Design Document

Abbo Giulio  
Accordi Gianmarco  
Bonetti Massiliamo

**1.Introduction**

1.C Definitions, Acronyms, Abbreviations

- Client: is intended as the final costumers, can be a user or a municipality;

Definitions  
Client: the user or the municipality

**2.Architectural Design**

**2.A Overview: High-­‐level components and their interaction**The highlevel architecture of the SafeStreets’ system is highlighted in the below Figure 1:

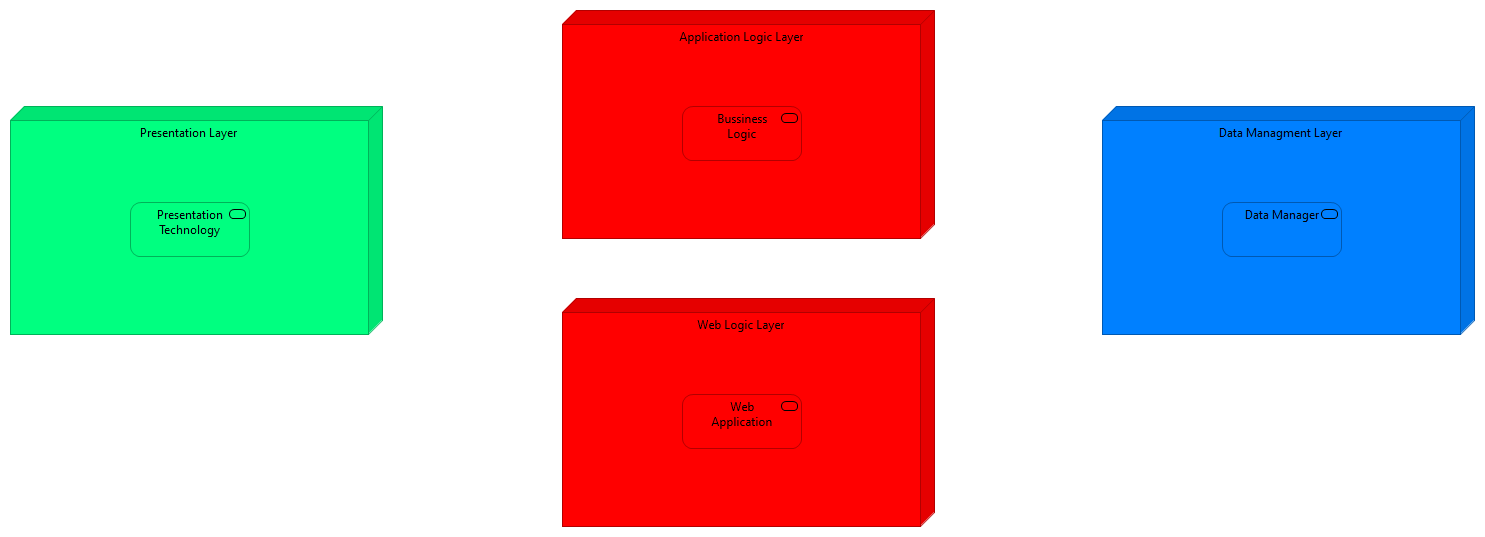


Figure 1

The defined architecture is a Four Tier architecture in which each components is briefly described below:

* *Presentation Layer* is the part that manage the visualization of the data and the possible interaction with the system in a human readability way, and also human friendly as much as possible, so this part knows how to show to the final user the results that comes from the Application or from the Web layer, also the user interact with this layer in order to interact with the system by using the application or the web access point through a browser;
* *Application Logic Layer* implements all the business logic of the Safestreets’ system, it receive all the request from the application on the users’ devices, and also the request coming from the web access, then it elaborates them, by retrieving all the information contained in the data manager, also it takes care of managing the data, by integrate them with the data provided by the municipality;
* *Data Management Layer* has the task of managing the physical allocation of the data, and to responds to: the queries that come from the application layer, and to store the data that the application layer wants to memorize.
* *Web Logic Layer* is used to respond to the web application request made by the user, but it doesn’t implement any type of business logic, so it’s only used to decouple the logic from the web visualization of the data, then all the requests made by the user through a web page are redirected to the application layer

The different levels of abstraction allow to manage different functionalities offered by the system on different machines, that need only to implement the required interface, allowing the replicate of the different machines, in order to scale in case of necessity and to be fault tolerant. The user is an aware of the different distribution of the levels: their distribution must be as possible transparent; he only needs to communicate throw a graphic interface. The data used by the system are obtained directly through the users’ registration, but it also need to retrieve this data from the municipality data manager, so periodically the system will integrate the new data obtain by the municipality with the data present inside the data manager of the Safestreets’ System.

To allow the communication to system different from the SafeStreets’ system the architecture makes use of adapter, this allows all the other component inside the system to use the same sets of operation, but the implementation of the adapter will change during time based on the implementation of the third party it refers to.  
The different level of abstractions are decoupled as much as possible, this means that they communicate through well defined interfaces, that allows us to extend some layer if necessary or to change them, for example it we want to change the data manager is sufficient to change the component on the Application Level.

**2.B Component View**  
The Safestreets’ System is composed by a component structure defined in the below Figure 2:

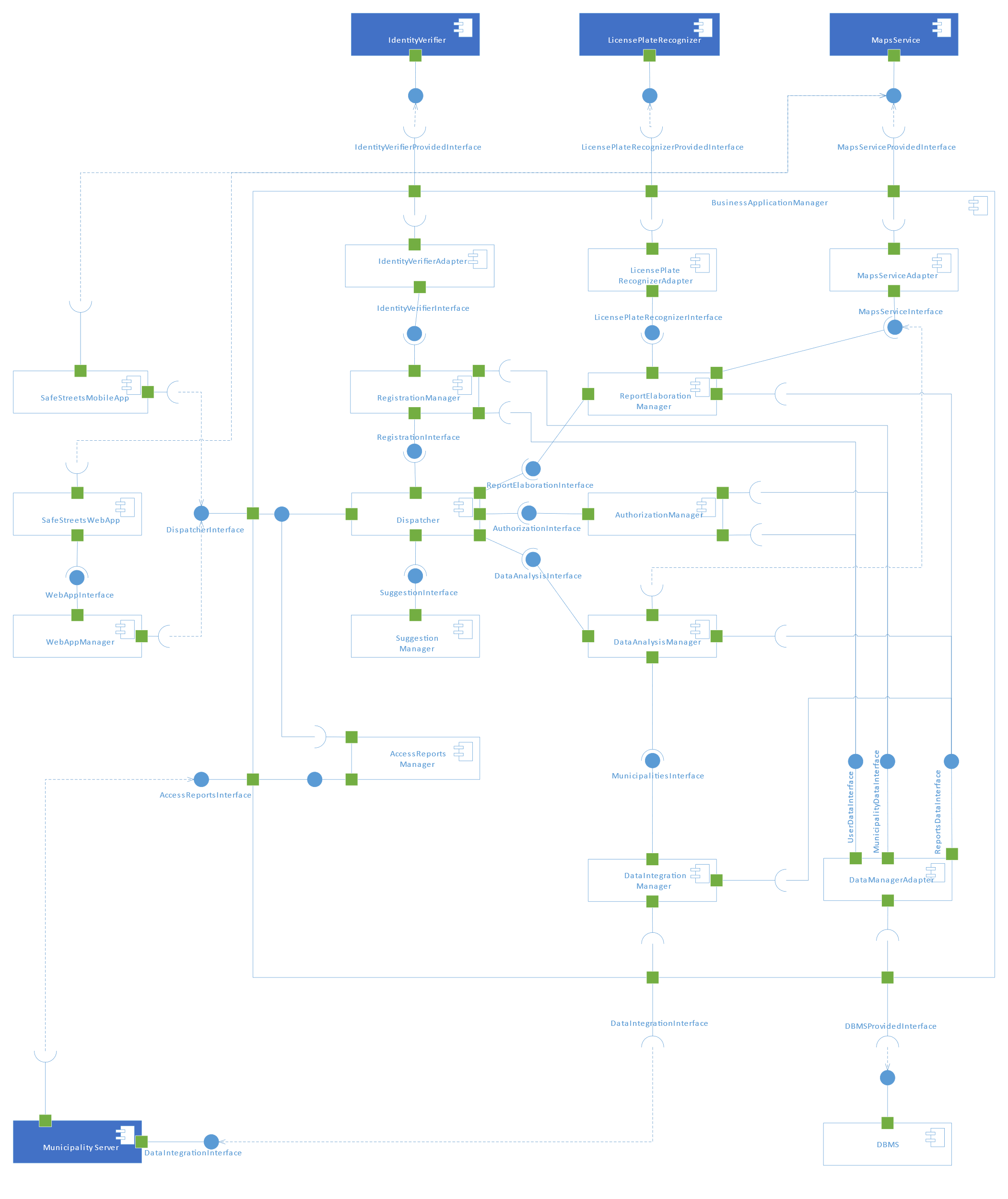


Figure 2 – Component View of the SafetStreets’ System

The figure illustrated how the components are linked between them: the figure highlight ,as already state in the high level architecture, that the module communicates among them through well-defined interface, so the figure defined which components expose an interface and which other components are interested in the same interface.  
In this chapter we will defined into details all the components that characterize our system.  
The blue coloured components indicates which component the system relies not, that are not develop with the system, in fact the system will use the services expose by some third parties, so the other components will be implemented by the software to be, if not contrary explicated.

The following list define the sets of tasks carried out by each single component that is external to the implementation to the *Business Application Manager*:

* ***SafeStreets’ Mobile App*** is the component that works on the client’s device, in fact the user interact with the system also trough a graphical interface that is similar to the one developed for the web page, this component allows the user to see how to interact with the Safestreets’ System, this means that this component sends directly to the *Business Application Manager* component all the requests made by the user, so it needs to work with the *Dispatcher Interface* of our system, in fact all the request will be dispatched by the *Dispatcher* component(in a transparent way to the user), the component also display to the user all the data that come from the *Business Application Manager*, this component needs also to have an understanding of the geographical coordination, given by the method invocation on the *Maps Service Provided Interface*;
* ***Maps Service*** this component provides to the system a geographical meaning to geographical space the system works with, the *Maps Service* implemented by a third party and not during the development of the system: the system relies on a third parties that offers this service(like for example any GIS), so an adapter will be used to connect the *Maps Service* to the system, this service needs to provide an interface that allows our system to retrieve the address of the violations, and also to get an overlay map that we can modify in order to provide a better graphical representation to the user, and in general it helps the system to have a better understanding of the geographical space;
* ***SafeStreets’ Web App*** is another method that the user can use in order to interact with the SafeStreets’ system, it is a similar style to the Mobile App, but in this case it can be used to any kind of internet browser, this in fact will contact the *Web App Manager*, through the *Web App Interface*, and it will provide the required page, then the component’s request will be forwarded to the business logic requests by the *Web App Interface*, also in order to interpret the coordinates it makes use of the method exposed by the *Maps Service Provided Interface* ;
* ***Web App Manager*** takes care of all the requests coming from the Web App of the system, this means that it has to provide all the page requested and due to the fact that it has no understanding of the business rules, he simply forward the request to the *Business Application Manager*, and when it responds the *Web App Manager* will sends back to the User’s Web App a web page with the results;
* ***Municipality Server*** is the component managed and deployed by the Municipality, this components needs to expose the *Data Integration Interface*, so as to improve the analysis of the data provided by the SafeStreets’ system, that periodically ,or when needed, will try to retrieve the latest violations from the municipality, and then it memorize them, also the municipality needs to use the *Access Reports Interface* exposed by *Access Report Manager*, in order to retrieve the violations sent by the user to the Safestreets’ system;
* ***License Plate Recognizer*** as highlighted by Figure 2 this component will not be implemented by the system, but the system will rely on a third party that offers this service, that carries out the task of recognize the plate from the report’s photos provided by the user, as already state in the RASD the response that we get from this component will be trusted by the system, so it will not be subject to other verification, and in order to communicate with the component the system implements an adapter;
* ***Identity Verifier*** the system relies on this component developed by a third party, that offers the service of verify the identity of the user, so when the user provides its credentials during the registration, the system use the *Identity Verifier Provided Interface*  in order to trust the user’s provided identity, similar to the *License Plate Recognizer* we trust the results obtained through this service;
* ***DBMS*** this component is responsible of the physical allocation and management of all the data used by the system, it exposes the *DBMS Provided Interface*, that will be used by the system to query the data and also to memorize the required data, the exposed interface will be wrapped by an adapter internally to system, in order to easily change the implementation of the query or of the technology.

Now we take a closer look to the subcomponents that implement the ***Business Application Manager***, it provides the business logic of our system:

* ***Dispatcher*** the figure makes clear that the dispatcher is used as an ingress point to the request coming from the different access method to the system, so the component will implement the *Dispatcher Interface*, used by the external component to communicate with the system, the dispatcher than is able to redirect the request based on his type, to the different interface to the other component, as better clarified in Figure 3:A close up of a map

  Description automatically generated  
  the Dispatcher need to use the interface provided: by the *Suggestions Manager* component, to complete the suggestion request made by the Municipality, by the *Registration Manager* in order to complete the registration requested by the user or by the municipality, by the *Report Elaboration Manager* that will take care of elaborate the report sent by the user, by the *Data Analysis Manager* to retrieve the analysis of the data requested by the user or by the municipality, by the *Authorization Manager* that will verify the credential of the user and his access rights, this operation is repeated for every request made by the user, the *AccessType*  returned by the *Authorization Manager* indicate which operations the requester can carry out;

Figure 3 – Interface used by the Dispatcher

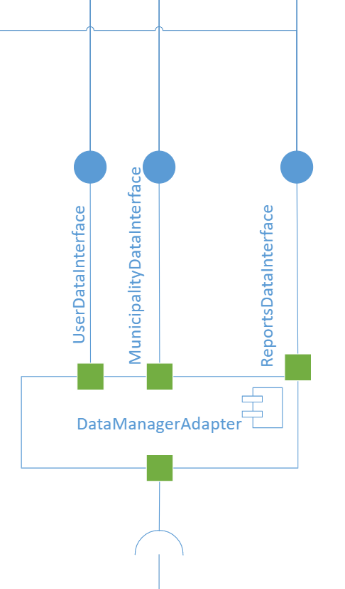
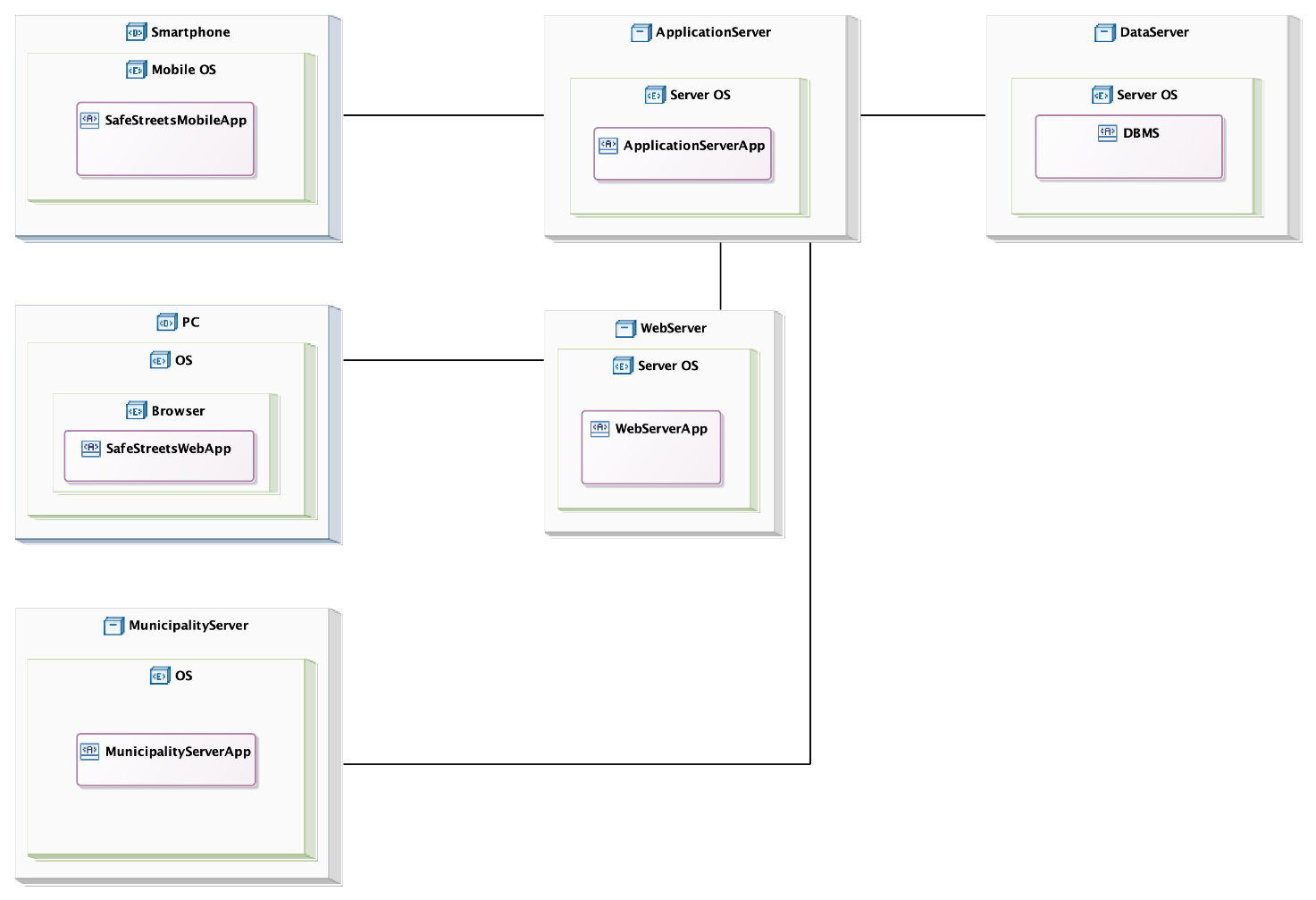
* ***Registration Manager*** this component will get from the *Dispatcher* the registration request of the user, so in this case it needs to verify the user’s identity through the service of the *Identity Verifier*, accessible with the *Identity Verifier Provided Interface*, and then if everything is fine it memorize the use in the system by using the *User Data Interface*, or it gets the registration request from the municipality, that contains the contract code of the registration, that will be verified, and then it memorize the registration by using the *Municipality Data Interface*;
* ***Identity Verifier Adapter*** as already mentioned in the description of the *Identity Verifier* component, the access to the *Identity Verifier Provided Interface* is not made directly through the system, but the access needs to pass from this adapter, that expose the *Identify Verifier Interface*, of which the implementation is used as a wrapper of the real interface, thus to allow to system to be decoupled from the implementation of the Identity Verifier: in fact if the Identity Verifier change, is sufficient to change the implementation of this component;
* ***License Plate Recognizer Adapter*** similar to the previous component also this one is an adapter, that expose the *License Plate Recognizer Interface*, that can be used by all the internal component, to retrieve the license plate from the photo provided by the user’s report, this interface is used as a wrapper for the *License Plate Recognizer Provided Interface* exposed by the *License Plate Recognizer,* this allows us to easily change it;
* ***Maps Service Adapter***  also this component is used to expose an interface, the *Maps Service Interface,* that is used as a wrapper to real interface exposed *Maps Service Provided Interface* exposed by the Maps Service, this component is used to the internal component the possibility of understand the geographical coordinates, and to manipulate the data;
* ***Report Elaboration Manager*** it receives the report sent by the user through the Dispatcher, so it implements the *Report Elaboration Interface*, when a report is received it verifies the identity of the user, and then if necessary it validate also the position of the violation or to recognize the license plate present in the report’s photo, and ,as stated in the RASD, the result retrieved from the *Maps Service*  and the *License Plate Recognizer* is trusted, also the component have the task to notify the user if the something goes wrong during the elaboration, then if something had gone right, the component use the *Report Data Interface* to memorize the data;
* ***Authorization Manager*** this component is responsible to verify the credentials provided by the user or the municipality during the login and to set the correct access rights for the municipality or the user, that will be used during its navigation in the system, so it gets the login request from the *Dispatcher* through the *Authorization Interface* that this component implements, and then he access to user or municipality data by using respectively the *User Data Interface* and the *Municipality Data Interface*, this component will be used by the *Dispatcher* every time when a request arrives in order to verify which action the requester can perform, by using the *AccessType* returned by this component, in all the internal components is implicit that every time the access right of the requester are verified*;*
* ***Data Analysis Manager*** has the task to carry out the data analysis requested by the user or by the municipality, when the request comes from the *Dispatcher*, then the component starts to aggregate the data that he can get through the *Report Data Interface* exposed by the *Data Manager Adapter,* when the result is available it is sent back to the user or the municipality, if the area of the analysis include a municipality that gives us the possibility to access to their data, then the component verifies that the system have received also the last update from the municipality, in order to produce results updated as possible, the component also use the service of the *Maps Service,* by using the method exposed by the *Maps Service Interface*;
* ***Suggestion Manager*** this component identifies the possible intervention that a municipality can carry out in order to reduce the number of report, so when a municipality asks for some, the request is sent from the *Dispatcher* to this component, and the component will receive the identity of the municipality that asks for suggestions and also the most common violations in his area, that has been previously requested by the *Dispatcher*  to the *Data Analysis Manager*, and then based on the information received, it finds out some pattern that can be improved, the suggestions found are sent back to the municipality;
* ***Access Report Manager*** it’s the component that expose the method used by the municipality to access the violations reported to the Safestreets’ system by the user, so the component expose the *Access Report Interface* used by the *Municipality;*
* ***Data Integration Manager*** is the component responsible to maintain the data memorize by the system up to date, it uses the method expose by the *Data Integration Interface* exposed by the municipality that allows the Safestretts’ system to get the data from their repositories, so the component periodically or on request asks for the latest violations, then through the method exposed by the *Report Data Interface*, it memorize the latest violations in the *DBM,* to make them visible to the future requests;
* ***Data Manager Adapter***  similar as above this component is used as an adapter for the *DBMS Provided Interface*, so in this case allows us to modify only this component if some change is made to the *DBMS*, the interface exposed by this component are highlighted in the Figure 4: the component needs to implement the interface used to the *User Data Interface* that give the possibility of access the data of the user, similarly the *Municipality Data Interface* allows to access to the data of the municipalities, and both of them inherit from the *Client Data Interface,* then the component implements also the *Report Data Interface* that allows to retrieve the report memorized in the system.

Figure 4 – Interface of the Data Manager Adapter

**2.C Deployment View**

The next picture shows how the system will be deployed on the various machines.



The architecture is four tier, there are: the machines of the user and of the municipality, the web server, the application server and the data server.

The presentation layer is split between the machines of the user and of the municipality and the web server.  
The presentation layer is split between the machines of the user/municipality and the web server.  
The application layer is on the application server and the data layer is on the data server.

The user and the municipality can use the SafeStreets’ mobile app on the smartphone and the SafeStreets’ web app on the browser of the personal computer. The municipality can have also a server in which are stored its reports and from which the application server, with the permission of the municipality, can retrieve the reports. The municipality’s server can also retrieves from the application server the reports made by the user in its city.

The web server app is present on the web server, which communicates with the SafeStreets’ web app to provide the web pages and with the application server to forward the requests of the user or of the municipality.

On the application server there is the Application server app, it communicates with the web server, the data server to store and load the data, the SafeStreets’ mobile app to answer the requests of the user and the municipality’s server.

**2.D Runtime view**

In this section will be better clarified the behaviour of the system at runtime, in terms of functions call, in order to specify the sequence of operations done to carry out a certain task.  
The diagram better shown the central role of the Dispatcher, as said before, that act as an orchestrator for the completion of all the tasks. In all the following sequence diagram, except for the login and the registration, is implicit that in order to use the services exposed by the system, the client needs to perform a login operation.

In some case the operation between the Web Sever and the Web App are represents with a method that is an abstraction of the different method that the Web Server will expose, in the case of a RESTfull architecture are POST,GET,PUT and DELETE.

**User Login**

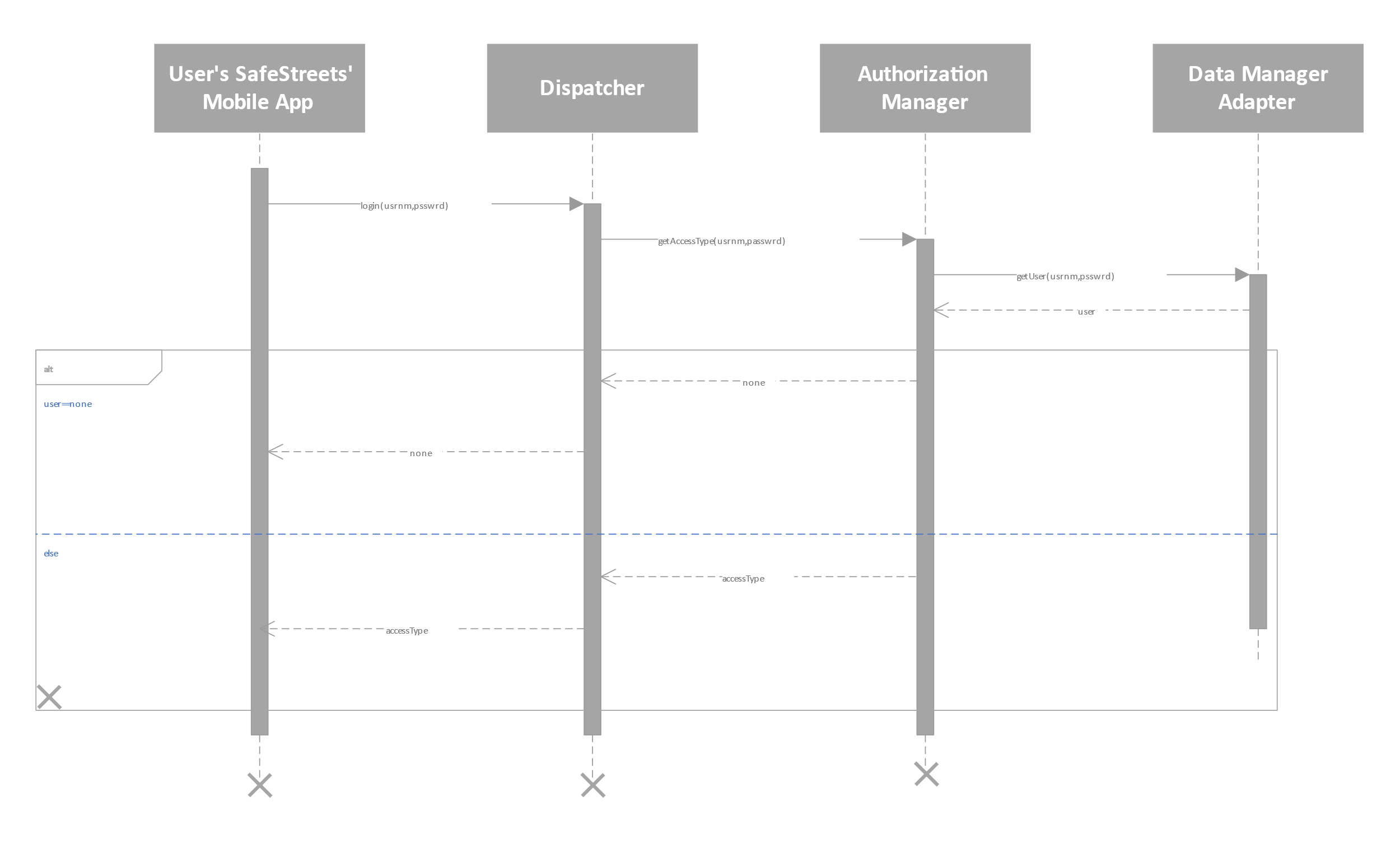
****

Figure 5- Sequence Diagram of the Login operation

This Sequence Diagram represents the orchestration of operation carried out in order to perform the login, that can be either the user or the municipality. The operation of login can be performed both from the mobile app, or from the web app through a browser, the sequence of operation is the same, but in the first part we need to insert an intermediate Web Server that forward the requests coming from the client to the Dispatcher. After the login operation the user is recognized by the system and all the services expose by the Safestreet’s system can be accessed.

**Submit Report**

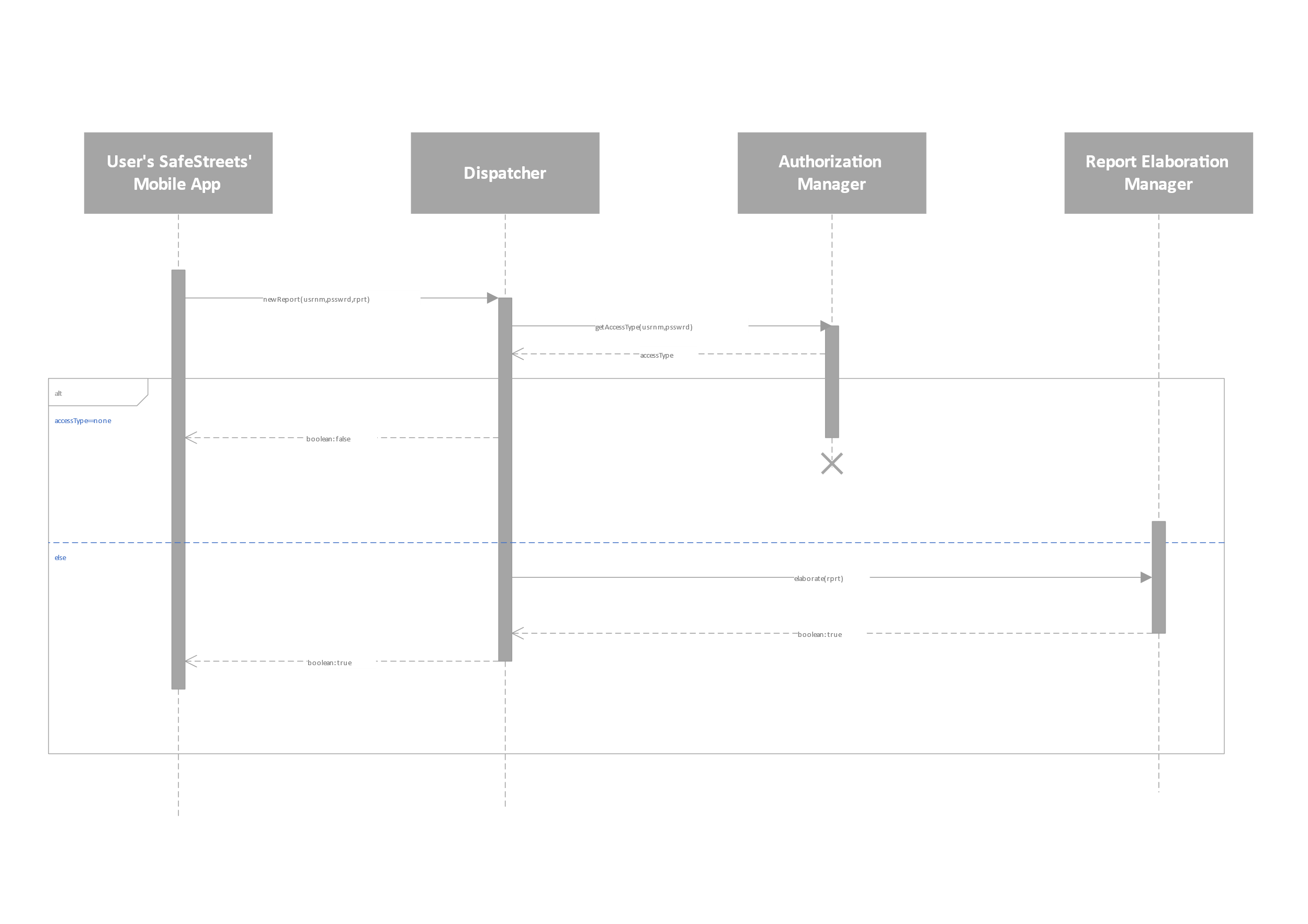


Figure 6 - Sequence Diagram of the submit report operation

This is the sequence of method invocation performed to submit a report violations, first of all the Dispatcher needs to verify the identity of the user, so it contacts the Authorization Manager that returns an *accessType* based on the credentials of the user, that needs to have performed a Login operation before. At the end there is no “X” because the computation goes on, as described in the below sequence diagram in order to elaborate the report, so the component are still working.

**Elaboration Report**

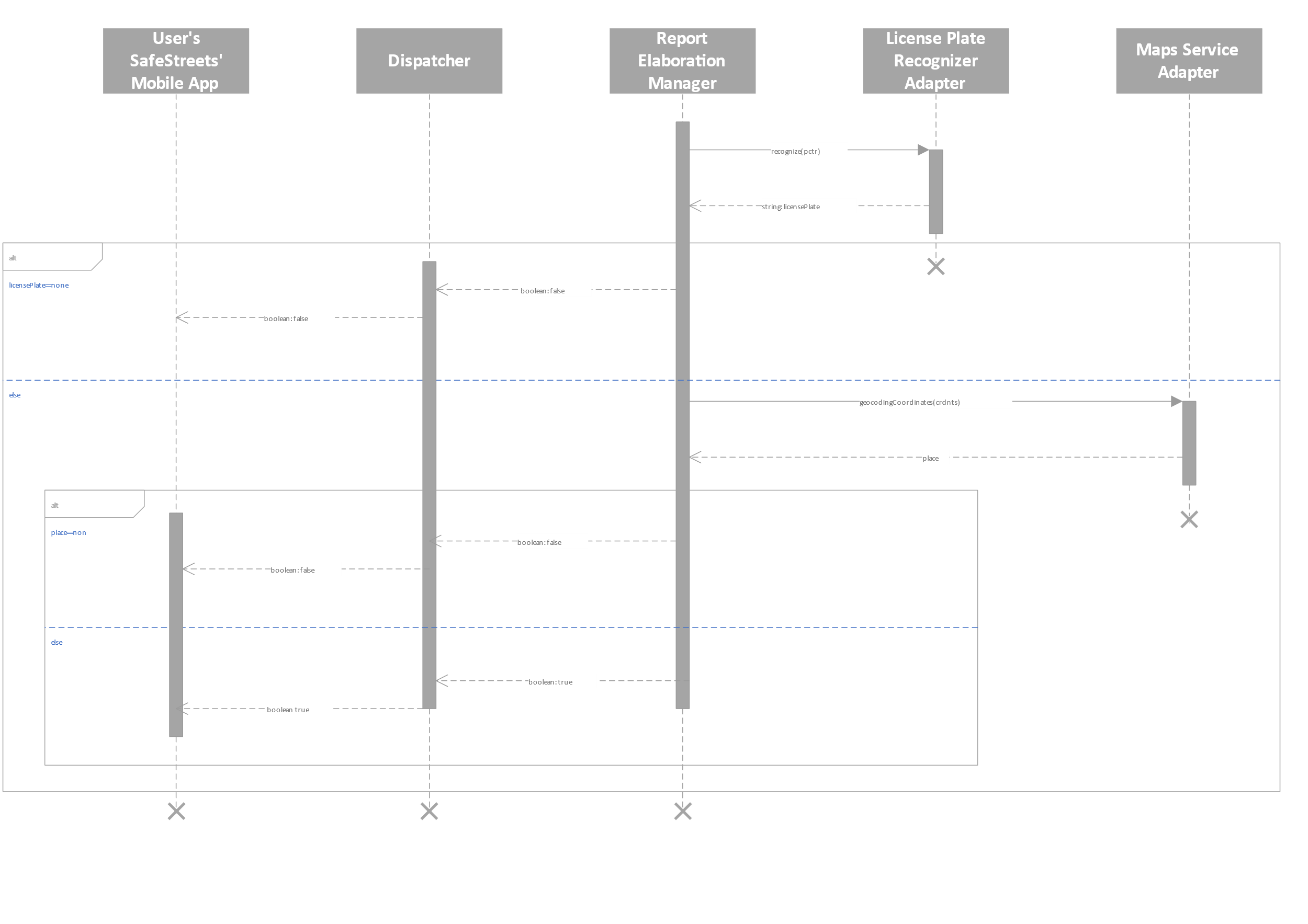
****

Figure 7 - Sequence diagram of the elaboration report operation

This sequence of operations defines how the elaboration of a report is done by the system: after the system has received a report from a user, after what happened in the previous sequence diagram, it tries to verify the report by looking at the license plate and the position of the violations, in the second case the system will maintain both the coordinates and the address of the position returned by the Maps Service in the *Place* data structure, so the system call the *geocodingCoordinates* method if the user sends us the coordinates and we need to obtain the *Place,* otherwise if the system receives the address then it calls the method *geocodingString*. This is done to help the work of the Data Analysis Manager. In this sequence of operation e refers to the assumption made in the RASD: the Reponses returned by the License Plate Recognizer and the Maps Service are trusted, so they are not father subject to validation.

**Available Statistics**

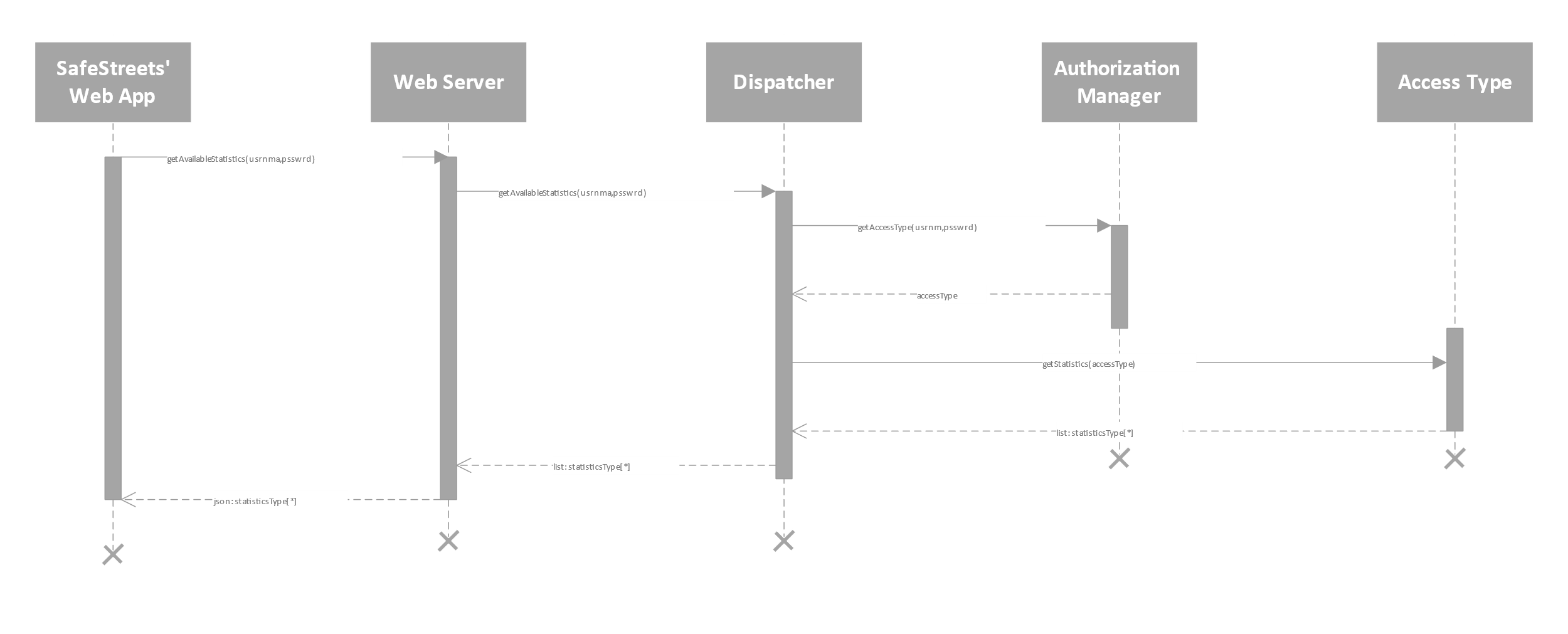
****

Figure 8 - Sequence Diagram of the get available statistics operation

In this case the sequence diagram highlight how the get statistics operation is performed: the user or the municipality(they both use the Web App or the Mobile App) wants to get some statistics from the system, so it asks to the system which are the available statistics according to the *accessType* returned by the Authorization Manager, the *accessType* will be an object that contains an enumeration of different access type, an each users will be associated with an *accessType* of type User, while the municipalities to an *accessType* of type Municipality. Also, each *accessType* is associated with a set of operation that can be performed with it. So what happens is that: based on the returned *accessType* from the Authorization Manager, the system find out what are the statistics, in this case, that the user can gets.   
This sequence of operation returns only the available statistics to the user or municipality, that the sequence of operation done to get the result of a statistics is defined in the following diagram.  
The *getAvailableStatistics* method is an abstraction of the invocation of the method on the Web Server in order to perform this operation, for example for the method invoked on the Web Server in a RESTfull architecture.

**Request Statistics**

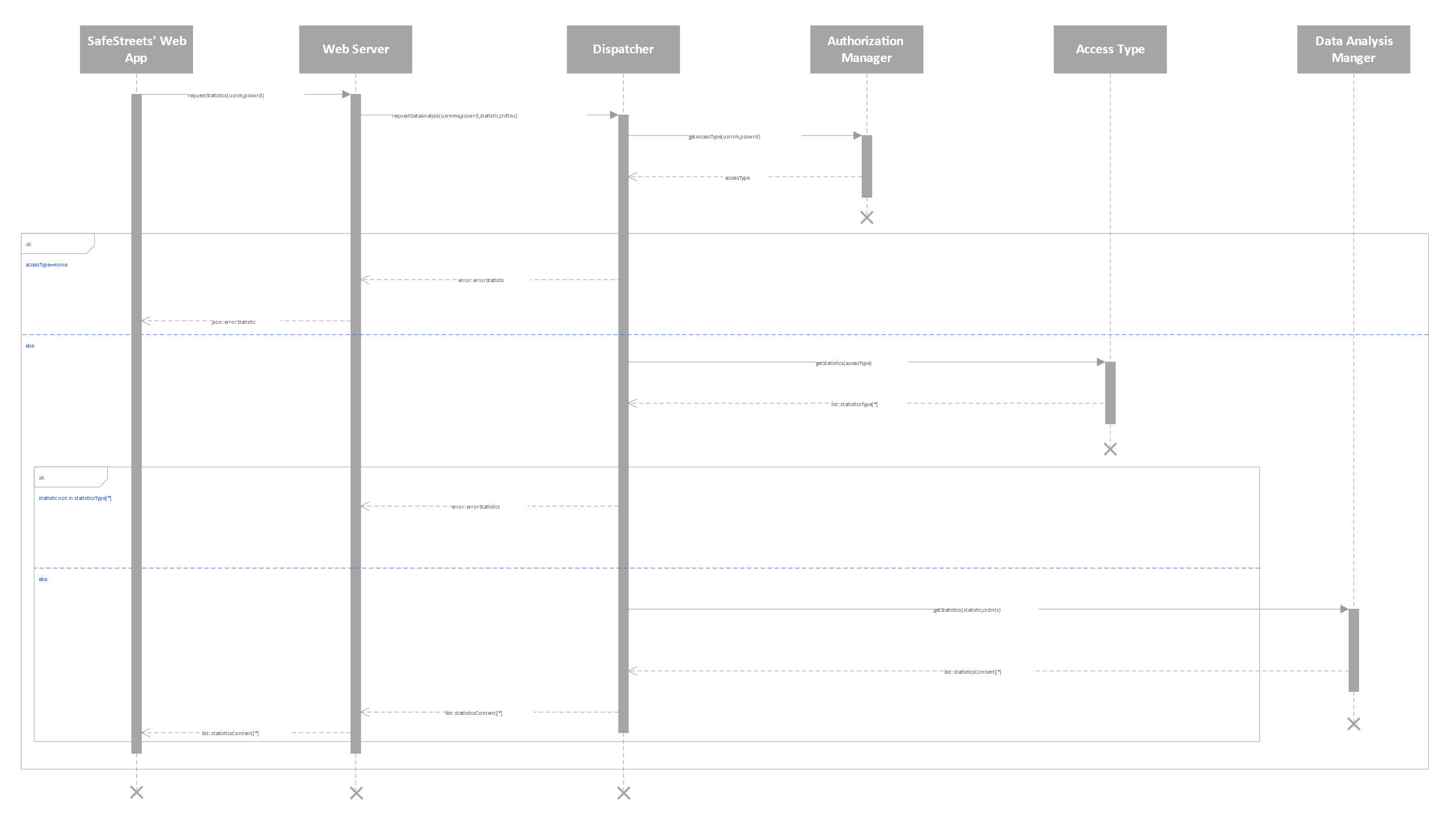
****

Figure 9 - Sequence Diagram of the request statistics operation

The previous diagram describes how the user or the municipality (they can use both the Mobile app and the Web App, in this case we need to add the Web Server as an intermediate) can get some statistics. In order to get the statistics, the client needs to have done the login, then he can access the services of the system. When the request is received, the dispatcher verifies that the client can gets the required statistics (similar as described in the previous sequence diagram), then the ­­­Data Analysis Manager is used to get the required statistics.  
The *requestStatistics* method invocation on the Web Server is an abstraction of the method invocation performed by the user on the Web Server to get this operation, for example for the method invoked on the Web Server in a RESTfull architecture.

**Municipality Registration**

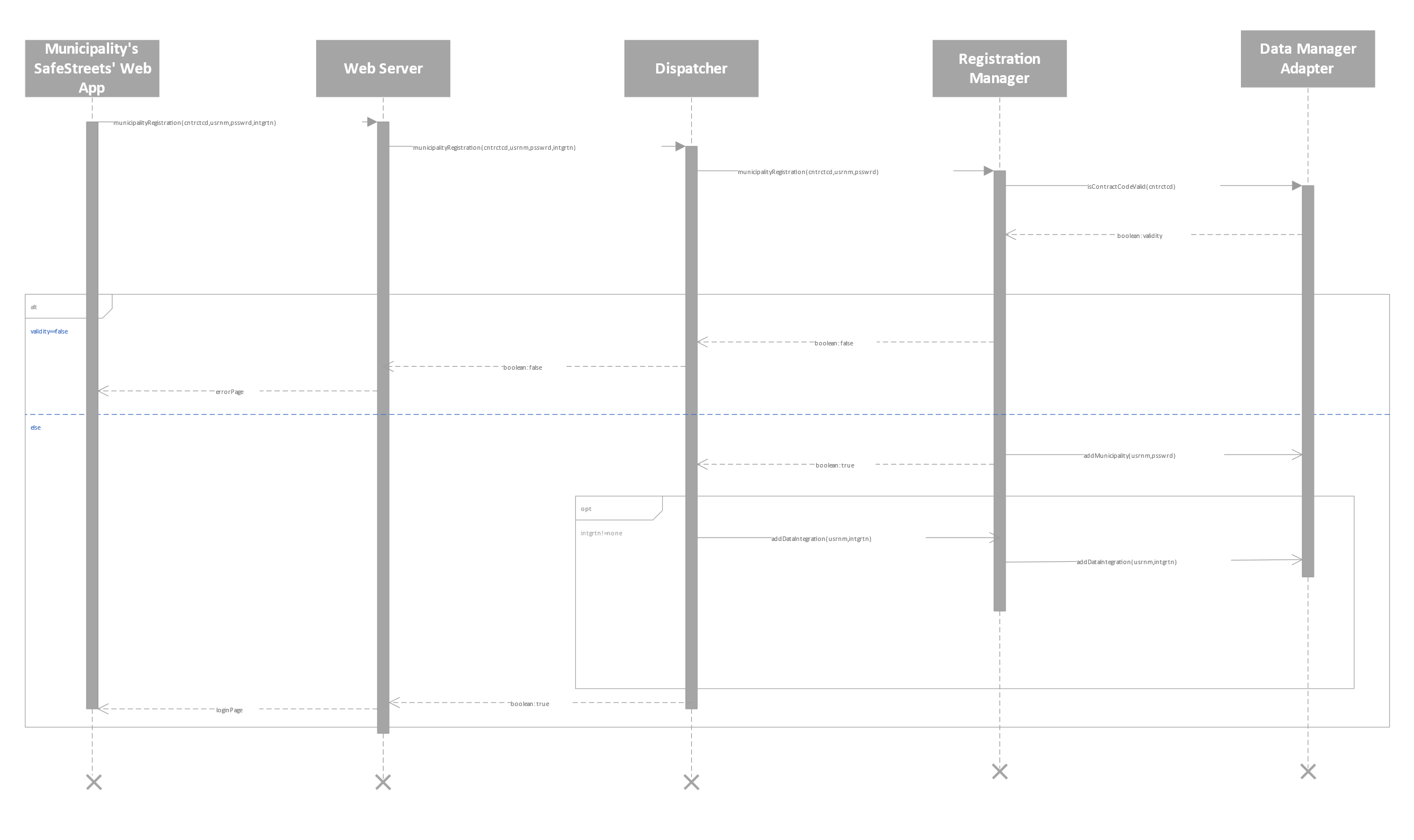


Figure 10 - Sequence Diagram representing the registration of a municipality

This sequence diagram describes how the municipality can perform a registration. As stated in the RASD, to perform a registration the municipality needs to previously stipulate a contract with SafeStreets’, then a contract code with the information about the municipality are memorized in the system, and this information will be used during the registration in order to authenticate the municipality and to retrieve the useful information about them (for example about their jurisdiction area).  
The invocation of the method m*unicipalityRegistration* on the Web Server is an abstraction of the method called on the Web Server to make the registration request, for example for the method invoked on the Web Server in a RESTfull architecture.

**User Registration**

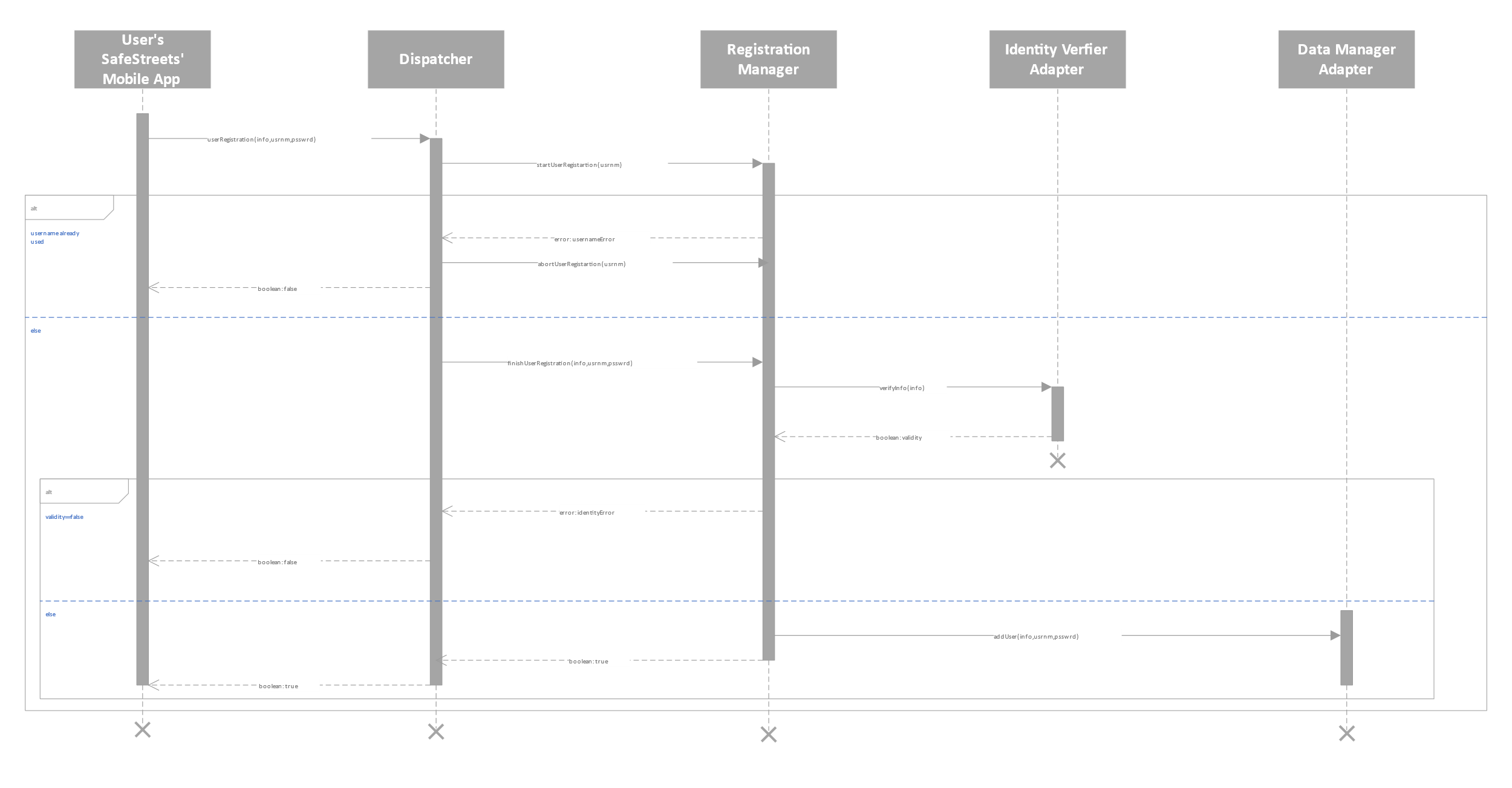
****

Figure 11 - Sequence Diagram about the user registration operation

The user’s registration operation is performed as indicated in the upper sequence diagram: the registration manager tries to start the registration of the user with *startUserRegistration*  and then to finish it with the invocation of *finishUserRegistration*, that can be interrupted at any time with the invocation of the method *abortUserRegistration*. The operation can be interrupted by some errors, for example if the username has been already used, or if the identity verifier cannot authenticate the user.

**Municipality Report Request**

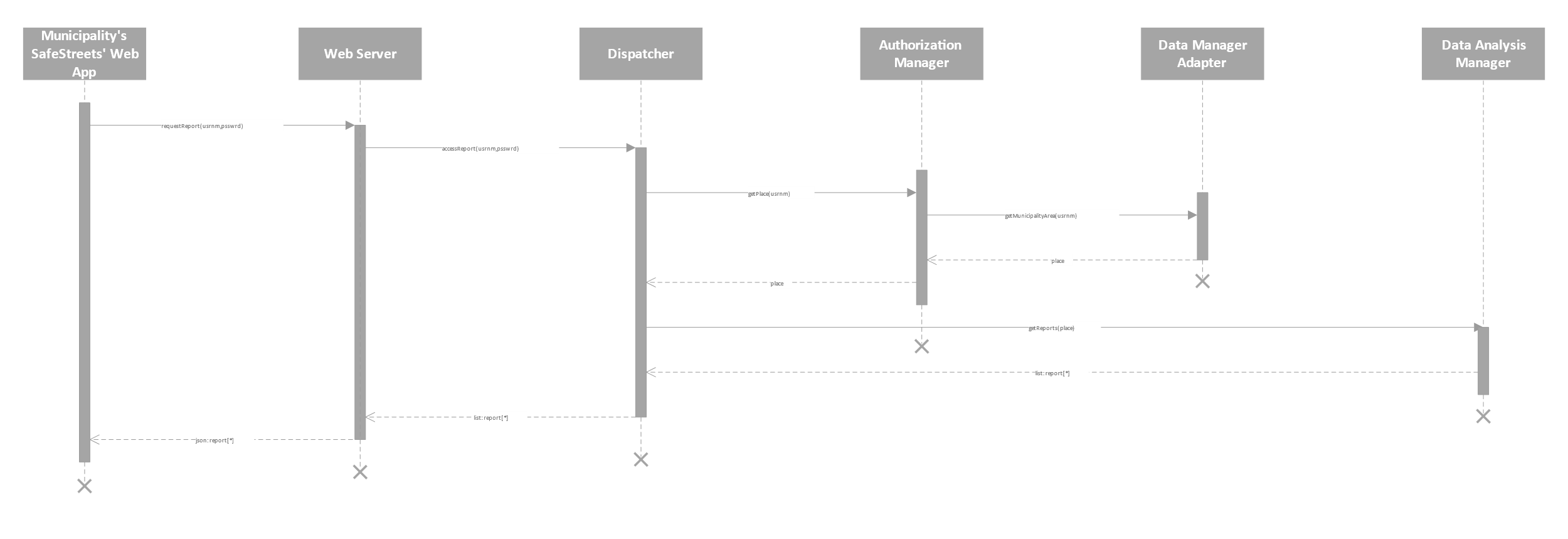
****

Figure 12 - Sequence Diagram about the reports request of the municipality

In this sequence diagram is better clarified how the municipality can access to the report sent by the user to the SafeStreets’ system. In order to get them the municipality can for example use the Web App, and when the system gets this request then the access right are verified by the Authorization Manager, that returns the competence are of the municipality, that will be used by the Data Analysis Manager, to return the reports for the municipality.  
The invocation of the method *requestReport* on the Web Server is an abstraction of the various method invocation performed on the Web Sever in order to get the latest report, for example for the method invoked on the Web Server in a RESTfull architecture.

**Data Integration**

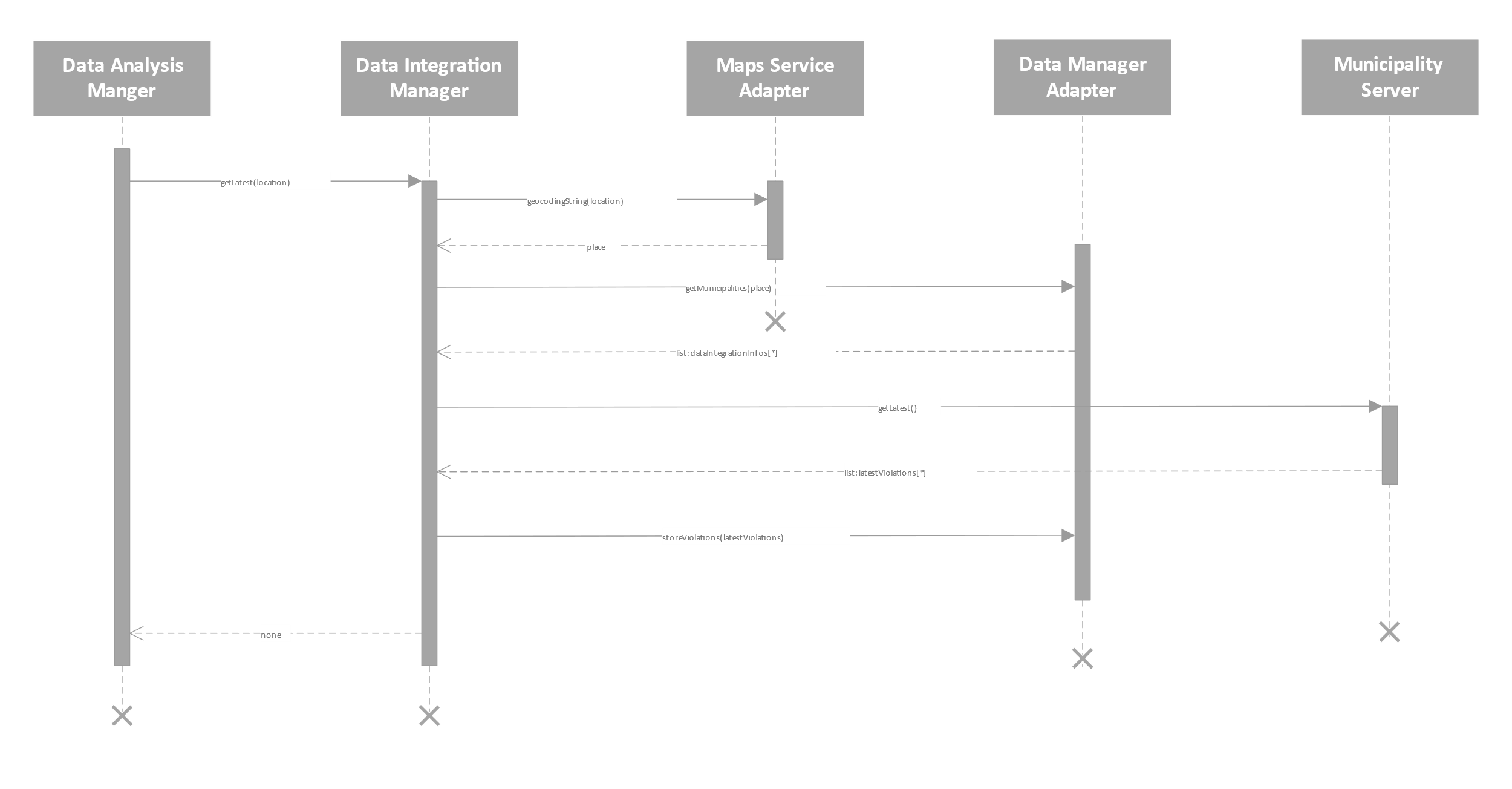


Figure 13 - Sequence Diagram about the integration of the data with the system

The diagram shows how the data are integrated inside the SafeStreets’ system: periodically (for example when the system is not overloaded with work to do) or when a request comes from the Data Analysis Manager (as in this case, when the work of the Data Integration Manager is triggered by the analysis request on the location passed by the Data Analysis Manager), the Data Integration Manager starts to merge the information memorized by the municipality that offers to the SafeStreets’ system the possibility of get the violations occurred on their competence area. In this case is the Data Analysis Manager that want the data brought up to date to returns a statistics about the given place, so the Data Integration Manager asks to the municipality that is included in the place, to return the latest violations.

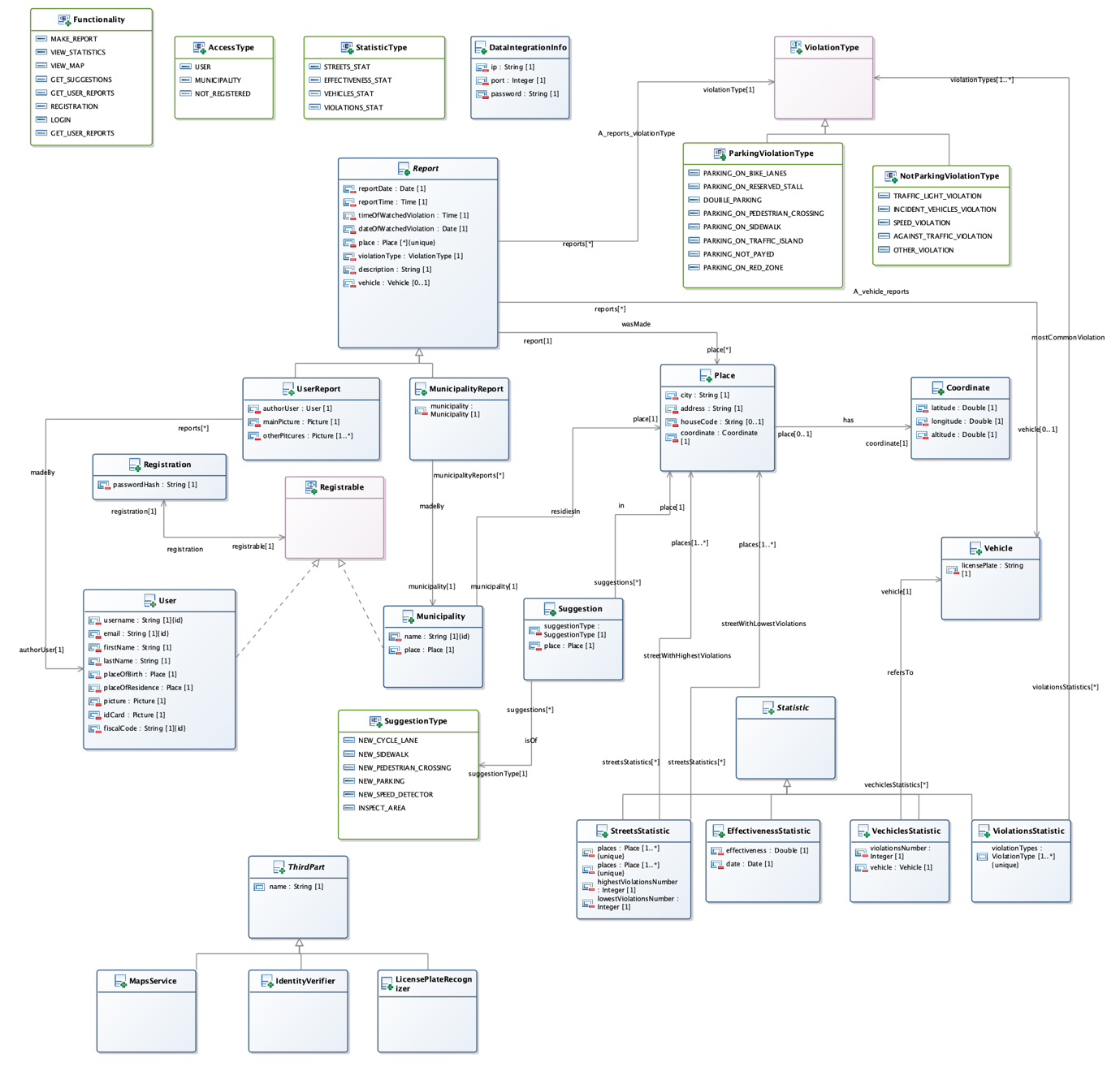
**2.E Component interfaces**Immagine che contiene screenshot

Descrizione generata automaticamente

Here the main interfaces used in the component diagram are presented. The diagram also contains some other interfaces – the ones with the tag “Provided” – that are provided by external parties, also missing is the interface through which the web app fetches the code from the web app manager. These interfaces depend heavily on the implementation and are omitted here.

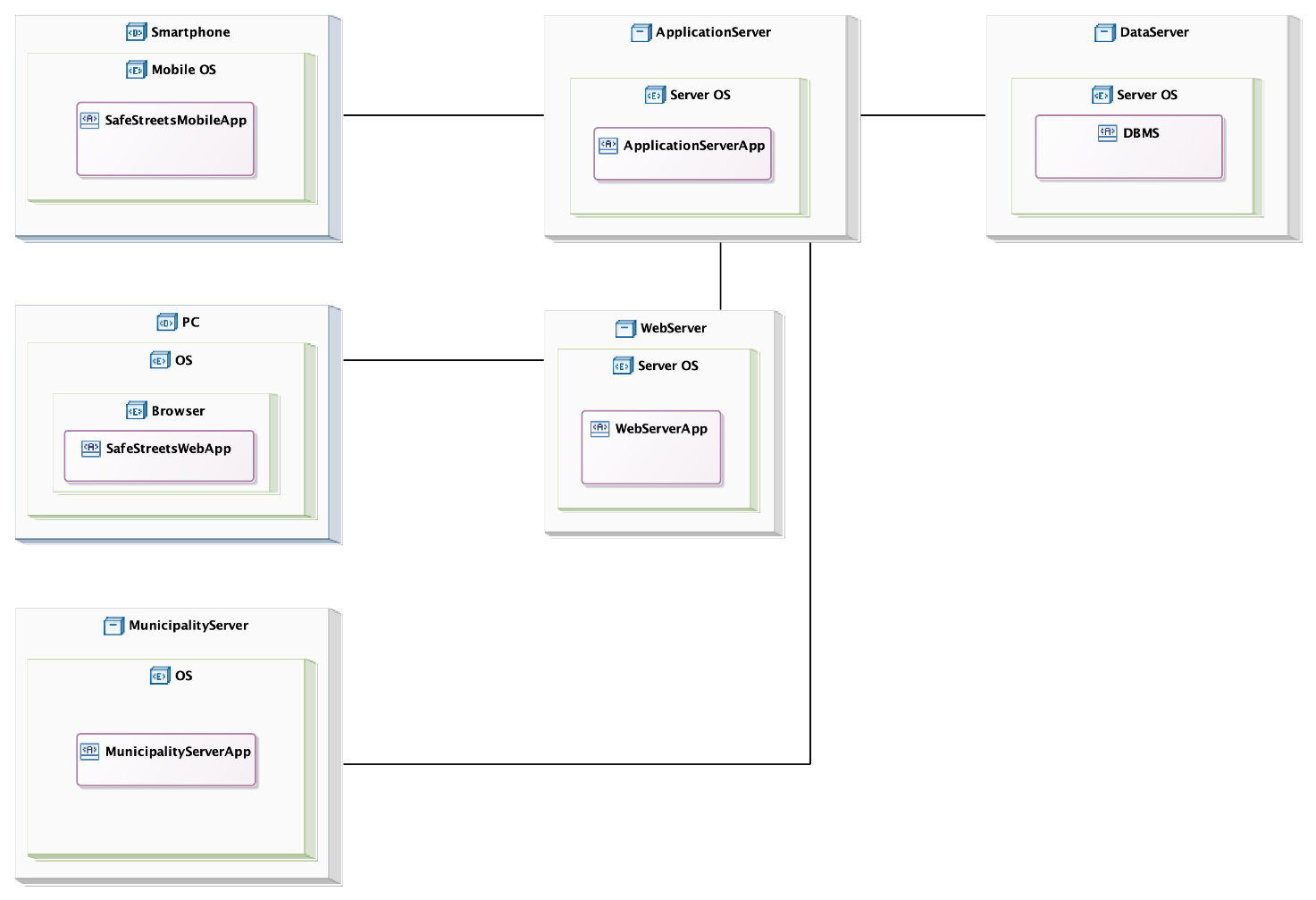
In the image, the first row contains the interfaces that interact somewhat directly with the outside; on the second line the interfaces that allow the internal workings are shown and in the third row there are the interfaces that allow the communication with the data layer. Among these it is worth pointing out that UserDataInterface and MunicipalityDataInterface extend ClientsDataInterface, this was done to clarify that they have some methods in common and that these methods do not depend on the access type.

In the next picture there is the class diagram of the model of the application server, it contains the most important data structures that will be used in the implementation phase.



**2.F Selected architectural styles and pattern**

**Four tier architecture**

As described in the deployment view the architecture is four tier. The next picture resumes the architecture.

There are: the machines of the user and of the municipality, the web server, the application server and the data server.

The presentation layer is split between the machines of the user and of the municipality and the web server.

The application layer is on the application server and the data layer is on the data server.

This architecture allows a separation of the logical layers between the tiers without mixing some logical layers in a single tier. This simplifies the interactions between the tiers, it allows to create redundancy in the servers to improve the performance and the security.

It is possible also to put a firewall between the application server and the web server, given that the web server is subject to frequent attacks.

**Thin Client**

The application layer is on the application server and on the clients there is only the presentation layer. This architecture allows to have a light client which presents only the information elaborated by the application server.

**RESTful Architecture**

The RESTful architecture is used in the communications between the clients, the application server and the web server.

The properties of the Restful architecture are the following:

**Client-server architecture**

The client asks the server for a service and the server answers the questions done by the clients.  
There are two clients: the SafeStreets’ mobile app and the SafeStreets’ web app. There are also two servers: the web server and the application server.

**Statelessness**

The application server doesn’t store the context of the client between its requests. All the requests from the client contain all the information to provide the service.

**Layered system**

The client doesn’t know whether it is connected directly to the end server or to an intermediary server. Future intermediary servers can improve the scalability of the system.

**Uniform interface**

The server’s resources are identified by using the Uniform Resource Identifier.

**Cacheability**

The client stores a cache with the data that refers to its state.

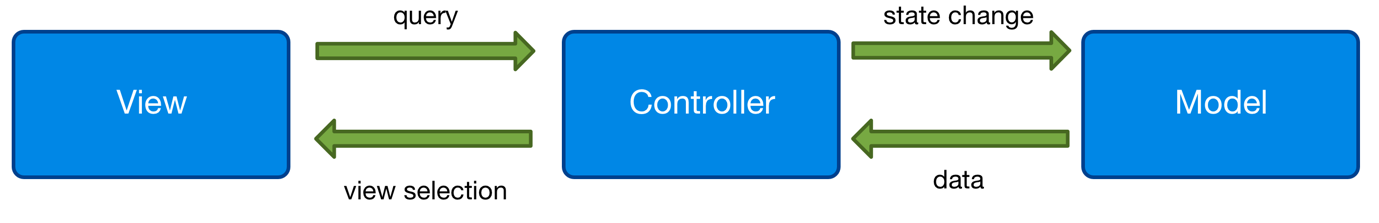
The REST architectural has been chosen for the following advantages:

* performance in component interactions;
* scalability, it allows the opportunity to increase the system when the number of clients will increase significantly.
* simplicity;
* modifiability of components to meet new requirements;
* reliability in the presence of failures within components.

To exchange the data between the clients, the application server and the web server there will be used the protocol HTTPS, which permits also to transmit securely the data.  
The data will be transferred with files of the format JSON. It is one of the most common standard of file format with the purpose to transmit the file.

**MVC**

The Model-View-Controller design pattern will be used. The next picture shows how the pattern is adapted for the system.



The view, through the gestures of the user or of the municipality, queries the controller. The controller elaborates the request by retrieving the data from the model and by updating the data in model. Finally, the controller selects the view with the new data to show to the user.

The view is present in the SafeStreets’ mobile app and in the SafeStreeets’ web app.  
The controller is implemented in the application server.  
The model is present in the database.

**Adapter design pattern**

There are four components that act as adapters: IdentityVerifierAdapter, LicensePlateRecognizerAdapter, MapsServiceAdapter and DataManagerAdapter.

These components interact with a system developed by a third part, that provides a fixed interface. The adapters allow to work in the application server without needed to worry about the interfaces of the third parties. If there is a need to change an external system, only the corresponding adapter must be changed without touching the other components.

**2.G Other design decisions**

**Relational Database**

The data will be stored in a relational database, which is a standard model very reliable. It offers a lot of good properties, that are summarized in the ACID properties: atomicity, consistency, isolation and durability.

The DBMSs of the relational database are provided by a lot of organizations, among which we have chosen the **MySQL** DBMS.

MySQL is an open source DBMS, it is present from the 1995, it is adopted by a lot of organizations and private users and it is currently supported.

**Google Maps**

For the Maps Service we have chosen the well-knows Google Maps. It is a service of a very good quality and it is used every day by a lot of people. It has a lot of experience and it is supported by one the largest companies on the world, Google.

Furthermore, Google Maps is easy to use and it has a good user interface, which fits with the design of the SafeStreets’ client applications.

**4. Requirements traceability**

This chapter explains how the functional and the non-functional requirements can be fulfilled through the components.

* **R1**: The reports about the violations are correctly stored.
  + - **ReportElaborationManager**, it allows to compile a report about a violation and it forwards the report to the DataManagerAdapter.
    - **DataManagerAdapter**, it interacts with the DBMS by passing the report.
    - **DBMS**, it saves the report in the database.
* **R2**: The user can view the statistics calculated by the System with some exceptions
  + R2.A: The vehicles that have committed the highest number of violations.
    - **DataAnalysisManager**, it calculates the statistics asked by the user.
    - **AuthorizationManager**, it indicates the type of access of the current actor that is interacting with the server.
* **R3**: The Municipality can access only the data of the violations of its competence area.
  + - **AccessReportsManager**, it offers an interface towards the municipality to retrieve the reports done by the users.
    - **DataMiningManager**, it retrives the user reports from the database.
    - **AuthorizationManager**, it allows to access only to the reports that refer to the city of the Municipality.
* **R4**: Violations registered by the Municipality can be retrieved by the system.
  + - **DataIntegrationManager**, it allows to retrieve the reports done by the Municipality.
* **R5**: The system must avoid the manipulation of the violations.
  + - **DBMS**, it ensures the persistency of the data stored in the database, so the persistency of the reports done by the users.
    - **DataManagerAdapter**, it is the only way to modify the data stored in the database and it doesn’t offer any interface to modify the them.
* **R6**: The system must be able to retrieve the position from the user or from the GPS
  + - **SafeStreetsMobileApp** or **SafeStreetsWebApp**, it offers a user interface to insert the position of the user and if it is not inserted, the SafeStreetsMobileApp (or SafeStreetsWebApp) asks the GPS the position of the mobile phone.
* **R7**: Only the Municipality can access the submitted parking violation of its competence area
  + - **AuthorizationManager**, it allows to access only to the parking violations that refer to the city of the Municipality.
    - **AccessReportsManager**, it offers an interface towards the municipality to retrieve the reports done by the users.
    - **DataMiningManager**, it retrives the user reports from the database.
* **R8**: The system must allow the User to take a picture or to select one from the device.
  + - **SafeStreetsMobileApp** or **SafeStreetsWebApp**, it asks the user if he wants to select one picture from the device or to take a picture, in these cases the SafeStreetsMobileApp (or SafeStreetsWebApp) calls the functionalities of the operating system.
* **R9**: The system accepts reports from the User.
  + - **AuthorizationManager**, it allows only the users to make a report.
    - **ReportElaborationManager**, it allows to compile a report about a violation and it forwards the report to the DataManagerAdapter.
    - **DataManagerAdapter**, it interacts with the DBMS by passing the report.
    - **DBMS**, it saves the report in the database.
* **R10**: The System must calculate some statistics
  + R10.A: The system must calculate the streets with the highest and the lowest number of violations.
  + R10.B: The system must calculate the effectiveness of the service.
  + R10.C: The system must calculate the vehicles (identified by the traffic plate) that have committed the highest number of violations.
  + R10.D: The system must calculate the most common violations of a given area
    - **DataAnalysisManager**, it calculates the statistics asked by the user or by the municipality
* **R11**: The municipality can view all the statistics calculated by the system.
  + - **DataAnalysisManager**, it calculates the statistics asked by the municipality.
    - **AuthorizationManager**, it allows the municipality to view all the statistics.
* **R12**: The system must suggest interventions to the Municipality.
  + R12.A: Inspect an area
  + R12.B: New cycle lane
  + R12.C: New sidewalk
  + R12.D: New pedestrian crossing
  + R12.E: New parking
  + R12.F: New speed detector
    - **SuggestionManager**, it calculates some possible interventions to do in the city of the municipality.
    - **AuthorizationManager**, it allows the municipality to view the suggested interventions.
* **R13**: The system accepts only reports with a valid plate number and position.
  + - **ReportElaborationManager**, it accepts the report done by the user only if the LicensePlateRecognizer returns a plate number and if the MapsService can calculate the right position of the violation.
    - **MapsServiceAdapter**, it allows to communicate with the Maps Service.
    - **LicensePlateRecognizerAdapter**, it allows to communicate with the Plate Recognizer Service.
* **R14**: The system must allow the user to perform the registration and the login.
  + - **AuthorizationManager**, it allows the user to make the login.
    - **RegistrationManager**, it allows the user to make the registration.
* **R15**: The system must allow the municipality to perform the registration and the login.
  + - **AuthorizationManager**, it allows the municipality to make the login.
    - **RegistrationManager**, it allows the municipality to make the registration.
* **R16**: The system must ask the User the non-mandatory attributes of the report.
  + - **SafeStreetsMobileApp** or **SafeStreetsWebApp**, it asks the non-mandatory attributes of the report.
* **R17**: The system must communicate with the Identity Verifier.
  + - **IdentityVerifierAdapter**, it allows to communicate with the Identity Verifier.
* **R18**: The system must communicate with the Plate Recognizer Service.
  + - **LicensePlateRecognizerAdapter**, it allows to communicate with the Plate Recognizer Service.
* **R19**: The System must communicate with the Maps Service.
  + - **MapsServiceAdapter**, it allows to communicate with the Maps Service.
* **Performance requirements**
  + - Dispatcher and **AccessReportsManager**, they offer some interfaces to communicate with the application server at any time.
* **Reliability**
  + - All the modules will be tested properly in order to be reliable.
* **Availability**
  + - All the modules are designed to be ready at any time, the Dispatcher and the **AccessReportsManager** allow to access the application server whenever needed.
* **Security**
  + - **DBMS**, it ensures the persistency of the data stored in the database, so the persistency of the reports done by the users.
    - **AuthorizationManager**, it manages the accesses to the server and it authorizes. according to certain rules, who wants to access the server.
* **Maintainability**
  + - All the modules are designed to be expanded in the future and to be readable with the comments and the documentation.
* **Portability**
  + - The various adapters allow to interact with the systems of the third parts by changing only a few modules.

In the mapping the non-functional requirements are mapped to the most important requirements, but they depend strongly on the entire system, on the hardware and on the technologies that support the system.

In all the requirements the component Dispatcher is used, because it redirects the requests of the client to the right component.

**5. IMPLEMENTATION, INTEGRATION AND TEST PLAN**

**RoadMap**



Figure 14 - Project Roadmap

The project Roadmap is highlighted in the above picture. The overall project will take approximative a month of work, and the project is divided in three main tasks: Frontend, Application Server, Data Management, Integration and Test. This tasks can be carried out almost in parallel with synchronization on the interface method previously defined. In this way also the Test can be made in the same way.