Politecnico di Milano
A.A. 2019-2020
Software Engineering 2: "SafeStreets"
Design **D**ocument

Frederik Saraci, Lorenzo Amata

December 11, 2019

Contents

1 Introduction

1.1 Purpose of this document

In this document we are going to describe software design and architecture of the SafeStreets system. The software architecture of a system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

1.2 Scope

SafeStreets is a crowd-sourced application which aim at providing cooperation among citizens (end users) and authorities (municipality, local police etc.) in order to make more livable our urban environments. Indeed, by means of SafeStreets authenticated users can notify authorities when traffic violations occur, in particular parking violations.
Before notifying traffic violations the guest have to fill the registration form providing all the mandatory fields (including a valid ID document) in order to be recognized by the authority. The application allows users to send pictures of the various type of violations(vehicles parked in the middle of bike lines or in places reserved for people with disabilities,double parking, and so on), including their date, time,position, type of violation, notes and license plate.
As we all know, a single vehicle is able to commit multiple traffic violation at time, in order to make the system simpler: traffic violations committed by a vehicle will be individually reported using one or more pictures (if needed) for each of them and the type of the traffic violation, avoiding multiple type commit of traffic violations for a single vehicle.
SafeStreets stores the information provided by users, completing it with suitable metadata, in particular when it receives a picture, it runs an external algorithm (OCR) to read the license plate,the license plate is even provided by the user during the report, especially if the environment condition are not favorable (insufficient lighting).
The information provided by users can also be used by themselves or authorities (whose type of visibility may vary depending on the type of the user who is interfacing with the application) for statistic purposes, e.g. highlighting the streets (or areas) with the highest frequency of violations, or show a graph with the vehicles that commit most violations or the type of violations for a specific street.
Logged Users will be able to see a pie chart that show the type of vehicle that commit most violations and the type of violations for a specific street, while the Logged Authority Users will see highlighted streets with highest frequency of violations.
SafeStreets is intended to be a means to help people becoming more civilized improving the efficiency of authorities to guarantee the respect of the traffic laws, the municipality (and in particular local police) could use the data stored in SafeStreets about traffic violations to generate traffic tickets, in particular a single traffic ticket for each traffic violation committed by the vehicle.
In this case, the authenticity of the stored data must be guaranteed before issuing a traffic ticket, indeed the Local policeman is in charge to check it and eventually notify that the data have been manipulated by the user, and in this

case the data must be discarded and the user is banned.
After generating a traffic ticket the local policeman (Logged Authority User) notifies that the traffic ticket has been generated by using a flag. From the opposite side, SafeStreets can use the information about the generation of traffic tickets to build statistics about the effectiveness of SafeStreets initiative.

1.3 Glossary

1.3.1 Definitions

- *System:* the SafeStreets software we are to develop
- *Guest or Guest User:*: the person who access the system as no logged user
- *Logged user or Authenticated user:*: authenticated person (not authority) who is interfacing with the system
- *User:* logged user or Authority user in general
- *Banned User:* A logged User or Authenticated user that cannot interface with the system anymore due to violations of terms and conditions of SafeStreets
- *Logged Authority User or Authenticated authority or Logged Authority:* The authenticated public organization which use Safestreet
- *Local Police or Police:* The department of a local police
- *Registration:* when user provide all the necessary data for accounting him into the system
- *Violation or Traffic violation:* a violation of the traffic laws in general

1.3.2 Acronyms

- RASD: Requirements Analysis and Specification Document
- DD: Design Document
- OCR: Optic character recognition
- DBMS: Data Base Management System
- DB: Database
- API: Application Programming Interface
- GPS: Global Position System
- S2B: Software to be
- ID: ID document number

1.3.3 Abbreviations

- *i.e.:* that is
- *w.r.t.:* with respect to
- *i.d.:* id est
- *i.f.f.:* if and only if
- *e.g.:* exempli gratia
- *etc.:* et cetera

1.4 Reference documents

- Specification document: ”Mandatory Project Assignment AY2019-2020”
- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications

1.5 Document overview

This document is structured as

- Introduction: it provides an overview of the entire document
- Architectural design: it describes different views of components and their interactions
- User interface design: it provides an overview on how the user interfaces of our system will look like
- Requirements traceability: it explains how the requirements we have defined in the RASD map to the design elements that we have defined in this document.
- Implementation,integration and test plan: Identify the order in which you plan to implement the subcomponents of your system and the order in which you plan to integrate such subcomponents and test the integration.

2 Architectural Design

2.1 High level architecture

The SafeStreets architecture will work on a multi layer architecture:

- **Client layer**: will display the data and manage the interaction with the users.
- **Proxy layer**: will provide load balancing and provide a secure channel with the clients.
- **Application layer**: it will run behind the proxy and provide all the methods for the client layer and attach to the DBMS.
- **Database layer**: it will provide the DBMS service for the Application layer

The client application will be the only way for the users to interact with SafeStreets. The same application will provide the features for the end users, the authorities and the police. Only the relevant tabs will be visualized based on the permission level of the user.
The app will provide only basic visualization and interaction, all the critical operations will be performed server-side.
The communication will be asynchronous to provide a better user experience, this may cause a temporary inconsistency of the displayed data but this will be tolerated. The requests from the client will be signed and sent through a channel encrypted with TLS that provides confidentiality and authenticity.
The system will be designed to run on one or more nodes based on the expected load. The application server will be deployed under a reverse proxy that will provide encryption, caching and load balancing.
The application server and the DBMS will we containerized to facilitate migration and replication.
The OCR service can be embedded int the application server container or deployed as a standalone component in another machine to improve the plate recognition times. Firewall rules between each component will guarantee that only the intended traffic reaches the containers.
The database storage will be encrypted it will accessible only by the application server, the authorities accounts must be manually inserted by a system administrator to prevent potential abuse.
Encryption of the data on the links between the internal components is optional if they are running on the same node and mandatory if the components resides on different nodes.

2.2 Component View

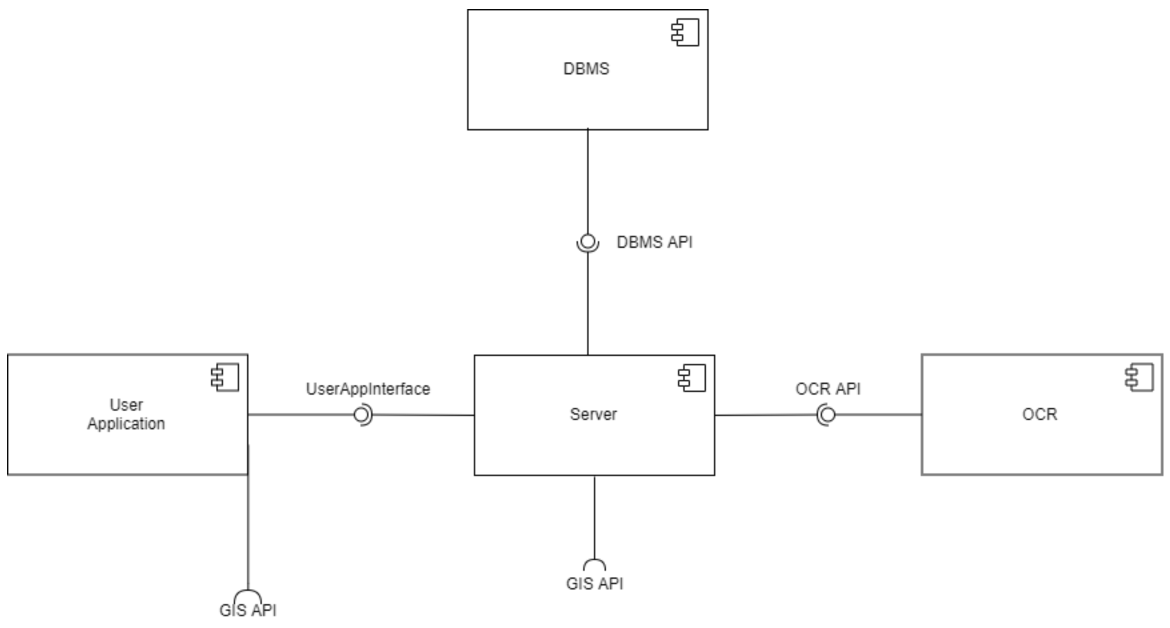

Figure 1: High-level components

In the previous figure we identify the high level components and interfaces. For the sake of simplicity we consider the DBMS as a unique high-level component even if it is distributed and fragmented over (at least) two different disks, User DB and Violation DB, in this way we are able to manage the load balancing by reducing the amount of accesses

- User Application
  - Register
  - Login
  - Report Violations
  - Get Statistic
  - Verify Users
  - Generate Traffic ticket

- DBMS
  - Store and retrieve data

- GIS API
  - Retrieve a reference to an up-to-date map centered on a given position
  - Get the latitude and longitude of a given location
  - Highlight to the aforementioned map

- OCR
  - Read license plates

2.2.1 Application Server components

The following diagram contains all the components (physical or logical ones) which belong to the Application Server.

- User Manager
  This component, as is shown in the figure, contains two subcomponents, Login Manager and Sign Up Manager, however it is also in charge to communicate with the User DB in order to retrieve information related to a specific User (e.g. if it is banned)

- Login Manager
  It is in charge to check whether the credentials provided by guests are correct by invoking the DBMS, it is important to point out that the DBMS, as we have explained in the previous paragraph, it's distributed and fragmented over two different disks, User DB and Violation DB, for the sake of simplicity let's consider them as a unique DBMS and the Login Manager will always interact only with the User DB.

- Sign Up Manager
  It is in charge to retrieve the credentials provided by the user and to store it in our User DB, it is also in charge to send confirmation emails when a guest has completed the sign up form.

- Violation Manager
  It is in charge to retrieve information and to satisfy tasks requested by the User Application by invoking the Violation DB, in fact, it is invoked during the report of a traffic violation when there is the need to store the information provided by the Logged User (insertViolation() or when there is the need to retrieve traffic violations in order to issue traffic tickets (getViolationsByLocation()).

- Statistic Manager
  It is in charge to check the effectiveness of our S2B with getTrafficTicketsRatio() which returns a float that is the ratio of the traffic violations to which a traffic ticket has been generated.
  It is also in charge to provide the statistic requested by the LoggedUser or LoggedAuthorityUser with : getStreetWithMostViolations() that gives back the Street within a municipality with the highest number of traffic violations, getViolationsForStreet(Street), getViolationsForVehicles().

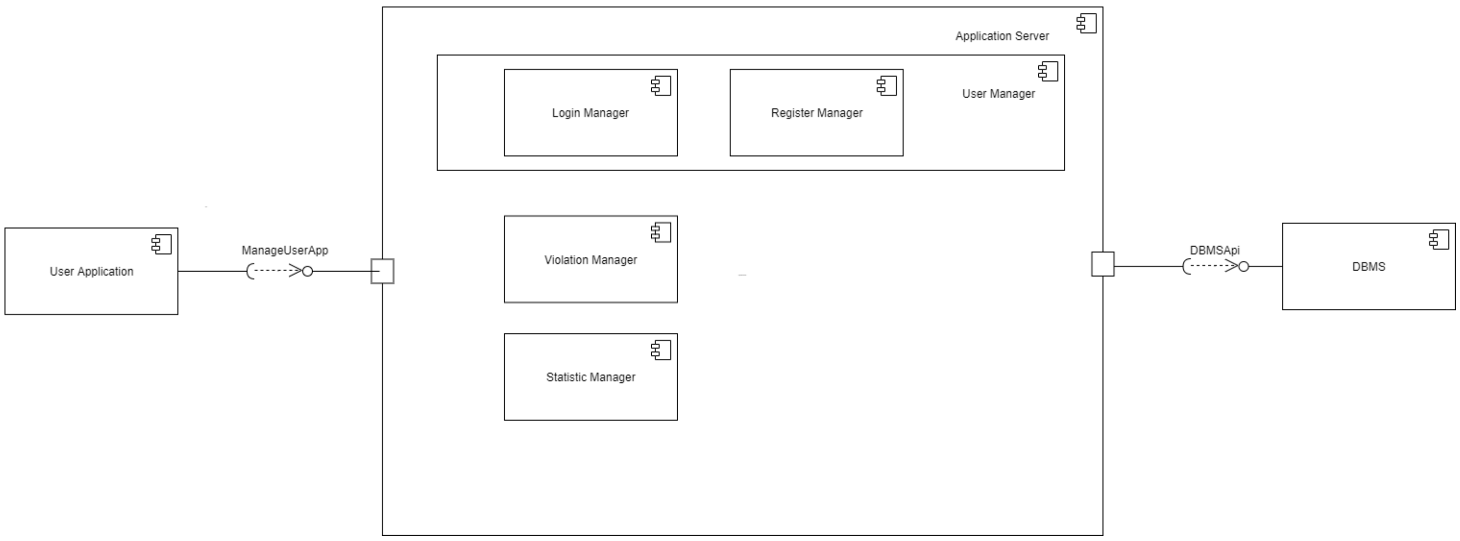

Figure 2: Components diagram

2.3 Deployment View

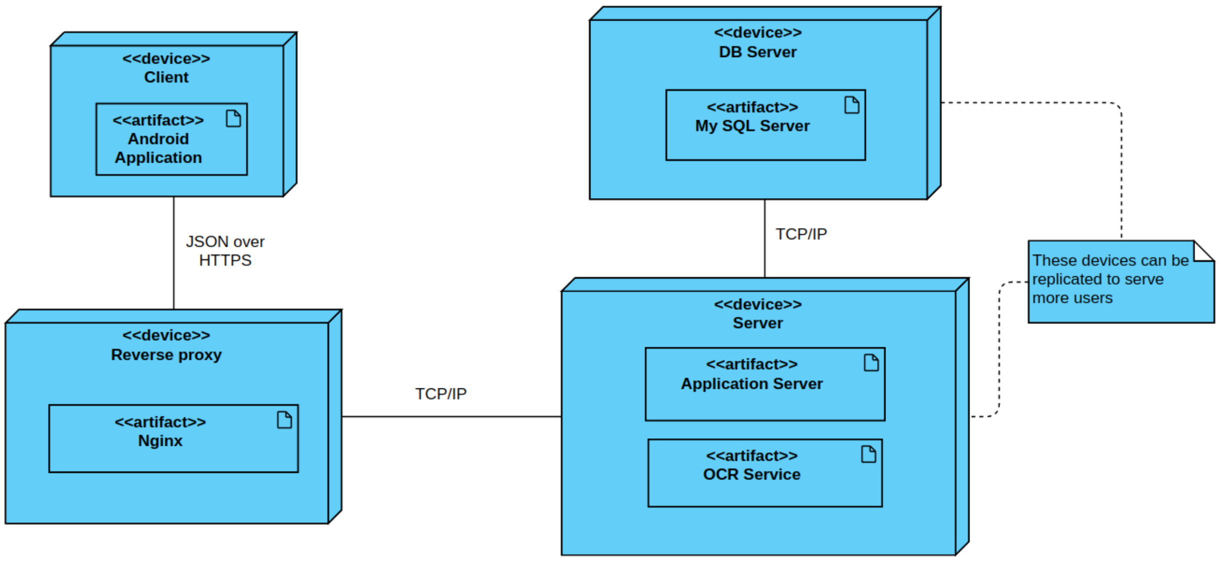

Figure 3: Deployment diagram

SafeStreets will be developed and deployed in a four tier logical architecture.

User Tier: consists in an Android application available for the download on the application stores. It will communicate with the server using the REST API with JSON and will rely on the Android system API to provide pictures, location and cryptographic primitives.
Tier 1: it contains nginx in a reverse proxy configuration that provides encryption and will provide load balancing if multiple application servers are deployed.
Tier 2: this node will contain the main Python application server which will manage the business logic and provide the REST API for the clients.
The OCR service will be deployed on the same node.
Tier 3: the My SQL DBMS will be deployed in the last layer providing the storage for SafeStreets.

2.4 Runtime View

In this section we represent some runtime views of the supposed interactions between the components of the SafeStreets system, for doing so we will use sequence diagrams.

2.4.1 Report Traffic Violation

Reports of traffic violations by a Logged User is the basic service of our S2B, we are assuming that the user is already registered and authenticated by an authority.
The logged user compiles the form for the report and after providing the mandatory fields he press the send button, in order to reduce the data exchanged over our links the client will firstly send an initial request attaching license plate,type of the violation and Timestamp of the traffic violation to the **Application Server**, then the **Application Server** invokes the **User Manager** and attach the User's unique ID in order to check whether the User is banned or not, hence the User Manager queries the DBMS that gives back a response.
After receiving it the Application Server sent a request using the method **violationExists**() to **Violation Manager** that queries the DB in order to check whether the traffic violation,that the Logged User want to report, is already stored.
If 'isBanned'= true or 'exists'=true the user hasn't the permission to report traffic violations and the process stops, otherwise the client can contact again the **Application Server** which by using **insertViolation**() contact the **OCR Service** that decodes the image in order to capture the license plate from the picture and gives it back to **Violation Manager** in order to check if it match, if it match the traffic violation will marked with 'correct' otherwise it will not, after that the **Violation Manager** will insert the traffic violation in the Violation Database that will return a status code to the client that will be shown in the client's device.

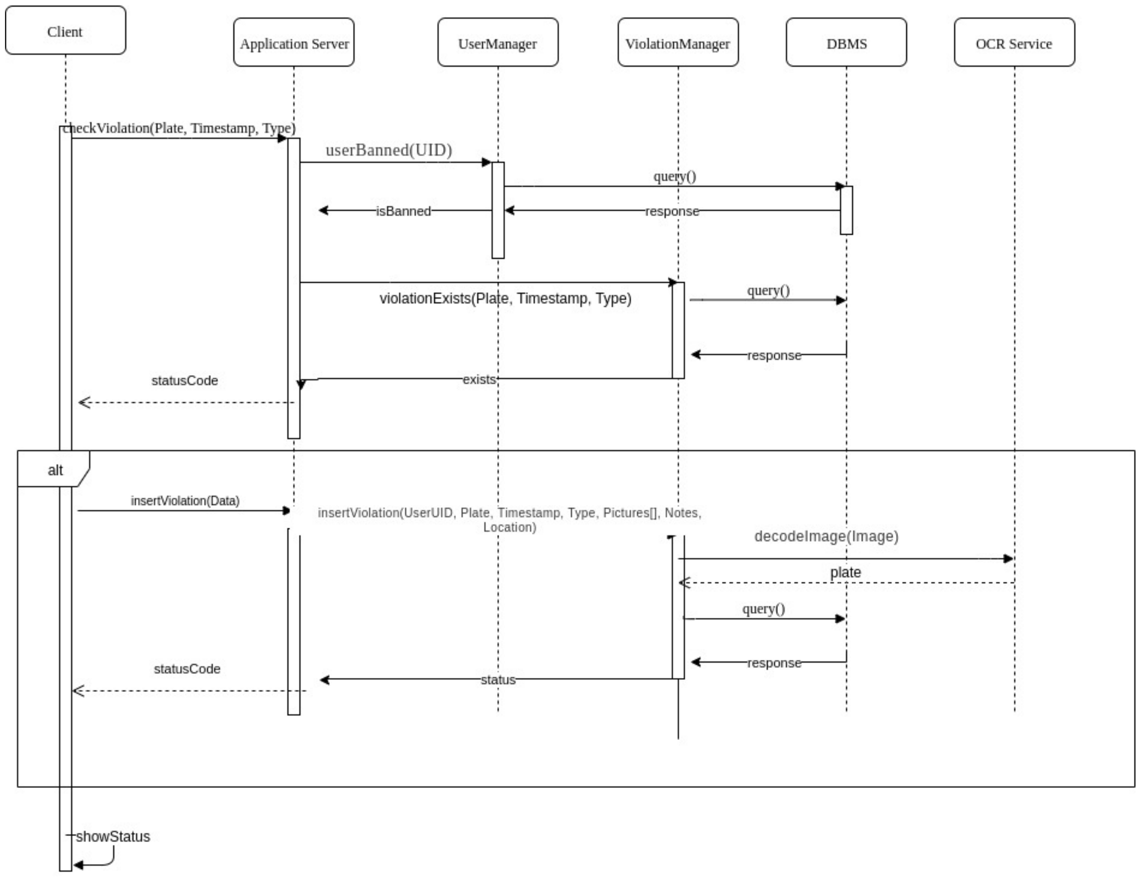

Figure 4: Report TrafficViolation

2.4.2 Get Traffic Violation

As a consequence of the report of a traffic violation by a LoggedUser now we have the need to access these stored data in order to issuing traffic tickets by Local policeman or for showing statistics about traffic violations.
The list of traffic violations within an area will be shown only to the LoggedAuthorityUser as already explained in the RASD, the LoggedAuthorityUser open the traffic violations tab on his device, on doing that the **Client** contact the **Application Server** which will contact the **Violation Manager** using the function getViolations(), then **Violation Manager** by using getUserLocation() get the Location of the LoggedUserAuthority by means of **User Manager** and return a list of Violations in the municipality of the LoggedAuthorityUser, moreover, the user can search for a specific area of his municipality by using a search filter, the message exchanged to achieve this request are similar to the mentioned before.
It is important to point out that during this acquiring phase of the data concerning traffic violations, we exchange only the fundamental fields of the traffic violation not including pictures or other heavy information.
These informations will be retrieved in the next phase as is shown in the sequence diagram of GenerateTrafficTicket.

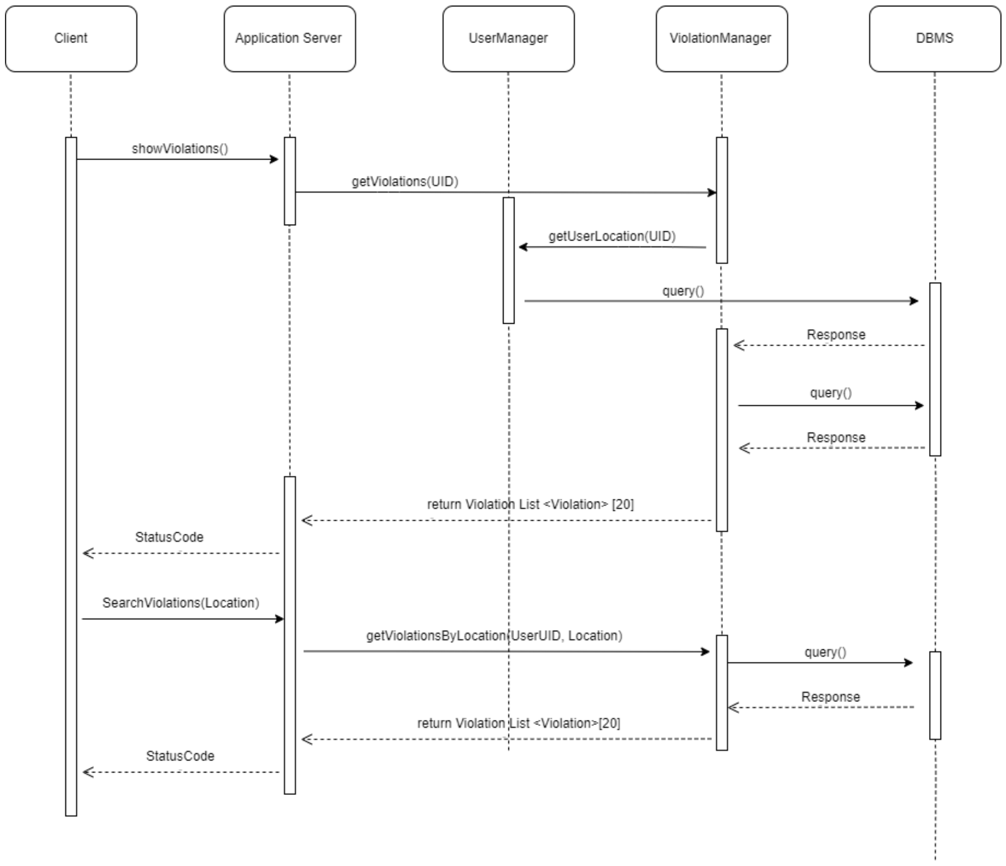

Figure 5: Get Traffic Violation

2.4.3 Generate Traffic Ticket

The issuing of traffic ticket is a consequence of the process explained before. A loggedAuthorityUser after querying for the traffic violation in his municipality, by clicking on a specific traffic violation, gets additional data concerned a specific traffic violation by means of the method getSpecificViolation(), the **Application Server** hence contact the **Violation Manager** which queries the DB and gives back the complete Traffic Violation structure.
After receiving the data the LoggedAuthorityUser (type Local Police) will generate the traffic ticket.
After this process in order to maintain the DB consistent the client by means of setGenerated() contact the **Application Server** which invokes the **Violation Manager** to set the flag Generated=true, after that a status code is sent back to the client.

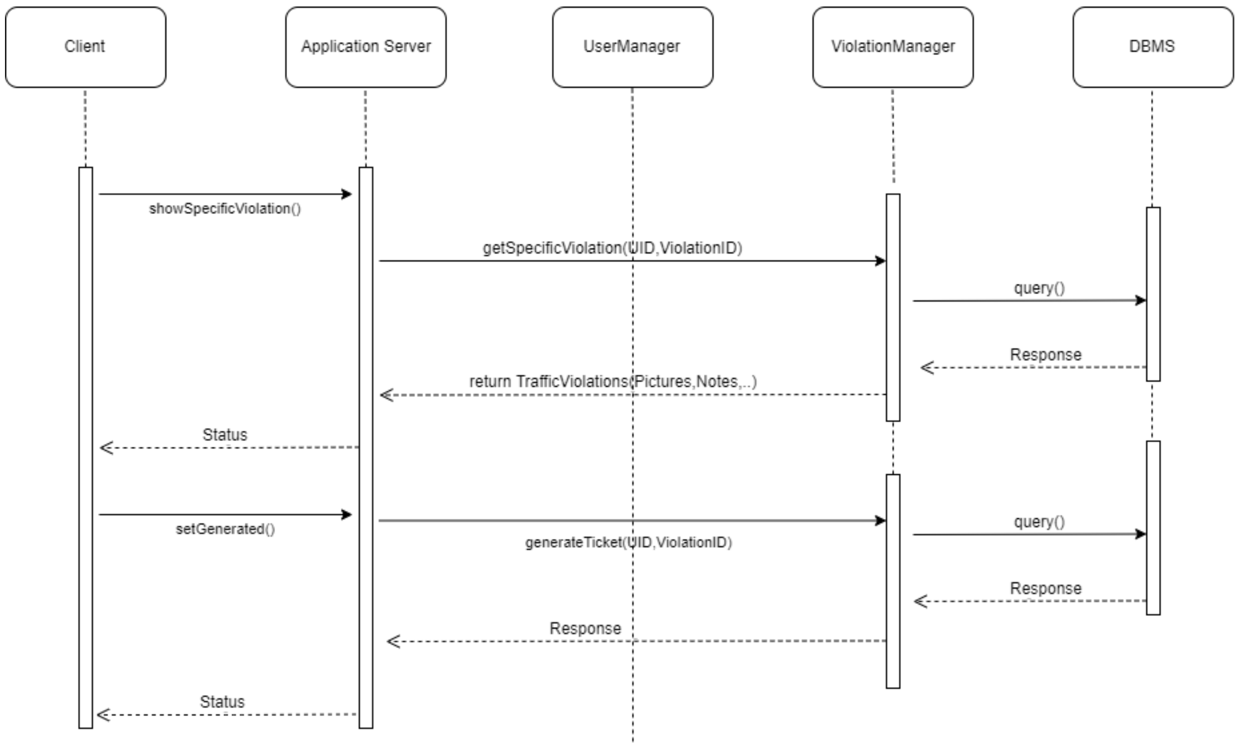

Figure 6: Generate Traffic Violation

2.4.4 Verify Users

As mentioned in the RASD the LoggedAuthorityUser is in charge to verify the document ID of the Users and their identity, for doing that the LoggedAuthorityUser opens the Verify Data tab in his device and the S2B will show the first 20 UnverifiedUser (User with flag verified set false), to show them the **Application Server** contacts the **User Manager** which queries the DB and gives back the list of Unverified Users.
It is important to point out that in order to reduce the amount of data exchanged over the links we will retrieve only few attributes of the requested Users, leaving heavy attributes for a future specific request.
After that the LoggedAuthorityUser (Client) by clicking on a specific unverified user contacts the **User Manager** which queries the **DB** for the missing attributes and the complete User structure is showed in the device, after the verify by the Client the S2B will set true the flag for that specific user.

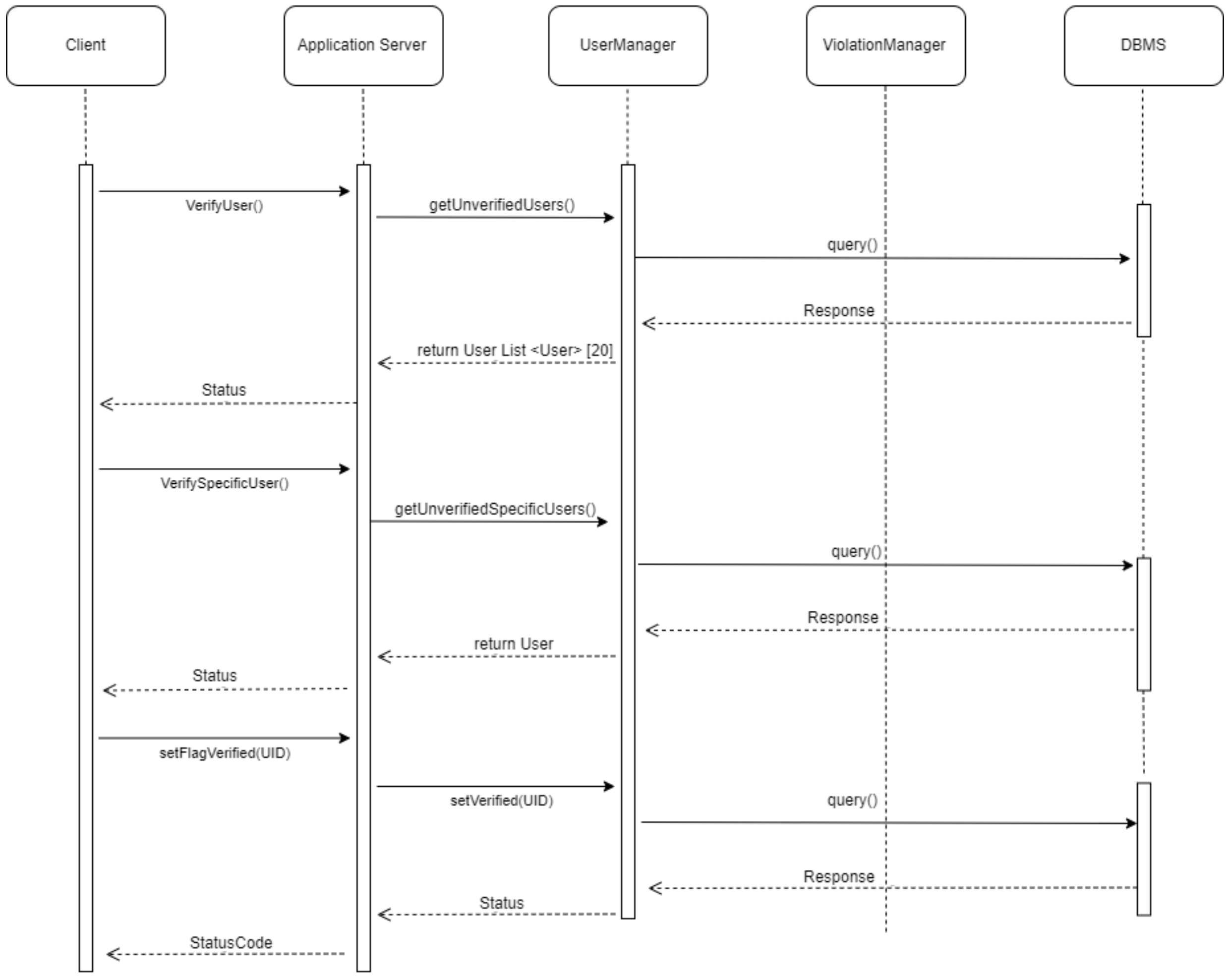

Figure 7: Verify Users

2.4.5 Get Statistics

LoggedUser and LoggedAuthorityUser are able to request different type of statistics, each statistic will have its method, however the messages exchanged over the links are almost the same.
The client (LoggedUser and LoggedAuthorityUser) open the statistic tab, the **Client** contact the **Application Server** which by means of getStatistic() invokes the **Violation Manager**, it queries the **DB** that gives back the Statistic class that is showed in the Client's device.

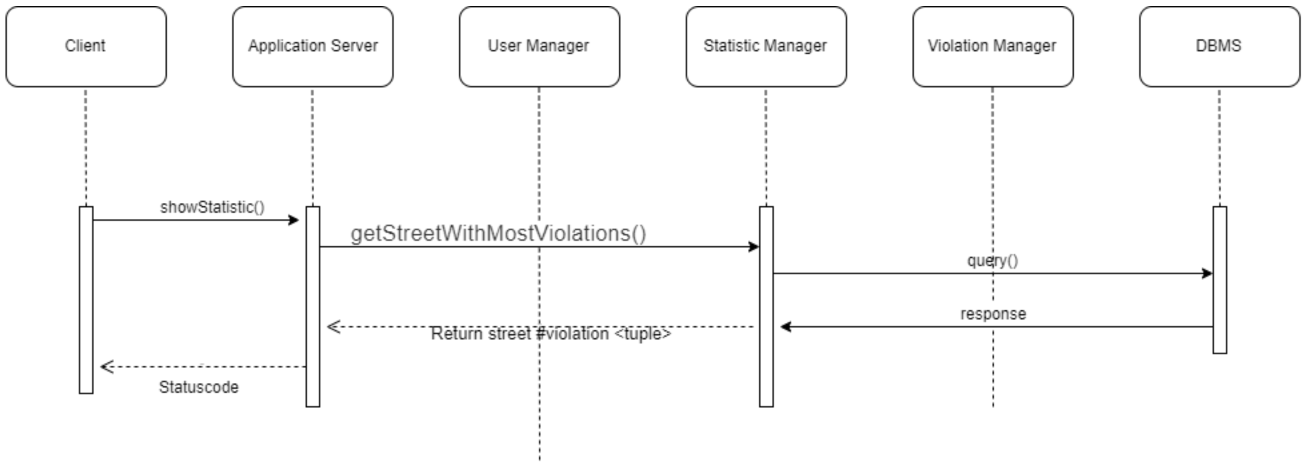

Figure 8: Get Statistics

2.5 Component interfaces

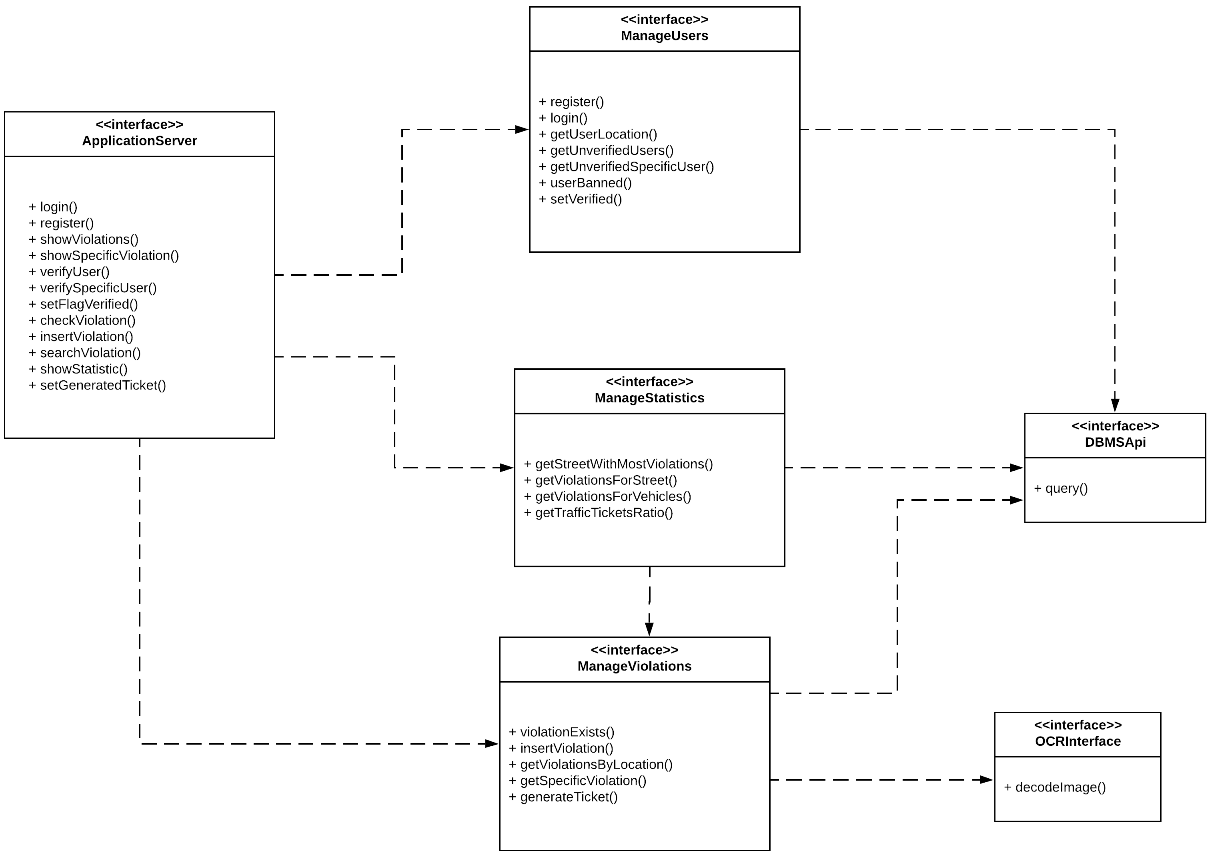

Figure 9: Component interfaces

In the above diagram are represented the interfaces of the server components. The user application communicates only with the application server interface which then routes the requests transparently to the other components. This approach allows to simply update only the Application interface to add more components or add other methods to the existing ones. Behind the Application Server interface we find the three major components of SafeStreets:

1. Manage Users
2. Manage Violations
3. Manage Statistics

2.6 Selected architectural styles and patterns

RESTful The RESTful architecture has been chosen because, nowadays, it's widely used for programming web application and is supported by different platforms. It's easy to develop and offers great flexibility to the programmer by hiding implementation details and decentralizing system's components. One of the cons of a RESTful architecture is that on a given time the system may see inconsistent data, for example, different clients may see different information on their device. We decided to compromise on this constraint for the sake of scalability and Reliability.

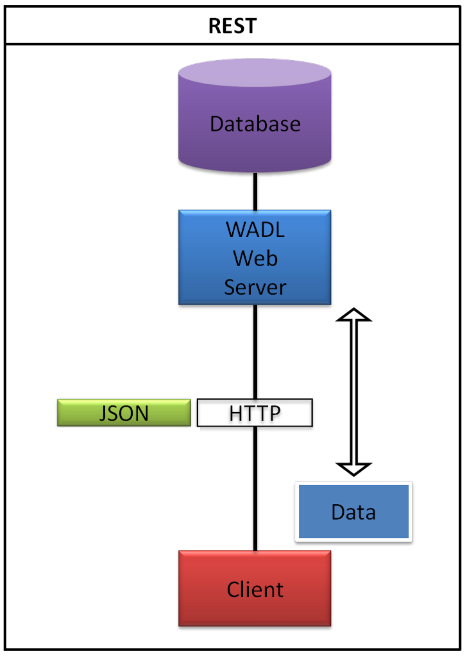

Figure 10: REST architecture

DB SafeStreets will rely on relational databases fragmented, distributed and replicated for the sake of fault tolerance and load balancing. The S2B use relational databases as a consequence of the need to have relational connection among the tables, in order to cross the stored information in our Databases, a clear example occurs whenever a user requests the Server and then the DBMS for a specific statistic.

3 User Interface Design

The mockups of the application were already exposed in the RASD document in the appropriate section (see appendix). For adding more details of the S2B in this section we provide the UX Diagrams to show the flow which the customer will follow to navigate inside the application, in accordance with the mockups in the RASD. The UX Diagrams below contain more additional interfaces that are not showed in the RASD in order to make more understandable the flow of events.

3.1 LoggedUser Interfaces

As we already explained, we have two kind of users which interface with our application, LoggedUser and LoggedAuthoriyUser with different kind of visibility in the statistic tab and different permission to access the S2B functionalities. The UX Diagram below shows the flow of events for what concerns a LoggedUser.

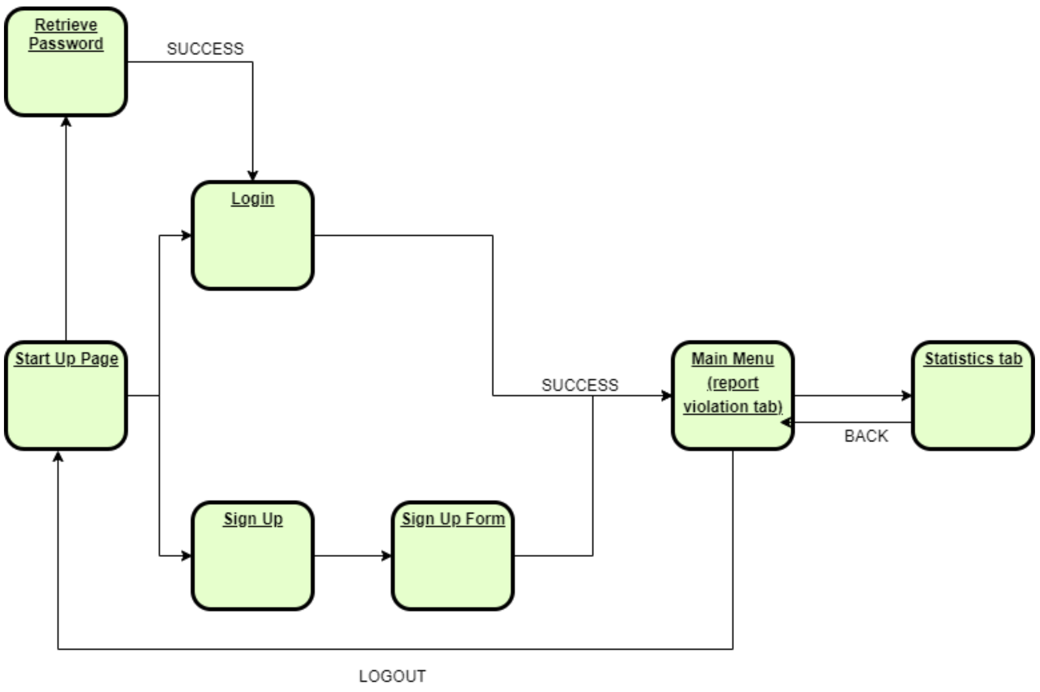

Figure 11: LoggedUser UX Diagram

3.2 LoggedAuthorityUser Interfaces

The UX Diagram below shows the flow of events for what concerns a LoggedAuthorityUser. A remarkable difference between the two UX Diagrams it's the lack of a sign up form interface for the LoggedAuthorityUser, this is caused because LoggedAuthorityUser accounts are directly provided and certified by our S2B.

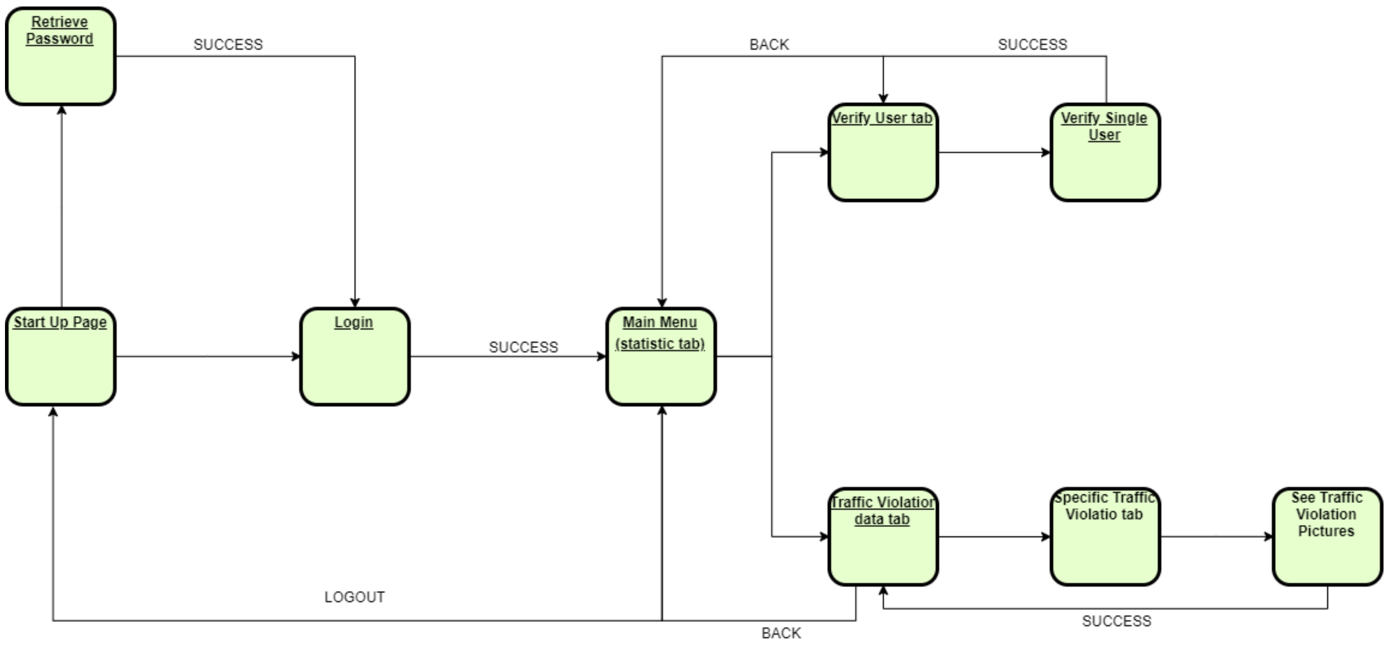
Retrieve Password
SUCCESS
BACK
SUCCESS
Verify User tab
Verify Single User
Start Up Page
Login
SUCCESS
Main Menu (statistic tab)
Traffic Violation data tab
Specific Traffic Violatio tab
See Traffic Violation Pictures
LOGOUT
BACK
SUCCESS

Figure 12: LoggedAuthorityUser UX Diagram

4 Requirements Traceability

In the table below is presented a mapping correspondence between the requirements defined in the RASD related to each goal and the components identified in the server component diagram.

4.1 Functional requirements

Requirements of goal	**Components**
G1 Allow a user to send data about traffic violations	Client Application Server User Manager Violation Manager OCR Service DBMS
G2 Allow the authorities to access to violation data	Client Application Server User Manager Violation Manager DBMS
G3 Allow the authorities to see specific kind of statistics about traffic violations	Client Application Server Violation Manager DBMS
G4 Allow the user to see specific kind of statistcs about traffic violations	Client Application Server Violation Manager DBMS
G5 Allow the local police to use SafeStreets system in order to issue traffic tickets	Client Application Server Violation Manager DBMS
G6 Allow to exploit the traffic ticket statistics to check the effectiveness of SafeStreets	Application Server Violation Manager DBMS

Table 1: Mapping goals on components

4.2 Nonfunctional requirements

Performance requirements To guarantee a short response time we have tried to decouple components in order to enable an instance pooling management of components and so an as much as possible concurrent management of requests.
During all the design process we also have kept in mind the scalability of the software w.r.t. the number of report of traffic violations trying to keep a linear complexity factor.

Availability To ensure availability requirements server will be running 24 hours for day. The architecture is designed with the purpose of enabling a redundancy architecture if availability constraints would make it necessary.

Security No security critical function will be performed client-side.
Secure protocols are used for transmission and storage of sensitive data.
Only the system administrators will have direct access to the nodes for maintenance.

Portability Users access the system through a web-based application that enables the portability and a potential cross-operative system compatibility.

5 Implementation,Integration and Testing

5.1 Implementation

Below is showed the list of features which are going to be developed. They are meant for multiple teams which work simultaneously. The System will be developed by using a bottom up approach in the following order:

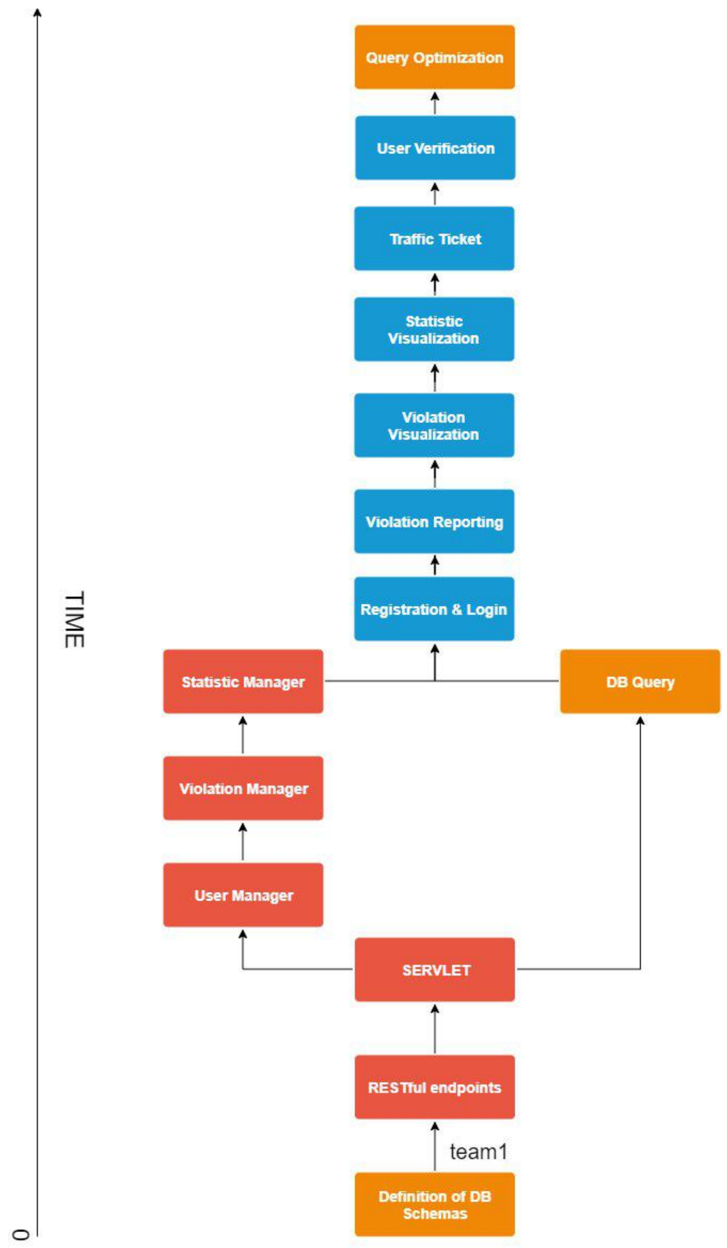

Figure 13: Single team bottom up approach

While for multiple team (e.g. 2 teams) the order in which the components will be developed is shown in the figure below.

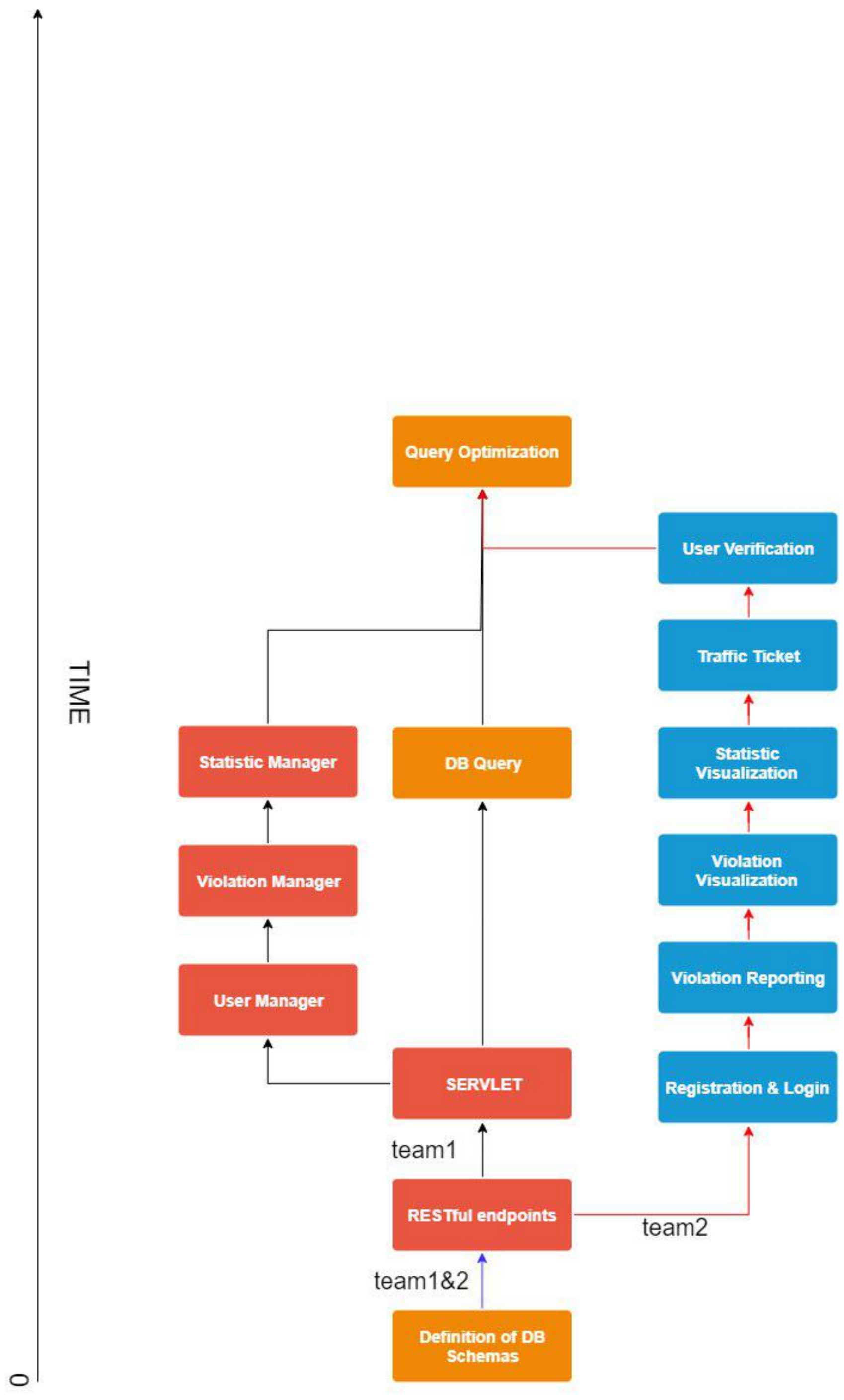

Figure 14: Single team bottom up approach

Android Application

- Registration and Login
- Violation reporting
- Violation visualization
- Statistic visualization
- Traffic ticket generation
- User verification

Application Server

- RESTful endpoints
- Servlet
- User Manager
- Violation Manager
- Statistics Manager

Database

- DB schemas
- DB queries
- Query optimization

DB schemas The first task will be producing the ER model of the database and the subsequent SQL schemas on top of which the application will be built

RESTful endpoint The endpoints of the REST application will be defined in the early stages of the development to provide a common interface for building simultaneously the Android Application and the Application Server

Servlet This task consists in the creation of the skeleton of the Application Server which will allow to easily deploy and test the other components

User Manager The User Manager must be the first component to be developed since the subsequent components will rely on the access control and the data provided by it.

Violation Manager The Violation Manager is the core component of SafeStreets, the main feature that must be developed first is the violation reporting followed by the traffic ticket generation. The last feature to be implemented is the license plate recognition using the external OCR Software

Statistics Manager This is the last component that will be developed since the statistics it provides are not critical to the system's main function.

Statistics Manager This is the last component that will be developed since the statistics it provides are not critical to the system's main function.

DB queries After the finalization of the data structures needed by the application the queries to the DBMS will be developed to provide the required data efficiently

Android Application The development of the Android application can proceed in parallel based on the the common REST endpoint. For the development a mock server can be used.

5.2 Integration and Testing

After the components have been developed they are unit tested and then integrated with each others and tested again in order to check the reliability and performances of our S2B in both of cases. In this phase the performance will traced and the queries to the DB will be optimized to reduce the load on the servers. After this phase the recommended non-functional requirements for the machines will be defined. We stress the S2B by means of https://locust.io/, which is a Software System that performs load testing by sending thousands or millions of requests and measures the time required to satisfy those request under stressed conditions.

6 Hours of work

This is the amount of time spent to redact this document:

- Lorenzo Amata: $\sim$ 52 hours
- Frederik Saraci: $\sim$ 48 hours