POLITECNICO
MILANO 1863

Computer science and engineering
Software engineering 2 - Project 2019/2020

SafeStreets

Design Document

Deliverable:	DD
Title:	Design Document
Authors:	Maldini Pietro , Paone Angelo
Version:	1.0
Date:	9-December-2019
Download page:	https://github.com/pm390/MaldiniPaone
Copyright:	Copyright © 2019, Maldini Pietro, Paone Angelo – All rights reserved

Contents

1 Introduction

1.1 Purpose

The purpose of this document is going more in the technical details than the RASD concerning SafeStreets application.
The Design Document gives more details about the design giving guidelines over the overall architecture of the system. This document aims to identify the core design choices for developing the system:

- The high level architecture
- The components and their interfaces
- The design patterns
- The Interaction between the components
- Planning for implementation, integration and testing of the system

A mapping of the requirements to the architecture's components is also given in the underneath chapters.

1.2 Scope

1.2.1 Description of the given problem

SafeStreets is a crowd-sourced application whose intention is to notify the authorities when traffic violations occur. Citizens, thanks to the system, will be able to send information about violations to the authorities who will take actions against them. In this way, the service provided by the authorities can be improved because they will receive notifications through the app. The sources of notifications are the Citizens who take photos of violations and send them to the authorities through the application. The information provided by users are integrated with other suitable information and are stored by the service. The system also runs an algorithm to read the license plate of the vehicle in the photos. All collected data can be seen by Citizens and authorities to find which streets are the safest. Users can have different levels of visibility: authorities must be able to know the license plates of vehicles in the photos, while normal users can only see data in the form of statistics. Moreover, data are sent to the municipal district so that important information can be extracted, and the system makes statistics and suggestion in order to make decisions to improve the safety of the area. Finally, the system will have to be easy to use, reliable and highly scalable to fit perfectly with the mutable context in which it will be used.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- Violation: parking violations which can be notified by Citizens to authorities
- Report: Notification sent by Citizens to the system
- Mapping System: external software that provides maps and directions to reach the position of a violation
- Licence plate Recognition Algorithm: calculation process that identifies the alphanumeric number on license plate
- Spam: a series of messages that are undesired
- App: application software

- Blocked: means that the account is banned for a given period
- Metadata: data about a violation. Position , date, time and the username of Citizen.
- Assignment : Work Request for authorities generated upon the receiving of a notification made by Citizens.

1.3.2 Acronyms

- RASD: Requirement Analysis and Specification Document.
- DD: Design Document
- API: Application Programming Interface
- GPS: Global positioning system
- HTTP: HyperText Transfer Protocol
- HTTPS: HyperText Transfer Protocol over Secure Socket Layers
- UML: Unified Modeling Language
- JSON: JavaScript Object Notation
- UI: User Iterface
- SQL:Structured Query Language

1.3.3 Abbreviations

Rn : n-th requirement.

1.4 Revision History

- DDv1.0 delivered on 9/12/2019
- DDv2.0 delivered on 15/12/2019 :
  - Updated Mapping of requirements with Components of the server.
  - Other minor changes: clarified some parts which were not clear. Fixed grammar error.

1.5 Reference Documents

- RASD version 1
- Specification Document: “Assignments AA 2019-2020.pdf”.
- IEEE Standard for Information Technology—Systems Design—Software Design Descriptions

1.6 Document Structure

This chapter debates about contents and structure of DD, indeed this document is divided in seven different sections:

1. The Introduction provides a general appearance of the systems defining which are the goals to reach.
2. Architectural Design: This chapter illustrates the main components of the system and the relationships between them, supplying information about their workflow and deployment. This part of the document also focuses on the main architectural styles and patterns.
3. User Interface Design: This chapter provides a general idea of how the user interfaces will be structured.
4. Requirements Traceability:This chapter explains how the requirements defined in the RASD are correlated to the design elements that are defined in this document.
5. Implementation, integration and test plan: This chapter identifies the order of implementation and integration of the various components of the system. The testing of those components is also described in this chapter.
6. Effort Spent : This chapter shows the amount of hours spent by each member of the group to write the document
7. Appendix : In this chapter the tools used to create the documentation are listed.

2 Architectural Design

2.1 Overview: High-level component and their interaction

The SafeStreets App is a distributed App with three logic software layers. The Presentation layer wich manages the interaction of the user with the system and is responsible of maintaining the GUI, the App layer which handles the logic of the App and its functionalities the last layer is the Data access layer which manages the accesses to the database and allows the separation of concerns between business logic and data. This so called three-tier architecture is thought to be divided on three different hardware layers that represents a group of machines so that every logic layer has a dedicated hardware. This architecture makes it possible to guarantee scalability and flexibility of the system and also lighten the computational load of the server splitting it in two different nodes.

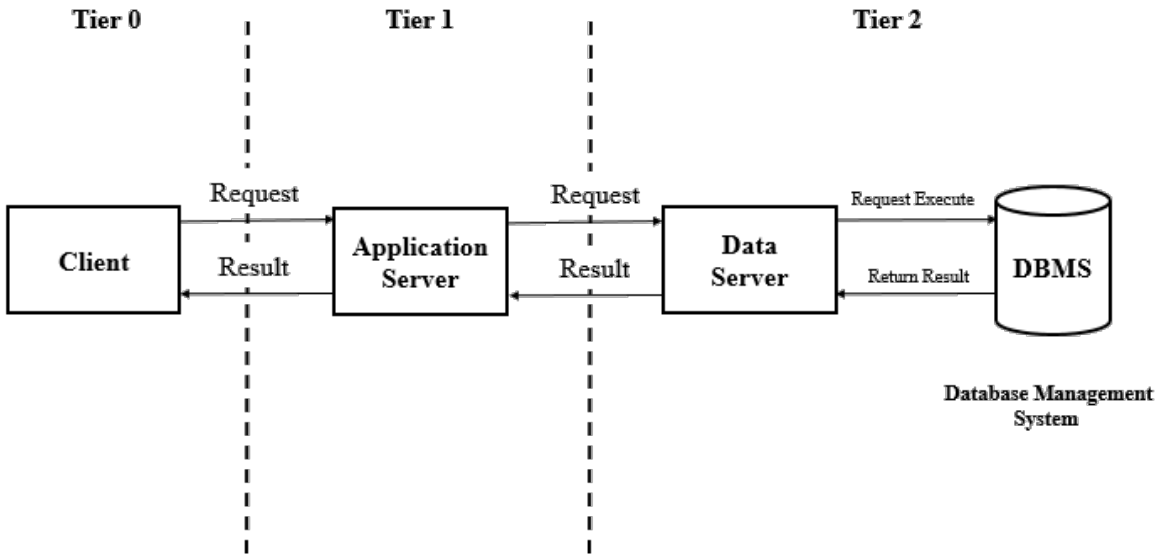


Figure 1: Three tier architecture

This architecture improves also the security of data since users can't access directly the data layer, but have to communicate with the App layer that will retrieve only the necessary data. Citizens access the system using their mobile phone App , which communicates with the App layer to send reports and get statistics; authorities can access the App in the same way as citizens do, but they can also receive assignments, see reports and take on assignments and finish them. The server of the App layer sends push notifications in asynchronous way to the authorities to warn them about violations in an area. Municipality and System manager communicate with the App layer to add new municipality and authorities to the system, to retrieve statistics and to read suggestions for improving safety on streets. The App layer communicates with data access layer synchronously to obtain information and asynchronously to store information about violations. This kind of architecture allows the server nodes to be replicated in order to improve the system scalability. Replicating nodes adds the need for a new component in the architecture, the load balancer, that is responsible for distributing the working load among the replicated nodes. This Approach also increases the fault tolerance of the service since a fault in a node doesn't affect the service availability. An error in the system may increase the work load for the other nodes, so the number of replications should be decided considering also this possibility to avoid the creation of a bottleneck due to a fault of the server. To assure security of data managed by the system the server has two firewalls to check packages exchanged with users. They both filter packages incoming and going out of the App logic one towards user devices and the other towards the data access layer.

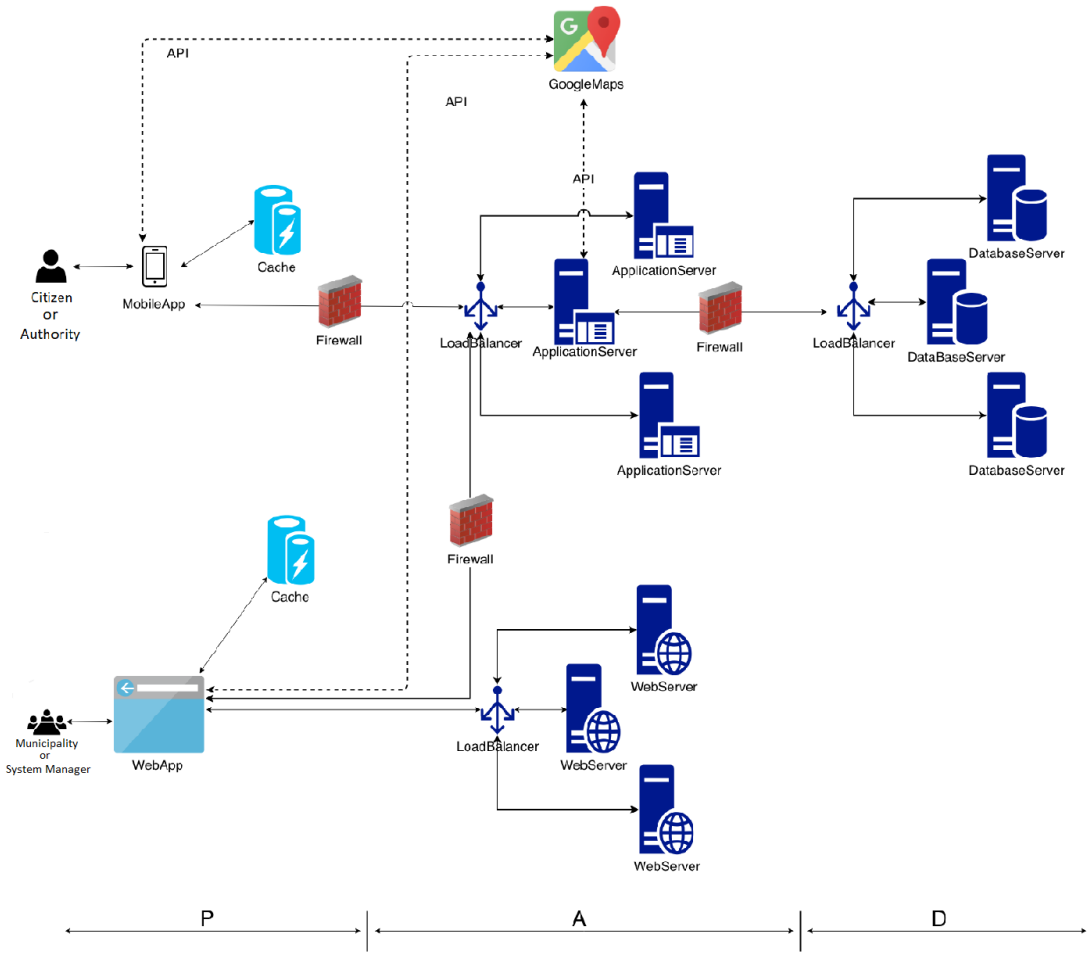


Figure 2: Application High Level Architecture

Citizen and authorities are provided of mobile Apps to access the functionalities offered by safestreets. Municipalities and system managers communicates with safestreets using a web App. The App layer is divided in two parts, one which sends static data (HTML, CSS, Javascript) to web App and another part which communicates with database and provides users dynamic contents. Our system uses only few caching capabilities because lot of data is dynamic and chages continuosly. The only information we cache are those about the violations when an authority takes the corresponding assignment; citizens have in cache the list of reports they have done and can choose how long that list would be kept in memory. For scaling purpose a non relational database could be used in later releases to collect data from databases with different structures and then use mining techniques to find useful information and pattern in recurrent violations and information. This could be useful to build statistics that may be queried on a regular basis and stored in a location which is faster to access for our system, reducing the number of queries to be done to external databases.

2.2 Component view

In this section we present the components of SafeStreets we start from a generic point of view showing the main parts which composes our system and then we divide the system in smaller parts to analyse the behaviour of the single components and their subcomponents.

2.2.1 General Component view

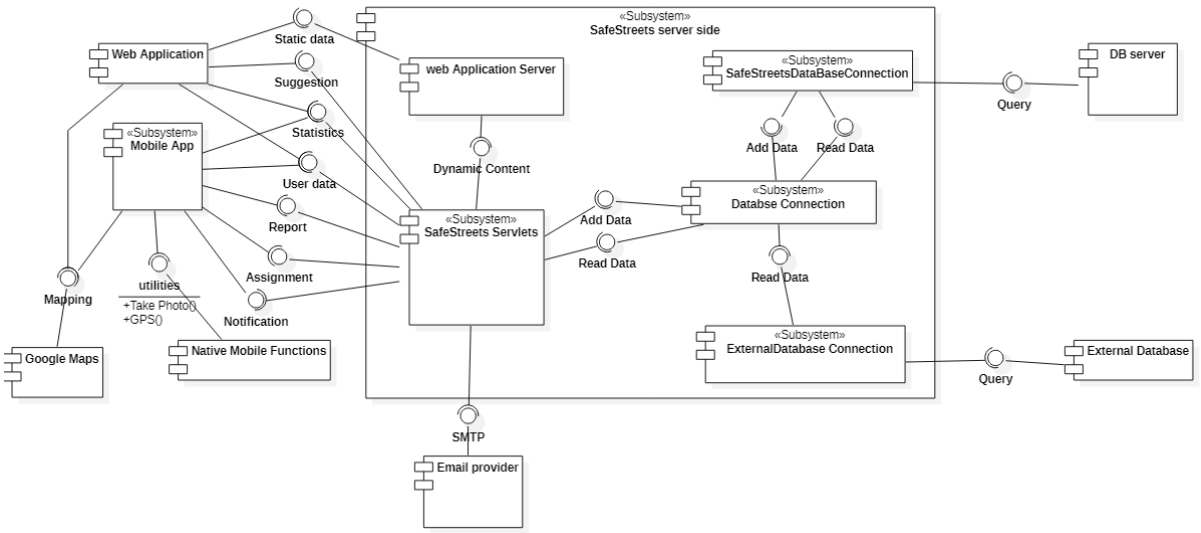


Figure 3: General Component diagram

In this diagram we represent a high level logic view on Subsystems and components. We show the interaction between the server and mobile App,web App, email provider and databases. The server is composed of five SubSystems and components:

- SafeStreets Servlets: this contains all the Servlets which allow the user to communicate with the server. To retrieve and save data this component communicates with the Database Connection subsystem. All responses to the user are sent in JSON format.
- Web App Server: this component provides the web App the static data of the App (HTML, CSS, JAVASCRIPT), this part is separated from SafeStreets Servlets for the sake of separation of concerns, dynamic content and static contents are different so different components should get request to retrieve them.
- DataBase Connection: this component acts as a facade which separates servlets from data access logic. This component communicates with SafeStreets servlets providing them the functionalities to access data. It communicates to Connection subsystems which communicates with SafeStreets Database and with External Databases to get data which is required by users.
- SafeStreesDataBaseConnection: this component communicates with SafeStrees database executing query to update and read data from it and provide data to DataBase connection component.
- ExternalDatabaseConnection: this component communicates with External databases executing query to read data from it and provide them to DataBase connection component.

SafeStreets Servlets communicates using SMTP protocol to send emails to users who have created a new account or have forgotten their credentials. It also communicates with Mobile App and Web App providing them with Access to their account data, the possibility to make reports for Citizens, to take assignments for the Authorities, to read suggestions for Municipalities and to see statistics for every type of users and visitors. The server also communicates with databases to retrieve and save data about violations, users and accidents. Mobile App and web App communicate with Google maps API to get maps to show to users. Mobile App also needs to communicate with functionalities given by the device such as taking Photos and obtaining position using GPS

2.2.2 Mobile App and Web App Component view

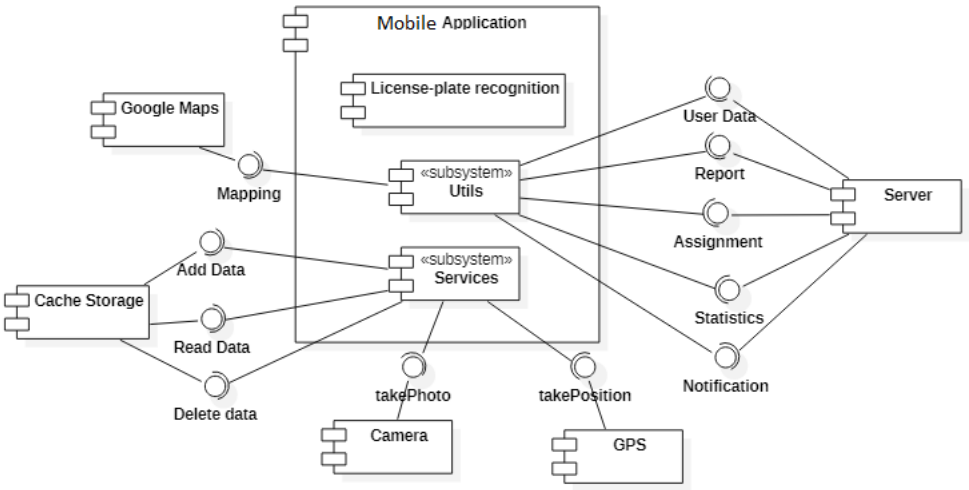


Figure 4: Mobile App Component Diagram

In this image we show how Mobile Apllication Subsystem can be seen in more detail. There are two main components :

- Utils: this component takes care of the communication of the App with SafeStreets Server and Google Maps mApping API.
- Services: this component communicates with Mobile phone's services which are needed for our App. Those components are GPS which is used to retrieve user position, the phone camera user to take photos of the violations and cache storage used to save locally some useful informations both for Citizens and Authorities.

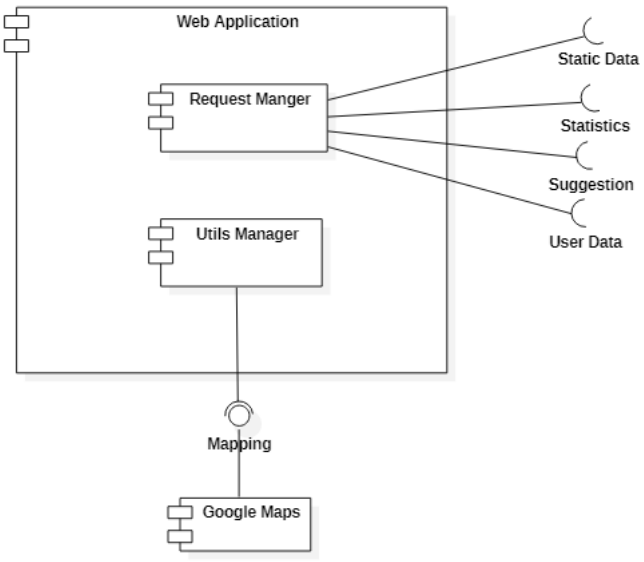


Figure 5: Web App Component Diagram

Web App works in a similar way to the mobile App. It must retrieve static data from webServer differently from mobile App.

2.2.3 Server Servlets Component View

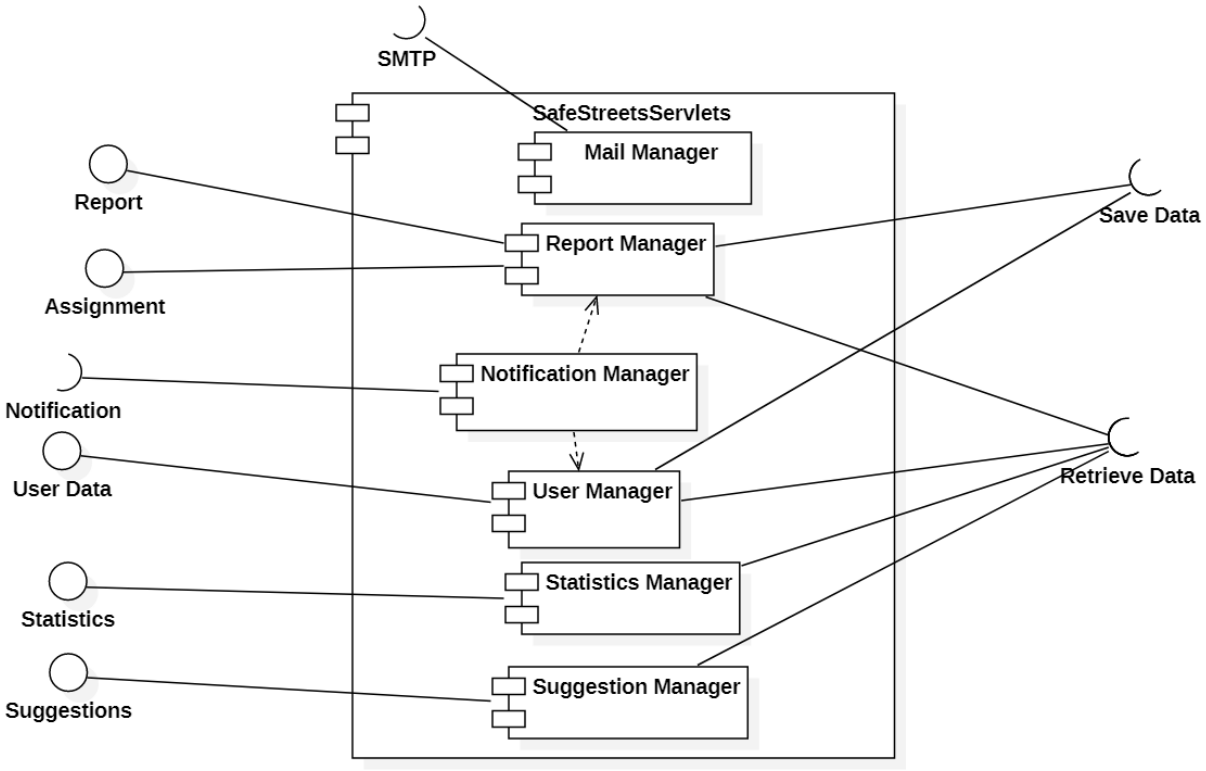


Figure 6: Servlets Compent

SafeStreets Servlets component is composed of 6 Managers:

- Mail Manager: this component allows the system to send email to Users when they create account or they modify their credentials. This component is associated with user manager which informs it when an account creation or modification is successful.
- User Manager: this component allows users to create and handle their account informations. They can create a new account, modify their data and Login. In order to allow these functionalities, this component must be able to both save data and retrieve data from database.
- Report Manager: this component allows citizens to create reports and manage them, and also allows Authorities to take assignments and terminate them. It communicates to the notification manager when new Assignments are created.
- Notification Manager: this component sends notifications to the authorities about new violations. It requires the Mobile App to open a websocket to communicate.
- Statistics Manager: This component retrieves data to make statistics requested by Users and visitors.
- Sugggestion Manager: This component retrieves suggestion requested by Municipalities.

All components take and save data using interfaces exposed by DataBase Connection component.

2.3 Deployment view

In the following image the deployment diagram for SafeStreets shows the distribution of the system components and the different deployment nodes. We have used a different color to represent External Database servers to show that the system should not implement these nodes but should only communicate with them. Google Maps API and Email Provider are not shown in this diagram to simplify it and because their interaction with our system is described in component view.

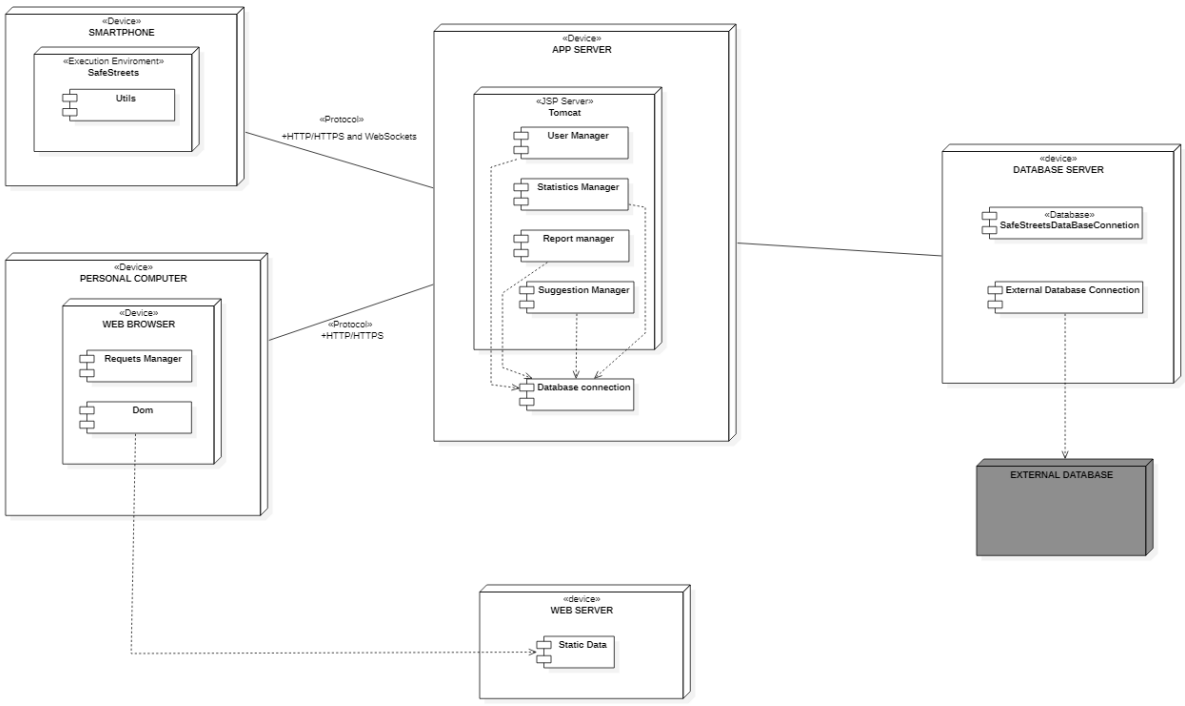


Figure 7: Deployment Diagram

This Diagram is divided into three sections to show clearly the separation of the different layers of the App:

- Presentation: this layer contains the presentation logic. Users in order to access data need the mobile App for Citizens and Authorities or the web App, which is accessible through a web browser by municipalities and system managers. In order to make the Mobile App more accessible as possible, it must be available for both Android and iOS, supporting not only newer versions. For the same reason, the web App should be compatible with major desktop web Browsers: Google Chrome, Mozilla Firefox, Safari and Microsoft Edge . Connection is made using HTTP and HTTPS for both devices in the figure, whereas it is made using also WebSockets for mobile App push notifications.

- BusinessLogic: this layer is divided into 2 different nodes types. Web Server communicates with webApp and sends it static data needed to render static components (HTML, CSS, and Javascript). App Server instead allows users of both Apps to access dynamic data acting as an intermediary to separate presentation and data but also to control access to data.

- Data Access: This layer executes a relational DBMS and provides functionalities to the Business logic to access data requested by users. In this layer, we have shown also the external databases which are another important source of data for SafeStreets. At first, the server will access external databases using the interfaces provided. Then, the App will grow and a different approach should

be implemented in order to reduce requests to external systems and improve performances. In order to reduce this load, a non-relational Database internal to SafeStreets may be used to store information from external databases using them as resources to create datamarts and to exploit technologies optimized for Big Data. If we use this approach, thanks to mining techniques, this non relational Database will build suggestions for municipalities.

2.4 Runtime view

2.4.1 Make a report

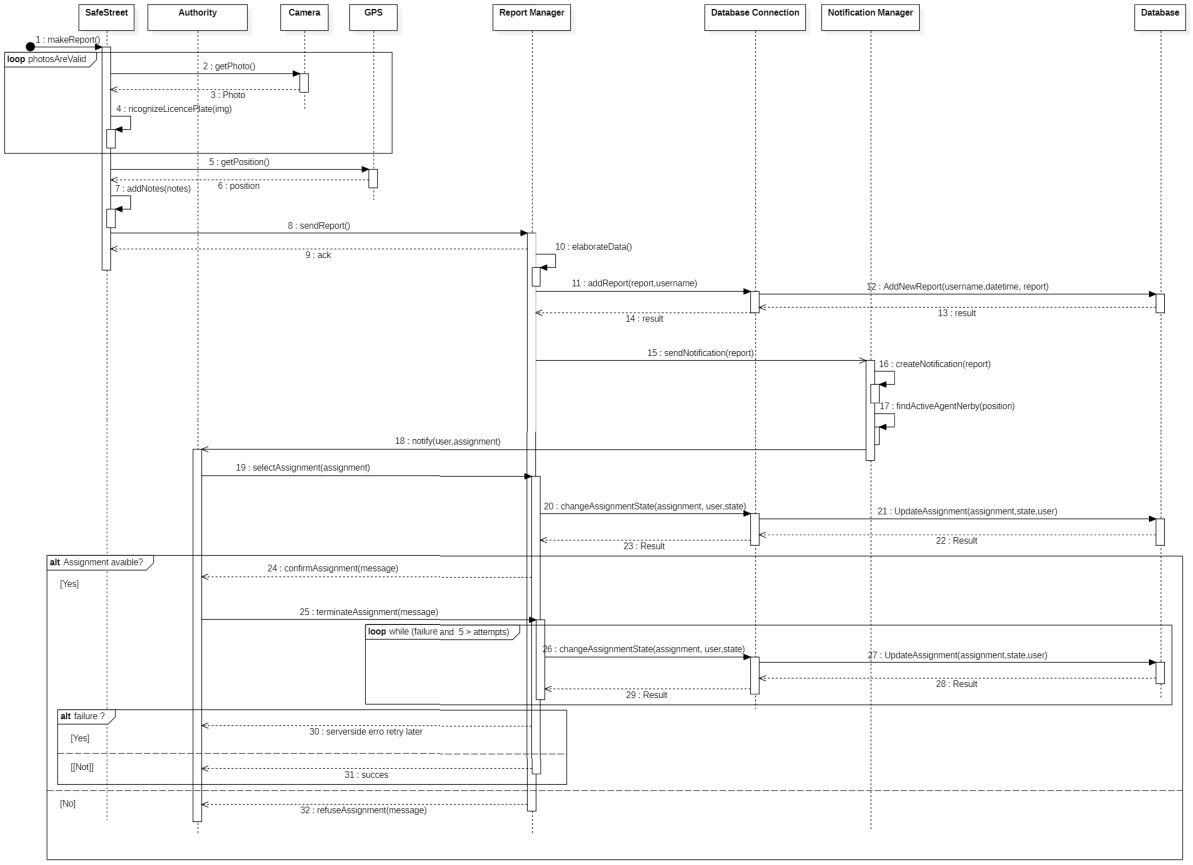


Figure 8: Sequence Diagram: Making a Report

In this sequence diagram is represented the process of making a report by Citizen, handling a request by the server and finally managing an assignment by authority corresponding to the report created in the beginning. Everything starts when the Citizen clicks on the Button "Make a Report" on the main screen; the phone camera opens automatically, so user is asked to take the first photo of the violation. After the photo is taken, it is elaborated by the Algorithm to Recognize Licence Plates. If it recognises a licence plate, the App gets the position of the user through the phone GPS, whereas in case of no recognition the user is asked to take another photo. To enrich the Report, the App allows the user to write some additional notes that authorities can read to get more information about the violation. Then, when the user decides to send the report, he clicks on the corresponding button; the App handles the request and sends the report and all the needed data to the Report Manager. The Report Manager elaborates the data and sends the report to the DatabaseConnection subsystem which manages to add the report to the DataBase. In this diagram we don't show the SafeStreets database connection subsystem to lighten the diagram, and also because it would be called with the same request as the DataBaseConnection resulting in no useful information being added. The Database recognizes if the report is new and creates a new Assignment if it doesn't exist yet, or it just adds a report connected to the assignment if it does. After that, the Report Manager sends the notification to the Notification Manager which manages to find the closest agents to the location report and warn them using push notifications of the assignment. The warned authority can decide whether to take the task or not; in case of acceptance, the Mobile App sends a request to the Report Manager that checks if the Assignment is still available in the database. If it is,

the Report Manager modifies the state of the assignment and confirms the authority to take charge of the assignment, otherwise a refusal message is sent to the Authority. Once the Authority terminates his assignment, he can notify the system using the Button "Stop Assignment" and, after that, the App asks the User to specify how the assignment ended and the type of violation occurred. Those data are sent to the Report Manager that modifies the state of the Assignment; if the modification is successful the Report Manager notifies the user, otherwise it retries this operation 5 times and then, in case of failure, sends an error message to the authority notifying him/her that there are problems in the server and asks the user to retry later to terminate the assignment.

2.4.2 Login from Mobile App

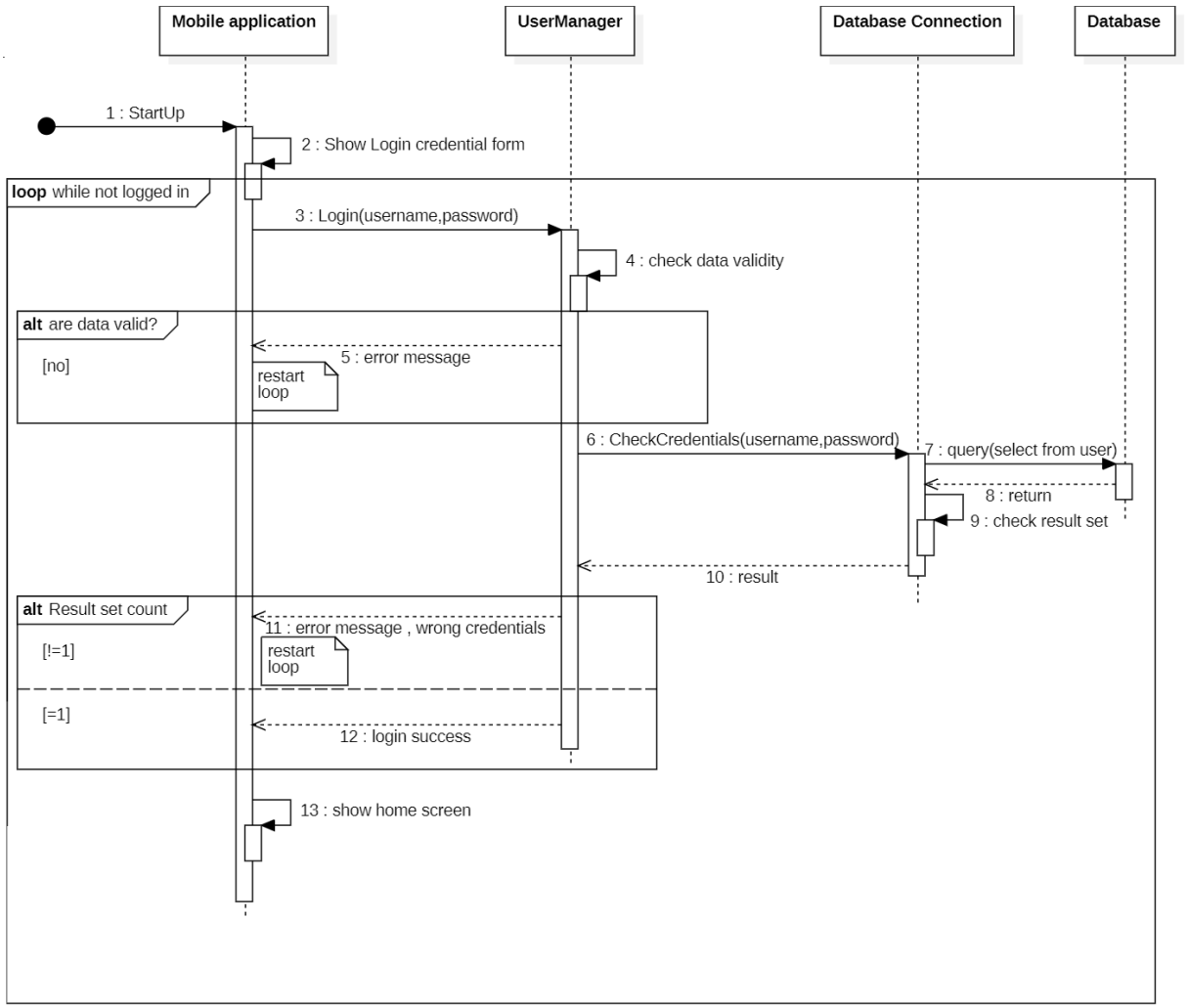


Figure 9: Sequence Diagram: Mobile App Login

This sequence diagram explains the process of a User logging through Mobile App, furthermore, this method is also used for the authentication via Web App. Once users get access to the App, they are asked to identify themselves in the "Login Page" inserting their credential in the dedicated text fields. Once the user submits the form pressing "Login" button, data are sent to User Manager that takes care of validating them. In case of failure an error message is sent to the user who must repeat the authentication sequence, otherwise, the User Manager asks the Database Connection subsystem if the credentials are associated with a user, as in the last diagram the SafeStreets DataBase Connection subsystem is not represented

because it wouldn't add relevant interaction for the App. If it is found an account who is associated to the inserted credential, the user is logged in and the homepage is shown to him, otherwise the login request is refused and the user is notified that the inserted credentials are wrong, and he/she must retry the Login process.

2.4.3 Authority registration

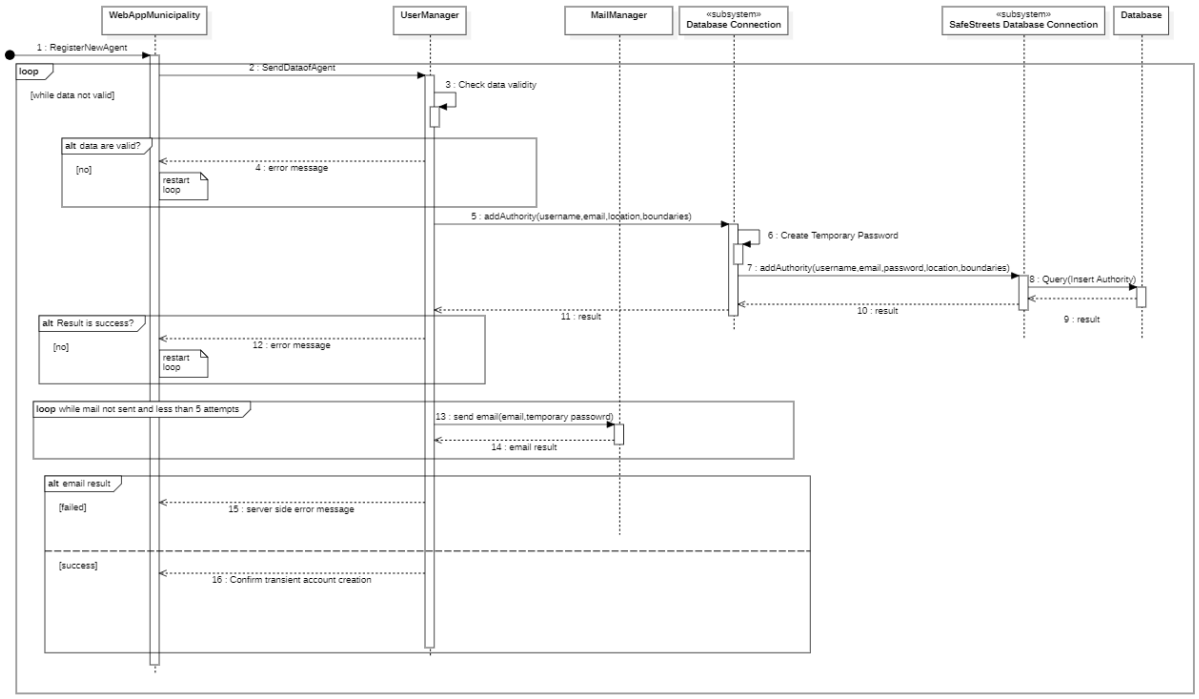


Figure 10: Sequence Diagram :Authority Registration

This sequence diagram shows the process of the registration of a new authority to SafeStreets service. The registration of an authority must be done by a Municipality, which starts the task pressing the "Add Agent" button in the Homepage of Web Application. Then, the application shows a page for registering a new authority which contains a form to be compiled with his/her data (email address, username and the boundaries of the area he/she takes care of). After that, the Municipality, using the "Add" button, sends the request to add the agent to the User Manager, which checks if data are valid. In case of invalid data, an error message is sent to the municipality that must refill the form and redo the registration sequence. Otherwise, the data are sent to the database connection, which generates a temporary password for the authority account; after that, the SafeStreets Database Connection subsystem checks the existence of an account with the same username or email. If there is a match in the database, the new authority is not registered, and the Municipality gets an error message. If the creation is successful, the User Manager forwards the request to the Mail Manager to send an email to the Authority containing the temporary password. If server fails 5 times to send the email, it informs the Municipality that there are problems with the email and that it will take care of sending it. If the email is sent successfully, the authority is informed of the creation of his account.

2.5 Component interfaces

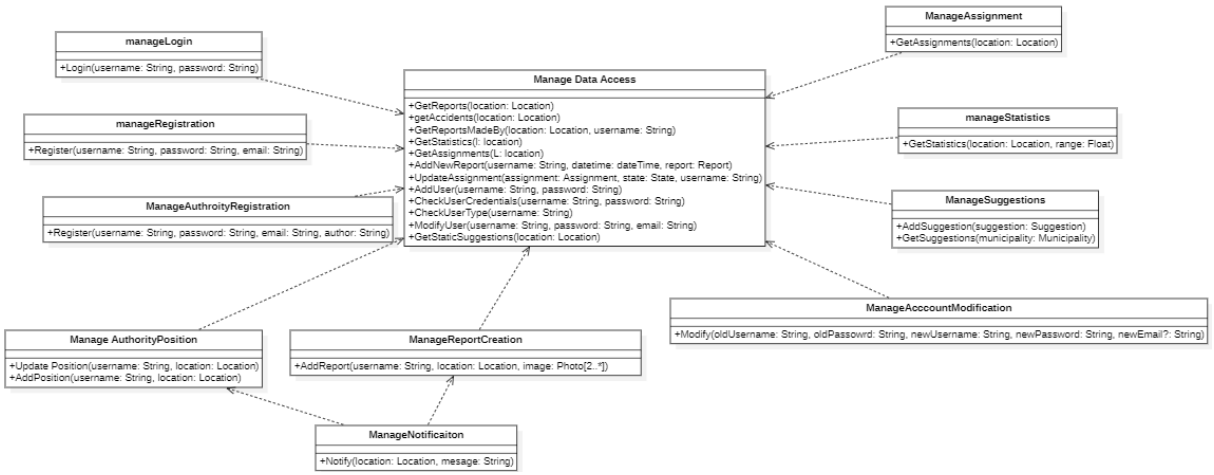


Figure 11: Component Interfaces

In the picture above the component interfaces of the business logic and the facade provided by data access layer are shown. The arrow represents dependencies between interfaces. All interfaces, except those of ManageDataAccess and ManageNotification, are required by the web servlets to respond to user Requests. ManageDataAccess acts as a facade provided by the Data Connection component and gives the other interfaces the needed data to respond. ManageNotification takes care of the notification sent to authorities, it requires the Manage Authority Position interface that informs it when an authority moves from one position to another, and the ManageReportCreation that communicates to ManageDataAccess interface to create a Report object. When the object creation is successful, it asks the ManageNotification to send notifications to every authority who is active near the location of the report. Those interfaces are provided by the Manager components of the system:

- User Manager: provides ManageLogin, ManageRegistration, Manage Authority Position, Manage Authority Registration and Manage Account Modification interfaces
- Report Manager: provides the ManageReportCreation and manage Assignment interfaces
- Notification Manager: provides manage Notification interface
- Statistics Manager: provides Manage Statistics interface
- Suggestion Manager: provides Manage Suggestion interface

In this diagram we used the symbol '?' after newEmail in ManageAccountModification to indicate that the old email isn't modified if it not specified by the user. This difference is done to show how all other parameters must be provided by the Servlets, which gets them from users and are mandatory for the functionalities to work. The most important interfaces needed by servlets to provide user the main functionalities of SafeStreets are:

- ManageReportCreation: this interface allows the Creation of requests, it needs the username of the user who has created the report, the location of the Violation and the photos associated with it which must be more than 2 [2...n]. It creates an object of type Report and uses the method provided by Manage Data Access to create a new report in case of failure the user is notified, in case of success two different scenarios may happen, the report is linked to a new assignment or the report is linked to an already existing assignment, if the already existing assignment is being taken care by an authority, the server communicates it to the user who reported the violation. In all the

other situations the Notification manager is asked to send notification to agents near the location of the violation and are asked to take the assignment.

- ManageAssignment: this interface allows authorities to find all the pending assignments which are close to their position. The assignments are sorted by number of associated reports within the last hour, if no assignment available respecting those criteria, also assignments from before the last hour are shown but are marked as old assignments.

- Manage Suggestions: this interface is mainly used to allow Municipalities to Get suggestions. Suggestions are made combining Report data and accident data close to the location of the municipality. In addition to dynamically created suggestions, Citizens and authorities can send proposals when sending reports for the former or terminating assignments for the latter.

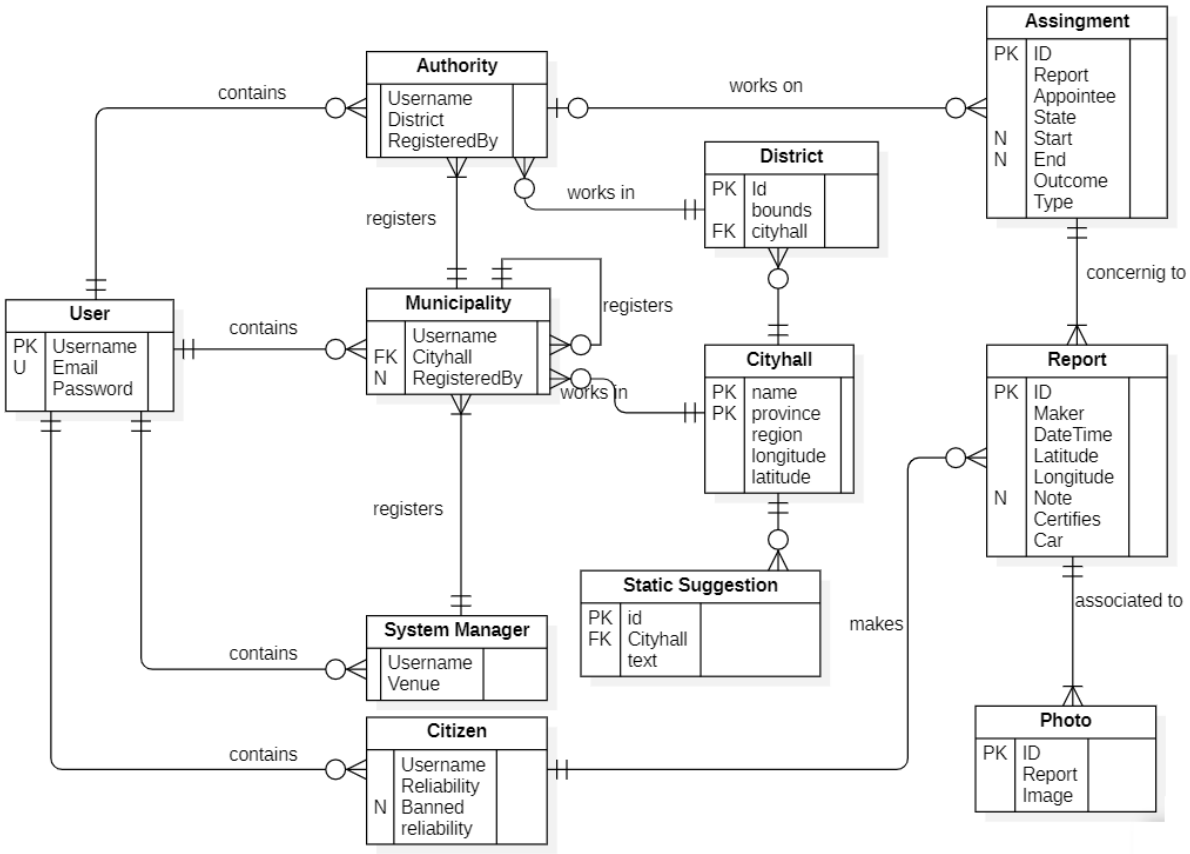


Figure 12: ER DIAGRAM

The image above represents the Entity Relation diagram to explain the data organization of SafeStreets database. The Different kind of users are identified by their username that works as primary key, each user has also a unique email address, because when user account information gets modified or credential are lost the right user must be notified by email. Municipalities have a nullable field RegisteredBy, which is used to keep track of who registers an authority or another municipality, it can be null if the registration happens through the system manager. To facilitate access to authorities and municipalities in a specific location the Cityhall table is used , this table must be indexed using longitude and latitude as indexes to making access to the table faster, Static Suggestions added by users and authorities must be indexed using Cityhall as an index to make easier access to suggestions. Report contains a field for notes made by users to authorities to give them information they couldn't provide using photos. Citizens have a reliability field which represents how dependable a user is, this is used by database to mark reports as

"possibly unsafe" when too many reports are done in a short time and reliability is not high enough, meaning those reports may not be consistent and if false reports are made, the user who makes them is banned.

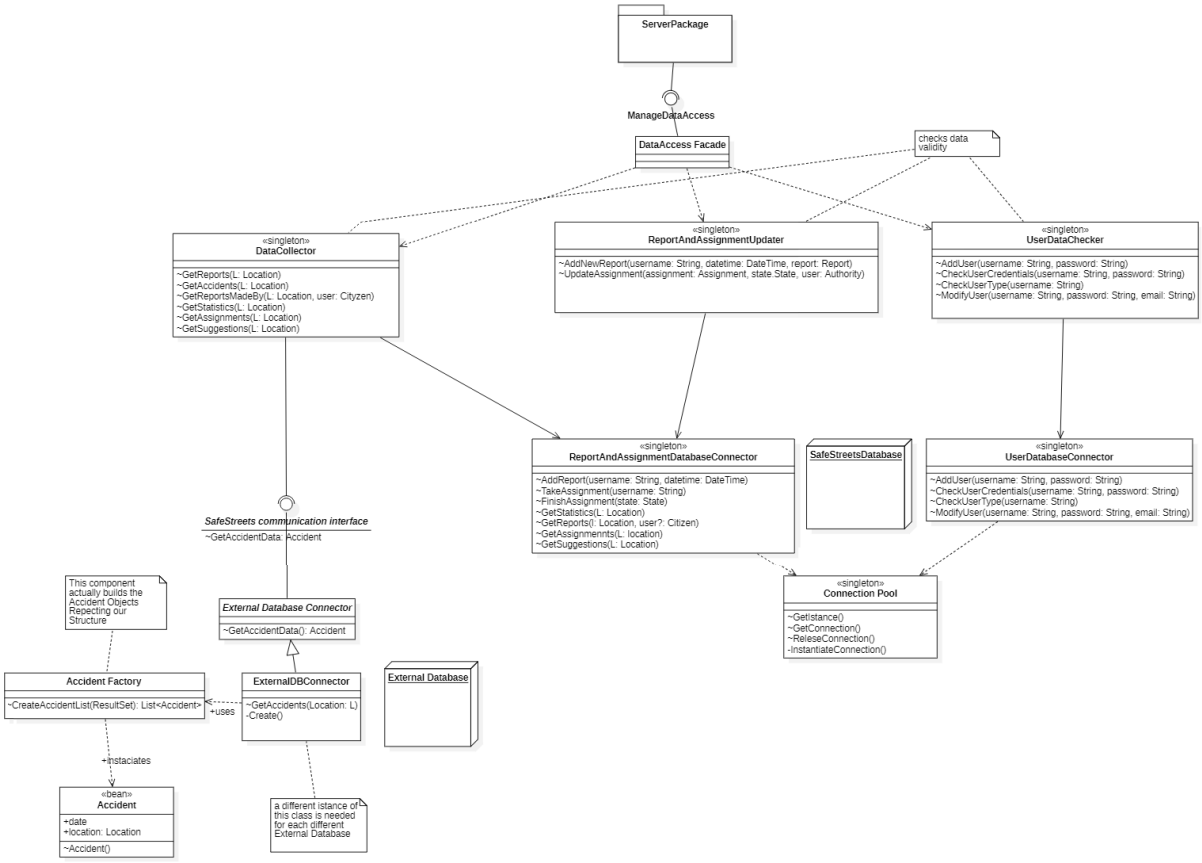


Figure 13: DataBase Connection Class Diagram

In this picture there is represented a part of the class diagram concerning Data Access Layer. This component provides the Business Logic (here shown as a package, the ServerPackage) an interface to access data. Internally different singleton classes handle the requests and a first group of classes checks input data validity: DataCollector, ReportAndAssignmentUpdater and UserDataChecker. The first one communicates with ReportAndAssignmentDatabaseConnector to get report, assignment and statistics from SafeStreets Database, and communicates with different ExternalDatabaseConnectors to get Accident data from external Databases. To communicate with external database the system has an abstract class of connector which contains the methods required by the system, for every database a different instance of child class is needed to implement the actual connection, and in case some information need further computation, a factory class is needed to create Accident instances which can be used by the business logic. A connection Pool for Connecting to SafeStreets Database to allow the reuse of connection and reduce the cost of memory allocation and deallocation in the server.

2.6 Selected architectural styles and patterns

2.6.1 Multiple Servlet server

We have chosen to use a server which provides multiple servlets to create different access points to the server for different requests. This allows us to make server manage different tasks using components that have few but specific functionalities, reducing the amount of interactions between them. This Approach allows the implementation and testing of components in parallel. The only components that must be developed and tested in advance are Database and its access components. Once those components are completed and their functionalities are tested, it is possible to develop different functionalities of the server in parallel, so the components must be tested and then integrated with the database access components. Moreover, this Approach allows us to make the system scalable: different functionalities may be separated in different hardware components and routing devices may be used to send the request to the right machine which can satisfy the request. The system can also be expanded in a really simple way: if we want to add new functionalities, new independent servlets can be added to manage them and a new manager to handle the request and send it correctly to the database connection components; as an alternative, an already existing component may be expanded to do it. Since different functions of the server are separated, in case some functionalities aren't working in the correct way, it is possible to pinpoint the parts of the system which aren't working correctly and fix them. For this Approach to work it is really important that the Database access components work correctly; in fact, a bug in one of those components would affect the overall App, making later maintenance more costly. So it is really important that all the functionalities of this part are tested; moreover, the documentation must be as clear as possible to allow future maintenance and expansions as simple as possible.

2.6.2 Three Tier Client-Server

The separation of the system in three different layers allows to make the system more flexible and reusable. Together with the multiple Servlet Approach it allows to add functionalities and modifying them without affecting the entire App. This also allows the separation of the presentation of data to the user from data access using the App layer to block access to information that a user should not access (I.e. a citizen should not access photos of violations or assignment lists).

2.7 Other design decisions

2.7.1 Thin client

A thin client is based to performing a single function which is to communicate with server to retrieve data and concentrates on presentation of them to the user.
Thanks to this design choice, it's possible to separate the business logic from the presentation. However, our Mobile App contains a layer of business logic, so, it is not purely a thin client, but this alternative is made to reduce drastically the draining of computational resources from server, in fact, in the App there is the Licence plate recognition algorithm which checks if at least one photo provided by the user contains a valid licence plate. Doing such a control from server side, all the photos should be sent to the server that takes care to check them, if there is a failure it should notify to the App the problem and ask the user to re-take photos and send them again to the server, considering also that that kind of algorithm can take lot of time to be executed this would reduce system performances. To clarify this statement this example may help to understand our choice: an execution time of a millisecond on a mobile phone may be seen as a bit of lag by the user, but if the algorithm runs on the server that millisecond of delay which must be multiplied by the amount of users sending photos and the number of photos sent by them can result in a delay of seconds. It is important to consider a lot those kinds of delays because nowadays, it must be guaranteed a satisfying user experience that invites the people to continue using the app.

3 User Interface Design

3.1 UX diagrams

The mockups of the application were already exposed in the RASD document.
The mock-ups of the application were already shown in the RASD document. We present UX diagrams showing the flow all type of users interacts with the system. There are repetitions in diagrams to show how mobile application users have lot of common interaction, the same applies also to the Web App. Every user when application starts up can login or ask to retrieve his/her lost login credentials. All user can also access statistics and modify their account's data and credentials. The differences are in some screen available only for specific type of users. Citizen can create new account and make reports, authorities can take assignments and terminate them, municipality can register authorities but can also see suggestions, system manager and municipalities can register new municipalities.

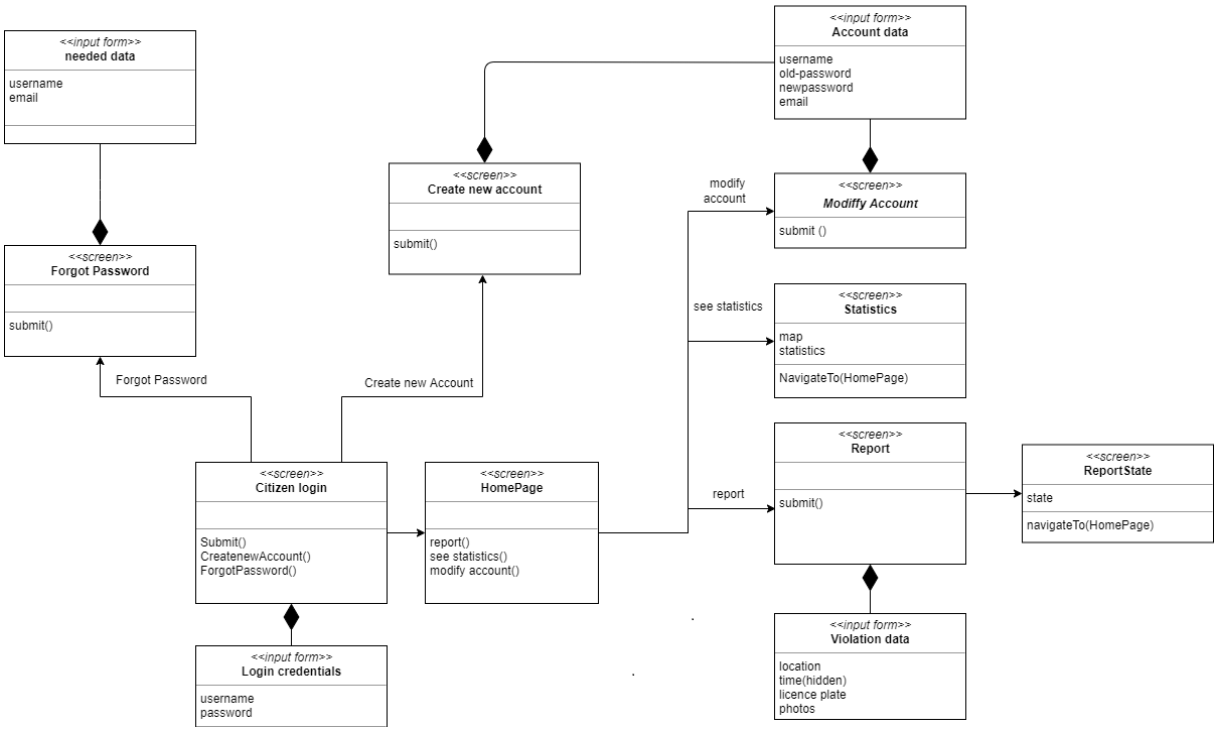


Figure 14: Citizen UX Diagram

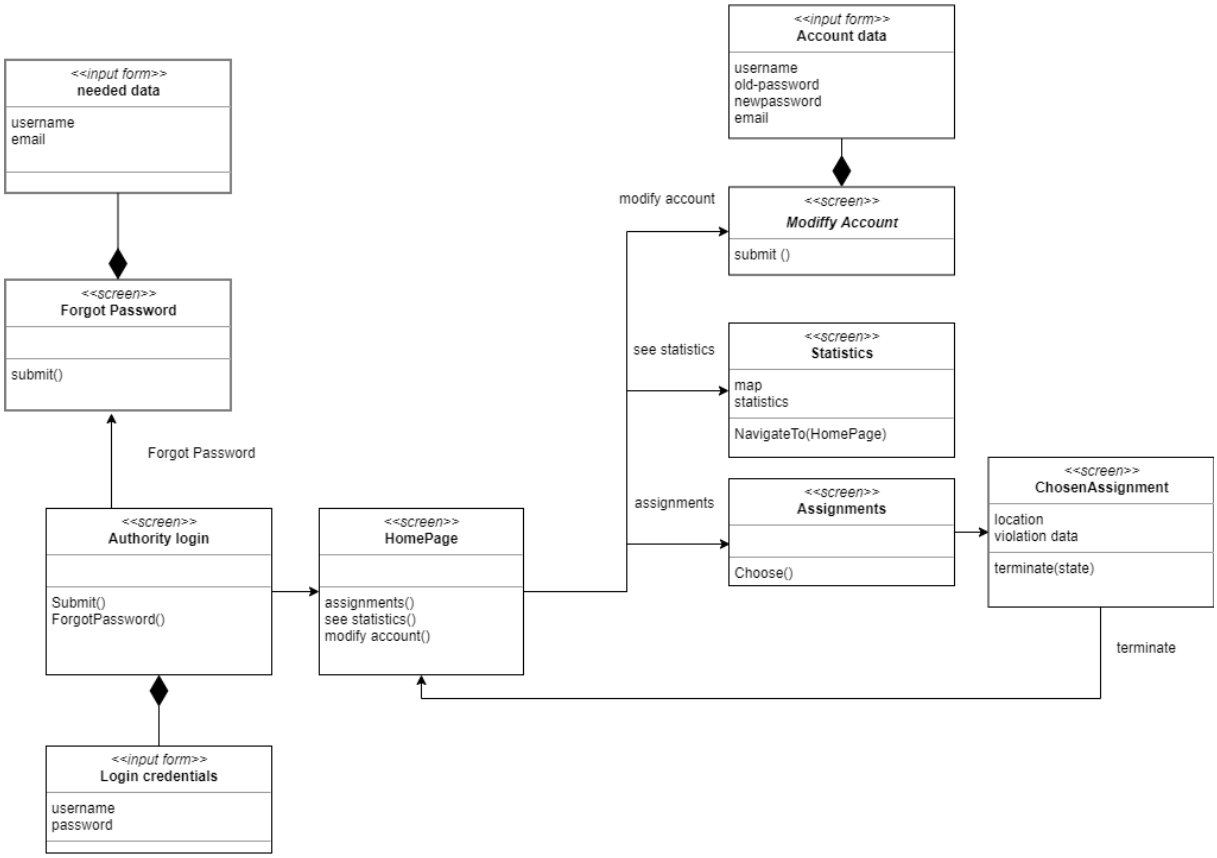


Figure 15: Authority UX Diagram

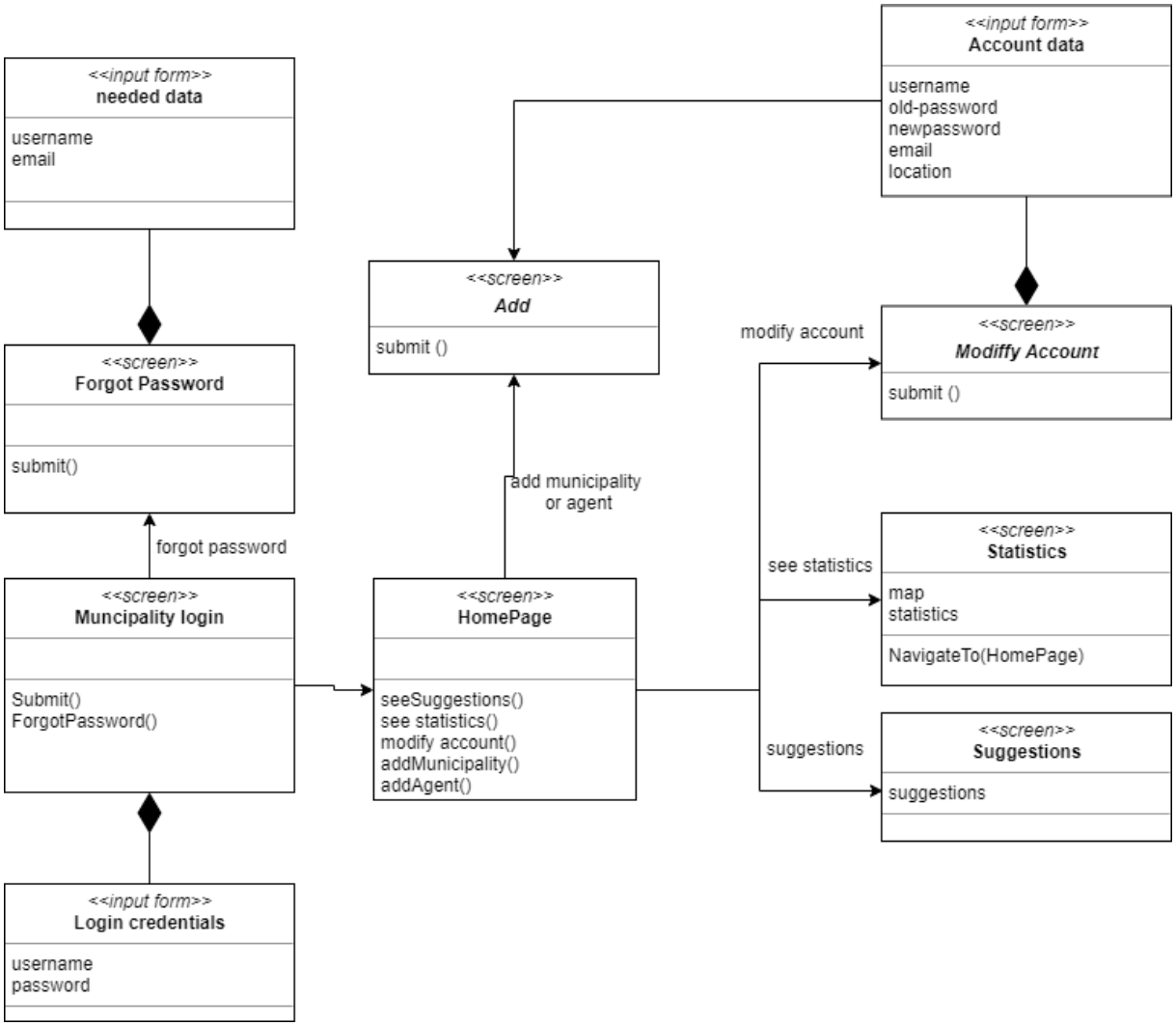


Figure 16: Municipality UX Diagram

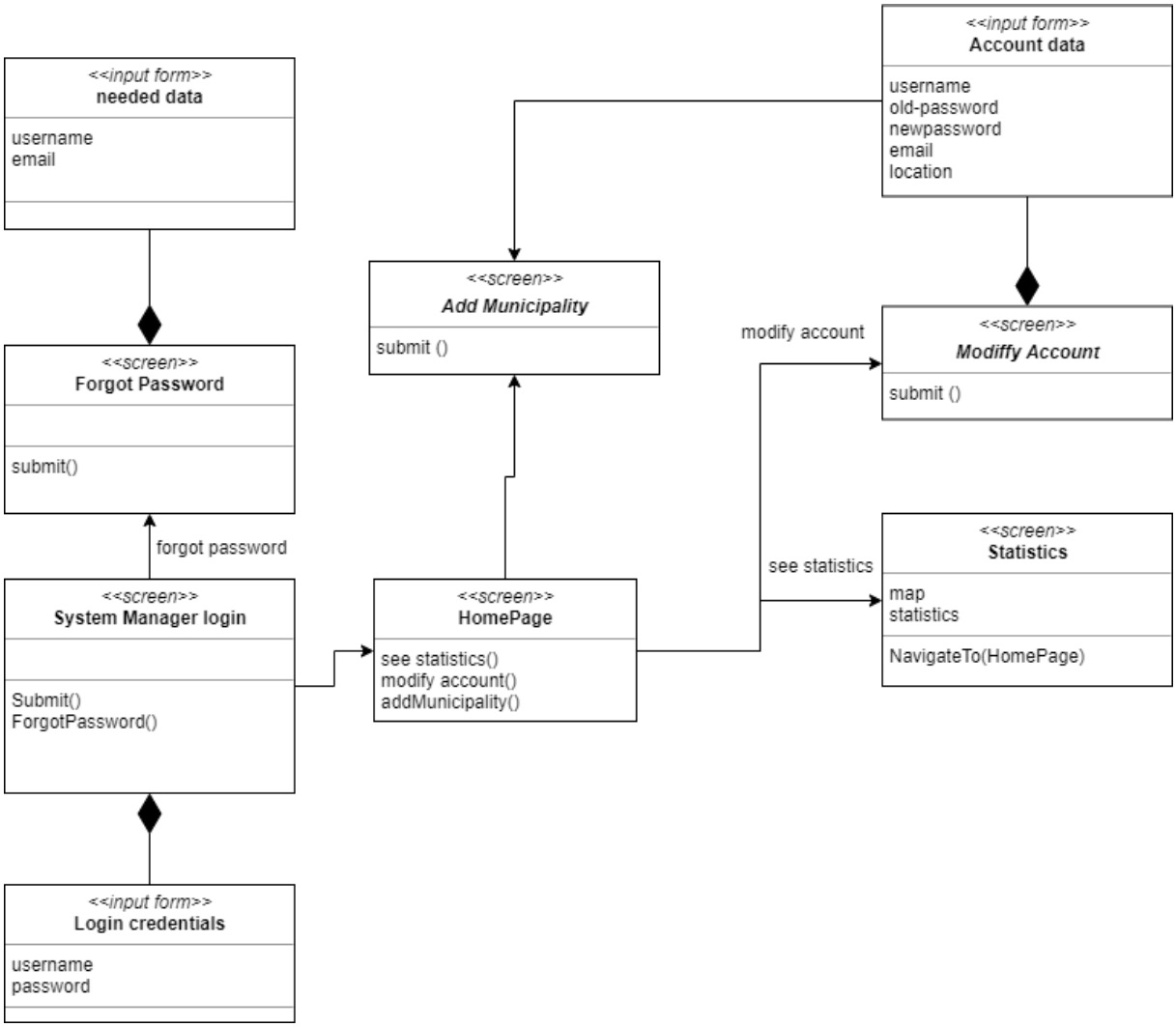


Figure 17: System Manager UX Diagram

4 Requirements Traceability

The DD is thought to guarantee that the components of the system can enforce the requirements presented in the RASD.
In this Chapter we show which components communicates with each other to enforce the fulfilment of the requirements.

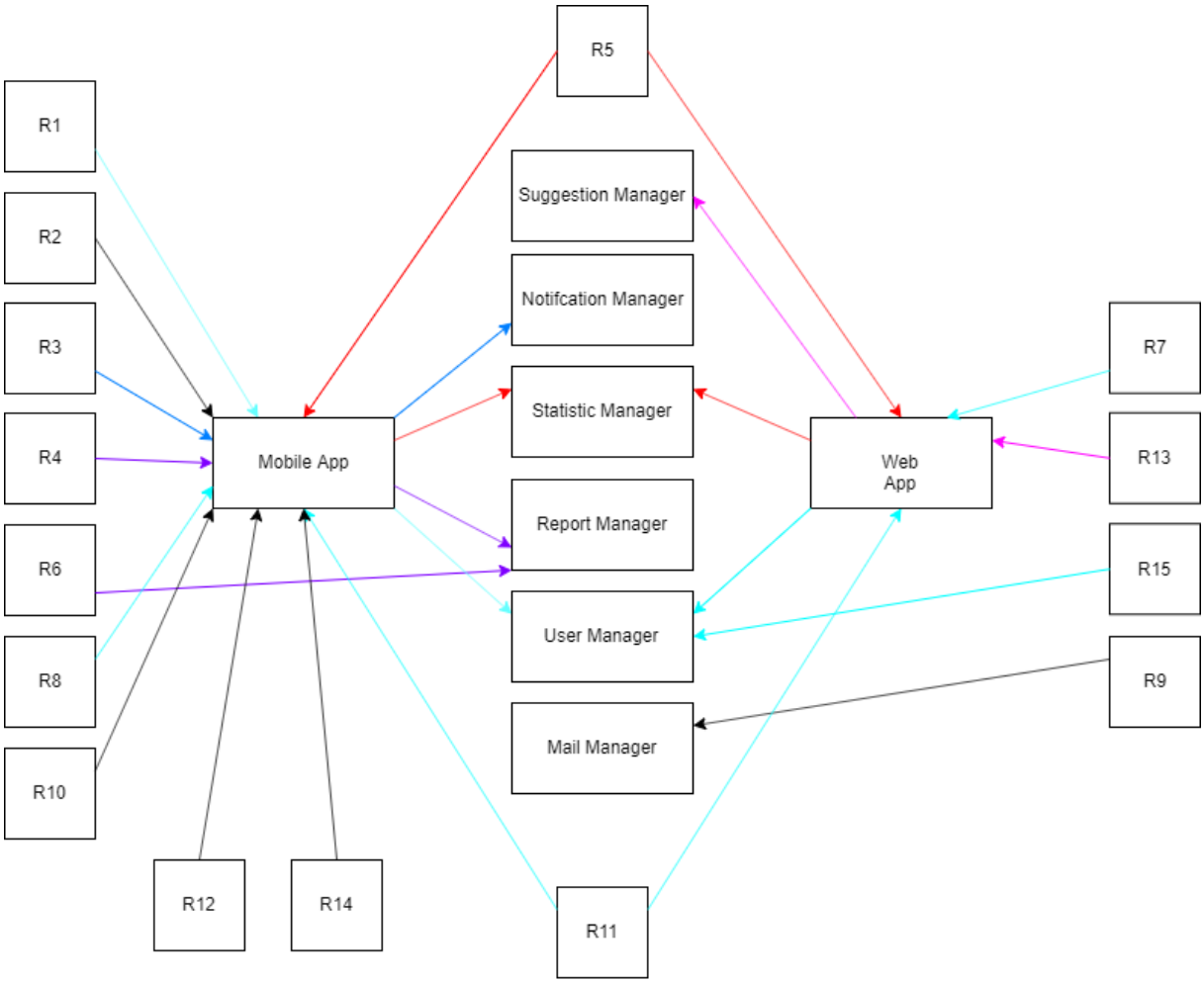
4.1 Requirements and system interactions

A small description of the interactions is here to show how the communication should be performed.

- R1) Authorities' location must be known by the system when they are in service: Authority Position Servlet gets data about authorities' location and through User Manager, the application sends data to the server every time the authority moves from the previous location of at least more than 200 meters.
- R2) When a Citizen makes a report the position is correctly added with the GPS when is available: Mobile application communicates with the GPS component of the Mobile phone, if available, to retrieve the position.
- R3) The right authorities are notified about violations: Notification Manager gets Authority position from database. The position is updated by User Manager which communicates with the DataBase Connection package through DataAccess Facade.
- R4) Authority must be able to provide the system how the assignment finished, resolved and the type of violation, no intervention needed when arrived, false report: Application after an authority accepted an assignment takes him/her to a screen where the assignment can be terminated. This Screen allows user to communicate with the Assignment Servlet which notifies the Report Manager.
- R5) The system must make Statistics available when asked: Data Access Facade allows all manager classes in the server package to access data which can be showed to users. User may query data making calls to the Statistics Manager.
- R6) Statistics are always updated when an event happens: Report Manager allows the system to get new reports and to terminate them, those data are stored in database using Data Access Facade and are used to build statistics.
- R7) For registering a Municipality his/her data must be provided to a System manager who will add those data to the service to sign up him/her: web app allows user to insert all the needed data and doesn't allow to send data if all the needed data are not correctly inserted. For further security also, Authority and municipality registration Servlet checks if all needed data are inserted.
- R8) A visitor must be able to begin sign up process in the SafeStreets App filling a form with his data: when a visitor accesses to the SafeStreets mobile app the application shows him the Login page which contains the form to authenticate and the link to Sign up form and sends the data to the User Manager.
- R9) When the creation of an account is successful the system must notify the Visitor sending an email to the address provided in the sign up process: After inserting data of the user in the database the system sends an email, containing a confirmation message, to the user email through the Mail Manager.
- R10) When GPS is not available the user can input the position from a map: Mobile Application's interface provides the possibility to select a position on a map using Google Maps APIs.

- R11)Users to use the full service must be able to login providing the right credentials: User Manager Checks if inserted credentials are correct allowing user to access SafeStreets functionalities if they are correct.
- R12) The camera of the mobile phone must be accessible to take photos of violations: Mobile App implements the communication with the mobile phone's camera.
- R13) Suggestions must be available when municipalities request them: Suggestion manager builds suggestions from data about violations, accidents but also static suggestions available on the database.
- R14) The User must be able to select the licence plate between the ones in output from the Licence Plate Recognition algorithm: Mobile Application's interface provides the possibility to select a Licence plate between the recognized ones.
- R15) Each Username is unique: User manager check the usernames during registration process. It doesn't allow different users to have the same username.

4.2 Requirement Mapping with Server Managers



5 Implementation, Integration and Test Plan

The System is divided in various subsystems:

- Mobile App
- Web Application
- Web Server
- Application Server
- DataBase Connection
- DataBase Server
- External Systems:Google Maps,External DataBases

The subsystems not external will be implemented and tested. They will also be integrated together, and with external systems. Since the Web Server provides user only static data (HTML, CSS, JavaScript) and is deployed to a different device than the web Application, this component can be implemented and tested in any moment during the development but the integration must be done when the full application is implemented. Given our design choice of having a multiple servlet server which communicates with DataBase through a facade, our implementation must have as a starting point the implementation of the DataBase Server and the connection with it.

5.1 DataBase Server and Connection

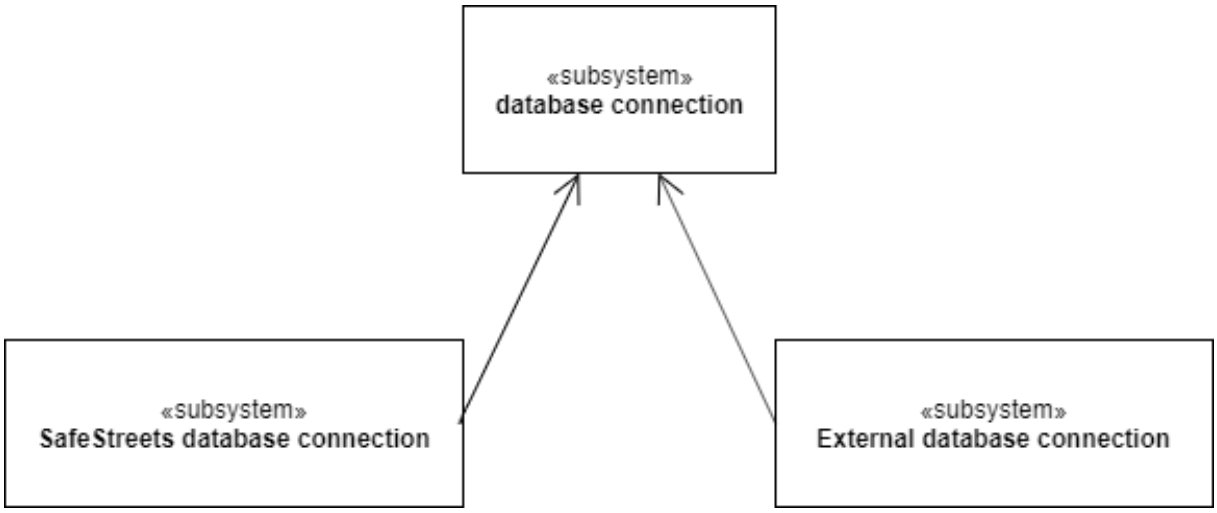


Figure 18: Database Server and Connection Integration

This phase is important so several test will be run to check the correct functioning of the connection and the functions for data access needed by the system. In this phase also the abstract classes for the connection to external DataBases will be implemented and tested with a dummy DataBase to test that the design choice for external DataBase works correctly. Then the DataBase connection is implemented and integrated with the DataBase Server and the External DataBase connection subsystems. The testing phase must cover as much cases as possible and different test approaches must be used to test the functionalities provided by DataBase Connection subsystem. After this initial phase the Application Server must be implemented

5.2 Web Application

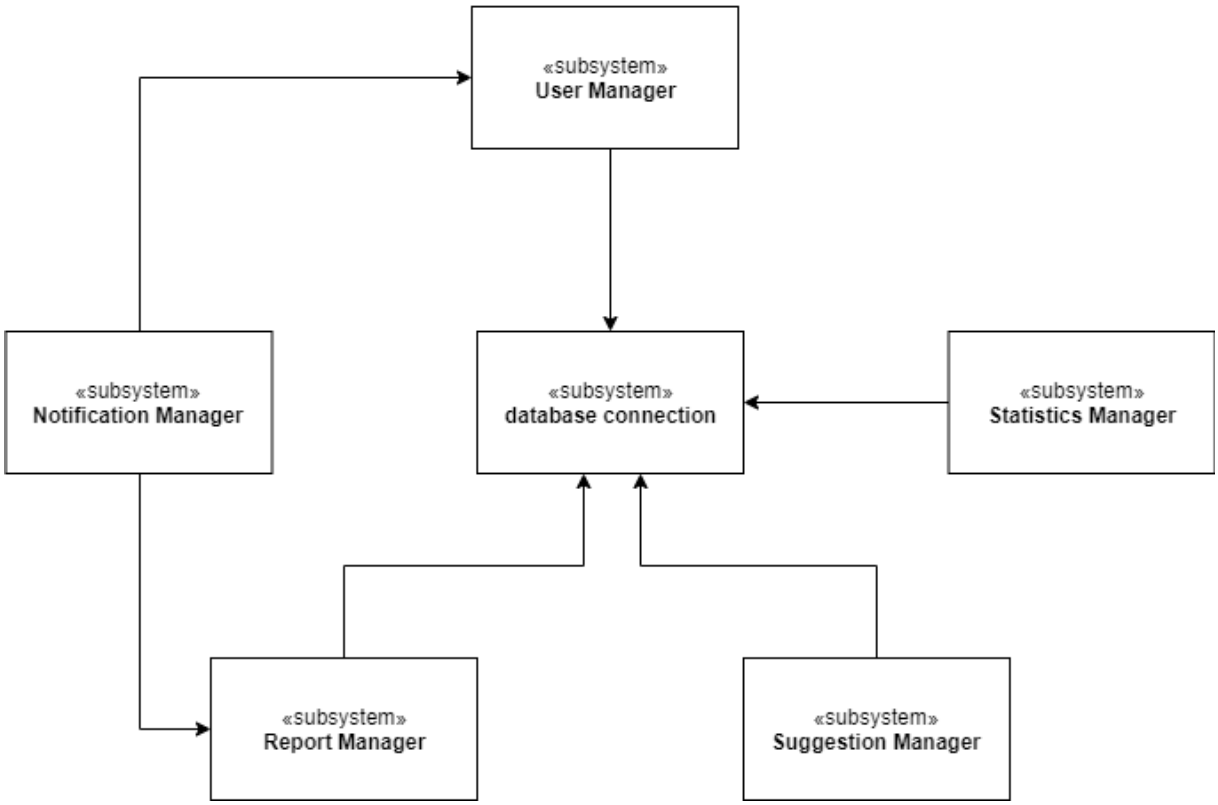


Figure 19: Servlet Integration

The implementation and testing can be performed in parallel for all the managers except for the Notification Manager which requires the implementation of the Report Manager and of the User Manager. All these components must be tested using test stubs to access and test functionalities of the managers, covering all of them and using coverage tests to check if all the checks done on data, and all possible kind of issues (I.e. missing DataBase, null values, exceptions) are handled and there are no controls which aren't reachable for any kind of input. The components are then integrated with the DataBase Connection subsystem

5.3 Mobile App and Web Application

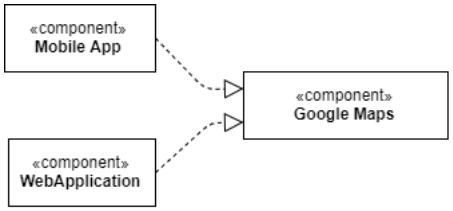


Figure 20: Google Maps Integration

In parallel to the implementation of Application Server the development of the WebApplication and Mobile App can be carried on.

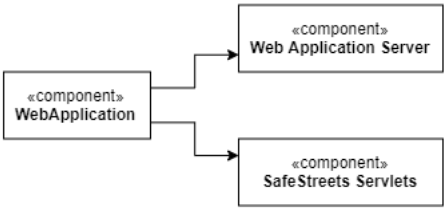


Figure 21: Web Application Integration

After the implementation both needs to be integrated with APIs provided by Google Maps and when Application Server development will be complete they will be integrated and tested together. Web Application also needs to be integrated and tested with Web Server.

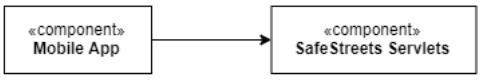


Figure 22: Mobile Applicaiton Integration

5.4 Main feature test plan after all components are integrated

Once all the systems are integrated an additional testing phase will be devoted to check the possible interaction of users with the system. For the main functionalities and interactions in the mobile application testing we will use some flowcharts, and do coverage testing over them and check that no unexpected results occurs with different values when citizens make reports, and when authorities accepts and terminate assignments.

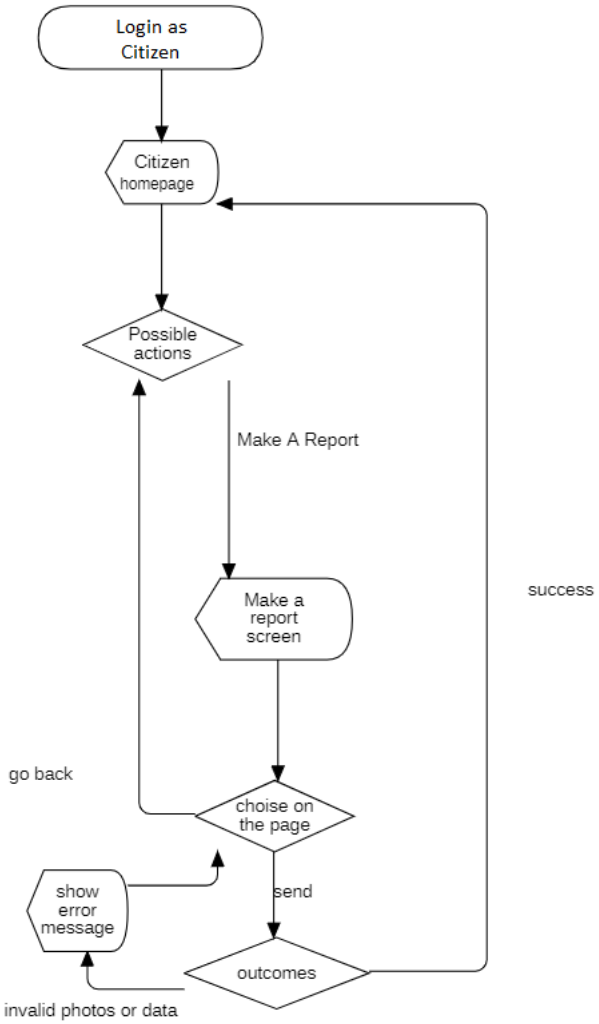


Figure 23: FlowChart: Citizen main features testing

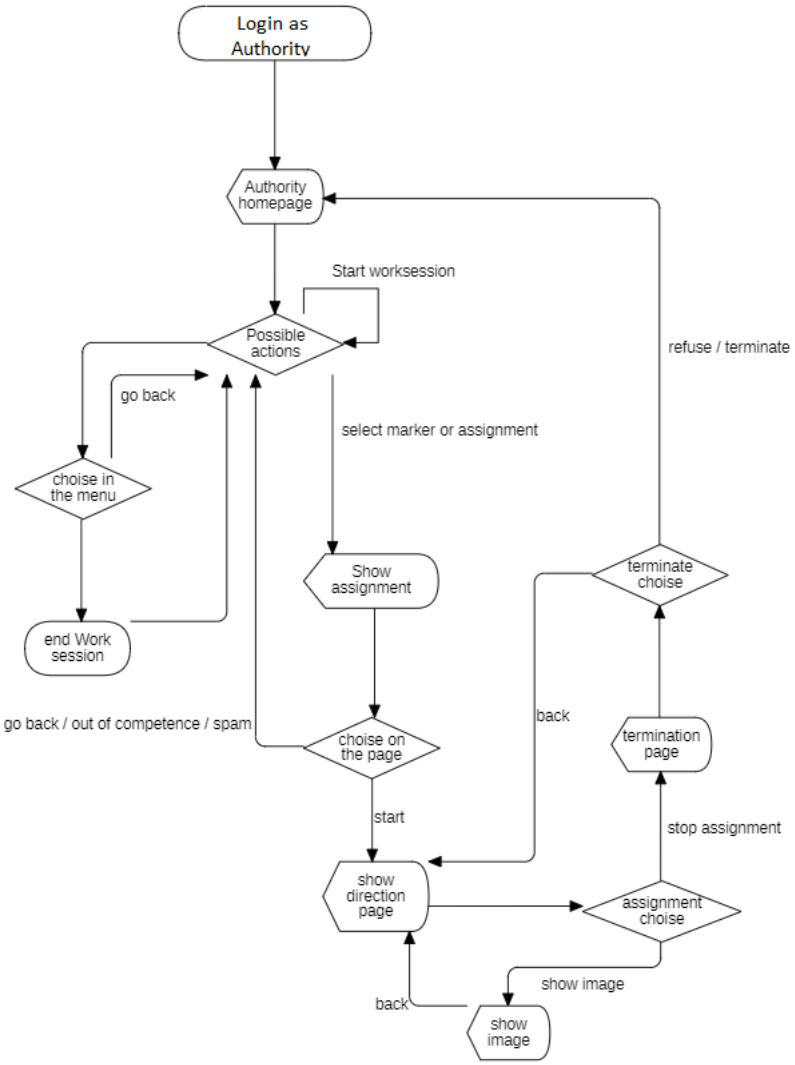


Figure 24: FlowChart: Authority main features testing

6 Effort Spent

Pietro Maldini

Section	Effort
Introduction	4
Architectural Design	30
User Interface Design	2
Requirements Traceability	5
Implementation, Integration and Test Plan	6
Total	47

Angelo Paone

Section	Effort
Introduction	4
Architectural Design	28
User Interface Design	2
Requirements Traceability	4
Implementation, Integration and Test Plan	5
Total	43

7 Appendix

7.1 Used Softwares

- UML Modeling Tools: StarUML, Draw.io
- Document Production: TeXworks
- Collaboration Environment :GitHub.com