POLITECNICO
MILANO 1863

**Safestreets project**

# Design Document


**Arash bahariye: 10596378**

**Niloofar salimi: 10648072**


---

Deliverable: DD

Title: Design Document


Authors: Arashbahariye, Niloofar Salimi


Version: 1.1

Date: 15 Dec 2019

Download page: https://github.com/arashbahariye/BahariyeSalimi.git

---

## Table of contents

# 1. Introduction

### 1.1 Purpose

This document will go into more detail on architecture and design of the software to be developed. With referencing the requirements present in RASD file, design and the architecture details will be explained in detail. General architecture, components and their interactions, classes, runtime sequences, implementation, integration and testing will be explained in high detail. The order of the sections is as follows:

- Overall Architecture
- Component Design
- Component Interfaces Services
- Runtime Views with Sequence Diagrams
- Deployment View
- Implementation and Testing Plan
- Integration and Testing Plan

### 1.2 Scope

As described in the RASD document, SafeStreets provides three services but for the purpose of this project we delve into two services. SafeStreets is an application that intends to bring public participation into reducing traffic offences, in particular parking violations. The application allows users to send pictures of violations, including their date, time, and position to system. There are 2 possible services, basic service and advanced service.

- for the basic service SafeStreets stores the information provided by users. In particular, when it receives reports of violations. the user inserts the plate number of the rouge vehicle along with a picture and scan of an RFID which has been assigned to each car by municipality and also the type of violation, which select by users and the name of the street where the violation occurred and also can be retrieved from the geographical position of the violation. In addition, the application allows both common users users and municipality to mine the information that has been received, for example by highlighting the streets (or the areas) with the highest frequency of violations, or the vehicles that commit the most violations. Of course, different levels of visibility could be offered to different roles.

- In addition, the municipality (and, in particular, the municipal agent) can benefit from a service that takes the information about the violations coming from SafeStreets and generates traffic tickets from it. In this case, mechanism should be put in place to ensure that the chain of custody of the information coming from the users is never change, and the information is never altered (e.g., if a manipulation occurs at any point of the image showing the violation, for example to alter the license plate, the application should discard the information). In addition, the information about issued tickets can be used by SafeStreets to build statistics.

### 1.3 Acronyms and Abbreviations

- DD: Design Document.
- RASD: Requirement Analysis and Specification Document.

- DB: Database.
- API: application program interface
- TDD: Test Driven Development.

### 1.4 Document's Structure

This *DD* project is composed by 6 chapters and appendix:

1. *Introduction* which contains the description of the given problem and basic information about this document in order to provide better understanding of the next chapters.
2. *Architectural Design* which contains an overview, high level architecture description, component view, class diagram, deployment view, sequence diagrams, component interfaces description and definitions and design patterns.
3. *User Interface Design* directs the user to the *RASD* file, where the user interfaces are presented.
4. *Requirements Traceability* relates the requirements and goals described in the *RASD* file to components presented in this file.
5. *Integration plan* which describes Implementation, Integration and Testing Plan which describes the need for approach to develop the described problems. Also Includes implementations and testing description and needed integration.
6. *Effort Spent* contains the relation of time spent in the project for each member of the group.

## 2. Architectural Design

### 2.1 Overview

A 3-tier architecture is a type of software architecture which is composed of three "tiers" or "layers" of logical computing which we use it for designing the architecture of our application. They are using in applications as a specific type of client-server system. 3-tier architectures provide many benefits for production and development environments by modularizing the user interface, business logic, and data storage layers. Doing so gives greater flexibility to development teams by allowing them to update a specific part of an application independently of the other parts. That is why we choose this method.

• **Presentation Tier:** The presentation tier is the front-end layer in the 3-tier system and consists of the user interface. This user interface is often a graphical one accessible through a web browser or web-based application or a mobile application and displays content and information useful to an end user. This tier is often built on web technologies such as HTML5, JavaScript, CSS, or through other popular web development frameworks, and communicates with other layers through API calls. In our project UI is developed using XCode.

• **Application Tier:** The application tier contains the functional business logic which drives an application's core capabilities. It's often written in Java, .NET, C#, Python, C++, etc. in our project we exploit swift.

• **Data Tier:** The data tier comprises of the database/data storage system and data access layer. Examples of such systems are MySQL, Oracle, PostgreSQL, Microsoft SQL Server, MongoDB, etc. Data is accessed by the application layer via API calls.

### 2.2 High level architecture

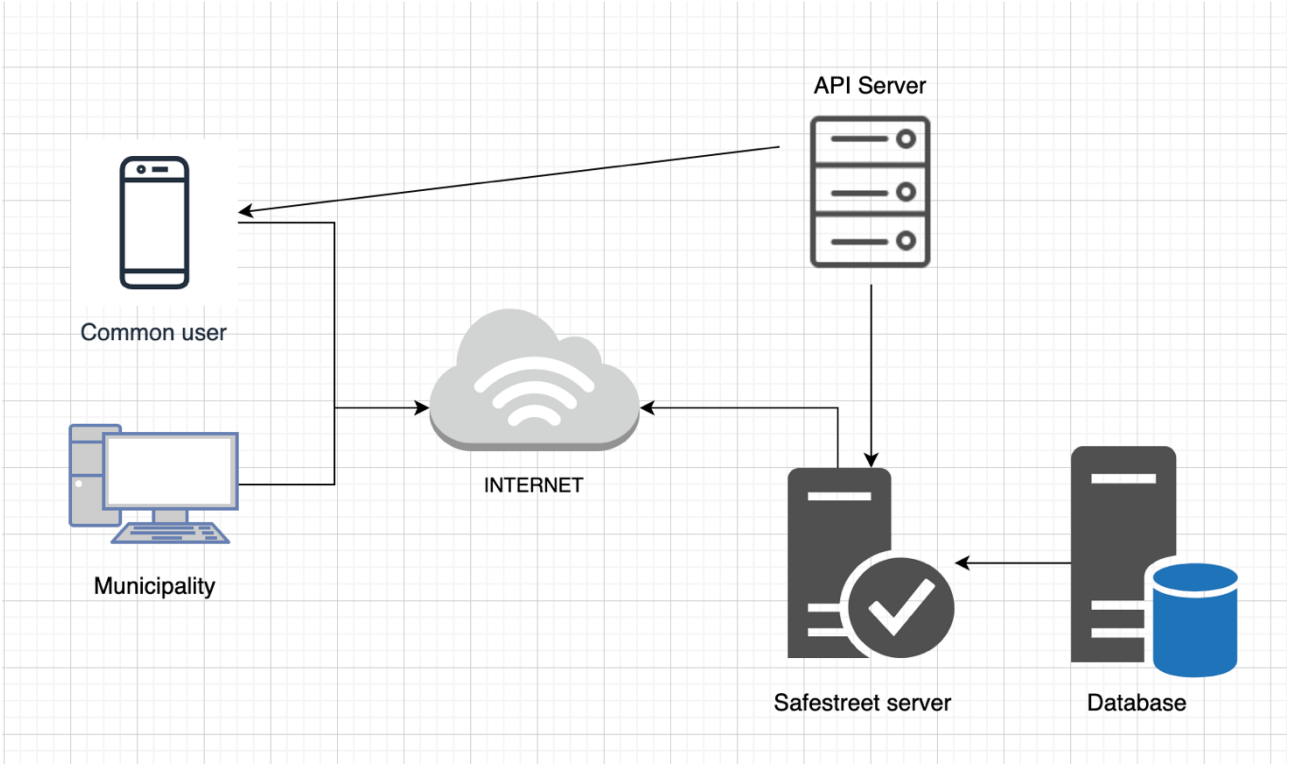


Figure 2.1: High Level Architecture Diagram.

The figure 1 shows the general architecture of the system. Both Registered User and municipal agent nodes are connected to the Internet in order to communicate with the External Server and Application Server nodes. Registered User and municipal agent are registered to the application in order to communicate with the application server. Application Server is the part of the program that encodes the business logic that determine how data can be created, stored, and changed. On the other way it is the logical part of the system. Application server uses the services which External Server provides. Application Server is connected to the Database Server to have access to the data. In the Database server all the data, information, pictures and reports which used in the system will be stored, such as the personal information of Registered User.

## 2.3 Component view

### 2.3.1 High Level Components Diagram

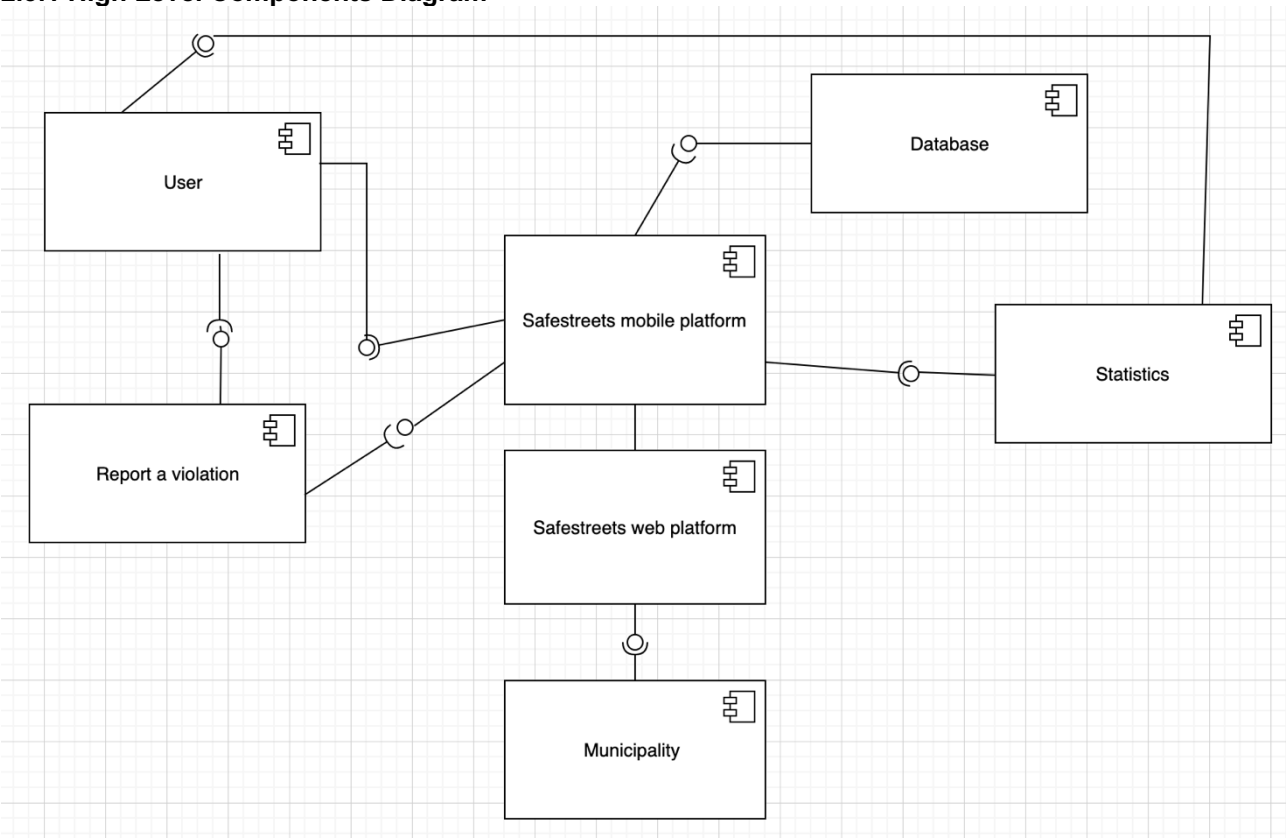


Figure 2.2: High Level Components Diagram.

### 2.3.2 Common user components diagram

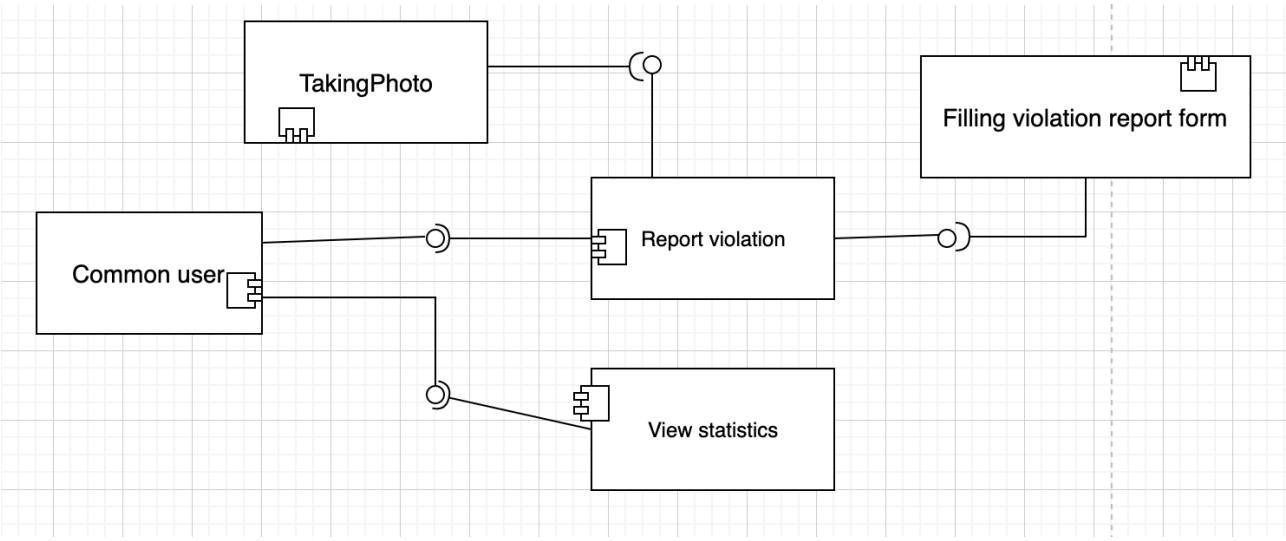


Figure 2.3: Common user Components Diagram.

## 2.4 Deployment View

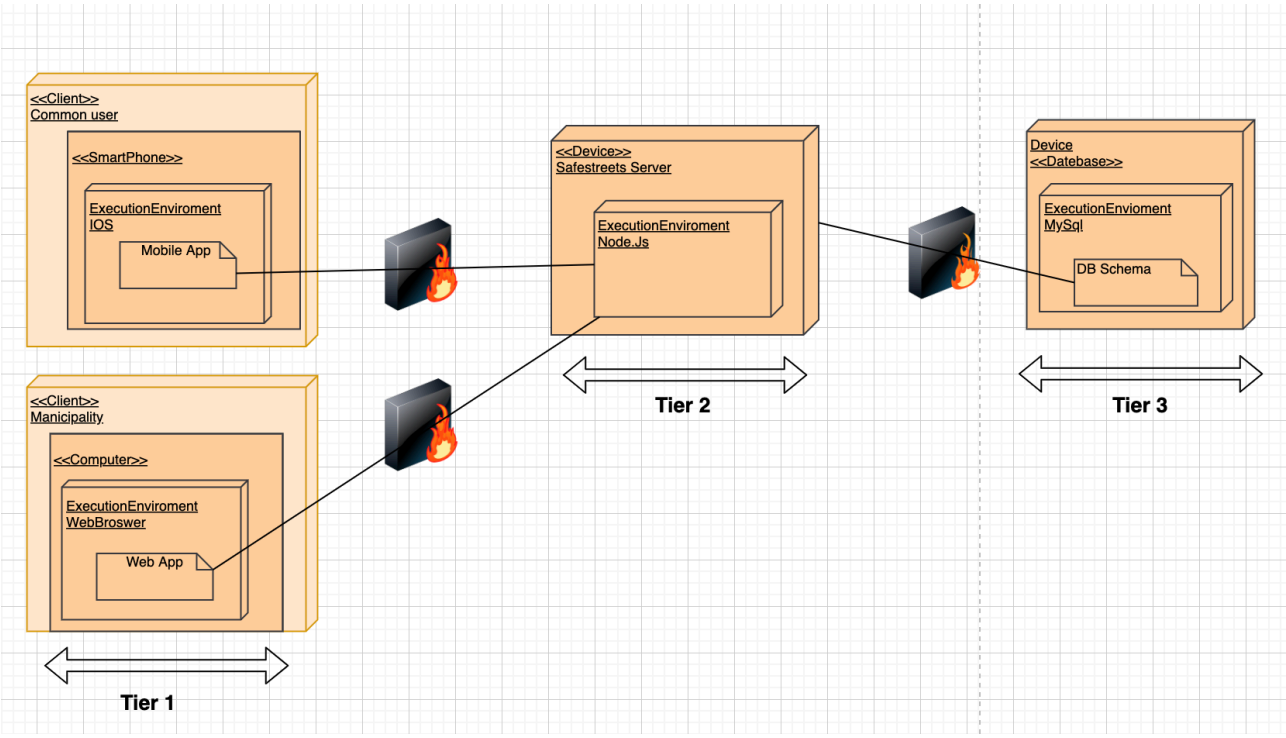


Figure2.4: Deployment View Diagram.

This view describes the environment into which the system will be deployed, including the dependencies which the system has on its runtime environment which can be seen in the figure 2.4.

The four boxes represent nodes, either software or hardware. Physical nodes should be labeled with the stereotype device, to indicate that it's a physical device such as a computer or smart phone.

Common user: This node consists of one node which is a Smartphone. Smartphone is a device which used by user and communicates directly to the Safestreets tier but since the information is critical, a firewall is needed between the Safestreets Server.

Municipality: an entity who is able to use web application by a web browser. They are another entity who communicates with Safestreets application but with different privileges and different platform.

Safestreets server: It has relation with all other nodes of the system. In this node, all the computation of the system will be done, hence this node is logical part of the system. Application server obtains data from the common users of the system and stores information in database server and manages data requests.

Database server: All the data and information which used in the system will be stored in the Database server such as the personal information of users. Between Safestreets Server and DB Server, a firewall will be used to ensure the integrity of the data transfers.

## 2.5 Runtime view

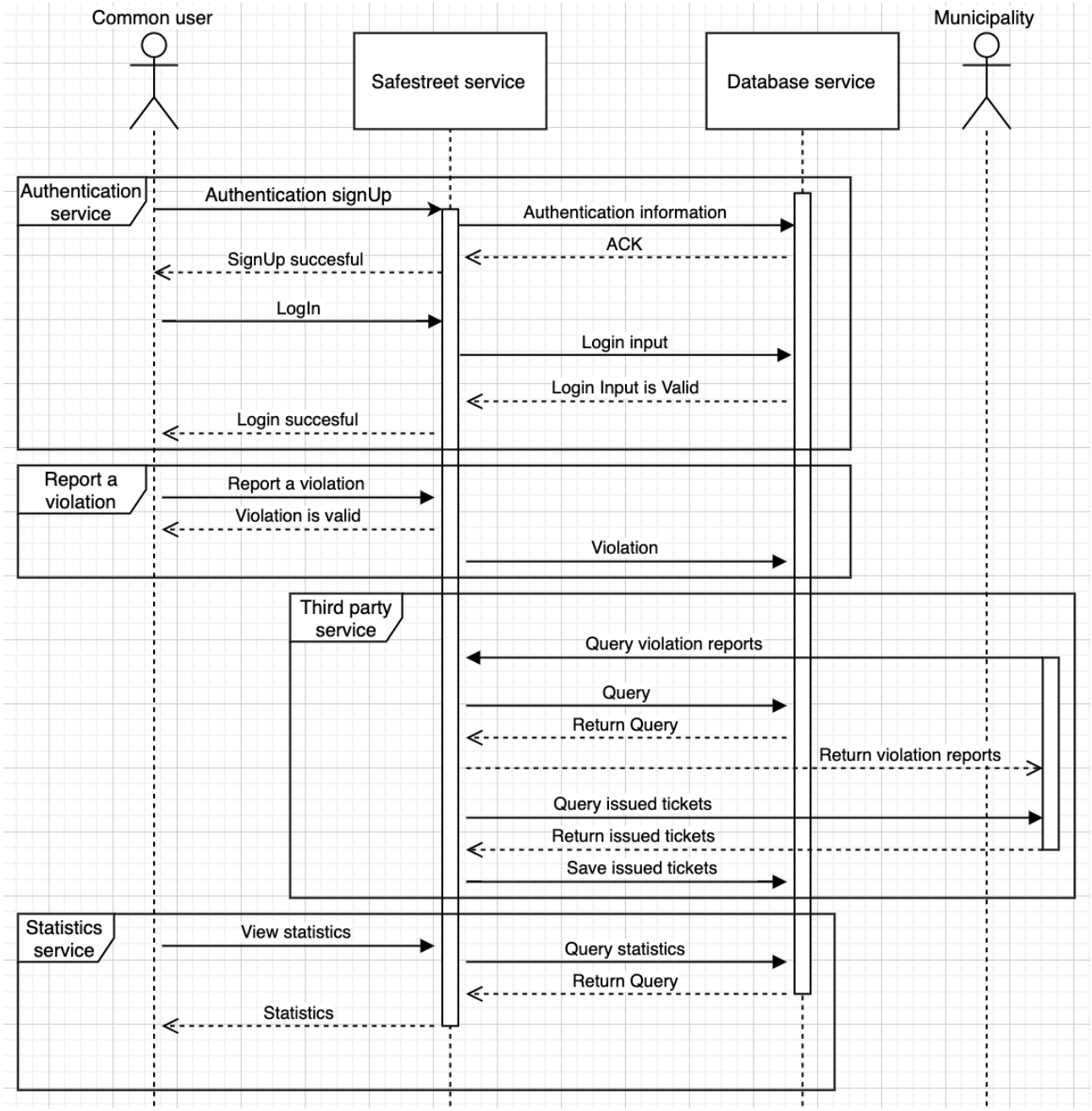


Figure 2.5: Runtime view diagram

The overall run time view of the entire system is depicted in figure 2.5. the workflow is as follows:

1. Authentication service:
    a. Common user first has to Sign up in the system by sending its request to Safestreet server. Then Safestreets servers save these informations into its databases.
    b. After successfully signing up into the system common user can login to the system. Common user sends a login request to Safestreets servers along with login information. Safestreets then validate existence of these

information through its databases and if successful it gives access to common user to be able to enter the system.

2. Report a violation:
    a. Common user sends report a violation request along with its information to the Safestreets servers.
    b. If the format of the reported violation is valid Safe street servers will save that report into databases.
3. Third party service:
    a. Municipality query for violation reports through its web platform by sending “query violation reports” message to safestreets servers.
    b. Safestreets servers query the request from the databases and forward queries to the municipality.
    c. Municipality exploit reports to issue tickets.
    d. Safestreets query from municipality the issued tickets. Then municipality reply to the query and Safestreets save the query reply into its databases.
4. Statistics service:
    a. Common user sends view statistics request to Safestreets server along with requested information.
    b. Safestreets servers query that request from its databases and forward the query to common user.

## 2.6 Selected architectural styles and patterns

### 2.6.1 Overall Architecture

As it has been explained in high level architecture, application will use three tier architecture approach. This choice will improve the applications maintainability and complement teamwork in high number of people.

Presentation Layer will be the most lightweight tier to ensure that mobile devices and smartwatches which have limited resources will not have any performance issues. Some of the most recent frameworks for both web applications and mobile applications are an excellent choice for this tier. Angular and Native Script will be used as the main front-end frameworks for web-based platform for municipality and XCode for mobile platform for common users.

Application layer will handle the core functionalities of this project. Without compromising security in mind, this layer will be able to handle high number of requests and responses with the help of Node.js and Express.js frameworks. With asynchronous task management, the application layer will not require a high specked hardware. This will reduce the maintenance costs significantly as well.

Data Layer is the data storage and data management tier. PostgreSQL database server will be used since it is completely free, and this is a big plus for the reduced costs of the project.

Of course, the separation of the layers will introduce performance drawbacks according to the data transfer between layers. But for maintainability and reusability in mind, this layered architecture is a much better choice in the whole development and runtime lifecycle.

### 2.6.2 Design Patterns

Client-Server Architecture: This separation of client and server will improve the performance of the client side. Since all of the business computations will be run on server, client will have a lightweight application. Also, this decoupling also improves development process with letting developers work without knowing what the business logic is running on server side.

MVC pattern in Angular: Angular framework introduces a MVC pattern which is very useful for decoupling of the components and request services. Three modules can be used for the development process:

- Components: This is the module which will mostly contain the data which is presented data certain time. These components can also reuse and extended according to the needs of the application. TypeScript is used to develop these modules.
- Views: This type of module is responsible for handling the presentation of the data to user. HTML and CSS is used to develop these modules which is mostly related to design for the user interface.
- Services: This module is handling the request-response cycle of the web application. All data requests from the components are sent to these modules and data transfer is always going through these modules. The design pattern of this module is Observable. These services can be injected to other components or even other services. For example, client-side authentication will be done through authentication service and this service will be injected to every service which handles the data requests from the application server.

Node.js and Express.js: Node.js is a cross platform engine which can be developed with JavaScript language. Express.js is a REST server implementation. With these two tools, business logic of the application will be developed. Since Node.js has a single non-blocking thread design, asynchronous development is necessary for the performance of the application server. With properly architecture code, this server is able to handle very high numbered requests and responses with respect to a very low-end hardware.

## 3. User Interface Design

All of the user interface mockups are presented in the RASD file. There are no further detailed mockups needed in this section but for coherence of this document to be independent from RASD, here we depict the main UX design of the application. Here we present some UX diagrams to show the flow which the Customer will follow to navigate inside the application, in accordance with the mockups contained in the RASD. It should also be noted that here we only depict main tabs and pages on the application.

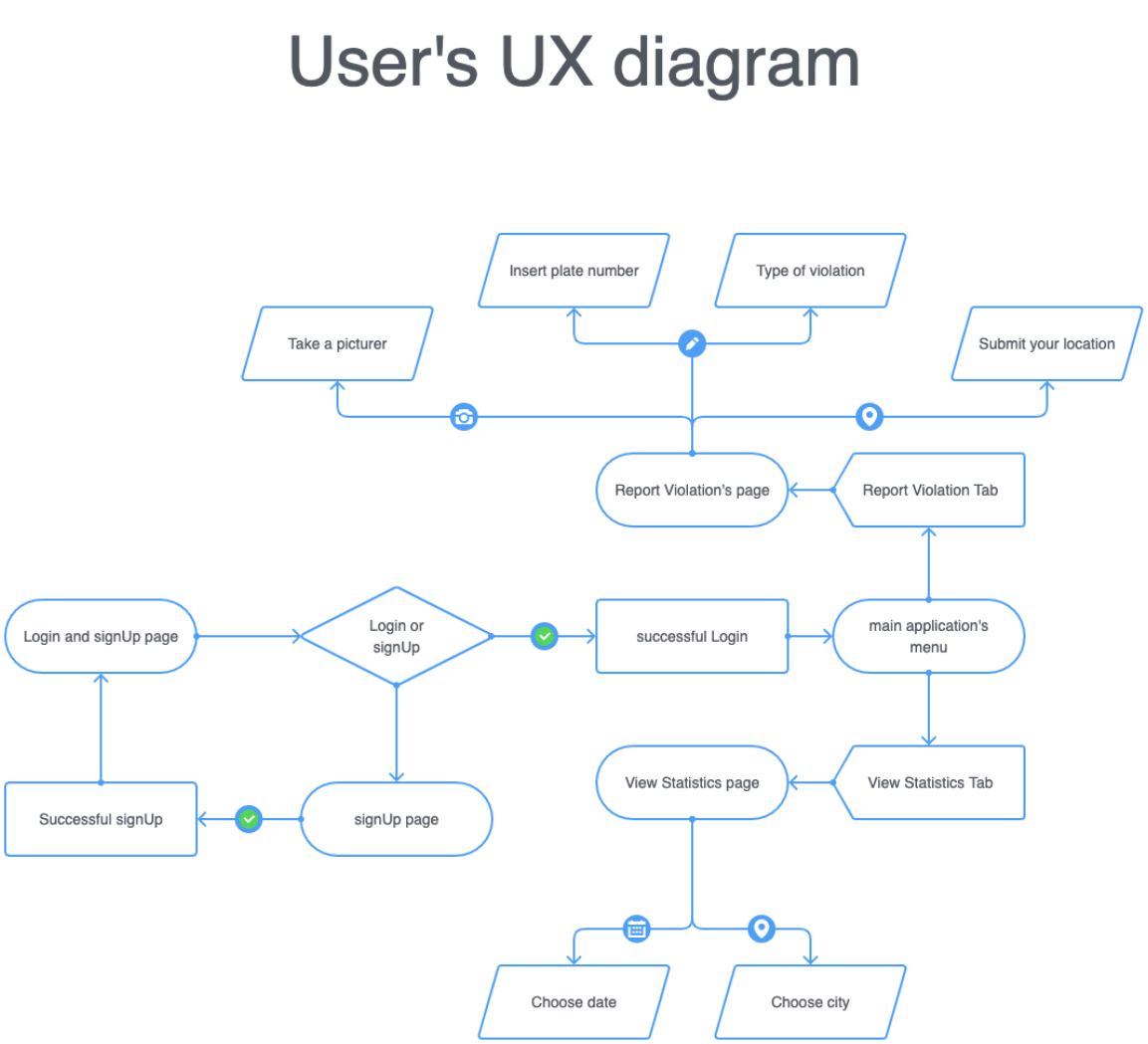


Figure 3.1 Common user UX diagram

## 4. Requirements Traceability

The whole design has been developed to guarantee that the system is able to enforce the requirements defined in the RASD (and, as a consequence, to achieve the goals defined in RASD). Here a mapping between those requirements and the design components in the application server that will ensure their fulfillment in a direct way (some other components that are indirectly needed to enforce some requirements are not mentioned in the list, but their role has been made clear through some comment) is shown:

[R.1]: Individuals and municipality can register to Safestreets services by providing identifiable information. (Goals G.1)

- Authentication Service

[R.2]: Allow users to report a violation. In specific Giving Access to use camera of his/her smart mobile phone, Providing a form for some additional information, such as the license plate number (it would be better to has this item which user can write license plate number in form), type of violation, details about location and vehicle's position, and other description. also Give access to users the location (it is possible by employing Map service to find the location of user and violation on the map and sending it to authorities).
(Goal G.2)

- Report a violation Service

[R.3]: Allow users to get useful information about general information and news about streets situation and Highlighting and warning most accident potential areas. (goal G.3)

- Statistics service

[R.4]: Give access the municipality the detail of violations. (goal G.4)

- Third-party service

[R.5]: Give access to municipal agent a panel for general announcements. (goal G.5)

- Third-party service

[R.6]: The system must reject the violation report if the place is not exact or if the photo is in poor quality or input information in specific plate number has altered. (goal G.2.2, G2.4, G.2.6, G.6)

- Report validation service

[R.7]: The system must communicate to the municipality system. (goal G.5)

- Third-party service

[R.8]: The system must be able to show zone with highest rate of violation based on type. (goal G.3)

- Statistics service

[R.9]: The system must be able to show car number with high rate of parking violation. (goal G.5)

- Statistics service

# 5. Implementation, Integration and Testing Plan

## 5.1 Implementation Plan

### 5.1.1 Agile Development Approach

This is a very popular approach for most of the software developed in recent years. Iteratively developing the functionalities of the software helps developers to focus only the important parts of the development lifecycle and improves the efficiency.

Agile approach will complement the development process since required features can be developed in each agile cycle. After each cycle, other requirements and any missing features can be determined and applied in the next cycle.

### 5.1.2 Server-Side First

The core business logic of the project needs to be implemented at first to ensure that presentation layer of the software will receive the correct representation of the data. Front-end layer will have to make API calls to get the data for presentation hence relevant back-end REST API has to be implemented first. After each completion of a component implementation, front-end part of the software will be implemented as well. This way developers can work simultaneously and reduce the development time significantly.

### 5.1.3 Top-Down Approach

In the implementation process of the core logic of the software, both Top-down approach will be used in the development process. Report-violation is the core component and other components rely on this component and services it includes. As Top-down approach in mind, Report-violation component will be developed at first to ensure that relying components will have a strong foundation. The following order of services will be implemented:

1. Report-violation
2. Reports validation service
3. Database Service
4. Statistics service
5. Third-party service
6. Authentication Service

Report-violation will be implemented first since most of the services will be using this service. Next report validation has to be developed since it is critical for both performance of the app and achieving desired goals, to have reliable and correct input data from users.

Database service has to be deployed in early stages as well since very important part of the application heavily relies solely on this service and that is statistical services which aim to give insights to both users and third parties.

Authentication Service is left to last since it is the last service to be implemented for individuals and third parties to register and use the services provided.

Below, the figure illustrates the decided approach.

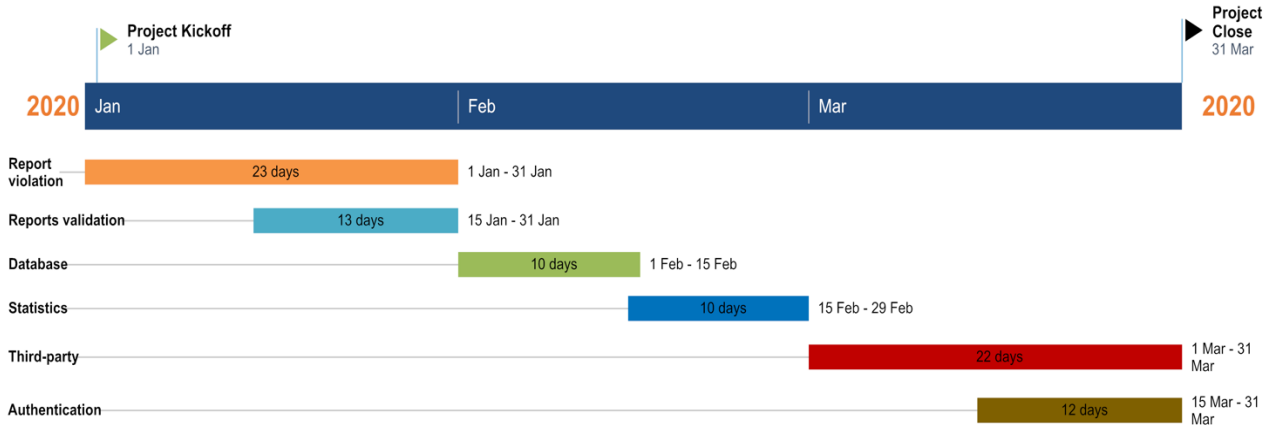


Figure 5.1: implementation Gantt chart

### 5.2 Implementation Testing Plan

Test Driven Development (TDD) approach will be used for testing of the implementation and integration of the software. For every service to be implemented, unit tests will be written first to ensure that implementation of the actual services will function properly. This approach also reduces any maintenance costs because of wrong implementation or addition of new features in the future. Also, this approach is excellent for continuous integration process for the software. For every feature to be developed, unit tests will be run before release and this will reduce the bugs which might not be encountered in development process.

For the components which require other components on development time, mock mechanism will be used to test the behavior of the required components, stubs will be used to test the state verification. These methods will reduce the development time since not all of the components will be developed at the same time and we can use mocks or stubs to replace the required components.

In case of any other functionalities are required by the stakeholders or any functionality must be added to ensure the requirements are fulfilled, same testing and implementation strategy will be applied. Firstly, tests of the functionalities will be written, then they will be implemented.

### 5.3 Integration Plan

Integration plan will follow the implementation plan with slight differences.

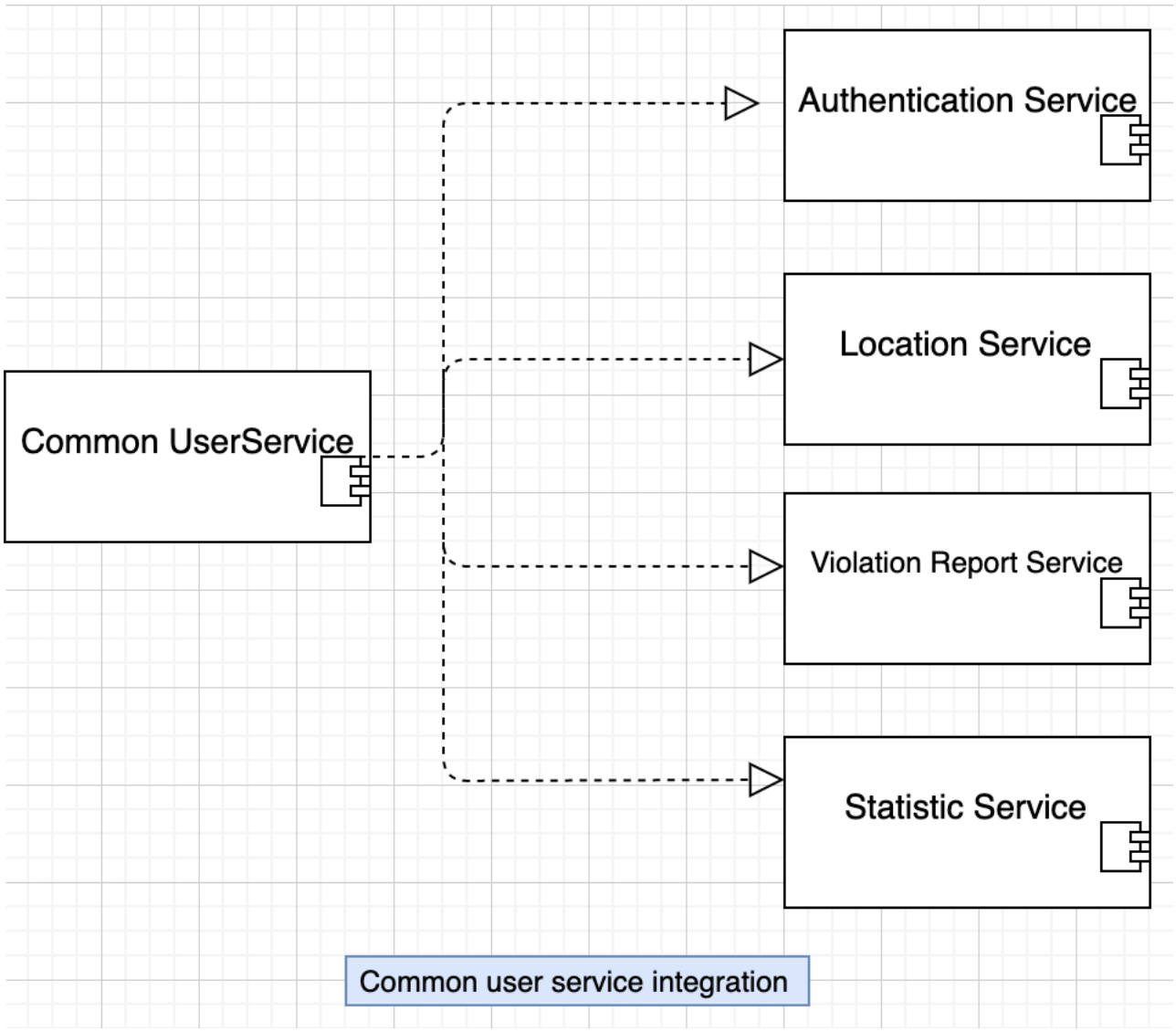


Figure 5.2 Common user integration services

Fist integration is between user and location service since it is needed for both report a violation service and statistics service. Next common user main functionality is to be able to report a violation hence common user and violation report service will be integrated next. Then as results of violation reports, common user has to be able to see statistics hence common user and statistics will be integrated next. For the system to be able to identify each and every individual, common user has to be authenticated hence next integration in between common user and statistics service.

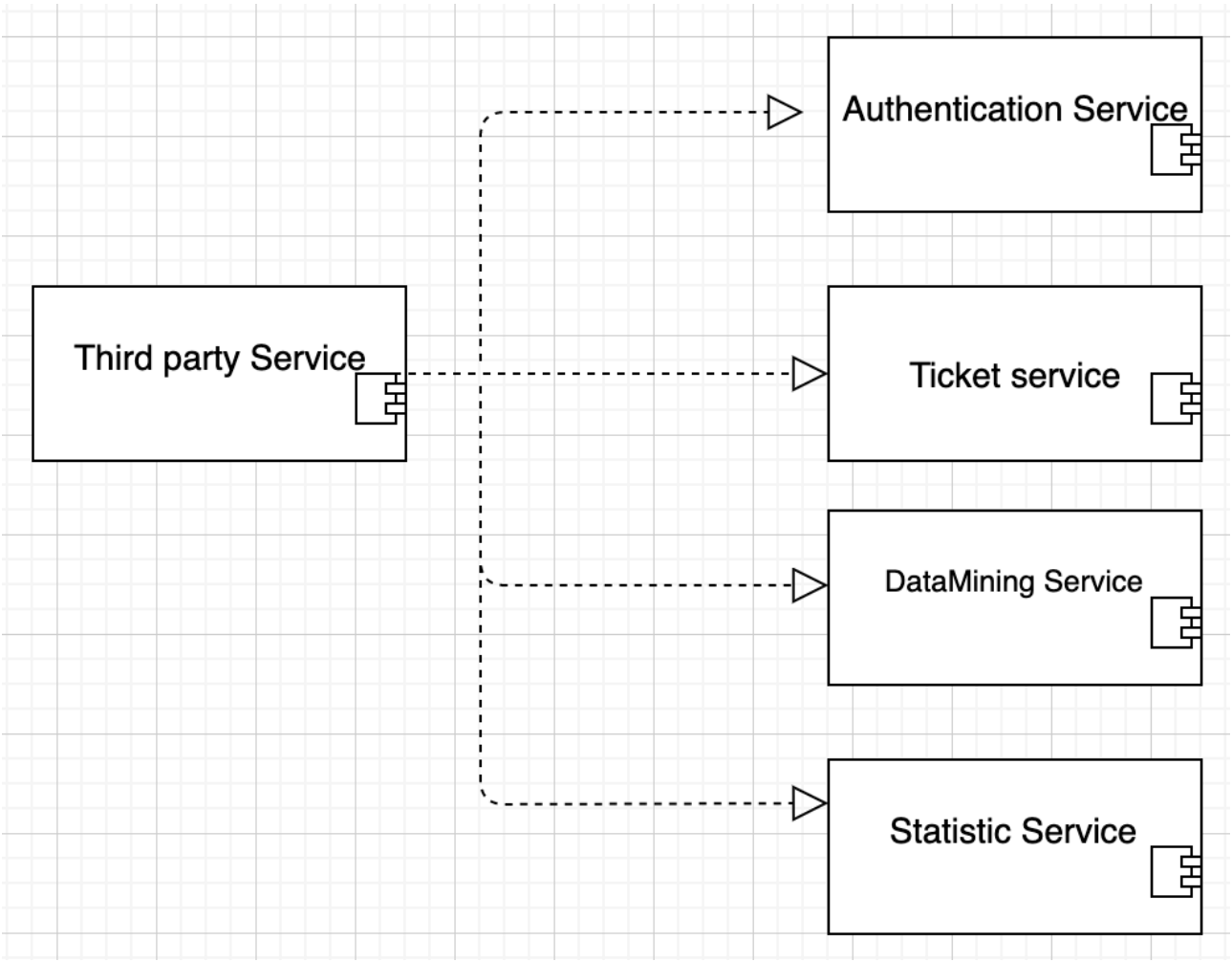


Figure 5.3: Third party integration services

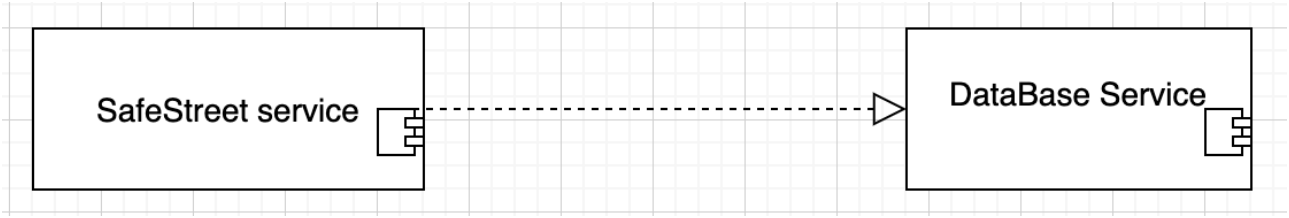


Figure 5.4: Safestreets and data base integration

With finished safestreets component implementation, Database integration has to be done to save common user and Third-Party login data to the Login Data module which is present inside Database Service which can be seen in figure 5.4.

### 5.4 Integration Testing Plan

Same as the implementation testing plan, TDD method will be used for integration of the components and external services. Before any integration of the components or services, integration test will be written to ensure that integration works properly. Even though integration tests consume more time because of the setup of required components and writing the tests for them, it is much safer and efficient in the long time.

For the integration of the external services, there won't be any extensive testing because these external components have their own internal tests. Very simple integration tests will be written to ensure that we have connection with these external services, and they are responding to the requests made by the internal services.

## 6 Effort Spent

**Arash Bahariye**

| | |
|---|---|
| Introduction | 1 hour |
| Component View | 3 hours |
| Component Interfaces | 2 hours |
| Overall Architecture | 2 hours |
| Design Patterns | 4 hours |
| Implementation Plan | 5 hours |
| Implementation Testing Plan | 2 hours |
| Integration Plan | 3 hours |
| Integration Testing Plan | 2 hours |

**Niloofar Salimi**

| | |
|---|---|
| Introduction | 1 hour |
| Component View | 2.5 hours |
| Component Interfaces | 2.5 hours |
| Overall Architecture | 4.5 hours |
| Design Patterns | 3 hours |
| Implementation Plan | 1.5 hours |
| Implementation Testing Plan | 4 hours |
| Integration Plan | 3 hours |
| Integration Testing Plan | 3 hours |