Department of Computer Science and Engineering

Design Document (DD)

SafeStreets

- v1.1 -

Authors:

Dlamini, Taras	10451348
Quran, Abdullah	10657807
Abdelaziz, Hesham	10654249

December 9th, 2019

1.Introduction

1.1 Purpose

The requirements elicitation and analysis activities concerning SafeStreets system were presented in detail in the RASD document (see the References section). After doing so, It is then proper to start exploring the design phase. This document aims at providing the technical details, architectural, and design choices that are, and should be, considered in order for SafeStreets' system to have the capability to fulfill the goals specified in the RASD, and to be used as a guide for the implementation and testing processes afterwards.

Below is the list of the aspects this document discusses:

- Overview of the high level architecture
- The main components and their interfaces provided one for another
- The runtime behavior
- The design patterns
- Graphical user interface
- Functional requirements mapping into the actual component of the system
- Implementation, integration, and testing plan

1.2 Scope

SafeStreets is an application trying to provide users with a user friendly and easy way to report traffic violations, such as double parking and car parked in a bike plane. Users would use the application to send authorities images, dates, times, location of violations via Safestreets. SafeStreets must have measures in place to make sure that the images and informations is tamper proof and valid.

In addition, users can see statistics about violations as well as safety of particular areas. Safestreets, would crosscheck data with Municipalities to make give the users Safe notifications if they desire. Furthermore, SafeStreets can give suggestions to Municipalities on how to improve safety in certain unsafe areas.

Finally, Municipalities should be able to use the data they acquire from SafeStreets to issue tickets for particular violations. In this case it is particularly important that the information and images are tamper proof and that the chain of custody of information coming from the user is never broken.

1.3 Definitions, acronyms, abbreviations

1.3.1 acronyms

- API: Application Programming Interface
- DD: Design Document
- RASD: Requirements Analysis and Specifications Document
- GUI: Graphical User Interface
- HTTP: HyperText Transfer Protocol
- TSL: Transport Layer Security
- JSON: JavaScript Object Notation
- SQL: Structured Query Language
- MVC: Model-View-Controller

1.3.2 abbreviations

- [Rn]: n-functional requirement.

1.4 Revision History

- 9/12/2019 - Initial release

1.5 Referenced Document

- Specification document: “SafeStreets Mandatory Project Assignment AY 2019-2020”
- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications
- Requirements Analysis and Specifications Document: SafeStreets -V1.2-

1.6 Document Structure

Chapter **one** is and introduction of the design document. It describes the document’s purpose and scope. It also includes definitions and abbreviations used in this document.

Chapter **two** represents the core part of the document. It includes an overview of the architectural design of the SafeStreets’ system. It describes the system’s architecture from different paradigms, in particular; component view, class view, deployment view, run time view, and exact interfaces and their actions. Then it includes a justification on the choice of the design architecture and pattern

Chapter **three** complements the user interface section discussed in the RASD section. However, here all the pages of the SafeStreets web and mobile applications are included

Chapter **four** includes mapping between the functional requirements thoroughly discussed in the RASD into the design elements explored in this current document.

Chapter **five** presents the implementation, integration, and test plan of the system. It describes how these three phases are to be tackled.

Chapter **six** shows the effort each member of the group spent to generate this document.

2. Architectural Design

2.1 Overview

The architecture pattern to be adopted in SafeStreets system is decided to be 3-tier architecture which is basically client - server architecture consists of 3 physically separated tiers called presentation, application and data tiers. 3-tier architecture was chosen because it provides scalability since the application is thought to be crowdsourced application and flexibility if future development is needed. In addition the physical separation between tiers results in hiding the logic from the users and this potentially advantageous in terms of information security. Below is a brief description of the high level component of each tier. Further elaboration on the system architecture is provided in the following sections of this chapter. Figure 3.1 shows the higher level components of the system and their interactions

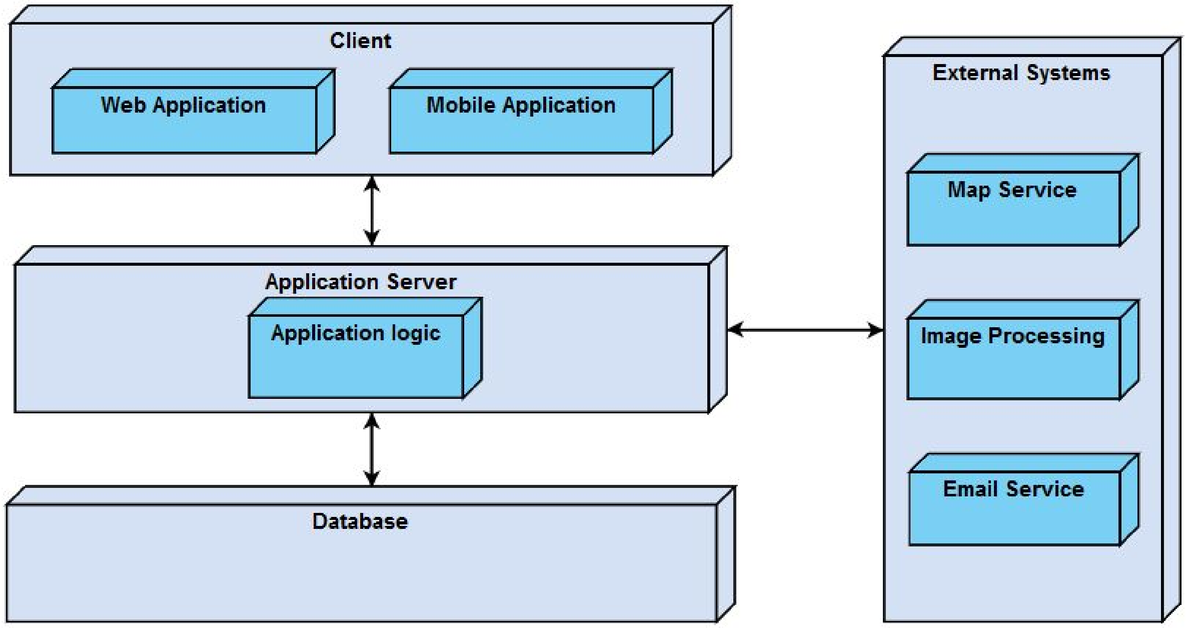


Figure 2.1: High level architecture of SafeStreets system

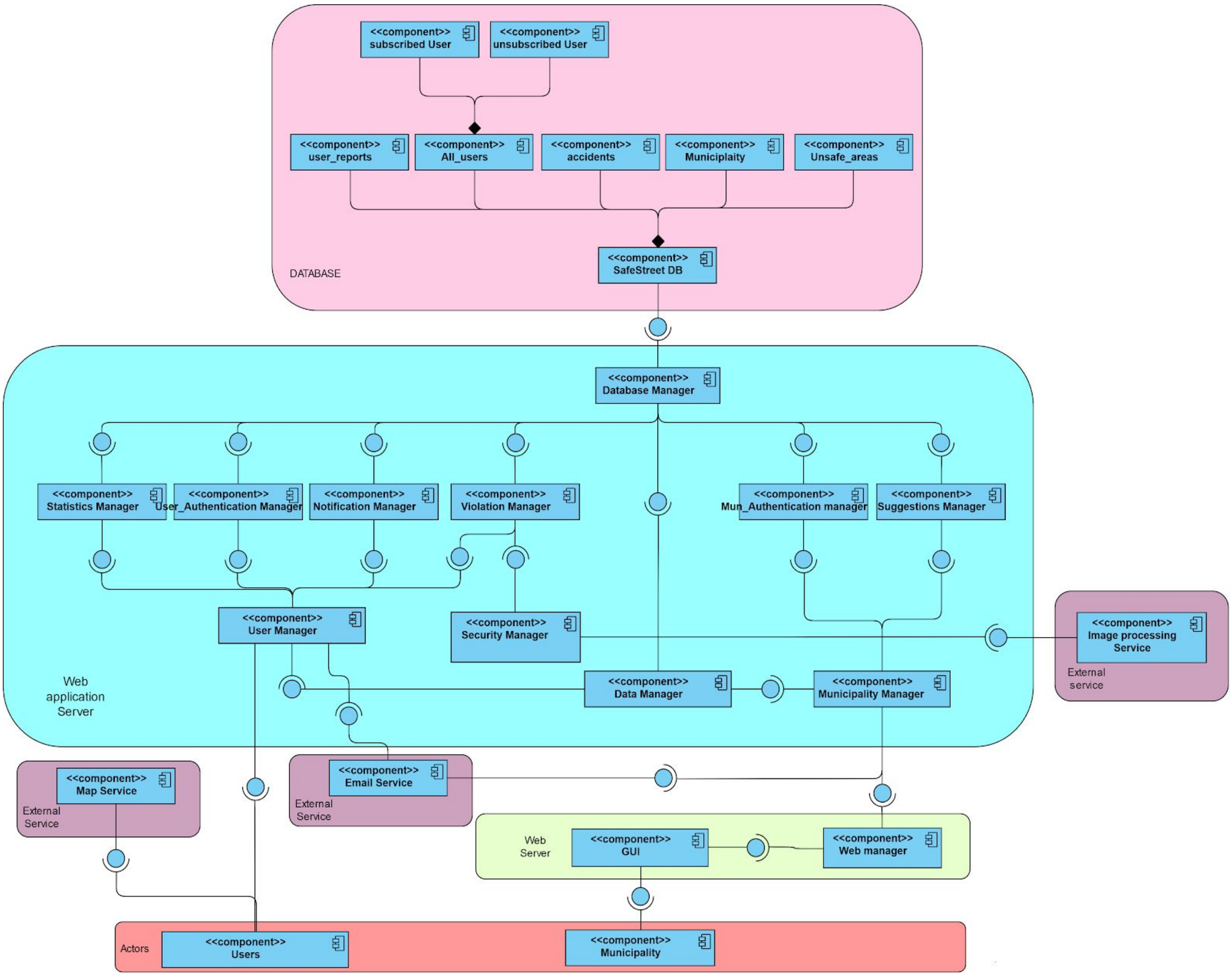
The **Presentation** (client) tier is comprised of the mobile and web application. It is the tier which users (regular users and third parties) have access to through GUI. We will see that small but important part of the business logic is placed in the mobile application in order to ensure that the chain of custody is never altered (this was defined in the RASD documents in details).

Then comes the **Application** tier that carries most of the business logic the system has to do. The application tier is positioned in the middle between the data tier and the presentation tier and it responsible for performing detailed processing. Also this tier has an interface with the external systems.

The last tier is the **Data** (database) tier which a database comprising both data sets and the database management system software that manages and provides access to the data. It communicates with the application server

2.2 Component view

In this section all the components presented in the following component diagram will be briefly explained in terms of functionality and interfaces provided by each component.



<<component>> subscribed User
<<component>> unsubscribed User
<<component>> user_reports
<<component>> All_users
<<component>> accidents
<<component>> Municiplaity
<<component>> Unsafe_areas
<<component>> SafeStreet DB
DATABASE
<<component>> Database Manager
<<component>> Statistics Manager
<<component>> User_Authentication Manager
<<component>> Notification Manager
<<component>> Violation Manager
<<component>> Mun_Authentication manager
<<component>> Suggestions Manager
<<component>> User Manager
<<component>> Security Manager
<<component>> Data Manager
<<component>> Municipality Manager
Web application Server
<<component>> Image processing Service
External service
<<component>> Map Service
External Service
<<component>> Email Service
External Service
Web Server
<<component>> GUI
<<component>> Web manager
Actors
<<component>> Users
<<component>> Municipality

Figure 2.2: component diagram of the system

2.2.1 Application server

- **User Manager** ”User control manager”: The user manager is the only component that interact with Mobile App , so it can be considered as the maestro that coordinates each request to its specific component , and provide the requested data to only the authentic users

    - **data_control**(email):boolean:this procedure responsible for forwarding the user request for the data to the ‘Data manager’ in order to download the data

    - **violation_control:**(violation_type,location,date,email,image):boolean:this procedure is responsible for forwarding the user request for the data to the

    - **notification_control**(location)

    - **authentication_control**(first_name,last_name,email,password,phone_number)

    - **authentication_control**(email,password)

    - **statistics_control**

    - **email_verification**

- **Municipality Manager**: It can also be considered as the control unit that authentic allow only authentic municipalities to download the data provided by the SafeStreet database and be able to see suggestions made by SafeStreet Server based on the existing data

    - **activation_control**

    - **login_control**

    - **data_control(start_date,end_date,violation_type,location)**

    - **suggestion_contro**l

- **Data Manager**: This component is responsible for handling the downloading , uploading of data by the Municipality and send a download link to the user email which contain the anonymized data.

    - **Mun_download**(violation_type, start_date, end_date, location, data_type)**:** data: a procedure that allows the municipality to download the data in order to mine it

    - **Upload**(data):void: a procedure to allow the municipality to upload the accident data happened in their territory

    - **user_download**(violation_type, start_date, end_date, location): data: a procedure that send an email to the user that contain a link to download an anonymized version of the data

- **Statistics Manager**: This component contains data mining algorithms that applies on the data provided from the database to be converted into a useful information provided to the users and the suggestion manager.

    - **retrieve_data**(start_date,end_date,violation_type,location):void: a procedure that allow the user to receive the mined data given some filter inputs from the user

    - **run_algorithm(**start_date, end_date, violation**)**:Data: a procedure that run the mining algorithm on the raw data provided by SafeStreet database.

- **User Authentication manager**: this component is responsible for checking the credentials provided by the user against the database “All user” table , and provide the “user Manager” an access token to access the other services provided by the Web application in case of an authentic user.

        - **User_check** (email , password):boolean: this procedure takes two parameters (email,password) and check these parameters against the database to allow him to access the system resources by generating a key token.

        - **user_register**(first_name,last_name,email,password,phone_ number)

        - **user_activate**(email)

- **Notification Manager**:this component keep track of subscribed users location and send them notification messages when the user in unsafe area , to give him awareness.

    - **subscribe_user(**email**)**:void:this procedure move the user from the unsubscribed user to a subscribed user in the database.

    - **unsubscribe_user(**email**):**void:this procedure move back a subscribed user to unsubscribed user in the database.

    - **safety_check**(location):boolean:this procedure is responsible for checking whether this location”street” is considered unsafe.

- **Violation Manager**:This component handle all he violation received from the users and uses an external service to validate the violation provided by the users, invalid violations or false violations will be discarded and not save into the database.

    - **receive_violation**(email):data: this procedure receive the reported violations from the user.

    - **is_validated**(image):boolean this procedure is to pass the images attached to the violation to the “Image processing manager” and return a boolean variable , true if the plate number returned , false if not returned.

    - **store_violation**(Violation):void:this procedure check the return outcome from the procedure “is_validated” if it's true ,the violation will be stored in the database, and if its false the violation with images will be discarded.

- **Security manager**: This component provide service to the “Violation manager” by verifying that the image associated with the violation report is never altered then send the image to an external service to identify the plate number and return it as string .

    - **Img_check**(image,string) :boolean: This procedure check whether the image is altered or not by calculating its hash and compare it with the hash associated with it.

    - **img_to_num**: (image):string:This procedure send the images associated with the violation to an external service to extract the plate number from the image and return the plate_number as String .

- **Suggestion Manager**: This component takes the data from the “Database” and apply them to a recommender algorithm , to provide some recommended action to the municipalities in order to make the street more safe.

    - **Get_data**(start_date,end_date):Data: This procedure get the data of violation from the database.

    - **run_algorithm**():string[]: this procedure runs a recommender algorithm on the given data and produce some suggestions.

- **Municipality authentication**: This component is responsible of authentic the municipality to be able to access the system functionality like downloading ,uploading accident data and be able to access the suggestions provided by SafeStreet application server.

    - **Get_register**(name,email):boolean: This procedure allows the municipalities to confirm their registration and receive their access credentials.

    - **Get_authentic**(email,password):boolean:This procedure checks the municipality credentials.

- **Database Manager**: This component contains all the functions required to provide all the other component in the system the data they need in order to work properly.

- **check_user**(email,password):boolean:this procedure queries the given user email against "All_users" table which contain all the users details and their saved password.
- **retrieve_accidents**(start_date,end_date,location):Data :this procedure retrieve all the **accidents** that occurs between the 'start_date' and 'end_date' in the given location.

- **retrieve_violation**(start_date,end_date,location,violation_type,actor) :Data : this procedure retrieve all the violations that occurs between the 'start_date' and 'end_date' and the violation_type in a given location.

- **add_user**(first_name,last_name,email,password):boolean:this procedure add the user information into "unsubscribed_user" table.

- **sub_to_unsub**(email):boolean:this procedure move an existing user from 'subscribed_users' table to 'unsubscribed_users' table .

- **unsub_to_sub**(email):boolean:this procedure move an existing user from 'unsubscribed_users' table to 'subscribed_users' table.

- **add_municipality**(name,email):boolean: this procedure add the new registered municipality information to the municipality table.

- **check_municipality**(email,password):boolean: this procedure check the municipality credentials.

- **add_violation**(violation):this procedure store a violation into the user_report table
- **Retrieve_unsafe**(start_date,end_date,location):data:this procedure run a query to bring all the unsafe areas given a date interval and the location.

- **safety_check**(location):boolean:this procedure is checking whether this location is safe or not by searching the ‘unsafe_areas’ table.

- user_activate(email):boolean:

DataBase:The database that provide service to the web application consists of five physical tables and two virtual tables .

Subscribed user: its a physical table which contain all the users details like (Email,password,firstname,lastname) that subscribed to receiving notifications

Unsubscribed users: its a physical table which contain all the users that is not subscribed for notification services

All users: its a virtual “view” that contain all the users in the system except for the municipality which have a different physical table

User reports: its a physical table which contain all the reported violation reported by the users and its structure like (violation_type, date, time, plate_number, street_name, email), note that the attribute email refer to the user that report for the violation.

Municipality: its a physical table which contain all information about the registered municipality like(Email, Mun_name, Phone_number, location)

Accidents: it's a physical table that contain all accidents reported by the municipalities in their territory. and its attributes (accident_type, date, time, street_name, Mun_name).

Unsafe areas: it's a physical table which contain all the unsafe areas

2.2.2 Mobile application

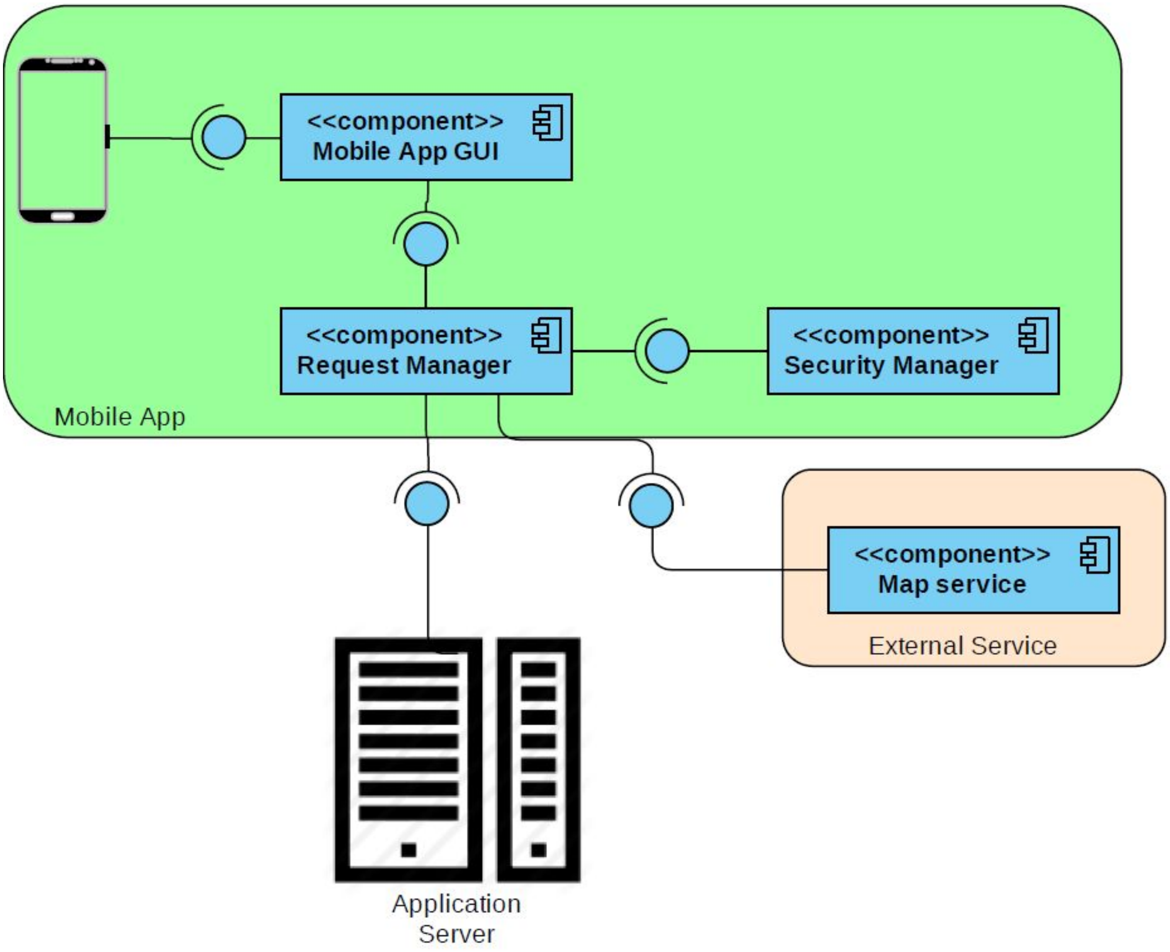


Figure 2.3: Mobile application component diagram and interfaces

- App GUI: this component is responsible for making the user to interact with the Request Manager graphically and view the received data in a user friendly way

    - **Login_screen**: this procedure is showing the login activity on the App

    - **Main_Screen**: this procedure show the main activity on the App

    - **Registeration_screen**: this procedure show the registration activity on the App

    - **myReports_screen**: this procedure shows the user reports on the app

    - **ReportViolation_screen**:this procedure show the report violation activity on the App

    - **Statistics_screen**: this procedure show the statistics activity on the App

- **Security Manager**: this component is responsible for preventing users from altering the images associated with the violation details.

    - **hashing**(image):String: this procedure calculate the hash of the image taken by the App associated with the violation details and store the hash along its image

- **Request Manager**: this procedure responsible for sending requests to the Application Server and receiving the data

    - **Register**(first_name,last_name,email,password,phone_number):boolean:this procedure sends to the web application a registration request

    - **Login**(email,password):boolean:this procedure is responsible for sending the Login Request

    - **Report_violation(violation_type,location,date)**:boolean:this procedure is responsible for sending the violation details along with the image meta-data “Hash”

    - **My_reports**(email)**:**data:this procedure request all the user reports from the web application

    - **View_statistics**(start_date,end_date,violation_type,location):data:this procedure request the violation in a given location and date

    - **change_notification**(email):boolean:this procedure is responsible for subscribing/unsubscribing the user from notification.

    - **get_notification**(email):boolean:this procedure is responsible for requesting a notification automatically.

    - **Request_data**(email):boolean:this procedure allow the user to request the data from the database and sent to his email as hyperlink reference.

    - **get_location**():location:location:this procedure call the map service to provide the user with his current location.

2.2.3 Web server

- GUI:
    - Login_page:
    - Suggesion_page:
    - Main_page:
    - Data_access_page:

- Web Manager:
    - Login(email,password)::
    - Activation(email)::
    - Data_request(start_date,end_date,violation_type,location,data_type): void:
    - suggestion_request():string:

2.2.4 External services

Map Service: its an API that can be used by the user in the reporting violation activity to retrieve the user location based on his current location, and it will also send to the Server the location of subscribed users periodically to retrieve notifications based on their current location.

Email Service:its an API that will send the users a downloadable link to download an anonymized version of the data.

2.3 Deployment view

Figure 2.4 Below show the deployment diagram of SafeStreets system.

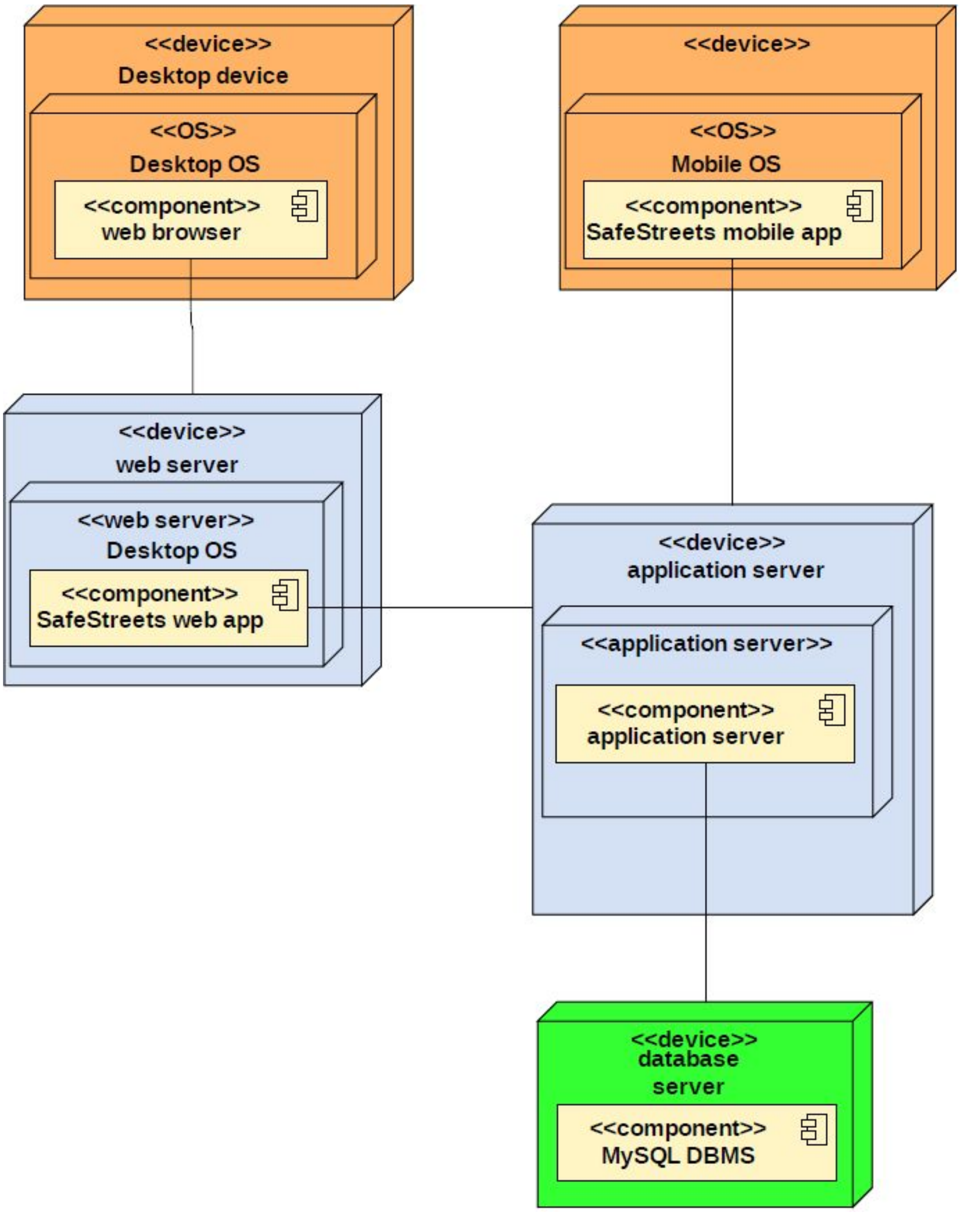


Figure 2.4: Deployment Diagram of SafeStreets system

2.4 Runtime view

In this section we will summarize some of the main behavioural functions in the system using the sequence diagram.

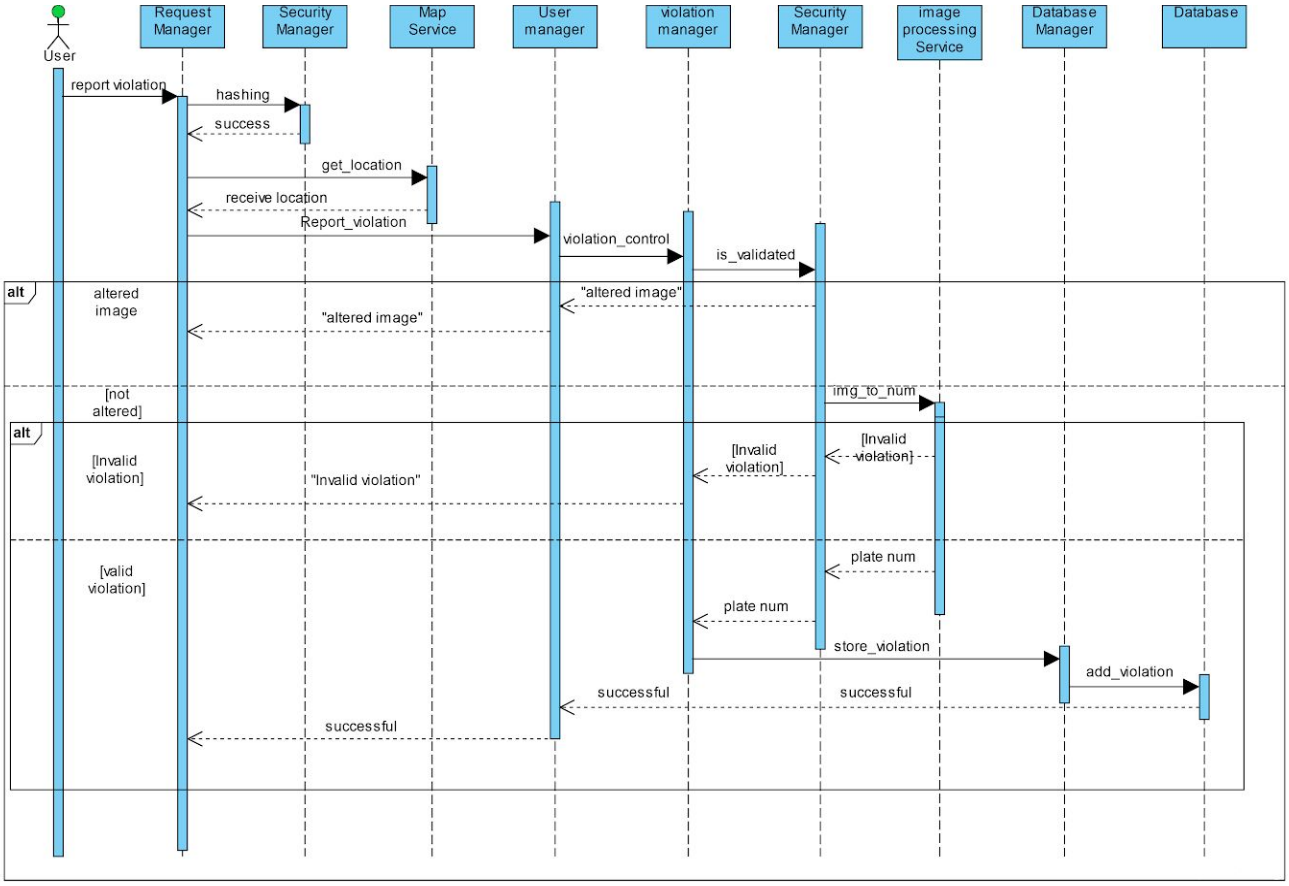


Figure 2.5: reporting violation sequence diagram

Figure 2.5 represents the behaviour of a user reporting a violation,the user fill the violation form and attach the violation image, then the violation is being sent to the web application including the image hash to make sure that the image is not edited and to extract the plate number from the image, then the violation is saved into the database.

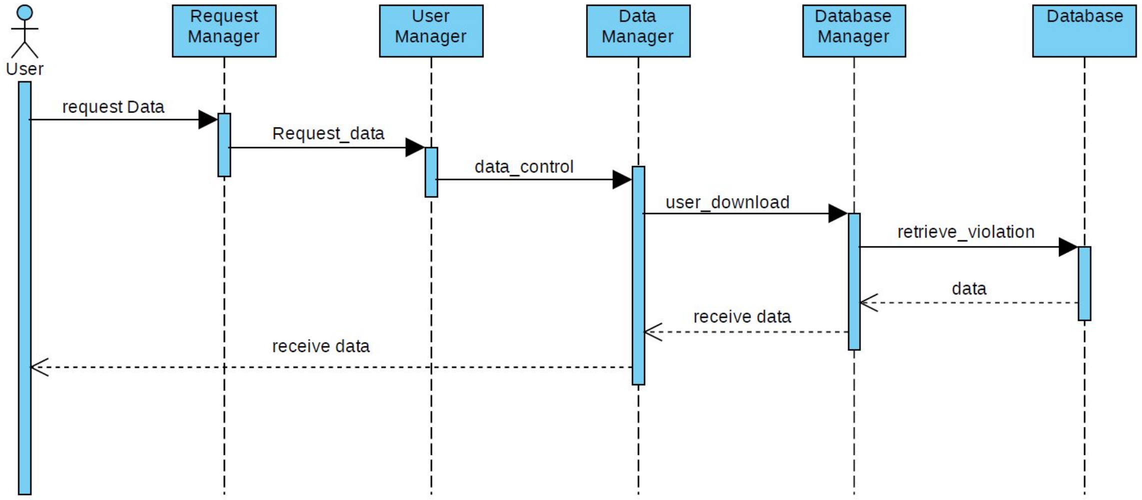


Figure 2.6: sequence diagram of access data request by users

Figure 2.6 shows the flow of procedures when a user request access of data of violation. The request indicate the desired filteration and the data passed to users should be anonymized.

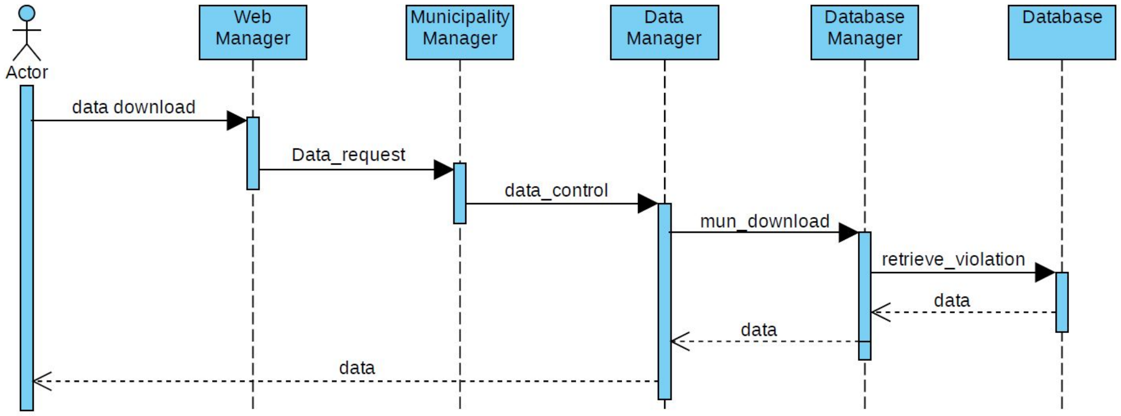


Figure 2.7: sequence diagram of access data request by municipality

Figure 2.7 in this figure sequnce of flow of procedures between components when a mmunicipality sends request for accessing the data of vioation. This request indicate the desired filteration. Municipalities have full access to the data and they should not be anonymized.

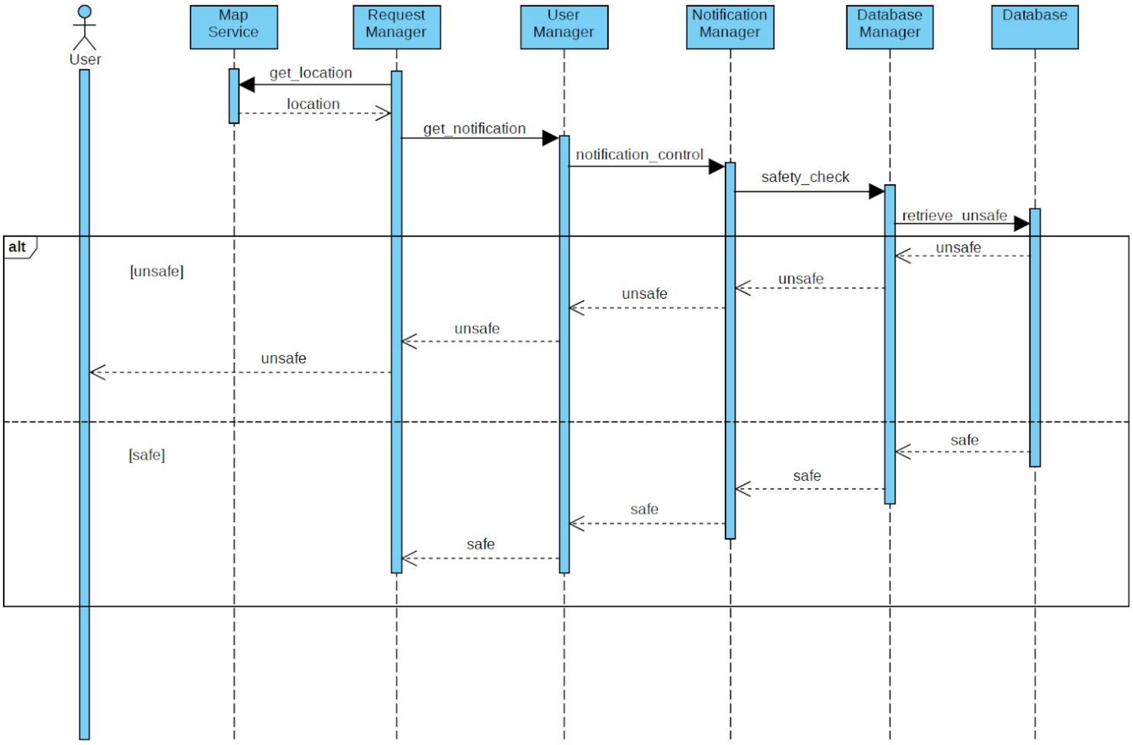


Figure 2.8: Notification update sequence diagram

Figure 2.8 shows how a user receive a notification when entering an unsafe area, firstly the Request Manager component asks the map external service to give his current location, then send this location to web server and search the database if this location is unsafe, if yes a notification will be sent back to the user, if not it will not send anything to the user.

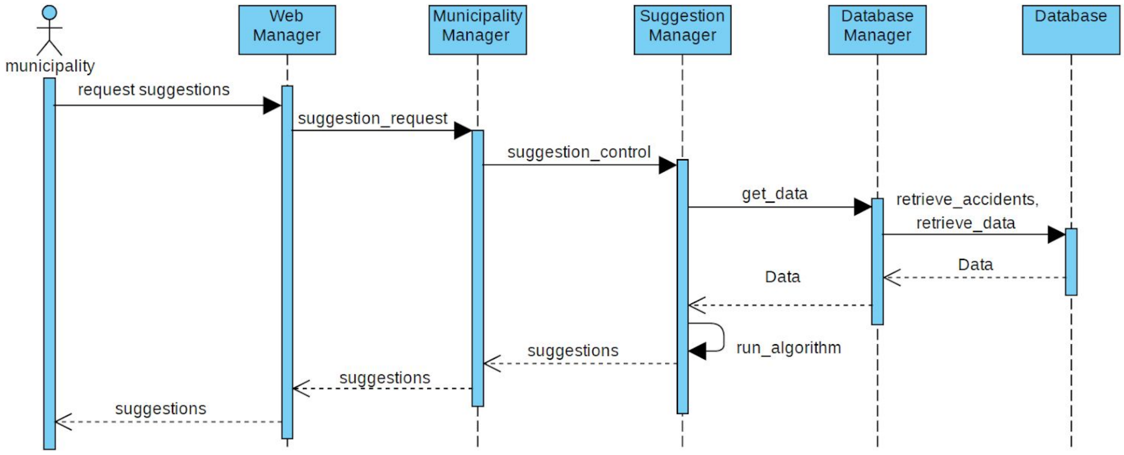


Figure 2.9: sequence diagram of sending suggestion request

Figure 2.9 shows how the application web server can generate the suggestion based on the accidents retrieved from the municipalities and the violation from the users report.

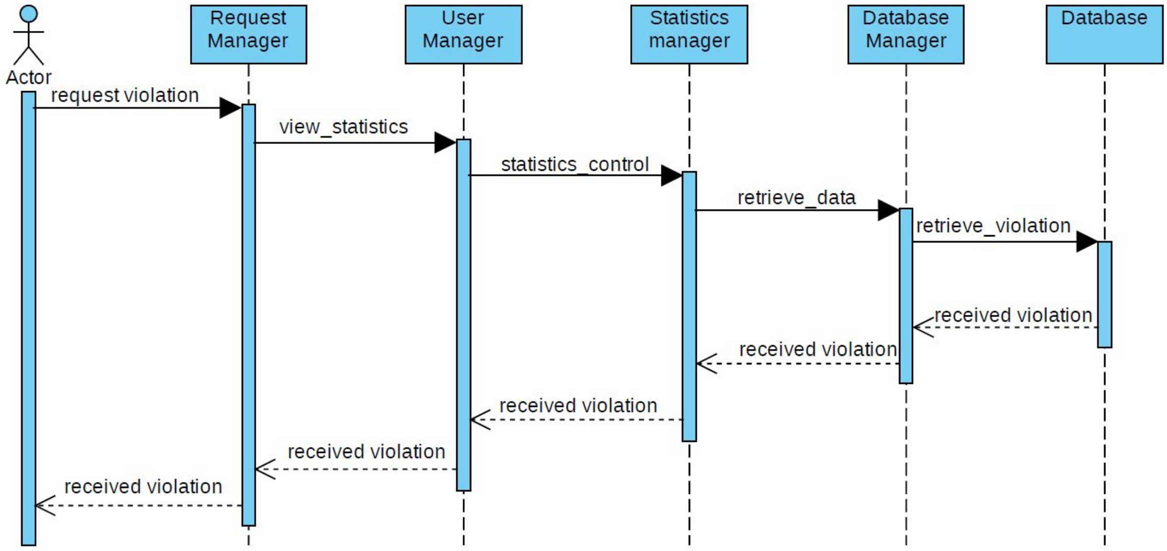


Figure 2.10: sequence diagram of sending see my reports request

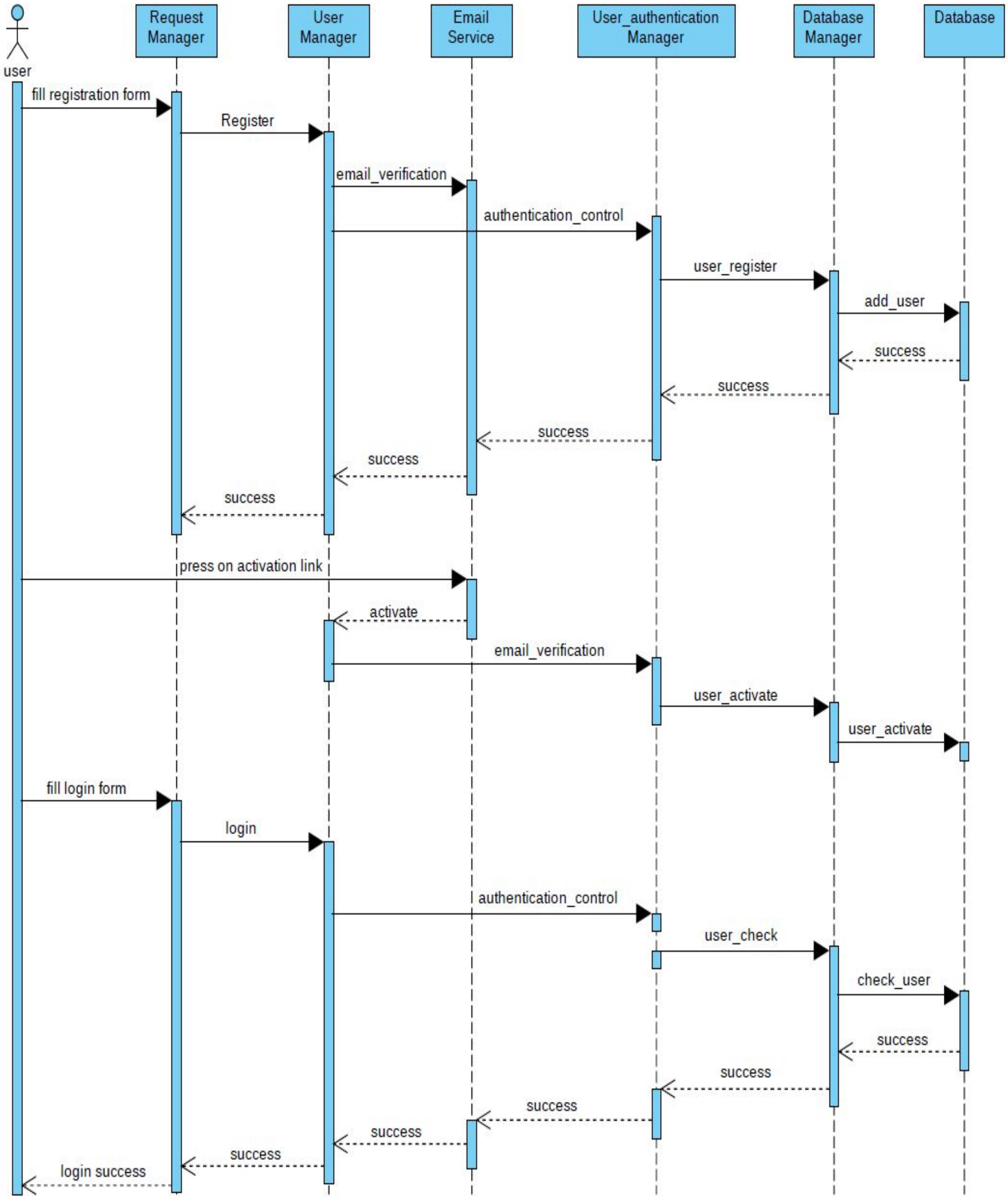


Figure 2.11: sequence diagram of sending suggestion request

Figure 2.11 shows user registration, activate his account after registration then login into his account, firstly the user fill the registration form then press confirm , then the user will receive an activation link into his email, after pressing confirm the user manager will send a confirmation of his activation.

2.5 Selected architectural design and patterns

In this section, we elaborate on the design choices and architectural styles that were chosen for the system and the reasons why we choose them.

2.5.1 Three-tier architecture

As already discussed in section 2.1, the main design pattern SafeStreet can fit in is 3-tier architecture. This is because we have established physical and logical connection between between three tiers following the characteristics of 3-tier architecture. The mobile application and the web-application fit in the first tier which is the presentation tier. The web server and the application server fit in the second tier which is the application tier. The database fits in the third tier which is the data tier.

Stability is on the advantages of 3-tier as failure in the one the tiers will affect grealty the functionalities of the system, but will not block the system from functioning. For example, if a major failure occurs on the web server, the mobile application is capable of doing the full functions its responsible of. In addition, by separating the application server from the database this make it harder for the user (users and municipality) to get to the database layer which overall improves the security of the system.

2.5.2 Model, View, Controller

The architecture of the system can also be seen as Mode, view, controller style in the sense that both the mobile application installed on the user's device and the desktop browser with web server are equivalent to the view layer that has direct contace with user. The application server that is the brain of the system serves as intermediary between the view and the model layer and it is reasonible to see it as the controller layer. Then comes the model layer that has contact with the database and can be considered as the DBMS that receives request from the application server and process them. This style brigs many indeirect benefit such as code reusability and reduction of complexity through which different developers can work in parallel on different layers.

2.5.3 client - server

There can been seen some pairs which follows client - server architecture. Client - server architecture can be defined as an architecture in which client request and receive service from a centralized server. The following pairs can fit in this architecture:

- SafeStreets mobile application service installed to the user mobile application as a client and the application server as a server that receive the request, process them and send responses.
- Municipality with browser as a client and the web server as server that receive the request and send responses with HTTP interface.
- Application server as client when it sends requests to the Database server (server)

2.5.4 Thin client

Active Municipality web application can be considered as thin client. They connect and get the services through web browsers. Municipalities don’t have to install any software on their computers since All the necessary information required to use the services are offered by the Web Server through HTTPS, including a graphical interface and the application logic.

In addition regular user mobile application can be considered as thin client even though the software is installed on the mobile and it carries a small part of the logic, it is still considered thin because it cannot perform any functionality locally without relying on the remote network application server.

2.6 Other design decisions

2.6.1 design deciosn on the security of data

To fulfill the stakeholders demand of ensuring the chain of custody of information, which are mainly images of violation, a set of security desions should be taken into account. We can divide the problem into two parts; Image protection at the mobile application from user's manipulation, and Image protection from attacks at the communcation layer as the images are transmitted over wireless links in the network.

With regards to image protection at the application layer, we introdeuced two mechansims that make it hard to user with bad intents to tamper with images. These mechanisms are, given that no the user has no possibility to take images from outside the mobile appliciton, locally generating a SHA512 hash of the image as soon as the image is taken and storing it with the application directory. The second mechanism is actually in the way the image stored. The directory of the mobile appliction should be hidden from the user and this is the responisbility of the security manager and design that the develpoers should give careful attention when develpoing the application.

Coping with active attacks that might happen the communcaiton layer can be minimized to acceptable treashold by using secure communication protocol. In fact, HTTPS is capable of doing this job and ensuring data integrity and privacy between the mobile applcaition and the applcation server. It should be noted that HTTPS is also used for the web appliction interface with Municipality browser. At the application server the server will verify the image by rehasing the image agian and comparing the new hash with the hash delivered with image.

3. User Interface Design

3.1 Regular User Interfaces

Below are mockups of the user interface of the app for regular users representing the most important functionalities that different users will access.Figure 3.1 represents the login screen for all kinds of users (regular and municipalities). Figure 3.2 represents the registration screen for regular user that have not created an account yet, note that municipalities register through the webpage.

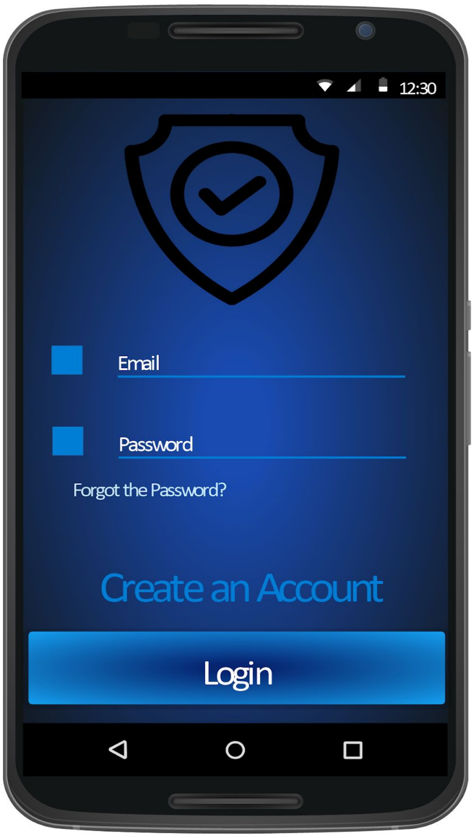


Figure 3.1 - Mockup: Login form

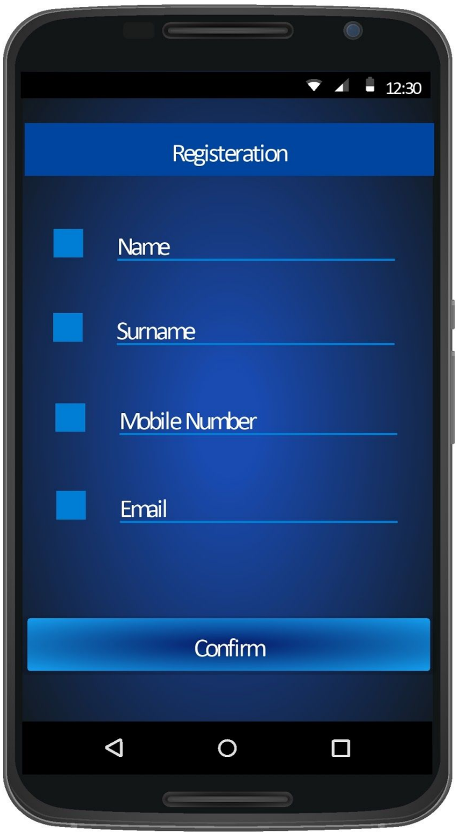


Figure 3.2 - Mockup: Registration

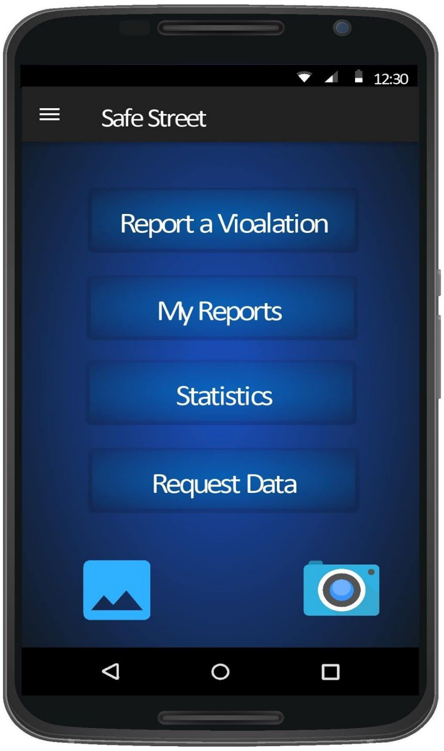


Figure 3.3 - Mockup: Regular user menu

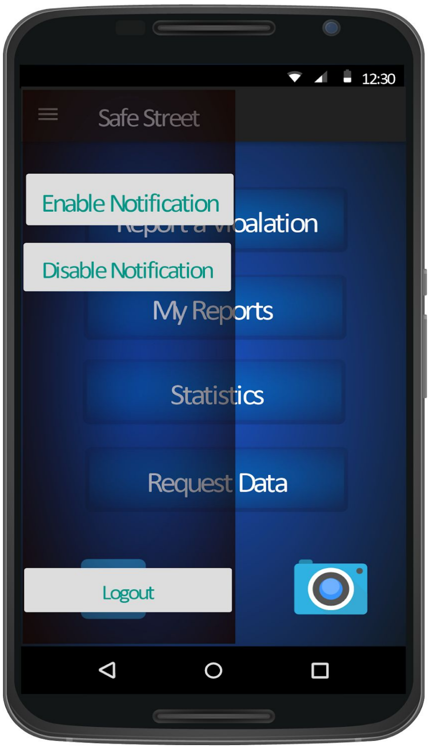


Figure 3.4 - Mockup: Account Settings

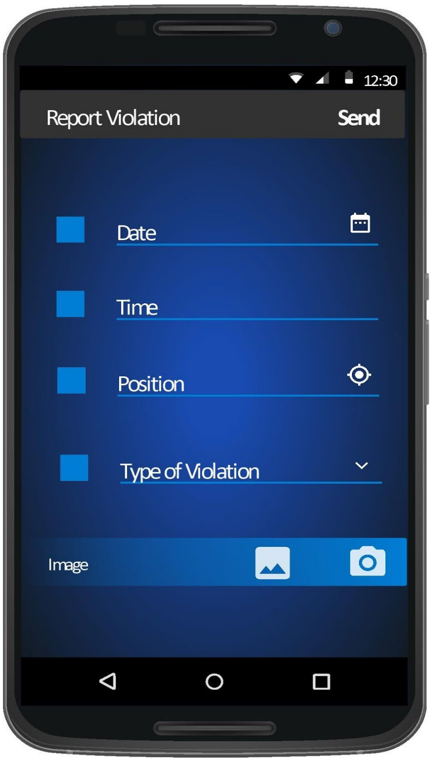


Figure 3.5 - Mockup: Reporting a violation

Figure 3.3 represents the menu for regular users where they can choose either to report a violation, to see their reports, to see statistics about violations in general or to get a data for mining purposes sent to their email. Figure 3.4 represents the pop up settings menu where users can opt to stop receiving or receive again notifications, furthermore they can logout and see account information. Figure 3.5 represents the regular user screen for violation reporting.

3.2 Municipality Interfaces

Below are mockups of the user interface of the website for the municipalities representing the most important functionalities that municipalities will access.

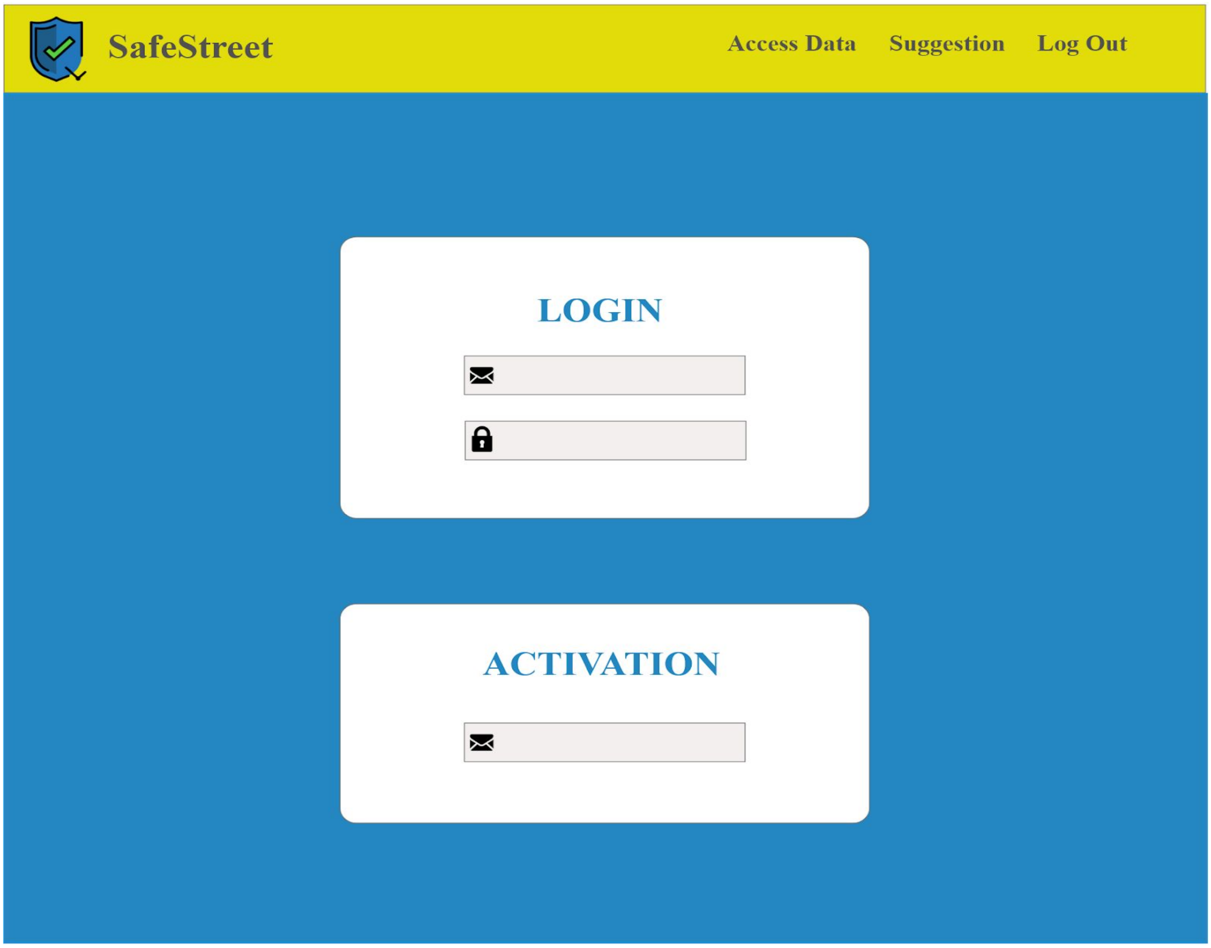


Figure 3.6 - Mockup: Municipality Login/Activation

Figure 3.6 represents the screen where a municipality can opt in to the SafeStreets initiative by typing in the email in the activation bar, they will receive an email from SafeStreets with the password and they can login on this same screen using their email and password.

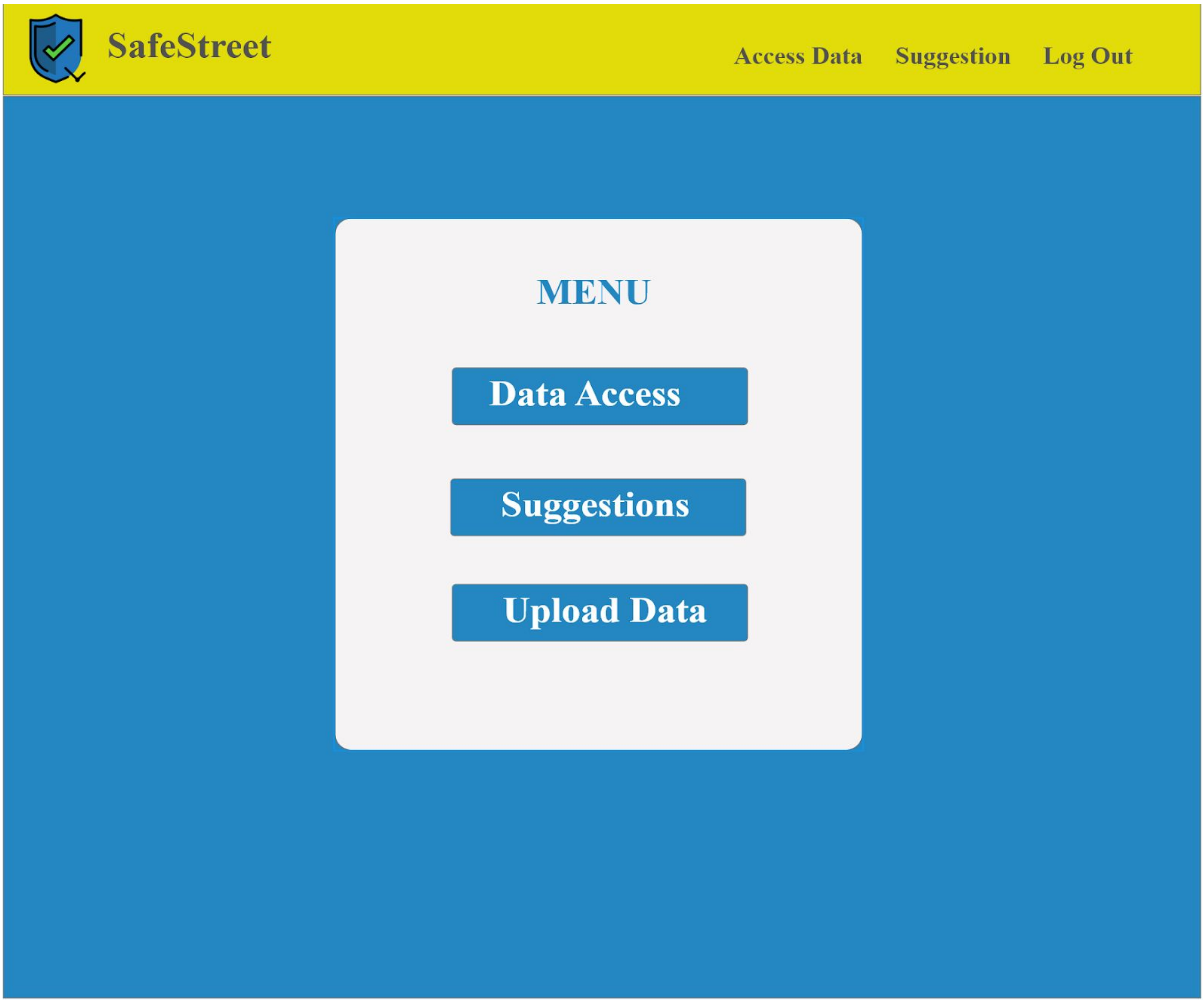


Figure 3.7 - Mockup: Municipality Menu

Figure 3.7 represents the Menu page where the logged in Municipality can select one of the options, if they click on the upload data, they will get a pop up where they can select an Excel, PDF or DB file to upload. The Data Access and Suggestion buttons will take them to the respective pages, described below.

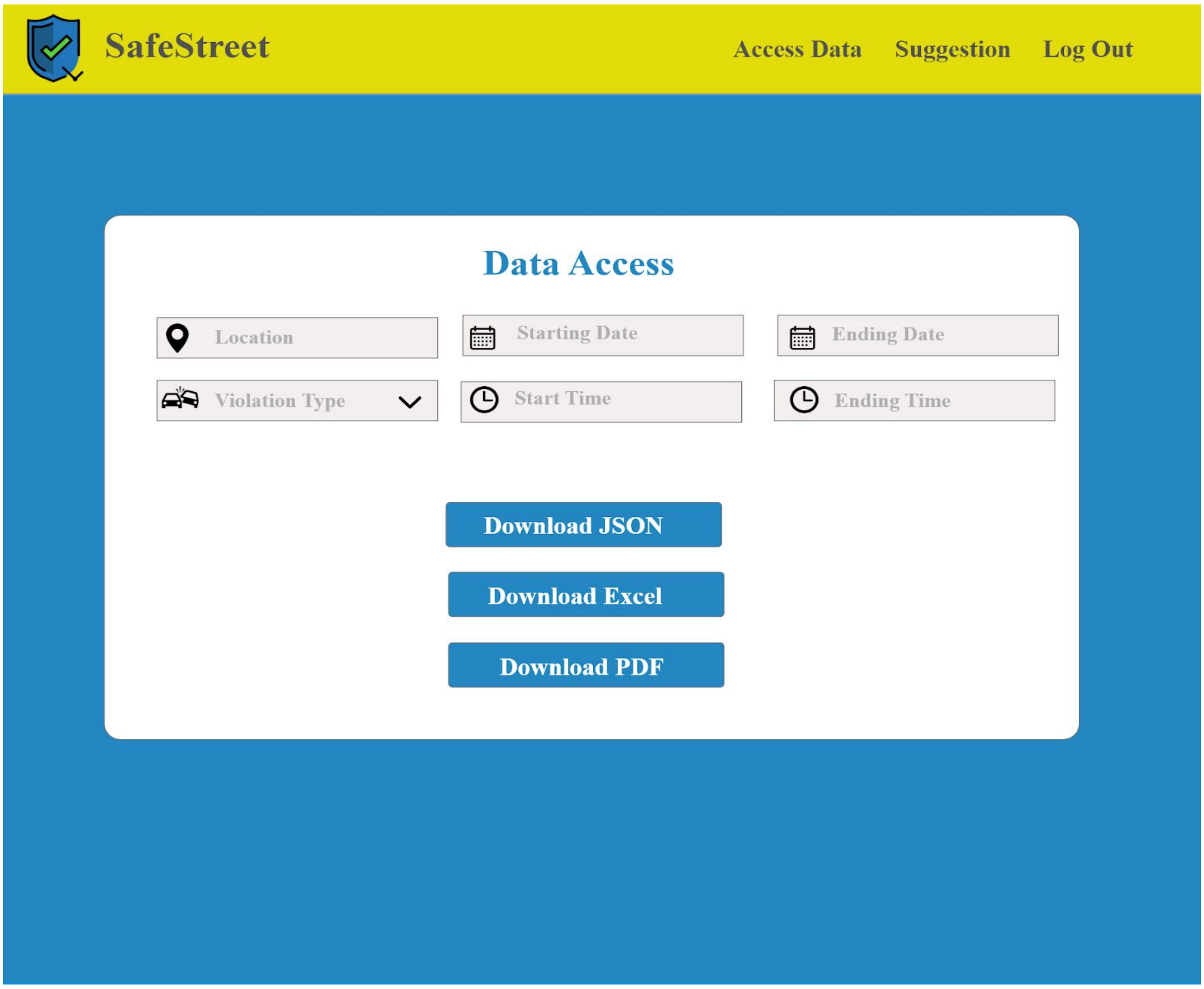
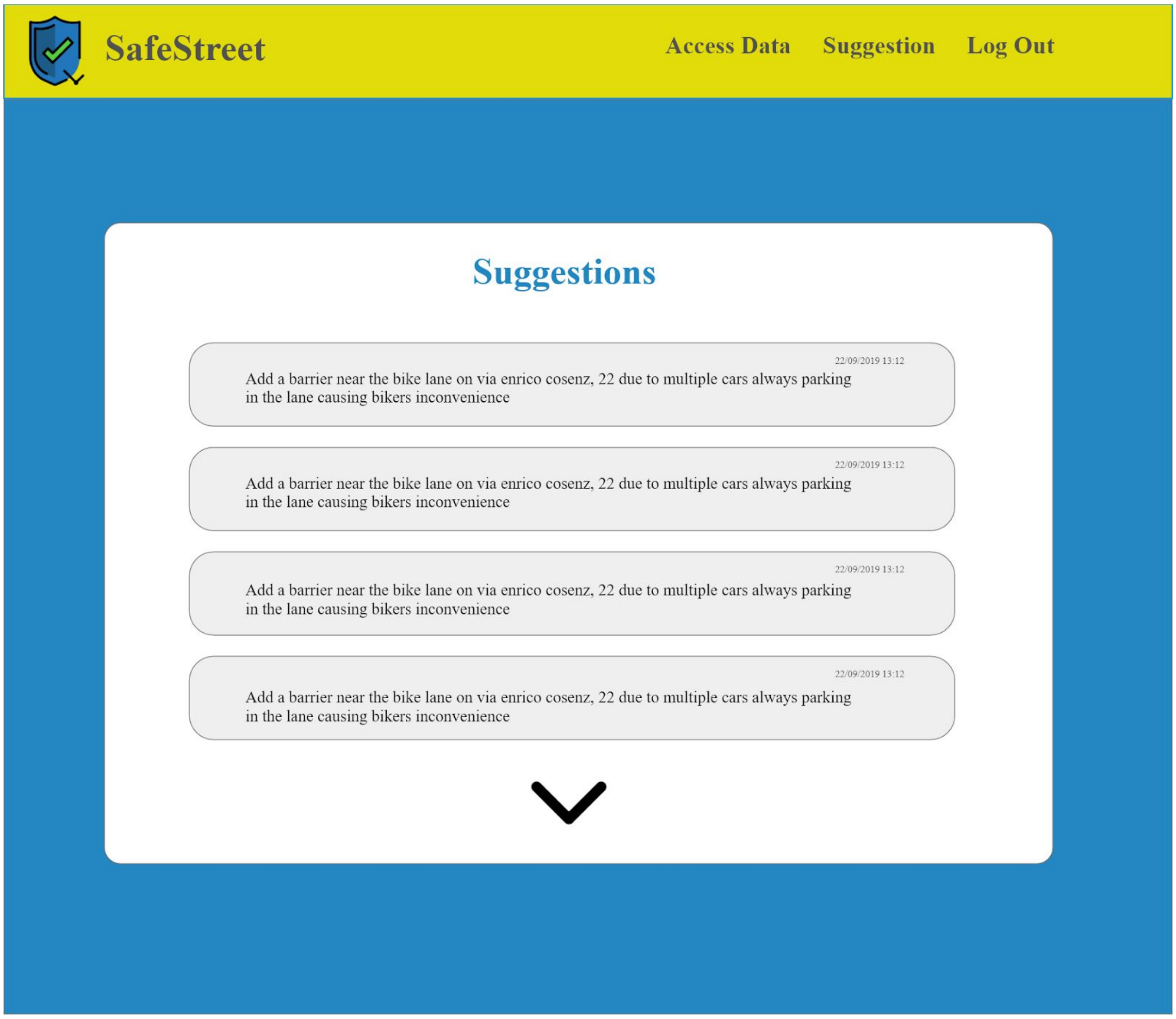


Figure 3.8 - Mockup: Municipality Login/Activation

Figure 3.8 represents the screen where a municipality can filter data that they would like a report sent in, they have the option of the report being sent in JSON, Excel or PDF file.



Suggestions

22/09/2019 13:12
Add a barrier near the bike lane on via enrico cosenz, 22 due to multiple cars always parking in the lane causing bikers inconvenience

22/09/2019 13:12
Add a barrier near the bike lane on via enrico cosenz, 22 due to multiple cars always parking in the lane causing bikers inconvenience

22/09/2019 13:12
Add a barrier near the bike lane on via enrico cosenz, 22 due to multiple cars always parking in the lane causing bikers inconvenience

22/09/2019 13:12
Add a barrier near the bike lane on via enrico cosenz, 22 due to multiple cars always parking in the lane causing bikers inconvenience

Figure 3.9 - Mockup: Municipality Suggestions

Figure 3.9 represents the suggestions page where the Municipality can see suggestions from SafeStreets about how they can improve certain streets and prevent accidents and violations occurring there.

4. Requirement Traceability

This section describes how the design components defined in chapter two are mapped to the functional and non-functional requirements predefined in the RASD documents. We decided to include the non-functional requirements since they play an important role in the performance of any software system. In addition to that, security, which is a non-functional requirement, is of paramount importance to the well-functioning of this system.

4.1 Functional Requirements mapping:

R1: The system must allow a user to register a new account

User Authentication Manager

R2: The system must allow a user to login to his/her account

User Authentication Manger

R3: The system must allow users to take images and to input details of violation

Violation Manager

R4: The system must allow users to upload images and data of violation to its database

Violation Manager

R5: The system must be able to communicate with its database

Database Manager

R6: The system must be able to store data of violation in its database

Database Manager

R7: The system must allow users to send a request for data of violation

Data Manager

R8: The system must be able to anonymize data accessed by users

Data Manager

R9: The system must be able to send the data of violation to the email the user had registered with.

Data Manager

R10: The system must allow active municipalities to filter data by type, time, date, and location of violation

Data Manager

R11: The system must allow municipalities to access the data of violation

Data Manager

R12: The system must be able to communicate with active municipalities to acquire data of accidents

Data Manager

R13: The system must allow users to see a list unsafe areas

Statistics Manager

R14: The system must allow users to filter the list of unsafe areas by location, type, time, and frequency

Statistics Manager

R15: The system must allow users to enable/disable notification about the area they are currently in.

Notification Manager

R16: The system must allow users to get safety notifications about the area he is currently in

Notification Manager

R17: The system must be able to notify the user if the safety status of an area has changed

Notification Manager

R18: The system must allow a municipality to activate its account

Municipality Authentication Manager

R19: The system must allow active municipality to access its account

Municipality Authentication Manager

R20: The system must allow active municipalities to download generated file of violations

Data Manager

R21: The system must allow active municipalities to check the suggestions offered by SafeStreets

Suggestions Manager

4.2 Non-Functional requirement mapping

As discussed in detailes in the RASD in chaper two of this documents, security of information is of paramount importance to the system. The direct design components that are responisble for protcring image integrity and confidentiaity is Mobile App Security Manager on the user side and Security Manager at the server side.

5. Implementation, Integration and Test Plan

5.1 Overview

In order to follow an organized way in tackling the project, we decided to follow the phases stated in Waterfall model. In fact, we have been following this model up to now by defining after finishing the RASD and the DD documents. The phases of this model in their sequential order are: **Concept Exploration, Requirements Analysis, System Design, Implementation, Testing, Deployment, and Maintenance.** It is intuitive to link Requirements Analysis phase to the RASD V1.1 document already written and explored. Then the System design phase which is similar to the DD almost complete.

It is wise then to plan for the next phases and in particular the implementation and Testing phases. SafeStreets software components described in a good depth in chapter two will be discussed in terms of followed implementation, integration and test strategies.

5.2 Implementation Strategy

5.2.1 Components to be Implemented

Implementation of each component in the system is necessary to develop the system as a whole. Following a list of the components and subcomponents of the system to be implemented. For the detailed description of these components please see chapter two. The main parts of the SafeStreets system are:

- **Mobile App:** this part is composed of the components: mobile app GUI, Security Manager, Request Manager as indicated in figure 2.3

- **Web server:** this part is composed of the components: web app GUI and Web manager.

- **Application server:** this part represent the core part of the business logic is composed of the following components: Data Manager, User Manager, Municipality Manager, Statistics Manager, User Authentication Manager, Notification Manager, Violation Manager, Image Processing Manager, Municipality Authentication Manager suggestion Manager, and Database Manager.

- **Database:** this part is composed of the following components: Database, Parking Violation, Regular users, Accident, Municipality, subscribed, and unsubscribed users.

It's worth to note that implementation of the **external services** is not relevant since we make the assumption that SafeStreet system deals with them as black box, so we only care about the interfaces between them and SafeStreets system and not the actual implementation of these services.

5.2.2 Implementation Order

The approach to be considered for the implementation of SafeStreets system is bottom-up approach; the implementation of each component will take place separately then piecing them together in a specific order to from the big system. With regards to the implementation order, we approached this by defining the features stakeholders look for in the system, and then to identify both its importance to them and its difficulty of implementation. Tables 5.1 and 5.2 summarizes the features and logically divides the features of regular users and municipalities.

Table 5.1: Features relevant to regular users

	Feature	Importance to regular user	Difficulty of Implementation
F1	Log in and Sign up	Low	Low
F2	Report violation	High	High
F3	Access data of violation	High	Medium
F4	See list of unsafe areas	High	Medium
F5	Enable/disable notifications	High	Low
F6	See my reports	Medium	Low

Table 5.2: Features relevant to Municipalities

	Feature	Importance to Municipality	Difficulty of Implementation
F7	Activation and Log in	Low	Low
F8	Access data of violation	High	High
F9	See suggestions about areas	High	Medium

Then we map those features to the corresponding design components defined in chapter two. This mapping is shown below in table 5.3. It should be noted that the responsible components for application server point of view only. Other components might be responsible for the well-functioning of that feature but they are not included here since we are starting with the implementation of the application server. The order of the implementation was decided based on the importance of the features to the customer metric. From a technical perspective, this order might has no benefit, but it is good to keep the customer in our consideration.

It can be deduced that From the way we designed the system, a logical separation between the components of responsible about fulfilling the features of the regular users and those responsible for municipality features can be made even though there are two components (Data Manager and Database Manager) responsible for both regular users and municipalities but this should not make an issue (we partially implement them). We made the decision to begin with the application server and with the components responsible for the functional requirements of the mobile application because it is kind of the essential part of the system in the sense that the dependency of this part on the web application is minimal while most of the functional requirements of the web application depends on the correct functioning of the mobile application.

Table 5.3 Mapping features to components

Feature	Responsible components	Implementation order
F1	User Manager, User Authentication Manager, Database Manager	6
F2	User Manager, Violation Manger, Security Manager, Database Manager	1
F3	User Manager, Data Manager, Database Manager	2
F4	User Manager, Statistics Manger, Database Manager	4
F5	User Manager, Notification Manager, Database Manager	3
F6	User Manager, Violation Manager, Database Manager	5
F7	Municipality Manager, Mun_Authentication Manager, Database Manager	9

F8	Municipality Manager, Data Manager, Database Manager	7
F9	Municipality Manager ,Suggestion manager, Database manager	8

5.3 Integration and Testing

5.3.1 Entry Conditions

The integration of components and its testing should start as soon as possible, however some preconditions must be fulfilled before it can begin. First of all, all the external systems and their interfaces should be available and ready for use.

secondly, the modules which are being integrated should have at least the procedures concerning one another fully developed except for the two components we talked about earlier, if not completed completely. The operations that have been developed should pass the unit tests assure that components are working separately, then if the integration test fails, we can say it is due to integration issues and not unit implementation ones.

5.3.2 Integration Test Strategy

The integration strategy followed is bottom-up approach; the integration should be done by integrating the already implemented components to each other in an incremental way and according to the interfaces depicted in the component diagram in chapter two. This will result in a compound components that should be tested after the integration. Bottom-up approach is useful in avoiding complex problems due to integration.

5.3.3 Flow of Integration

Integration of components of the application server

Figure 5.1 illustrates the order of integration followed in the application server. As explained earlier, the intuition behind this order is based on two factors: importance to the functioning of the system and importance from stakeholders's point of view. It is important to note that the integration of services is done along with the integration process, for example if some component relies on an external service then the order of integration of that service is the same as the order of the components, and not after the integration of the subsystem components. The intuition behind doing this is to ensure full functionality of the components and to minimize the probability of failures of the big subsystem to to the integration of external services.

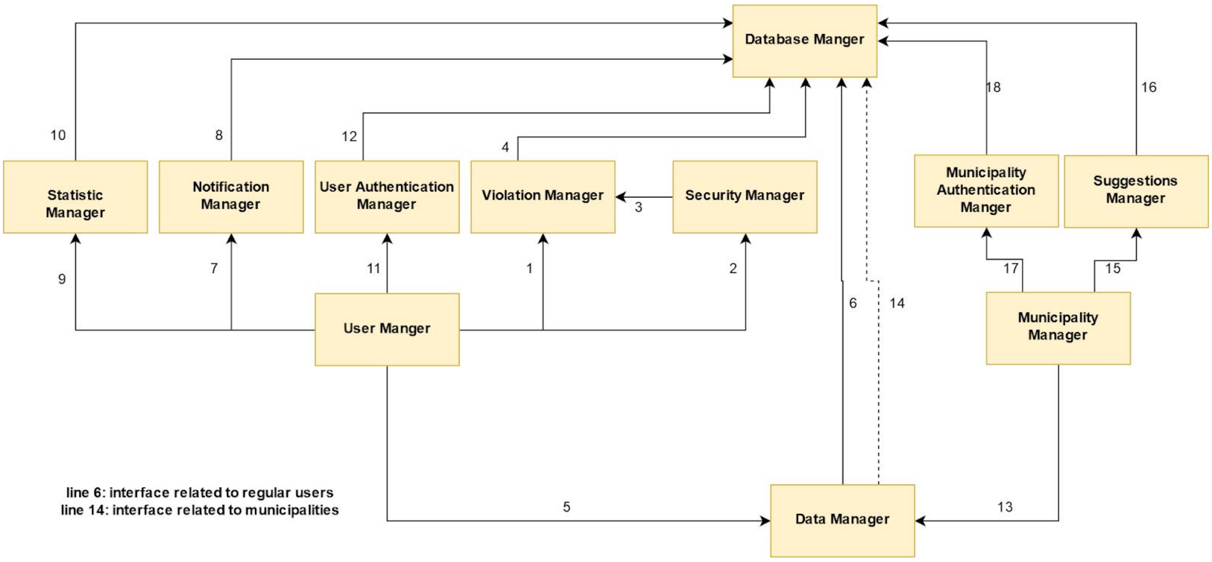


Figure 5.1: Integration order of the Application server components

Integration of components of the mobile application

Figure 5.2 illustrates the order of integration followed in the mobile application. Since the components presents in the mobile application are small in numbers and the business logic done there is relatively simple and easy to implement, the core part then is to develop user friendly GUI and to integrate it to other components. For testing purposes the finaliztion if the GUI might be done in the final steps after ensuring correct behavior of mobile application components. For what concerns the mobile service we only tackling the implementation, integration and testing of the interfaces and not the mape service component. The security manager as mentioned in chapter two is ensure the image is not tampered.

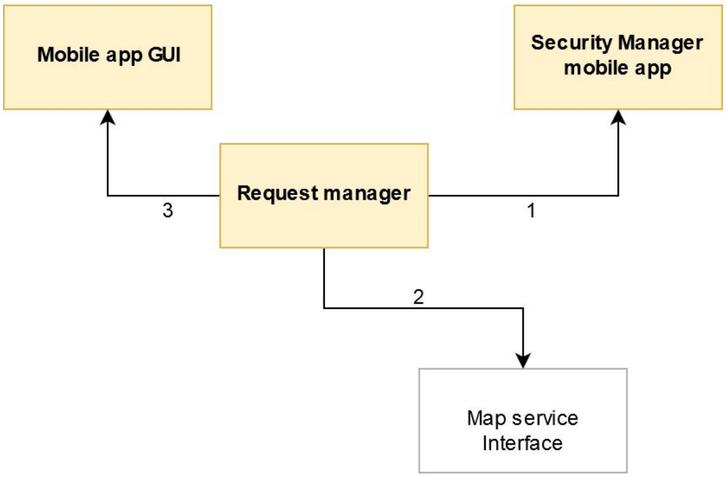


Figure 5.2: Integration order of the mobile application components

Integration of subsystems

After we explained the sequence of integration of components in the application server and mobile application, we move to the integration of the subsystems that constitute SafeStreets system. We did not get into the details of the implementation order of the internal components the web server or database since the number of components is small and it kind of intuitive. But we assume at this point that all the subsystems are fully implemented in isolation and only the interfaces between them are the interfaces left undeveloped.

order	Subsystems	component	Integrated with
1	Application Server, Mobile Application	User Manager	Request Manger
2	Application Server, Database	Database Manager	Database
3	Application Server, Web server	Data Manager	Web Manager

6. Effort Spent

Hesham Mohamed

Task	Hours spent
Component view	7
Run time view	7
Component interfaces	10
Architectural styles	2
Requirement Traceability	1
Implementation, Integration, test plan	3
Documentation	3

Taras Dlamini

Task	Hours spent
purpose	2
Scope	2
Definitions	2
Architecture overview	3
Component interface	2
User Interface layout	1
Requirement Traceability	2
Implementation Strategy	3
Documentation	5

Abdullah Quran

Task	Hours spent
Deployment view	1
Component view	2
Component interfaces	2
Run time view	1
Architectural styles	3
Other design deciosns	2
Implementation, Integration, test plan	7
Documentation	10