.4 Component Integration

As stated in the previous section, firstly component integration will be applied to the subsystems of the backend.

More in detail, the components to be integrated and tested together in this category are:

- User Manager and Model Interface, forming the *User subsystem*

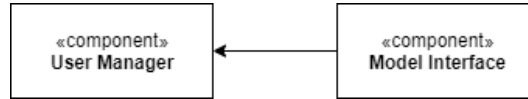


Figure 5.1: *User subsystem*

- Authentication Manager, Model Interface and Mail Service Interface, forming the *Authentication subsystem*



Figure 5.2: *Authentication subsystem*

- Violation Manager, Notification Manager and Push Service Interface, forming the *Notification subsystem*

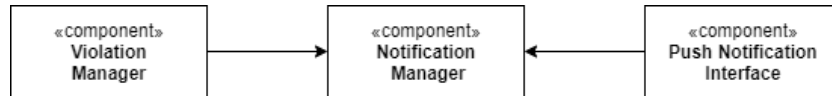


Figure 5.3: *Notification subsystem*

- Violation Manager, Authentication Manager, Map Service Interface and Model Interface forming the *Violation subsystem*

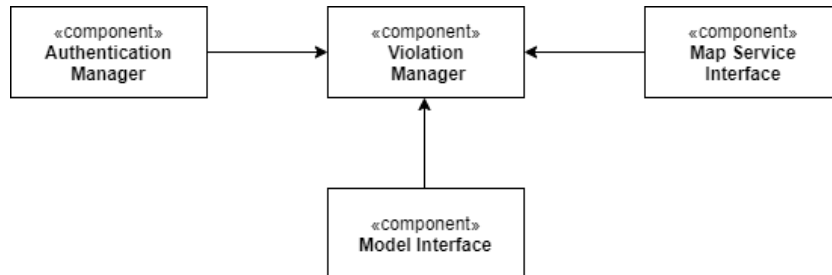


Figure 5.4: *Violation subsystem*

- Statistics Manager, Authentication Manager, Map Service Interface, Municipality Interface and Model Interface forming the *Statistics subsystem*

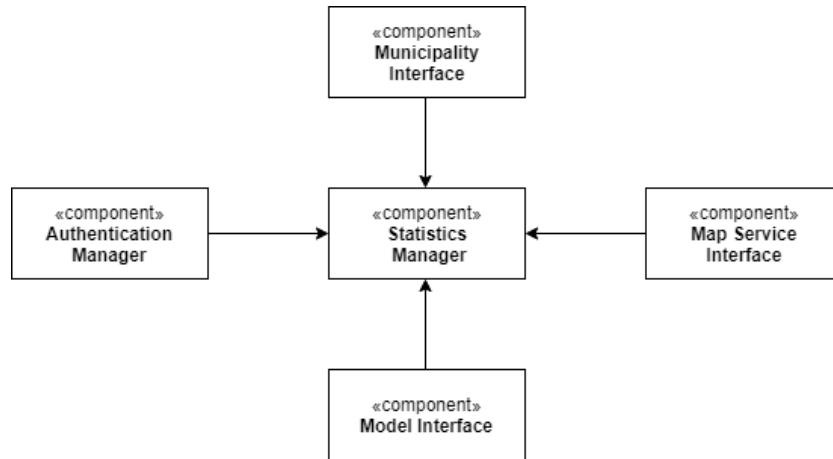


Figure 5.5: *Statistics subsystem*

Once achieved this result, the next step will consist in integrating the frontend with the backend. As mentioned before, the external subsystems are considered independent and will be used in place of stubs in order to test the interface, but won't be tested by themselves. Hence, the integrations between the backend subsystems and frontend subsystem that will be performed are:

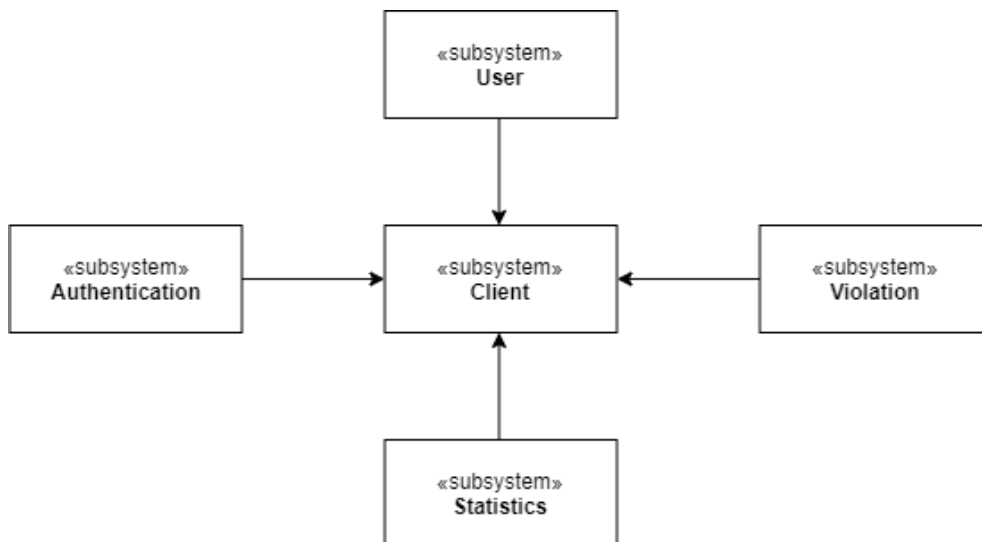


Figure 5.6: *Client subsystem integration*

- Client application and User subsystem
- Client application and Authorization subsystem
- Client application and Violation subsystem
- Client application and Statistics subsystem

At the end of the procedure explained before, the Frontend, Backend and External Subsystems are integrated and tested together.

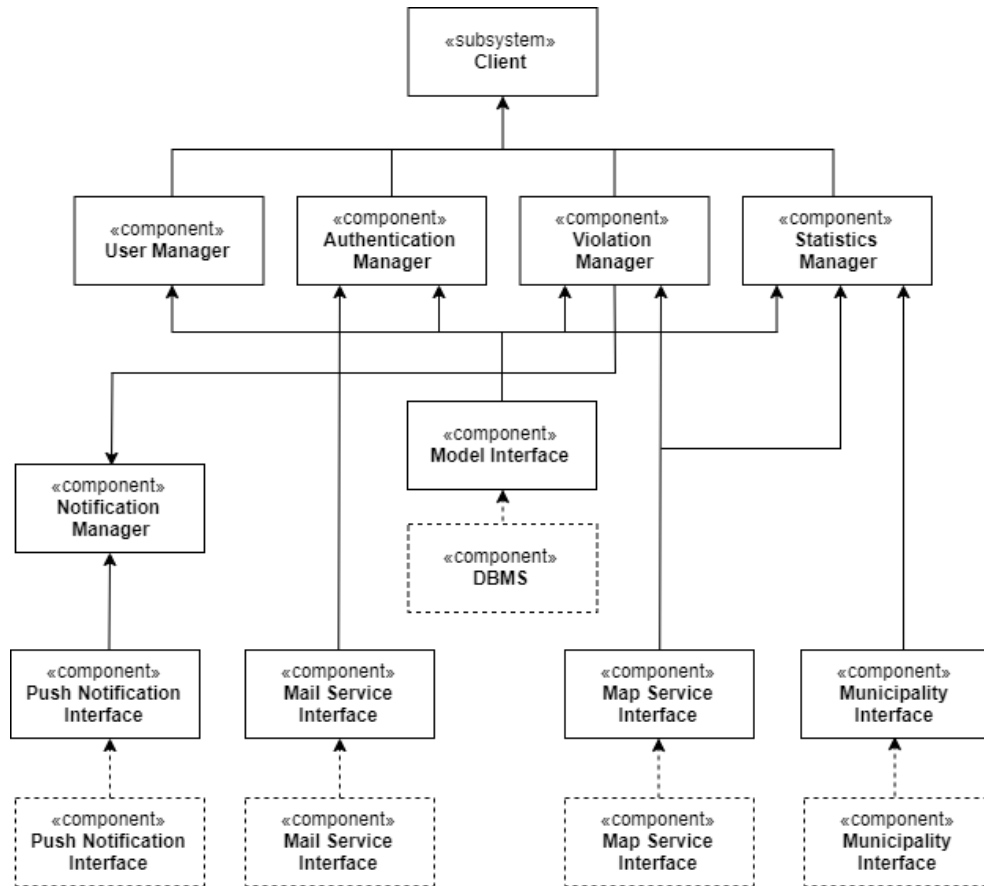


Figure 5.7: *Subsystems integration*

As a summary of what said previously, a table illustrating the main application's features and their impact on the development and testing is provided below.

Feauture	Importance for the customer	Difficulty of implementation
Sign Up & Login	low	low
Create Report	high	high
View Report History	low	medium
Delete Report	medium	low
View Pending Reports	high	medium
Accept/Refuse Report	high	medium
View Statistics	high	high

Table 5.1: *Main features and their impact*

6 Effort Spent

The effort spent from each member of the team to build the DD can be summarized with the following tables:

Samuele Meta

Task	Hours
First Meeting	1
Assignment Analysis and Workflow Decision	2
Purpose and Scope	2
Introduction Other Sections	2
Component and Deployment Views	5
Runtime View	5
Component Interfaces	4
Style, Patterns and Other Decisions	1
Mid-phase Meeting	3
UX Diagrams	3
Requirements Traceability	1
Implementation, Integration and Test Plan	4
Effort Tracking	1
Global Review	2
Total Hours	36

Stiven Metaj

Task	Hours
First Meeting	1
Git and Overleaf Setup	1
Assignment Analysis and Workflow Decision	2
Purpose and Scope	1
Introduction Other Sections	1
Component and Deployment Views	4
Runtime View	2
Component Interfaces	4
Style, Patterns and Other Decisions	5
Mid-phase Meeting	3
UX Diagrams	2
Requirements Traceability	3
Implementation, Integration and Test Plan	2
Effort Tracking	1
Global Review	4
Total Hours	36