Politecnico di Milano
AA 2019-2020

Computer Science and Engineering
Software Engineering 2 Project

DD – Design Document

Version 1.1 – 9/12/19

Authors:
Cem Uyuk – 942341
Sidem Isil Tuncer – 944006
Ali Ozkaya – 943637

Professor:
Matteo Giovanni Rossi

Table of Contents

1. Introduction

1.1. Purpose

The purpose of the Design Document is to delve into the details of how the requirements, which are specified in RASD, will be satisfied in terms of design and architecture choices. The objective of RASD was to describe in a worldly analogy how the users would make use of the application. The design document addresses the software development team to provide necessary information on what patterns will be used to create the application.

1.2. Scope

As stated in RASD, the scope of SafeStreets is that it is a crowd-sourced parking violation reporting service to be which aims to notify authorities about traffic violations. The service should also detect traffic violation hubs and offer possible solutions. The aim is to design an application to facilitate the digital needs for the service which includes gathering crowd-sourced and authority provided data, and sharing the relevant parts of this data with the authorities. Users will easily be able to submit traffic violation reports by taking a picture of the violation and the related car's license plate, instead of going the whole process of contacting the authorities directly. Simplifying the reporting process is also likely to increase the amount of reports coming from the public.

The application will focus on parking violations instead of all kinds of traffic violations because upon checking reviews of similar applications we came to the conclusion that most submissions of non-stationary traffic violations are denied as they either lack a clear display of the violation or the license plate. This creates frustration among the users, which results with low review scores and many users uninstalling the app.

Each user has to verify the submissions of other users while submitting a traffic violation. This system was implemented because according to the studies of similar services, around 75% of the submissions are denied because it is either spam or it provides insufficient information. These faulty submissions will be eliminated in a crowd-sourced way before they are presented to the authorities to decrease the workload of the authority users. A similar application which is used in a single city boasts 50000+ downloads and it is unfeasible to expect authority users to weed out all improper submissions.

Authorities will be able to view submitted violations in a structured order, verify them and generate traffic tickets accordingly, with the information provided. The regular users and authorities will also be able to view a heatmap of traffic violation hotspots, and authorities will be able to view license plates which commit the most violations. The system should be easy to use and reliable for both types of users and should be scalable in case of future content updates.

1.3. Definitions, Acronyms, Abbreviations

1.3.1. Definitions

1. Authority E-mail: The dedicated email address of an authority member.

2. Regular User Interface: The interface a regular user sees once logged into the application.

3. Authority User Interface: The interface an authority uses sees once logged into the application.

4. Violation: Traffic violation to be reported in the application.

5. Submitted Violation: Violation that is submitted by a regular user which is in the process of validation.

6. Validation: The process where a regular user is asked to confirm the information about the submitted violation type and license plate.

7. Validated/Pending Violation: The confirmed submitted violation, which goes into the pending violation queue.

8. Pending Violation Queue: The queue in which the validated violations are present for an authority user to see.

9. Approved Violation: A pending violation which is approved by an authority user, which means that the violation is ready to be ticketed.

10. City of Duty: The city in which the authority user is on duty.

11. Heatmap: The map of a city which shows the violation frequencies of streets in color temperature.

12. Unsafe Areas: The areas in which there are relatively higher number of traffic violations.

13. Intervention Map: A map which displays potentially unsafe areas and suggestions for intervention.

13. Crossed Data: The overlapping unsafe areas in between SafeStreets application's data and the data provided by the authorities.

1.3.2. Acronyms

1. RASD: Requirements and Analysis Specification Document

2. API: Application Programming Interface

3. AI: Artificial Intelligence

4. ID: Identifier

5. DD: Design Document

6. JEE: Java Enterprise Edition

7. DBMS: Database Management System

8. SoC: Separation of Concerns

9. UX: User Experience

1.3.3. Abbreviations

1. [Rn]: n-th Requirement

1.4. Revision History

- Design Document Version 1.0: Release Version
- Design Document Version 1.1
  - Runtime View is updated with respect to the enhanced design choices.
  - Component Interface is updated to be consistent with the sequence diagrams.

1.5. Reference Documents

- Specification Document: 'SafeStreets Mandatory Project Assignment'
- SafeStreets RASD Document by Cem Uyuk, Sidem Isil Tuncer and Ali Ozkaya
- Similar application app store statistics: https://play.google.com/store/apps/details?id=com.ichangemycity.publiceye&hl=en
- UML Diagrams: https://www.uml-diagrams.org
- ETH Zurich Design Document Template: http://se.inf.ethz.ch/courses/2011a_spring/soft_arch/downloads/Design_Document_Template.pdf
- JEE Framework Details: https://java-source.net/open-source/j2ee-frameworks

1.6. Document Structure

- Chapter 1 – The Introduction: Purpose, scope and relevant information required in the reading of the document is set forth in this chapter.
- Chapter 2 – Architectural Design: This is probably the most extensive information on the application. It contains the architectural design choices in the development of the system. It contains:

- Overview: High Level Components and Their Interaction
- Component View
- Deployment View
- Runtime View
- Component Interfaces
- Selected Architectural Styles and Patterns
- Other Design Decisions

- Chapter 3 – User Interfaces: The mock-ups and user experience diagrams are presented.
- Chapter 4 – Requirements Traceability: The mapping of requirements, described in detail in RASD, with components are shown.
- Chapter 5 – Implementation, Integration and Testing: Both the chronological and content related information on implementation, integration and testing are given in this chapter.
- Chapter 6 – Effort spent by the authors in the creation of the document.

2. Architectural Design

2.1. Overview: High-Level Components and Their Interaction

The application to be developed is divided into three layers, which are client, application and data access. The diagram that shows this division can be seen below:

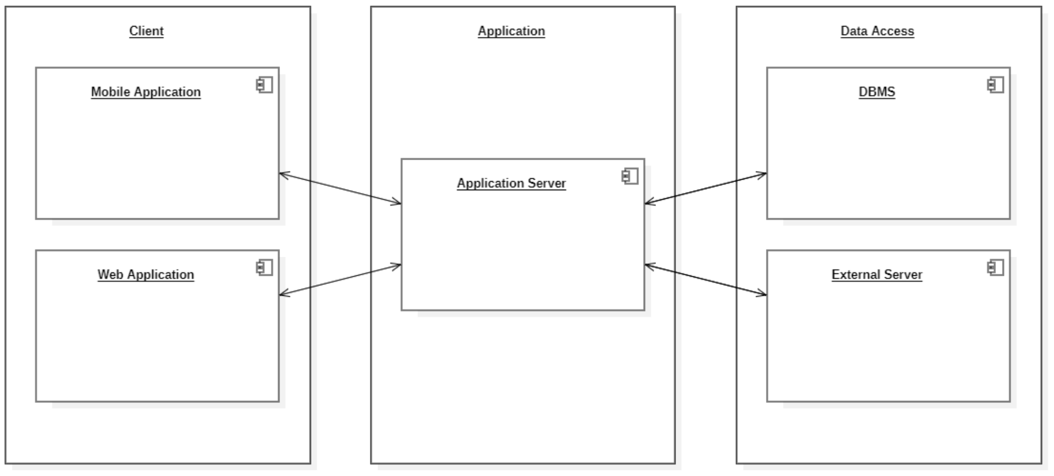


Figure 1: High-level architecture

The client layer is divided into mobile and web application because the application supports two different types of devices for their respective users. This layer contains the view that is presented to the users and is tasked to receive user requests via user interfaces. These requests are transferred to the application layer to be handled. The client also informs the user about notifications coming from the application layer and updates its view according to the response. Application layer contains the algorithms and functions necessary to handle user requests. Requests from the client are sent to the application server, which utilizes its internal components to handle these requests, application server relies on API's for map creation and license plate recognition. Data access layer is accessed by the application server in order to retrieve or update the stored data based on user requests. While DBMS supports both read and write operations, external server access is read only.

2.2. Component View

The following diagram contains all the components of the system and depicting their interactions. The depiction contains every component involved in the system, however the only physical component described in detail is the Application Server. It is the main structure that keeps the application functioning. The details can be seen in the following figure and, additionally, the explanations can be found after the depiction.

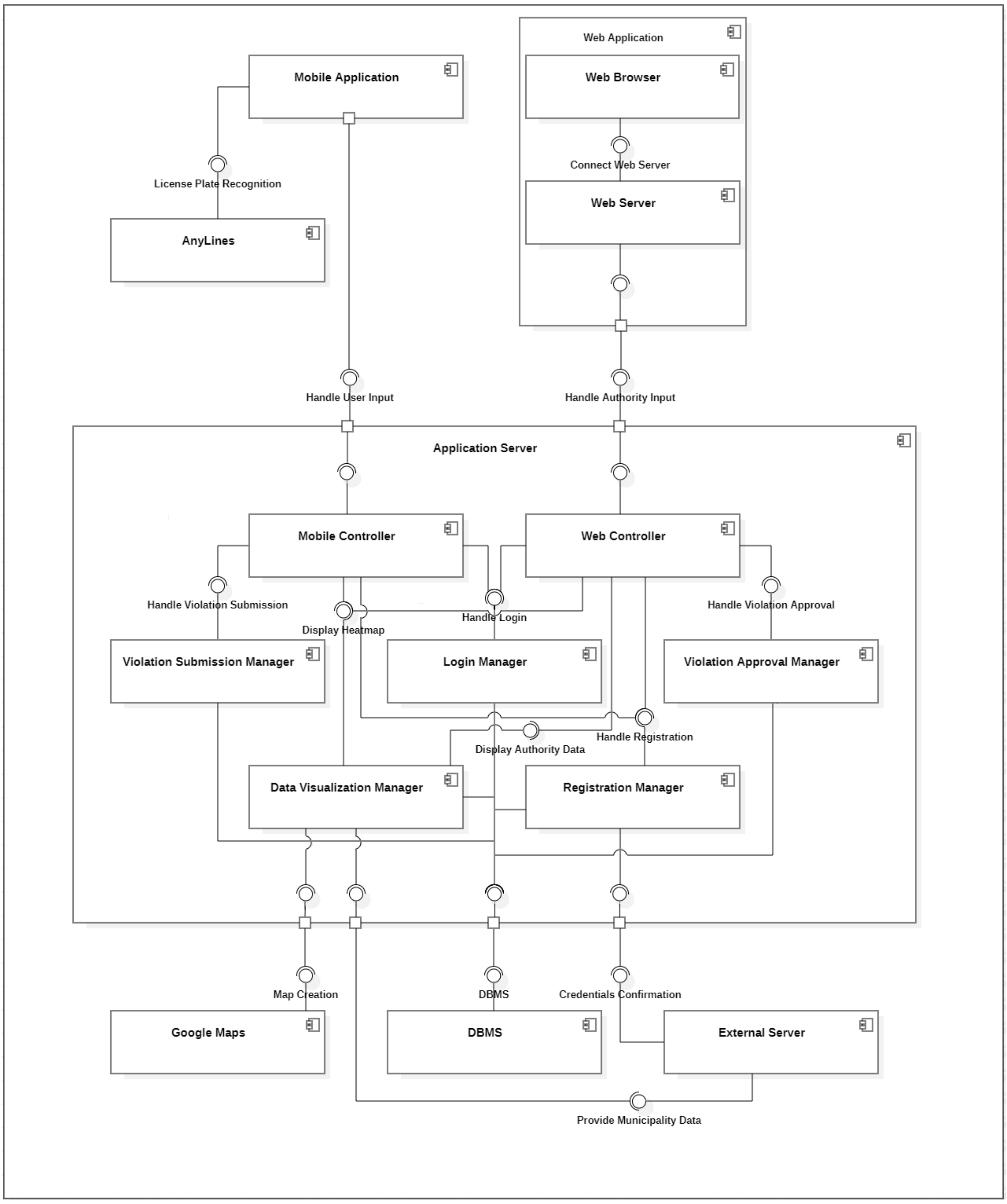


Figure 2: Component Diagram

- **Mobile Controller:** The Mobile Controller converts the requests of the regular user to commands and directs these commands to their respective components. It serves as a middle ground between the mobile application and the server. Mobile Controller's required interfaces are used to change the model according to regular user input. It also oversees violation creation process by providing the necessary user interfaces. These interfaces include **AnyLines** API for license plate recognition.

- **Web Controller:** The Web Controller is the web application version of the Mobile Controller and serves the same purpose for authority user requests. These components are separated from each other because they serve two different platforms and user types. Some required interfaces are shared among these components because they use the same server and some capabilities of the two user types overlap.
- **Login Manager:** The procedures of this component are necessary for a regular user to login to the mobile application or for an authority user to login to the web application. This includes checking the entered fiscal code and password in the database to find a match, so this component utilizes the DBMS interface.
- **Registration Manager:** The procedures of this component are necessary for a regular user to register to the mobile application or for an authority user to register to the web application. An interface of the external server provided by the municipality is used for the confirmation of the registration credentials. Once a registration is complete, DBMS interface is used to register the user information into the database.
- **Violation Submission Manager:** This component contains the necessary procedures for violation validation and submission. It also provides the violation types from the database for the Mobile Application to display during violation creation. This component also interacts with the database to retrieve and update violations during the validation process. The violation data provided by the Mobile Application is submitted to the validation queue by this component when the process is complete.
- **Violation Approval Manager:** This component handles the approval/disapproval updates on the violations which are in the pending queue. Approved violations are registered to be used in the creation data of the heatmap and the intervention map, while disapproved violations are deleted from the database. To realize this purpose, this component interacts with the DBMS interface.
- **Data Visualization Manager:** This component contains the logic which takes the violation data from the database and displays the data mining elements such as the heatmap, the intervention map and the list of most frequent offenders. It provides its interfaces to the mobile and web application according to the authorization of the user

types. The updates of the data mining elements are also handled in set intervals by this component. The external server provided by the municipality is utilized to access the accident records for the updates of the intervention map.

- **External Server:** This component is provided by the municipalities that the SafeStreets application works with. The interfaces of this component is used during the registration process to confirm credentials and during the update of the intervention map to retrieve accident records.

2.3. Deployment View

The classical UML deployment representation of the system is composed of five tiers and it is based on JEE frameworks. The internal and external parts are not shown on the diagram to depict the low level and core development of the system. Explanation of the deployment will be made after the depiction, which is as the following:

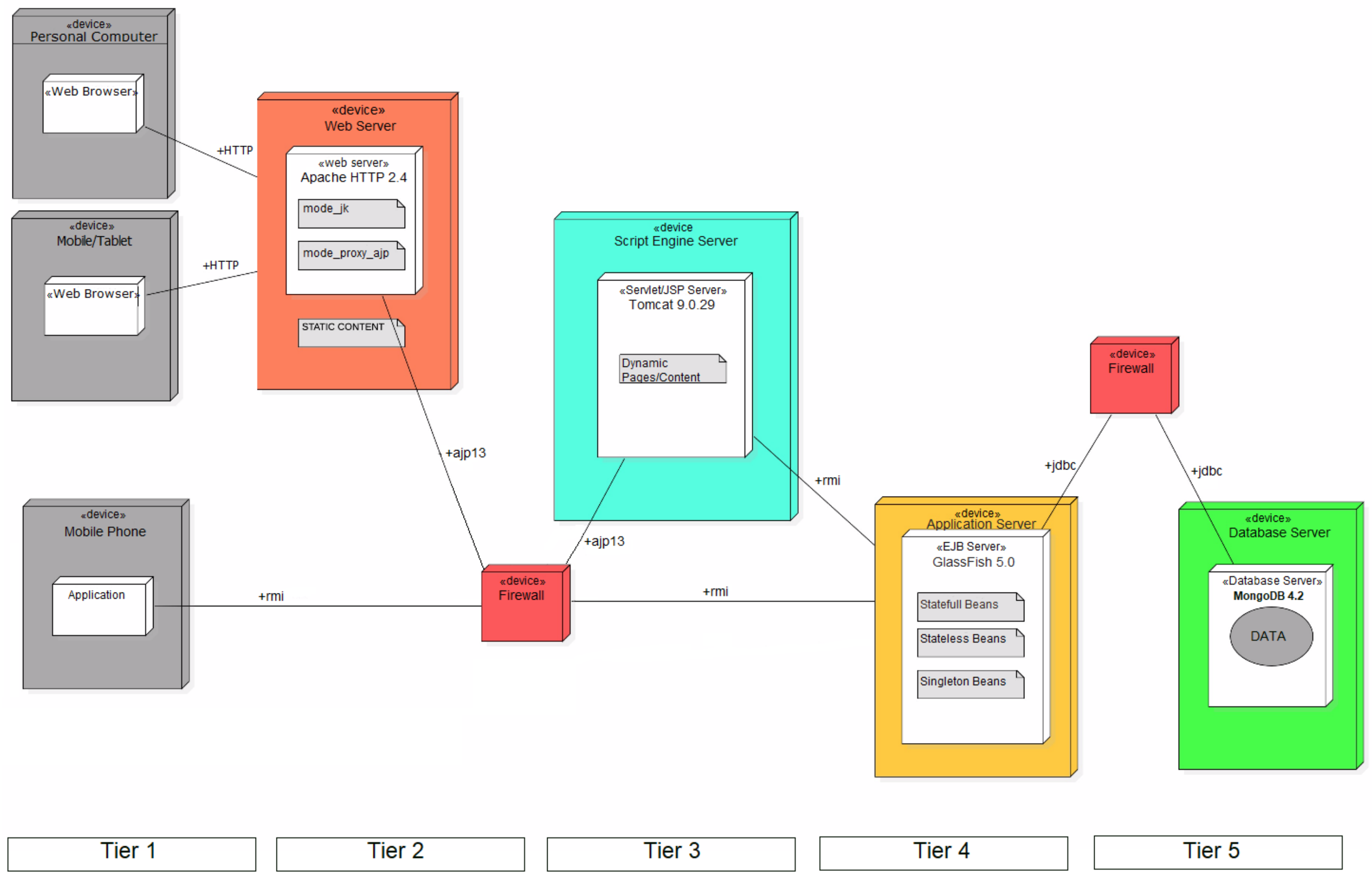


Figure 3: Deployment Diagram

1) Client Tier: There is a mobile application which is connected to the application server directly passing through a firewall. In addition to this, the web application is used by the authority users either on a PC or mobile/tablet. The second component is connected to a web server to go through HTTP protocols.

2) Web Tier: This tier contains the web server implemented on Apache HTTP, which is composed of many core and additional modules among which the mod_jk and mod_proxy_balancer are the important ones for connection with the Servlet/JSP server.

3) Script Engine Tier: This tier contains the Script Engine Server which is created by Tomcat, an extension of Apache, to create the dynamic content and feed the web server for the web application using ajp13 protocol.

4) Application Tier: This tier contains the application server designed with GlassFish that feed directly the mobile app through RMI and the Script Engine Server. The application server contains all the business logic and the beans to handle the relationship between the components.

5) Data Tier: This tier contains the Database Server which is created by MongoDB. The application server receives data through the JDBC protocol from the database.

2.4. Runtime View

2.4.1. Regular User Registration

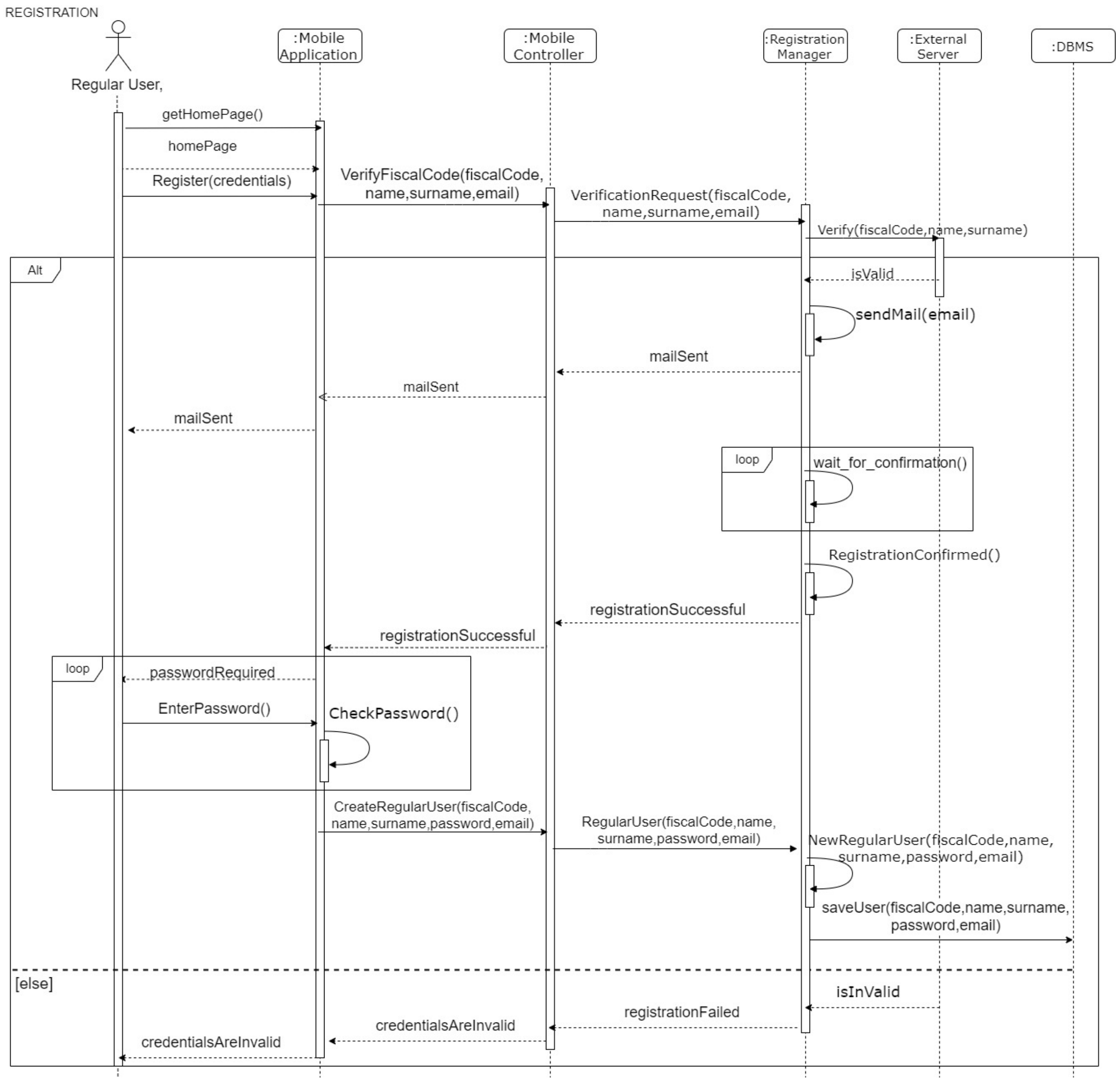


Figure 4: Regular User Registration Sequence Diagram

In Figure 4, registration process of a regular user is explained as a flow in sequence diagram. Registration is an important function in our program since it ensures that users of SafeStreets are real citizens and they are responsible for any malicious behavior under their fiscal codes. After taking the credentials from visitor, through Registration Manager fiscal code, name and surname are sent to an external server connected to the municipality to verify the given data. If the external server verifies the data, registration process continues by sending an email to the visitor to create a password which is handled by Registration Manager. Lastly, the program creates a new regular user and saves it to the database with fiscal code, name, surname, password and email.

2.4.2. Authority User Registration

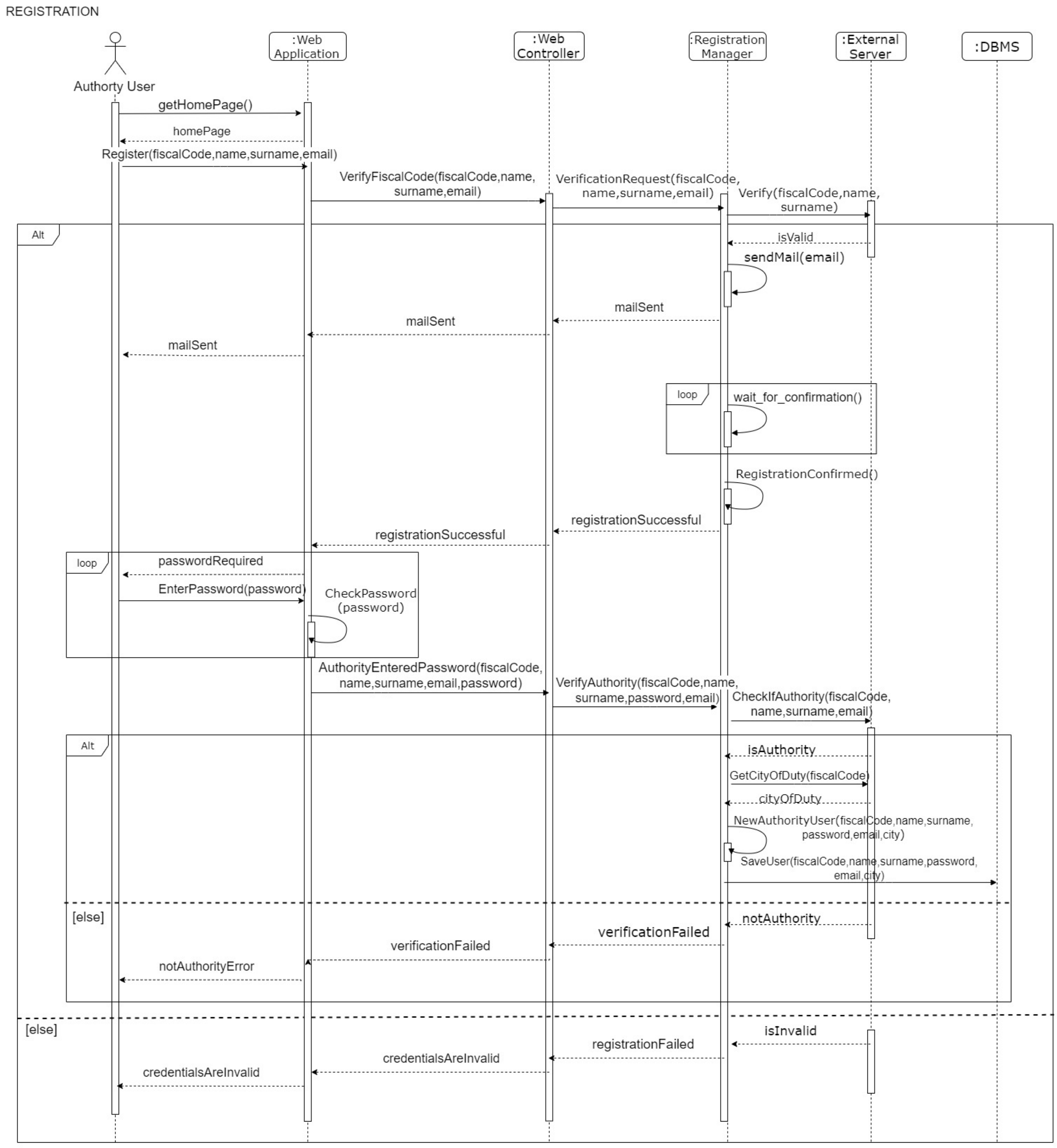


Figure 5: Authority User Registration Sequence Diagram

In Figure 5, similar to the regular user registration, this time authority user registration is shown including some differences in the process. Here, the program first verifies the credentials again, then it also checks if the visitor is an authority using fiscal code, name, surname and his/her dedicated email through the external server. If the visitor is an authority, system gets her/his city of duty from the external server and Registration Manager creates a new authority user and DBMS saves it to the database. If the credentials are invalid or the visitor is not an authority, then an error message is shown to the visitor.

2.4.3. Violation Submission

SUBMIT VIOLATION

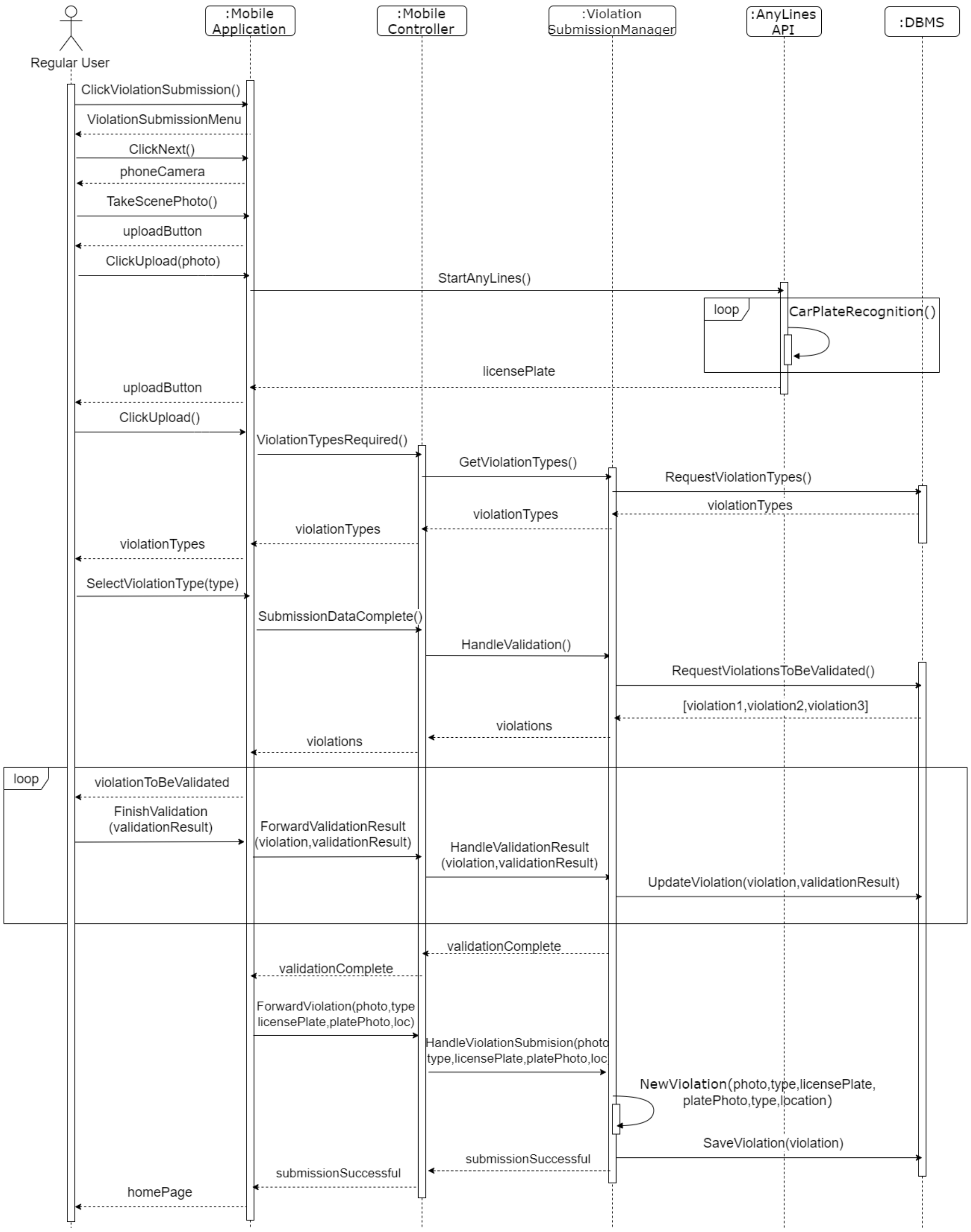


Figure 6: Violation Submission Sequence Diagram

The sequence diagram in Figure 6 shows the flow of a violation submission made by a regular user. After opening the violation submission menu, application opens the camera for the user to take a photo of the general scene of the violation. Upon uploading the photo, AnyLines API is started by the application which is a specialized license plate recognition software. AnyLines works until the user directs the camera to the license plate and it's recognized by the program. After uploading the license plate the user requires to select the violation type from the list on the screen. System fetches the violation types from the database through the Violation Submission Manager and DBMS, and shows them to the user. In order to complete the submission, in the last step user needs to validate three violations submitted before. The violations to be validated are kept in a queue in the database and three of them are fetched and directed to the user through Violation Submission Manager and DBMS. Application shows the violations to the user in turns and saves each result after the user checks the accuracy of the scene, license plate and the violation type. After the validation process is over, three violations are updated via Violation Submission Manager and DBMS accordingly. The new violation submitted by the user is created as a new violation by the Violation Submission Manager and saved to the database by DBMS.

2.4.4. Violation Approval/Disapproval

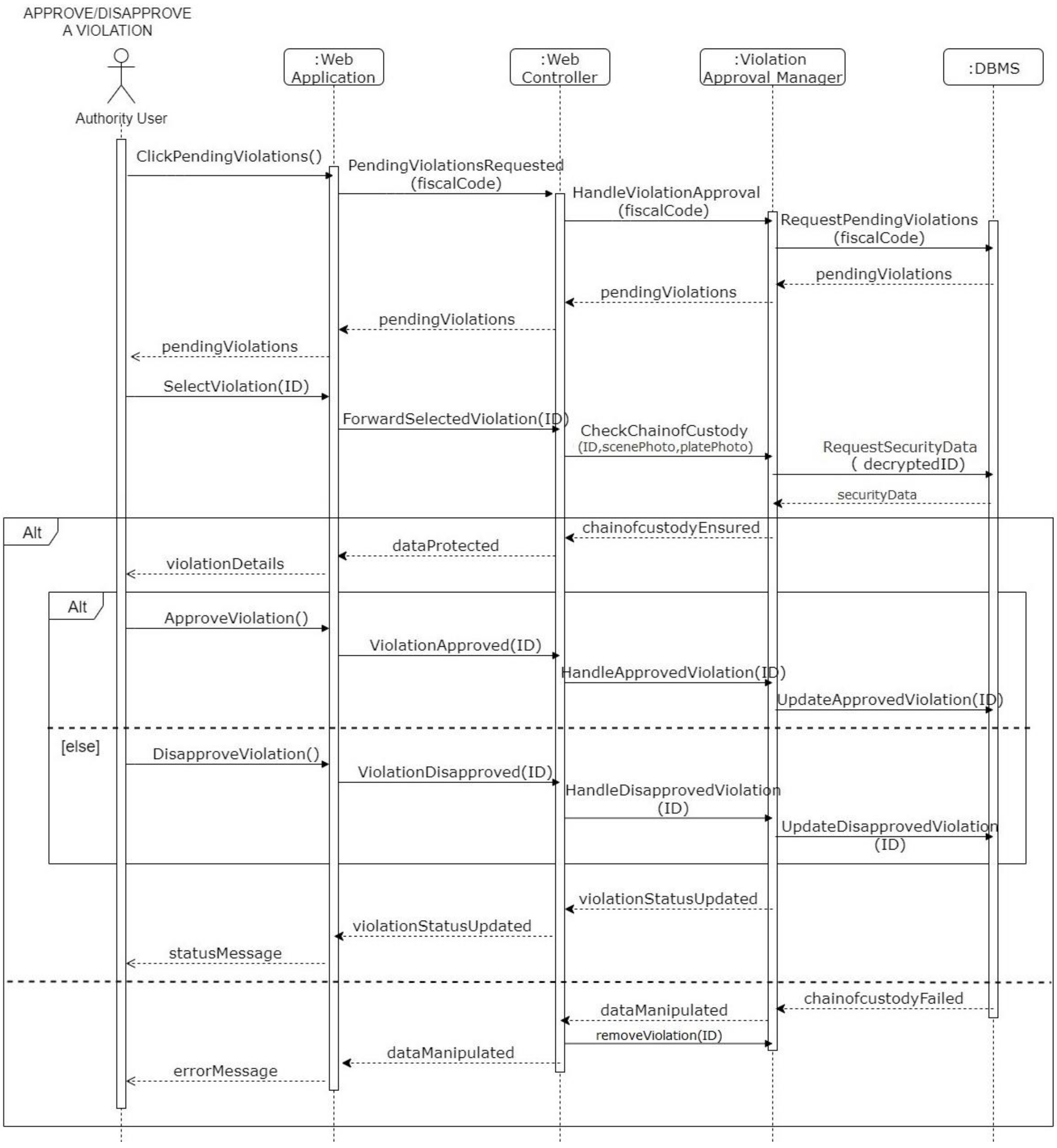


Figure 7: Violation Approval/Disapproval Sequence Diagram

In Figure 7 the flow of approval/disapproval of a violation starts with getting the pending violations from the database which are kept in the pending queue. Particular pending violations are fetched by looking at the authority user’s city of duty using his/her fiscal code by DBMS. When the user selects a violation, the system forwards the selected violation ID, scene photo and license plate photo to the Violation Approval Manager. Here the id is decrypted and security data which are kept in the database are fetched using the decrypted id. Violation Approval Manager compares two information with each other to ensure the chain of custody. Application follows this process in order to ensure no change has been made between the time interval of violation submission and approval. After the authority decides whether to approve or disapprove the violation, system updates the data of the violation in the database accordingly. If the chain of custody fails, the violation is removed via the Violation Approval Manager.

2.4.5. Violation Density Heatmap Visualization

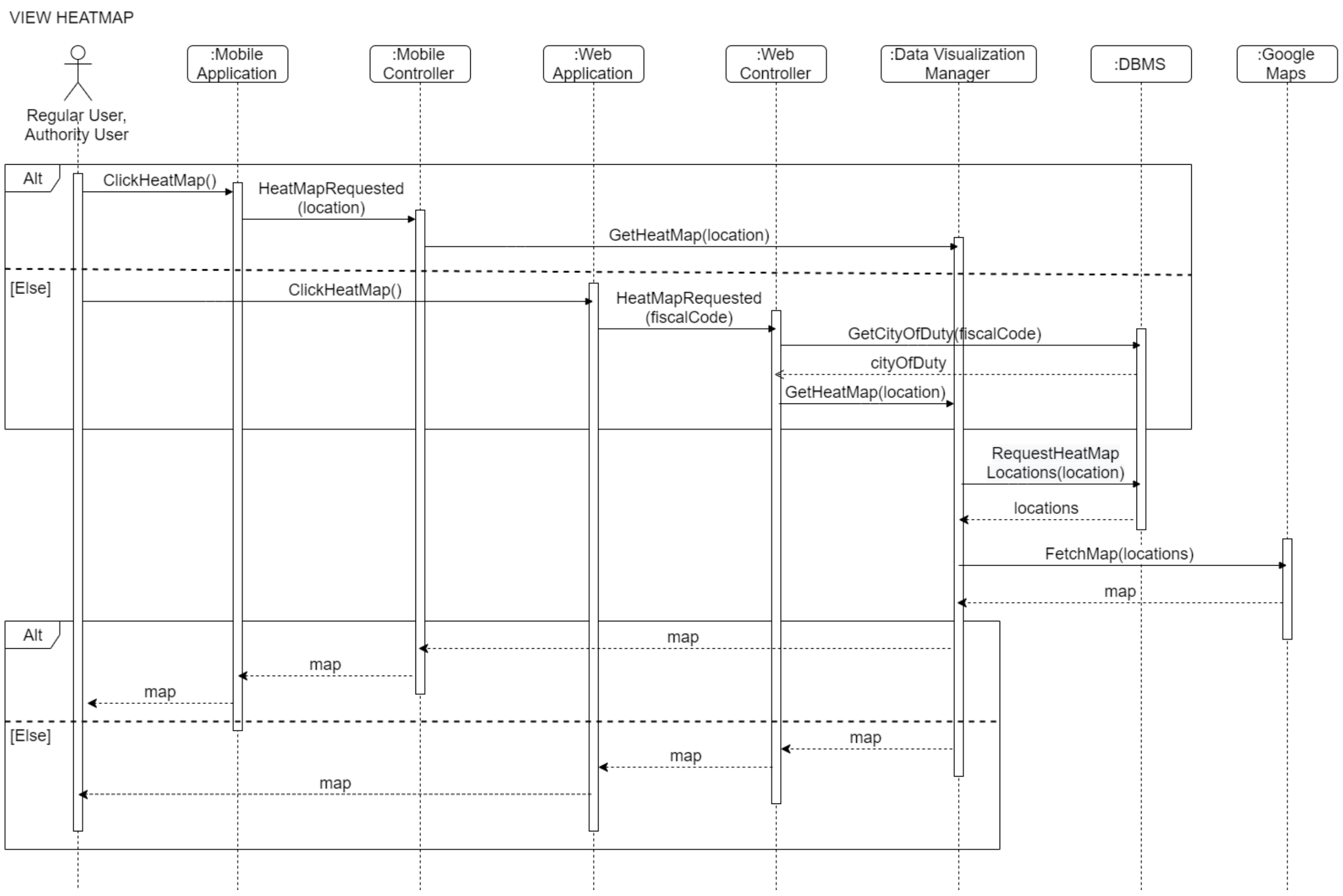
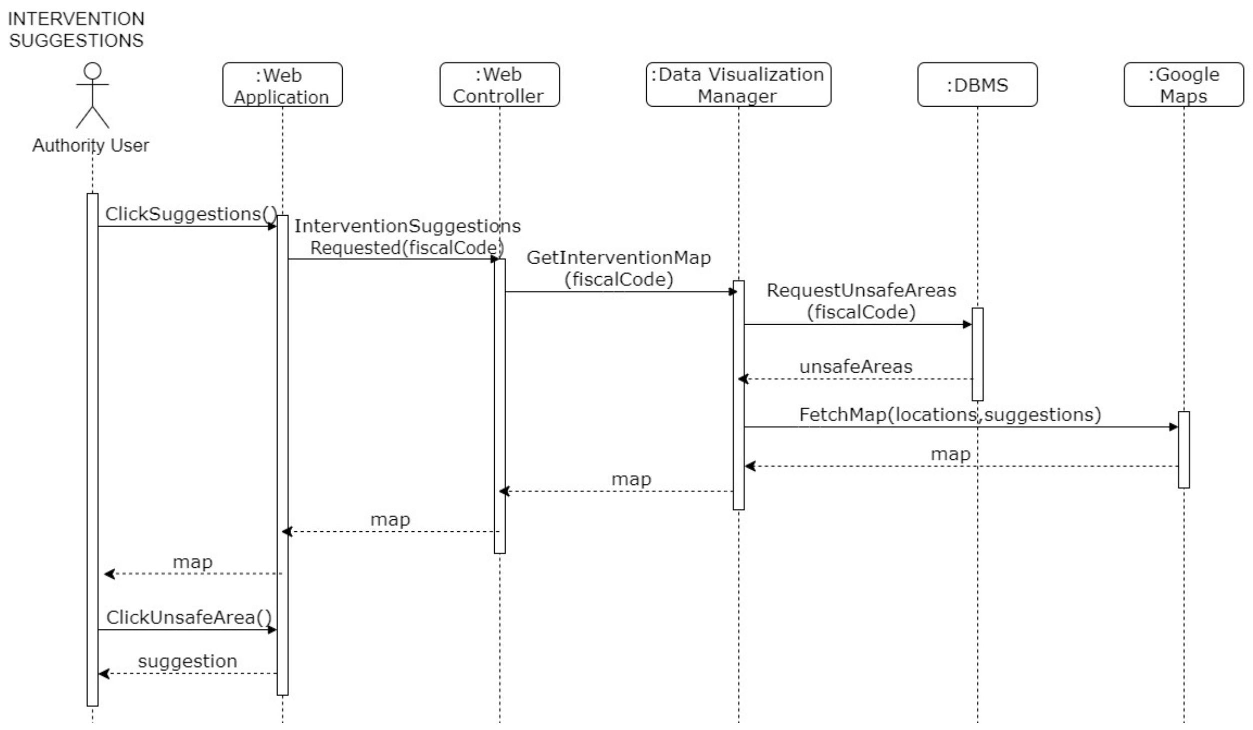


Figure 8: Violation Density Heatmap Visualization Sequence Diagram

The sequence diagram in Figure 8 shows the function of viewing the heatmap for both user types regular user and authority user. While for regular user, Mobile Application and Mobile Controller components are used, for authority user Web Application and Web Controller components are used as shown in an alternative flow in the diagram. To get the heatmap system needs the related location of both types of users. For regular user system simply forwards the location of the user to the Data Visualization Manager, for authority user Data Visualization manager gets the city of duty using his/her fiscal code through DBMS. After that for both users the process is the same, such that the locations that are going to be shown on the map are fetched from the database and forwarded to Google Maps API. Lastly the program shows the map to the user by way of Web Controller and Web Application or Mobile Controller and Mobile Application depending on the user type.

2.4.6. Intervention Suggestion Map Visualization



INTERVENTION SUGGESTIONS
Authority User
:Web Application
:Web Controller
:Data Visualization Manager
:DBMS
:Google Maps
ClickSuggestions()
InterventionSuggestions Requested(fiscalCode)
GetInterventionMap (fiscalCode)
RequestUnsafeAreas (fiscalCode)
unsafeAreas
FetchMap(locations,suggestions)
map
map
map
map
ClickUnsafeArea()
suggestion

Figure 9: Intervention Suggestion Map Visualization Sequence Diagram

The flow of showing the intervention suggestion map to an authority user can be viewed in Figure 9. As in the heatmap visualization, system needs to fetch particular unsafe areas from the database. Via the Web Application and Web Controller, user's fiscal code is forwarded to Data Visualization Manager and unsafe areas are returned here by the DBMS. After that, locations and corresponding suggestions which are present in the unsafe area data, are sent to Google Maps by Data Visualization Manager to get a map that includes both locations and corresponding suggestions. Lastly the map is shown to the authority user and suggestions appear on the map when one clicks an unsafe area on the map.

2.4.7. Frequent Offenders List Visualization

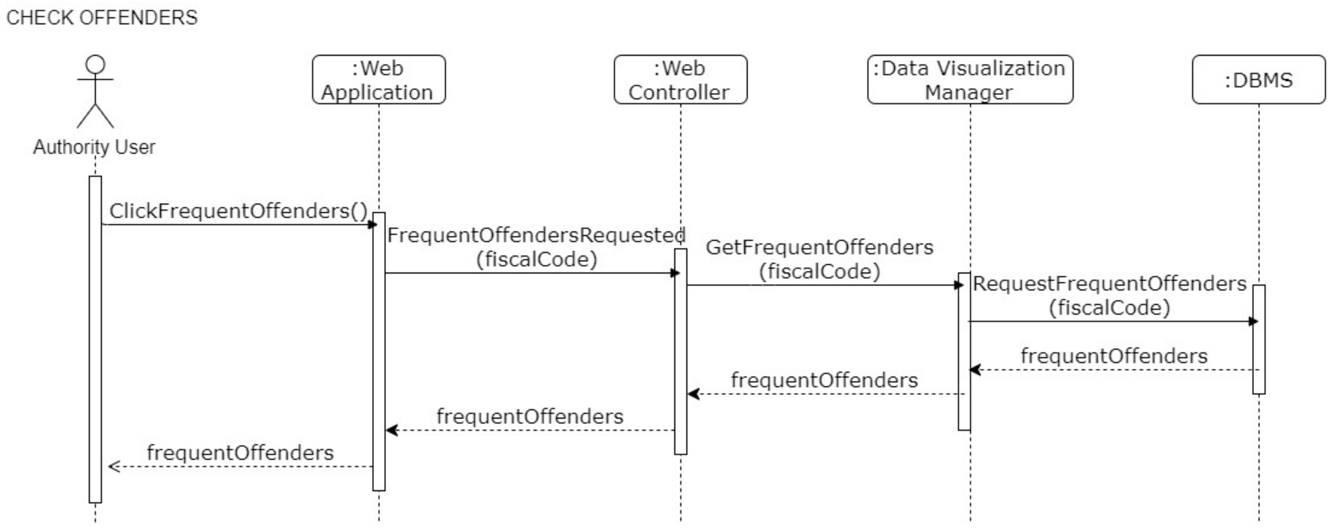


Figure 10: Frequent Offenders List Visualization Sequence Diagram

The application can easily get the frequent offenders related to the authority user's city of duty as shown in Figure 10. DBMS fetches the frequent offenders list from the database using the user's fiscal code and returns it to the Data Visualization Manager. Then the list is sent to Web Application from Data Visualization Manager through Web Controller.

2.5. Component Interfaces

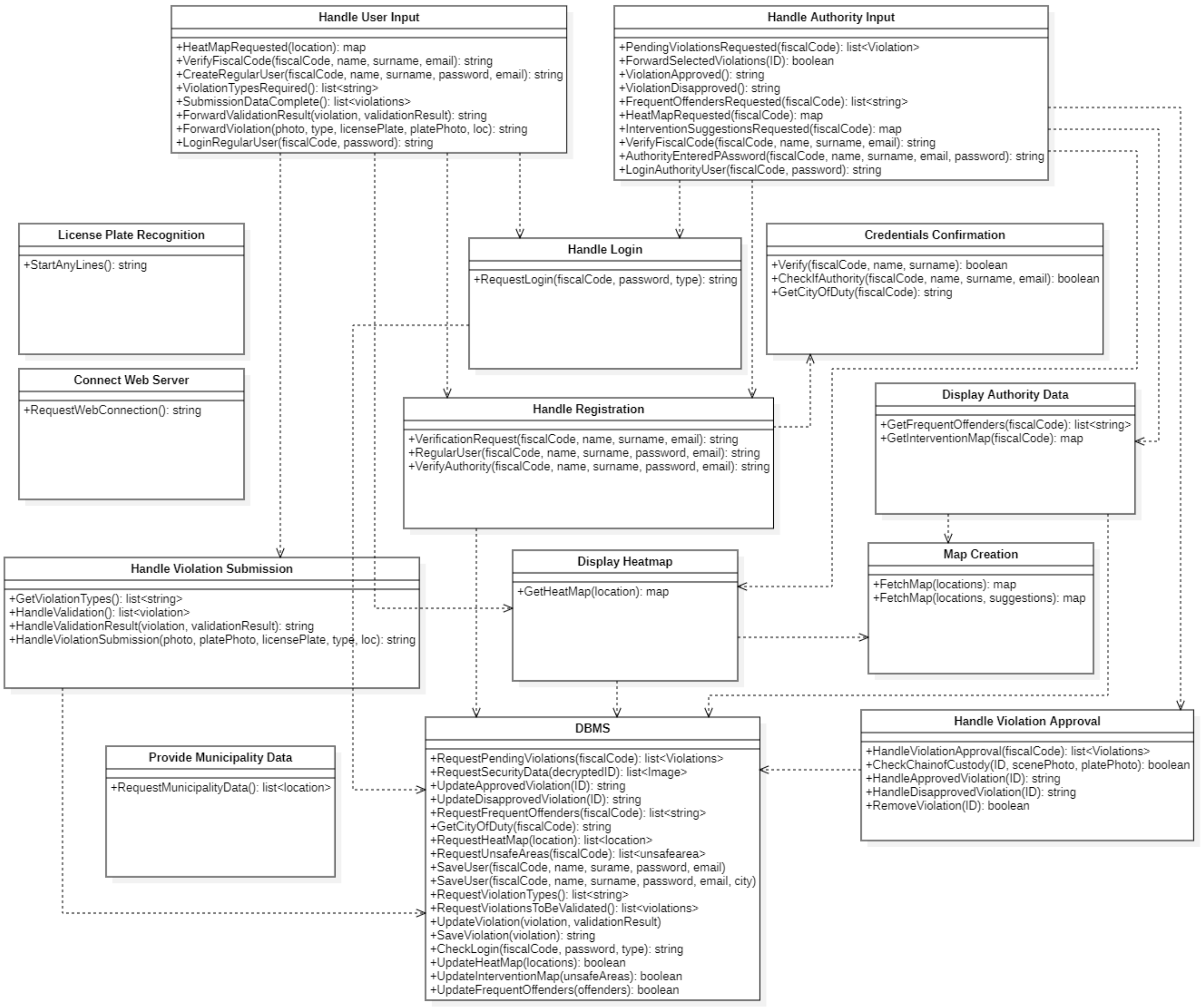

Handle User Input
+HeatMapRequested(location): map
+VerifyFiscalCode(fiscalCode, name, surname, email): string
+CreateRegularUser(fiscalCode, name, surname, password, email): string
+ViolationTypesRequired(): list<string>
+SubmissionDataComplete(): list<violations>
+ForwardValidationResult(violation, validationResult): string
+ForwardViolation(photo, type, licensePlate, platePhoto, loc): string
+LoginRegularUser(fiscalCode, password): string
Handle Authority Input
+PendingViolationsRequested(fiscalCode): list<Violation>
+ForwardSelectedViolations(ID): boolean
+ViolationApproved(): string
+ViolationDisapproved(): string
+FrequentOffendersRequested(fiscalCode): list<string>
+HeatMapRequested(fiscalCode): map
+InterventionSuggestionsRequested(fiscalCode): map
+VerifyFiscalCode(fiscalCode, name, surname, email): string
+AuthorityEnteredPAssword(fiscalCode, name, surname, email, password): string
+LoginAuthorityUser(fiscalCode, password): string
License Plate Recognition
+StartAnyLines(): string
Handle Login
+RequestLogin(fiscalCode, password, type): string
Credentials Confirmation
+Verify(fiscalCode, name, surname): boolean
+CheckIfAuthority(fiscalCode, name, surname, email): boolean
+GetCityOfDuty(fiscalCode): string
Connect Web Server
+RequestWebConnection(): string
Handle Registration
+VerificationRequest(fiscalCode, name, surname, email): string
+RegularUser(fiscalCode, name, surname, password, email): string
+VerifyAuthority(fiscalCode, name, surname, password, email): string
Display Authority Data
+GetFrequentOffenders(fiscalCode): list<string>
+GetInterventionMap(fiscalCode): map
Handle Violation Submission
+GetViolationTypes(): list<string>
+HandleValidation(): list<violation>
+HandleValidationResult(violation, validationResult): string
+HandleViolationSubmission(photo, platePhoto, licensePlate, type, loc): string
Display Heatmap
+GetHeatMap(location): map
Map Creation
+FetchMap(locations): map
+FetchMap(locations, suggestions): map
Provide Municipality Data
+RequestMunicipalityData(): list<location>
DBMS
+RequestPendingViolations(fiscalCode): list<Violations>
+RequestSecurityData(decryptedID): list<Image>
+UpdateApprovedViolation(ID): string
+UpdateDisapprovedViolation(ID): string
+RequestFrequentOffenders(fiscalCode): list<string>
+GetCityOfDuty(fiscalCode): string
+RequestHeatMap(location): list<location>
+RequestUnsafeAreas(fiscalCode): list<unsafearea>
+SaveUser(fiscalCode, name, surame, password, email)
+SaveUser(fiscalCode, name, surname, password, email, city)
+RequestViolationTypes(): list<string>
+RequestViolationsToBeValidated(): list<violations>
+UpdateViolation(violation, validationResult)
+SaveViolation(violation): string
+CheckLogin(fiscalCode, password, type): string
+UpdateHeatMap(locations): boolean
+UpdateInterventionMap(unsafeAreas): boolean
+UpdateFrequentOffenders(offenders): boolean
Handle Violation Approval
+HandleViolationApproval(fiscalCode): list<Violations>
+CheckChainofCustody(ID, scenePhoto, platePhoto): boolean
+HandleApprovedViolation(ID): string
+HandleDisapprovedViolation(ID): string
+RemoveViolation(ID): boolean

Figure 11: Component Interfaces

The figure above depicts the relations between the interfaces. Handle User Input and Handle Authority Input interfaces are Facade interfaces which directs user requests to their specialized handlers. DBMS handles the write and read operations on the database and for this reason is utilized by most of the interfaces within the Application Server. Operations which are used for user login and updates on data mining elements were left out in the runtime view for being trivial and not being relevant respectively. These operations are included in component interfaces.

2.6. Selected Architectural Styles and Patterns

Model View Controller: MVC is a common design pattern in developing user interfaces. It divides the system into three elements, namely model, view and controller and ensures the separation of concerns. The user inputs are sent to the controller to manipulate the model, which updates the view that is presented to the user. This style is beneficial because it allows logical grouping of related actions, makes future updates to the system easier to implement, allows the same model to be used for multiple views and promotes parallel development. This style also allows the use of Facade Pattern, because all user inputs are passed to the controller to be distributed to related components.

REST API: REST API introduces a set of constraints to be used in the creation of the application. Following these constraints ensures the application uses low bandwidth during client server communication. These constraints include uniform interface, stateless and cacheable server, client-server model, layered application system. Controllers provide uniform interfaces for their respective users, and the separation between mobile and web application and the application server ensures client-server model which is explained further below. The application server is stateless and relies on client information to fulfil user requests. Client information also states if the response is cacheable or not, which aims to eliminate unnecessary communication between the client and the server. Implementing a layered system is beneficial for future updates and increases system availability.

Client-Server Model: This model is used between the mobile and web applications and the application server. Mobile and web applications are service requesters which provides user input to the resource, which is the application server. This model partitions tasks between these two elements, allowing the mobile and web application to be thin because handling user input and storing data is left to the application server. Thin clients require less resources to function, which lowers the device requirements to run the application. This model also allows storing the data in a single place, which eases the process of data recovery and allows developing and updating the client and server separately.

Facade Pattern: Interfaces provided by the Mobile and Web Controllers follow the Façade Pattern as they supply a single interface which handles all user inputs by distributing them to their related interfaces. This pattern eases the communication between the applications and the server, as it is reduced to a single channel.

Strategy Pattern: Interfaces which overlap in use between the mobile and web application use the strategy pattern to provide polymorphism. This allows regular and authority applications to use the same interfaces despite running on different devices and serving different types of users. This pattern is beneficial for possible future updates because it allows changing the underlying algorithm without affecting the dependent components.

2.7. Other Design Decisions

Relational Database: Relational database is the most used database type. It offers several advantages such as simplicity, ease of data retrieval, flexibility and integrity. Data is naturally organized in relational database and relations between tables are clearly stated for ease of use and development. Data can easily be queried with advanced syntax, which allows combining related tables and filtering results as required during searches. Adding new tables and relations to relational databases does not require updating previous tables and virtually there is no limit to the data that can be stored, which provides flexibility. Data integrity is ensured with validity checks and data typing.

Our application aims to store violation data in several different tables, which will be updated regularly, so ease of data retrieval and update is key. Relational model is beneficial for handling data mining requests such as heatmap and intervention map creation. Data integrity is beneficial to further assist in ensuring the chain of custody of violation submissions.

JEE Framework: Java Enterprise Edition Frameworks allow building applications hosted on servers that run on a cloud service or a data center. It provides compatibility with HTTP client technologies, database and resource access as well as RESTful APIs. These compatibilities and functional uses match with our purposes for SafeStreets.

3. User Interface Design

3.1. Mock – Ups

As described in section 3.1.1 of the RASD document, the user interface design will be composed of two different versions:

- Mobile Application: By using the mobile application, regular users will be able to login, register, submit violation reports, validate others’ violations and visualize a violation heatmap.

- Web Application: By using the web application (either on PC or smartphone/tablet), authority users will be able to login, register, check pending violations, see violation density heatmap, see the list of most frequent offenders, get suggestions for traffic regulations.

In the following sections, the mock-ups provide an idea of how the mobile and web application will look like.

3.1.1. Mobile Application

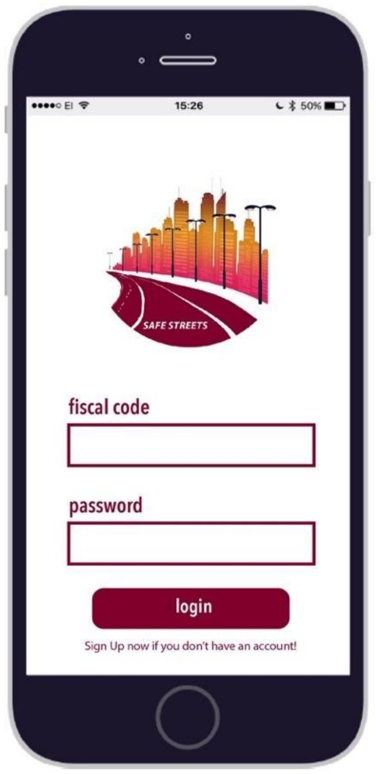


Figure 12: Login

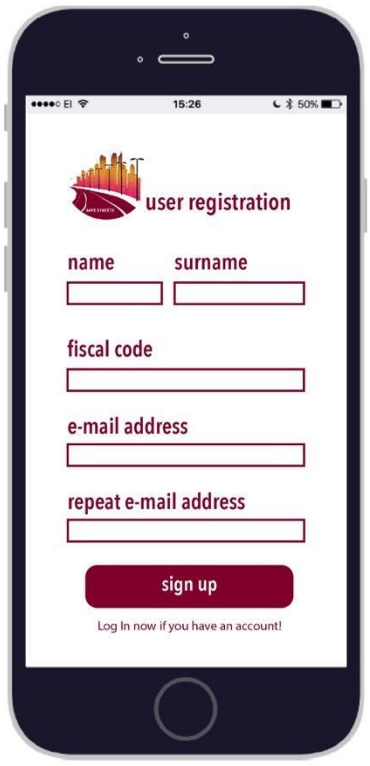


Figure 13: Registration

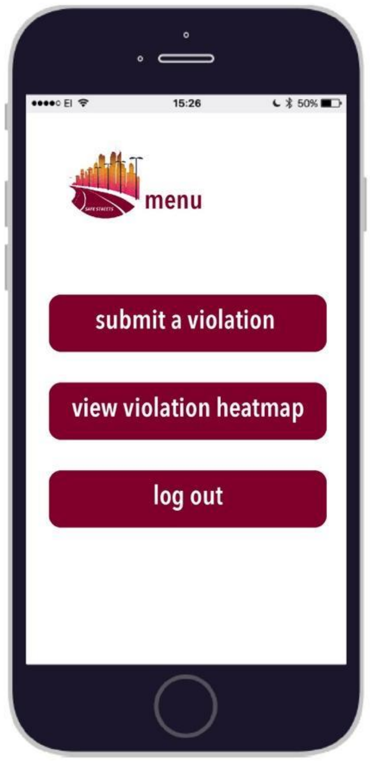


Figure 14: Menu

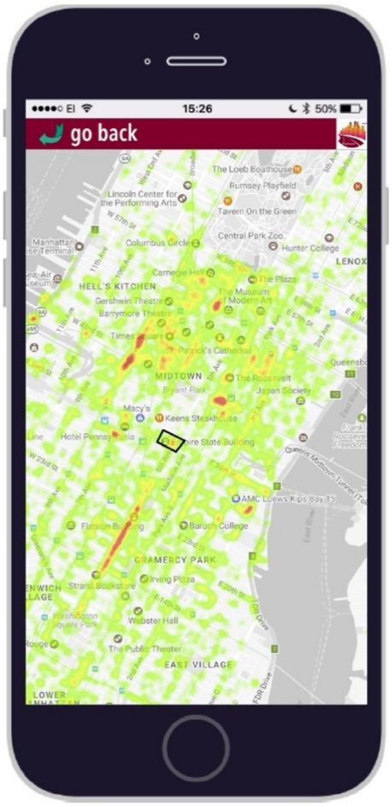


Figure 15: Heatmap

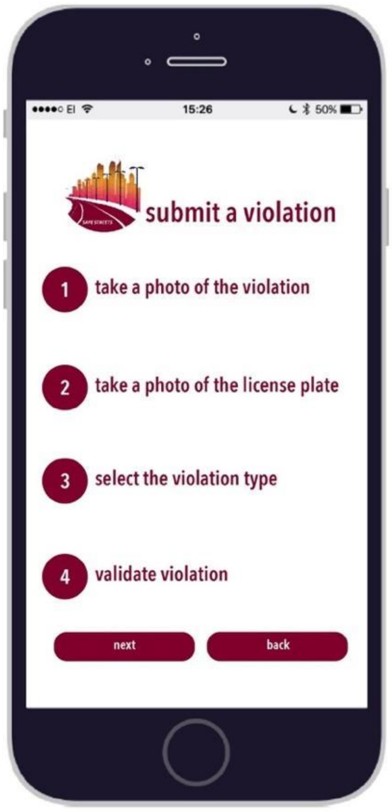

15:26
50%
submit a violation
1 take a photo of the violation
2 take a photo of the license plate
3 select the violation type
4 validate violation
next
back

Figure 16: Violation Submission

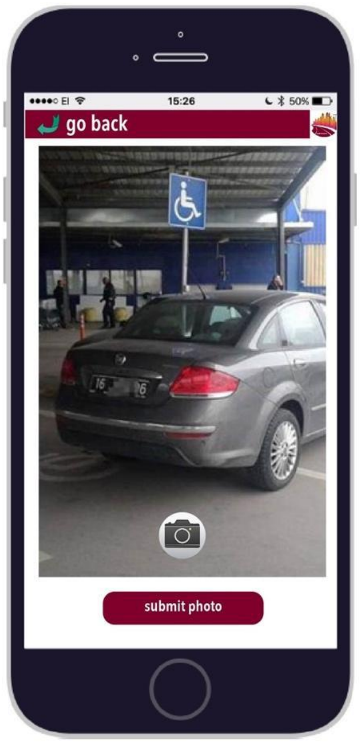


Figure 17.1: Violation Submission Step 1 – Taking a Photo of the General Scene

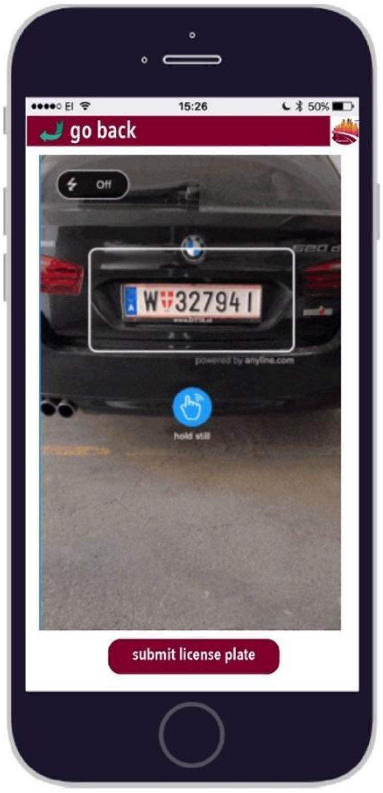


Figure 17.2: Violation Submission Step 2 – License Plate Scanning

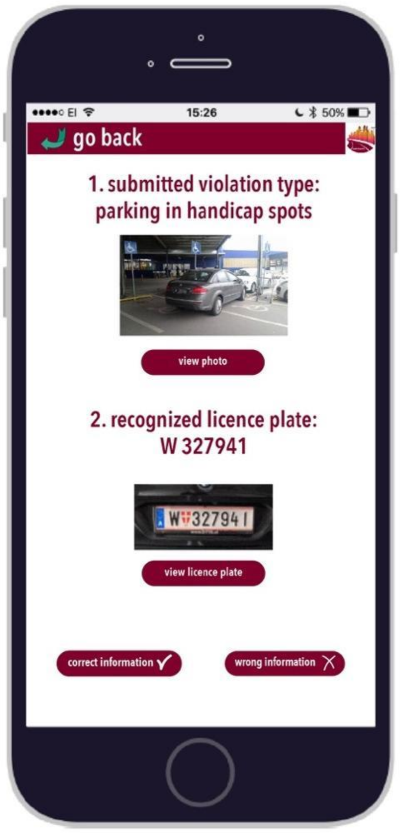

15:26
50%
go back
1. submitted violation type: parking in handicap spots
view photo
2. recognized licence plate: W 327941
view licence plate
correct information
wrong information

Figure 17.3: Violation Submission Step 3 – Violation Validation

3.1.2 Web Application

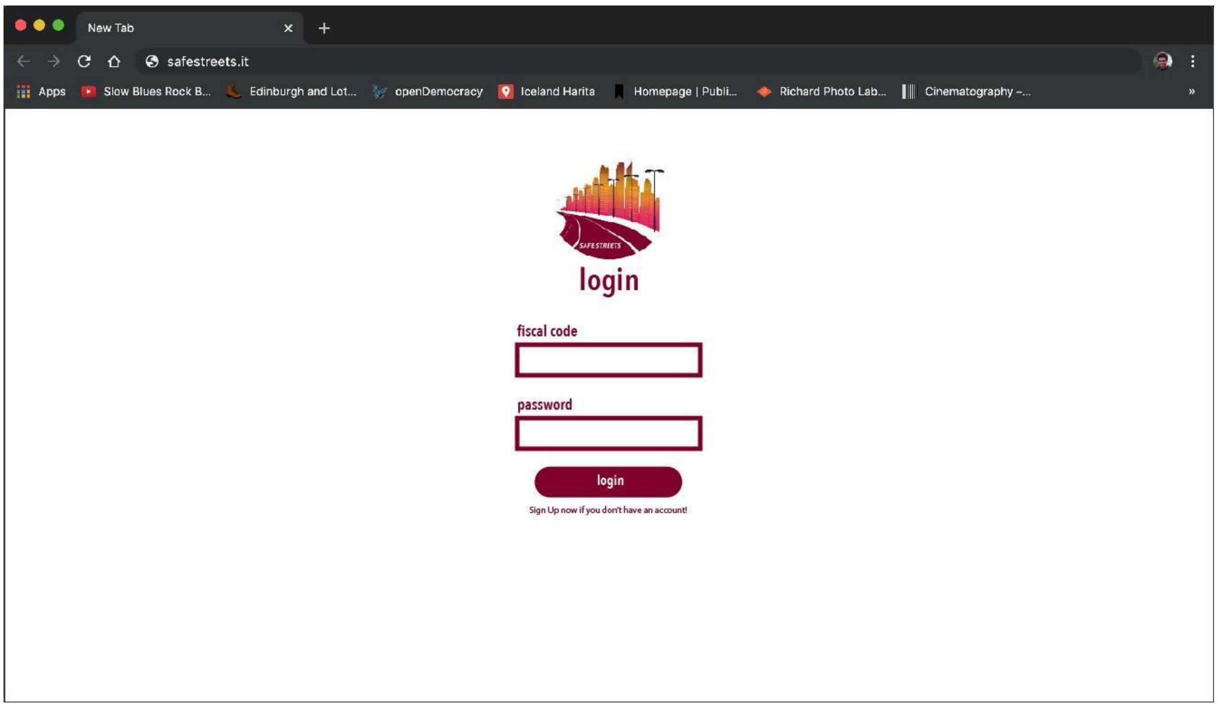


Figure 18: Web Application Login

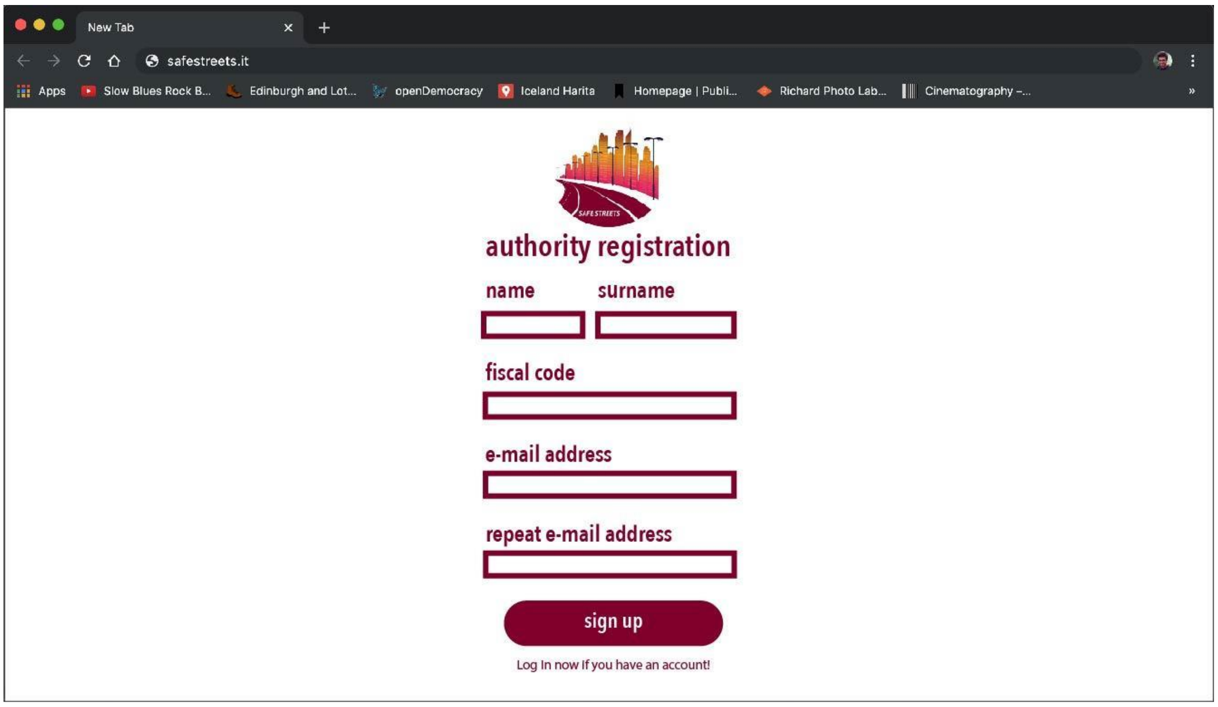


Figure 19: Registration

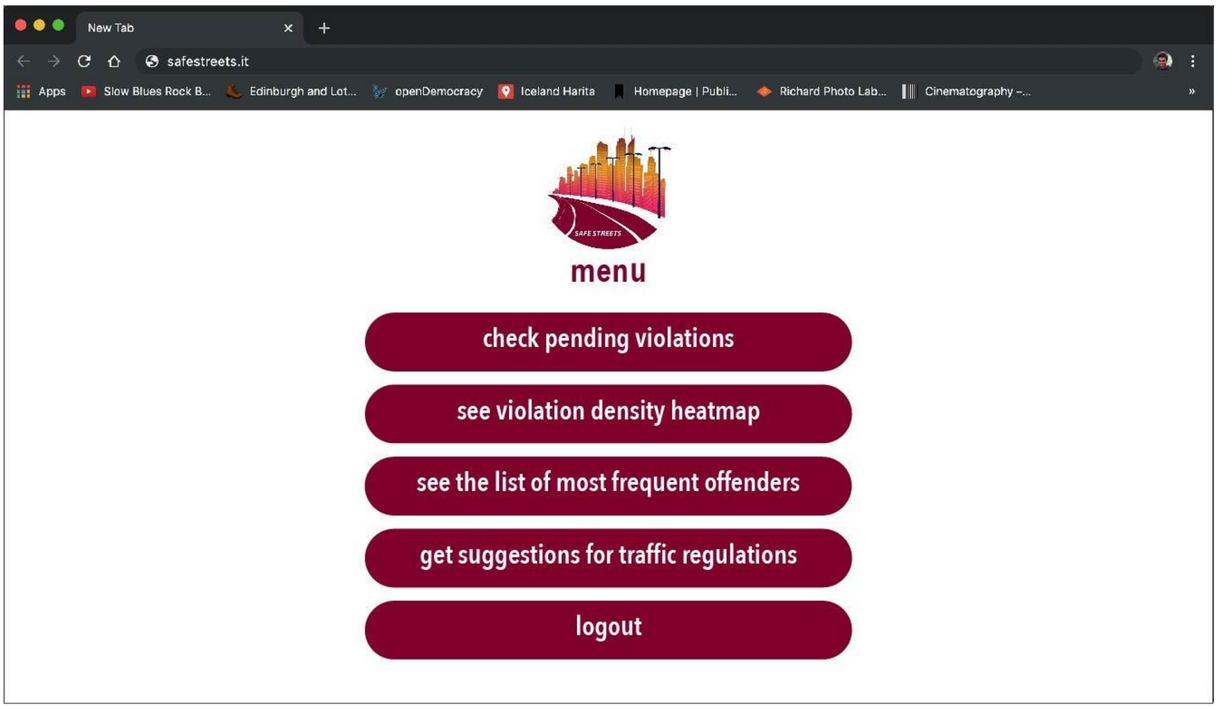
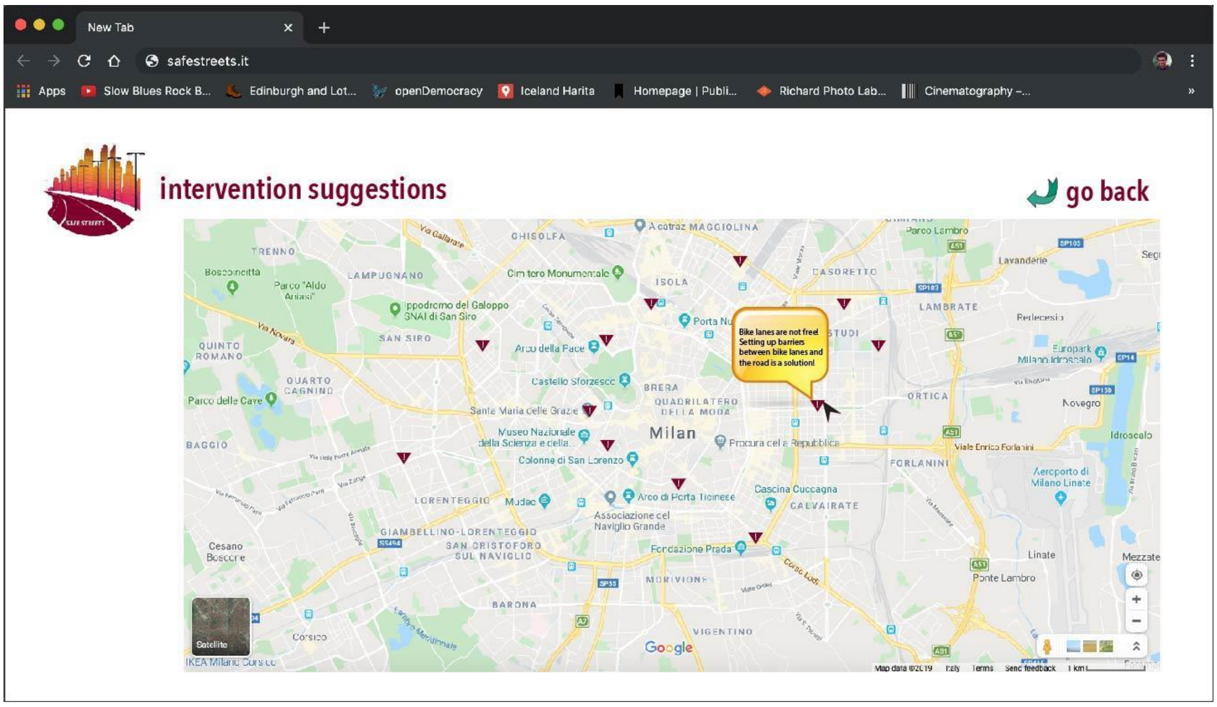


Figure 20: Menu

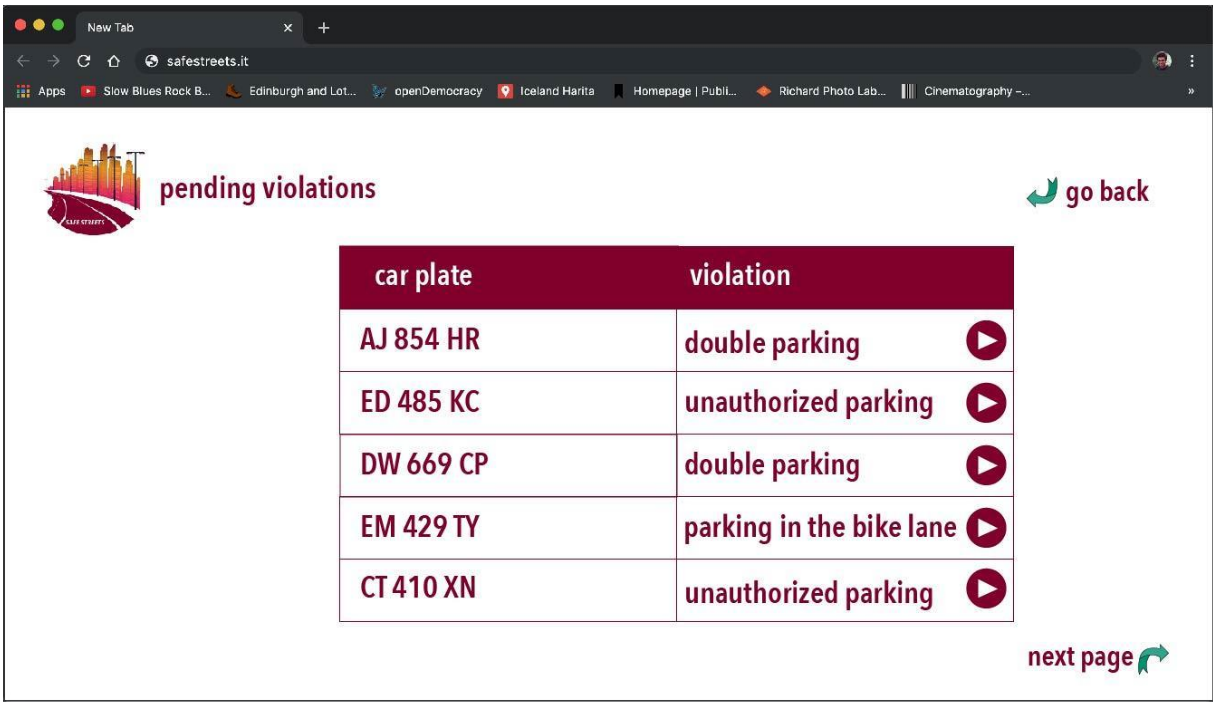


intervention suggestions

go back

Bike lanes are not free! Setting up barriers between bike lanes and the road is a solution!

Figure 21: Intervention Suggestions



pending violations

go back

car plate	violation
AJ 854 HR	double parking ▶
ED 485 KC	unauthorized parking ▶
DW 669 CP	double parking ▶
EM 429 TY	parking in the bike lane ▶
CT 410 XN	unauthorized parking ▶

next page

Figure 22: Pending Violations

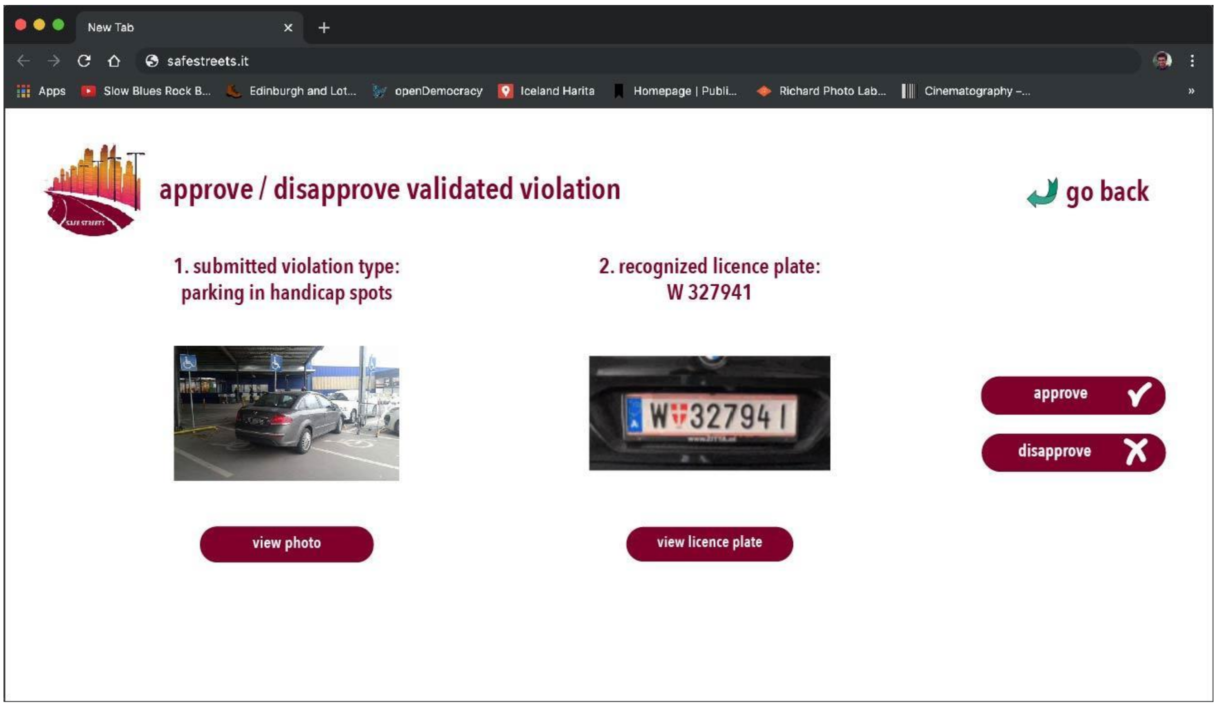
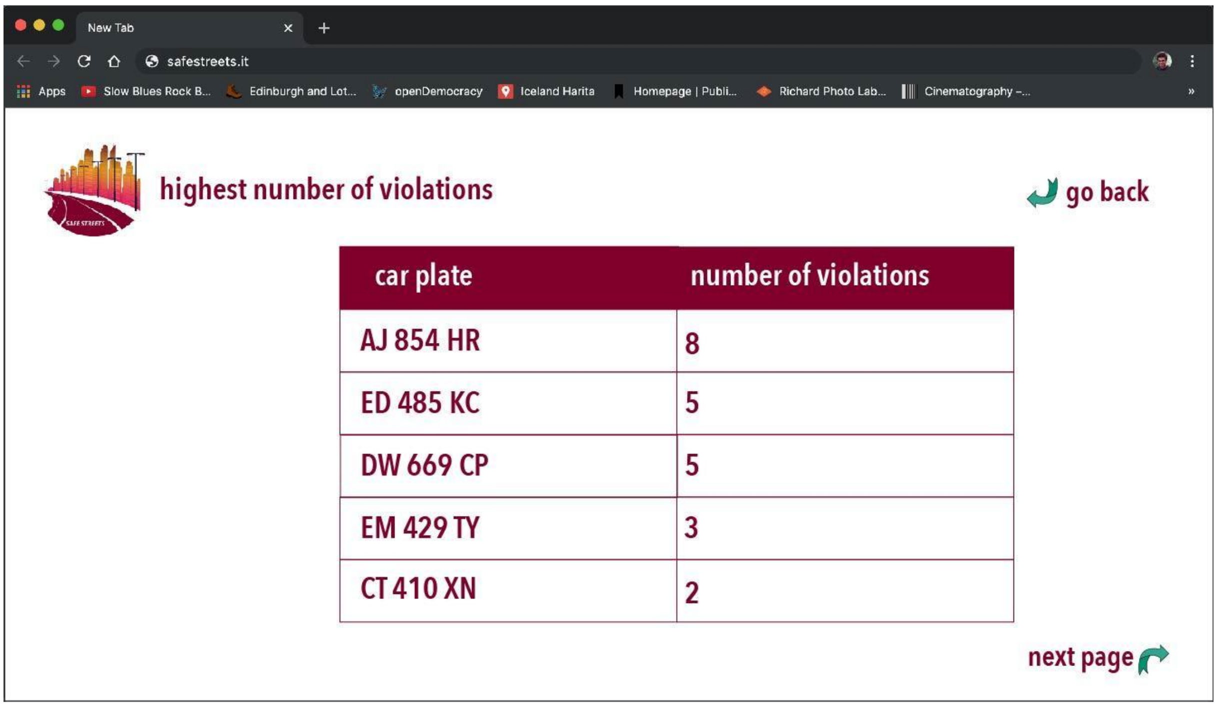


Figure 23: Approve/Disapprove Violations



car plate	number of violations
AJ 854 HR	8
ED 485 KC	5
DW 669 CP	5
EM 429 TY	3
CT 410 XN	2

Figure 24: The List of Frequent Offenders

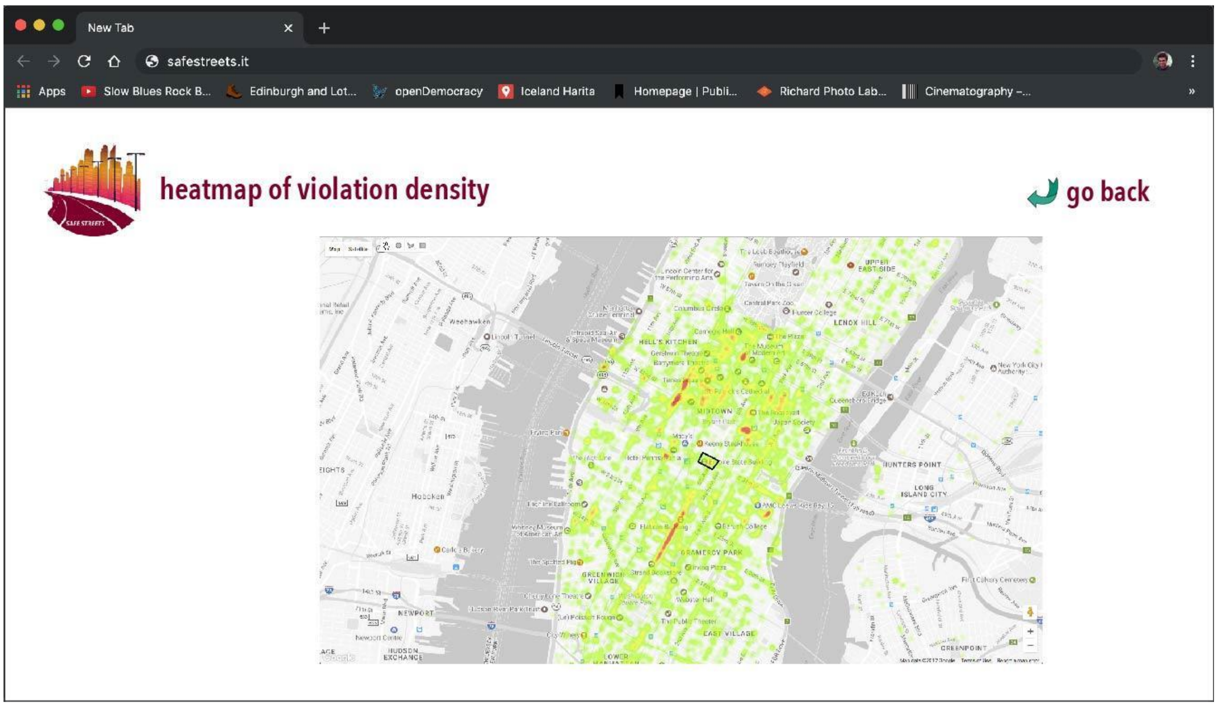


Figure 25: Heatmap

3.2. User Experience Diagrams

In the following User Experience (UX) charts the flow of above depicted mock-ups is depicted. The first diagram shows the flow for the mobile application, whereas the second for web application. In other words, the diagrams respectively represent regular user and authority user experience.

3.2.1. Mobile Application UX Diagram

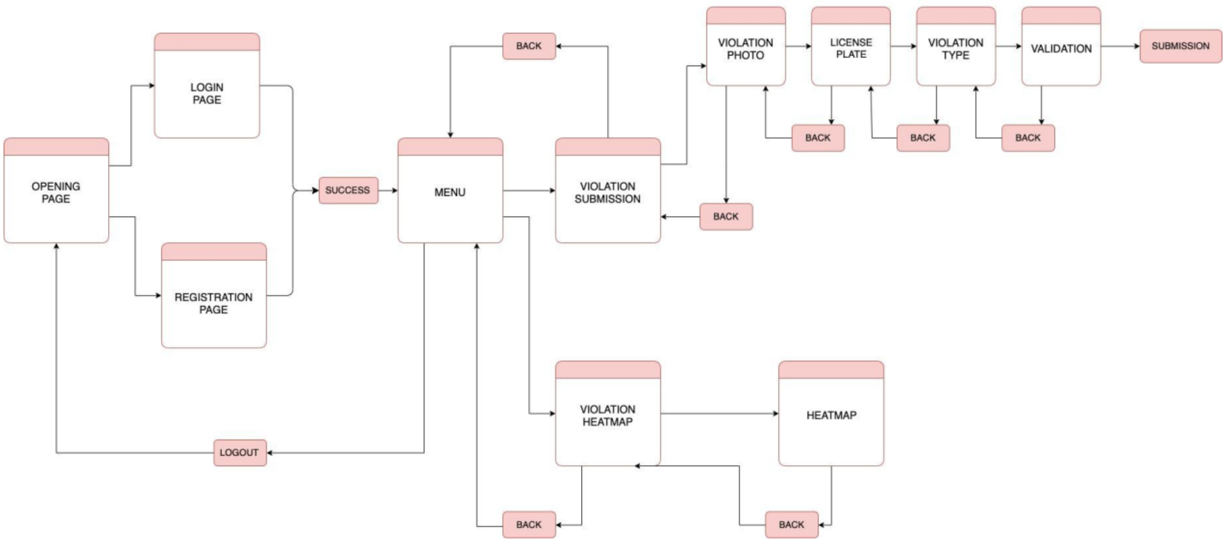


Figure 26: UX Diagram – Mobile Application

3.2.2. Web Application UX Diagram

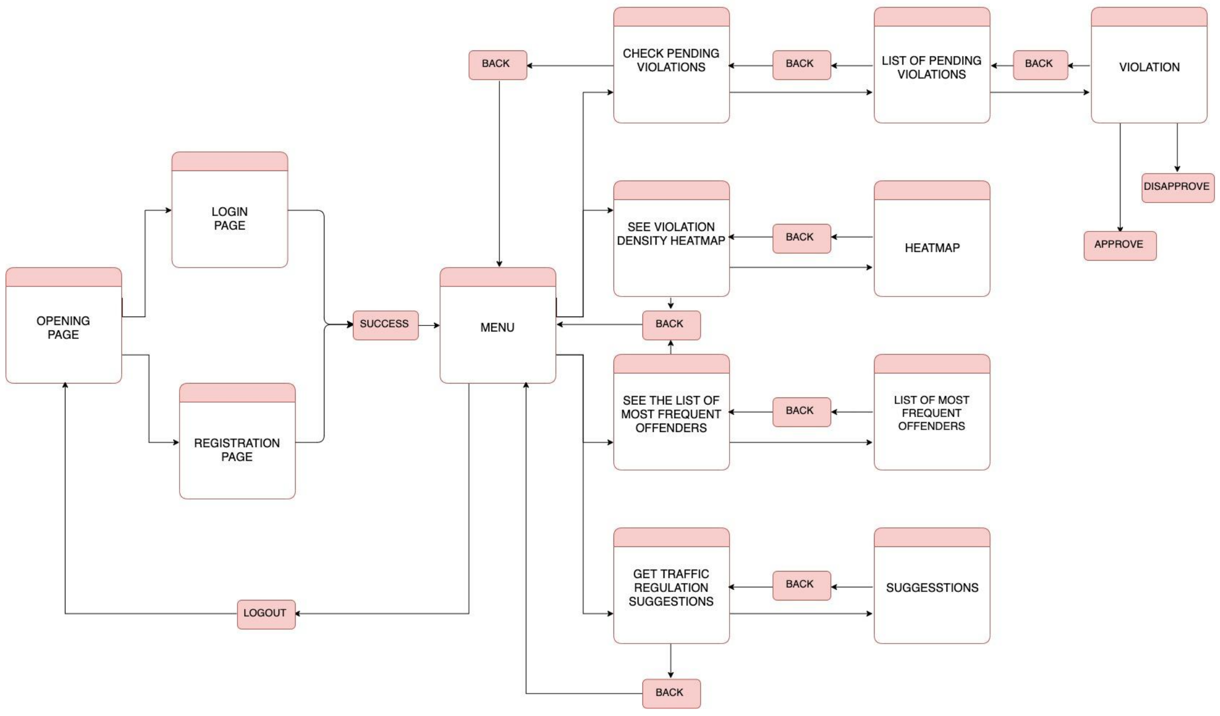

OPENING PAGE
LOGIN PAGE
REGISTRATION PAGE
SUCCESS
MENU
LOGOUT
BACK
CHECK PENDING VIOLATIONS
LIST OF PENDING VIOLATIONS
VIOLATION
DISAPPROVE
APPROVE
SEE VIOLATION DENSITY HEATMAP
HEATMAP
SEE THE LIST OF MOST FREQUENT OFFENDERS
LIST OF MOST FREQUENT OFFENDERS
GET TRAFFIC REGULATION SUGGESTIONS
SUGGESSTIONS

Figure 27: UX Diagram – Web Application

4. Requirements Traceability

The design choices for the software application SafeStreets have been made such that all the requirements, and hence, the goals mentioned in RASD are fulfilled. Below, a mapping between the requirements and components that satisfy the requirements are given:

[R1.1]. A visitor must start with the registration process. The system asks for fiscal code, name, surname, email and phone number to register.

[R6]. An Artificial Intelligence (AI) API is used to recognize the license plate number during taking a photo of the plate.

[R7]. The system presents a list of violation types for the regular user to choose.

[R8]. The application records the time and date of violation submission.

[R9]. The application utilizes GPS data to record the location of the submission.

[R12]. Application shows the pictures of the submitted violation in question and presents buttons for the user to validate the correctness of the car plate number and violation type.

- **Mobile Application:** This component contains the logic which displays the mobile application registration page. This page requests the fiscal code, name, surname, email and phone number of the user to start the registration process as a regular user. This component also handles client-side operations of the violation creation process, including using **AnyLines** to recognize the license plate number. Mobile Application receives the data required to display **[R7]** and **[R12]** from the database through the **Application Server**.

[R1.2]. A visitor must start with the registration process. The system asks for fiscal code, name, surname, email and phone number to register.

- **Web Application:** This component contains the logic which displays the web application registration page. This page requests the fiscal code, name, surname, email and phone number of the user to start the registration process.

[R2]. A user verifies the account and creates a password through a received email.
[R3]. Signing-up to the mobile application results in the creation of a regular user's account.
[R4]. Signing-up to the web application results in the creation of an authority user's account.

- **Registration Manager:** This component handles the registration process for both applications. It utilizes the **External Server** to confirm user credentials, and in the case of an authority user to set a city of duty. After the credentials are confirmed an email is sent to set a password and finalize account creation.

[R5]. A registered regular user must be able to login to the mobile application and utilize the regular user interface.
[R14]. A registered authority user must be able to login to the web application and utilize the authority user interface.

- **Login Manager:** The procedures within this component handles the login attempts for both applications, these attempts are only successful if the fiscal code and password has a match in the database.

[R10]. The application asks a regular user to validate, if exists, three random violations previously submitted by other regular users.
[R11]. Application asks user to validate, if present, three random submitted violations before their submission gets to the validation queue to be validated.
[R13]. A submitted violation goes through three validation processes, if all of them are positive, then the submitted violation becomes a pending violation. Else, the submitted violation is deleted by the application.

- **Violation Submission Manager:** This component oversees supplying the necessary data from the database during violation creation. It retrieves violation types to be listed. During the validation process this component retrieves the violations to be validated from the database and updates them according to user's input when the process is over. When the violation validation is complete, this component also handles the submission of the created violation into the validation queue.

[R15]. Application shows a queue of pending violations to an authority user.
[R16]. Application presents approve/disapprove buttons for each pending violation.
[R17]. Ensure the chain of custody from a regular user's submission to the view of an authority user.
[R18]. The system records the violations that are approved previously by an authority user.

- **Violation Approval Manager:** This component contains the necessary logic which is used to retrieve pending violations from the database for an authority user to view, and to update the approval status of the violations according to input.

[R19]. The system uses the recorded violations and their locations to create a heatmap.
[R20]. The system uses an existing map interface.
[R21]. The system updates the heatmap weekly.
[R22]. The application crosses its data with the information that comes from the municipality.
[R23]. The application makes meaning out of the crossed data in order to suggest possible interventions.

[R24]. The system updates the intervention map monthly.

- **Data Visualization Manager:** This component handles the updates and view requests of the heatmap, the intervention map. Updates are done weekly for the heatmap and monthly for the intervention map. For the updates of the intervention map, **External Server** is utilized to receive municipality's accident records. This component relies on **Google Maps** as an existing map interface. This component is also responsible for querying the database to create a most frequent offenders list on authority user request.

5. Implementation, Integration and Test Plan

5.1. Implementation

Our system is composed of 12 main components:

A. Mobile Application
B. Mobile Controller
C. Web Application
D. Web Controller
E. Web Server
F. Login Manager
G. Registration Manager
H. Application Server
I. Violation Submission Manager
J. Violation Approval Manager
K. Data Visualization Manager
L. External Components (e.g. Google Maps, AnyLines, DBMS and External Server providing data from municipalities)

The development generally will follow a bottom-up approach. The components needed for core functionalities will be implemented, which will, essentially, be building bigger components/subsystems such as the mobile application, web application, web server etc. When the components for core functionalities are ready, the subsystems will be completed, then they will be integrated and tested. Finally we will have a system that is completely implemented, integrated and tested. It is worth to notice that external components are considered reliable. Their integration to the system will later be explained in this chapter. Additionally, every component will be tested within itself first and then the integration of components will be tested. When the integration is complete, the whole application will be tested.

Each component mentioned above serves single or multiple functionality(s). Below table is created to show how the system is divided into tasks and how each component serves that specific task. Depending on the level of importance, implementation difficulty and the number of components involved in making, these functionalities will be graded by an ordinal scale of 0 to 5. Due to the fact that we want to follow a bottom-up approach in our implementation, higher importance tasks with less complicated implementations and smaller number of components will be graded higher. After the grading, higher numbers will receive the implementation priority, and essentially, which component to implement first will be certain too.

Tasks of the Application	Importance	Implementation Difficulty	Component Flow	Grade
Login	Low	Low	(A, B)|(C,D,E), F, L	1
Registration	Low	Low	(A, B)|(C,D,E), G, L	1
Violation Submission	High	High	A, B, H, I, L	5
Violation Validation	High	High	A, B, H, I, L	5
Violation Approval/Disapproval	High	High	C, D, E, H, J, L	4
Heatmap Visualization	Medium	High	(A, B)|(C,D,E), H, K, L	2
Frequent Offenders List Creation	Medium	Medium	C, D, E, H, K, L	2
Intervention Suggestions	High	High	C, D, E, H, K, L	3

Table 1: Core Functionality Priority Table

1. **Violation Submission:** For Violation Submission, it is necessary to build Violation Submission Manager component. External systems, as mentioned before, are considered reliable and will only be integrated. Also, Mobile Application, Controller and Application Server components will be implemented as the components necessary

for the core functionalities are built. Each component responsible will be tested within itself.

2. **Violation Validation:** Violation Validation functionality requires the same components as Violation submission. So, only the necessary extension for this component will be made. The interaction of the necessary components for this function will be tested.
3. **Violation Approval/Disapproval:** For this functionality, it is necessary to build Violation Approval Manager component. After the implementation, the component will be tested within itself and the integration with the external systems.
4. **Intervention Suggestions:** Intervention Suggestions functionality requires the same the Data Visualization Manager to be built. This component will be implemented and tested within itself at this stage.
5. **Heatmap Visualization:** Heatmap Visualization functionality requires the same components as Intervention Suggestions. So, only the necessary extension for this component will be made. The interaction of the necessary components for this function will be tested.
6. **Frequent Offenders List:** Frequent Offenders List functionality requires the same components as Heatmap Visualization. So, only the necessary extension for this component will be made. The interaction of the necessary components for this function will be tested.
7. **Registration:** Login Manager will be implemented and tested. Then it will be integrated with external components.
8. **Login:** Registration Manager will be implemented and tested. Then it will be integrated with external components.

Below is a flow that shows the implementation order:

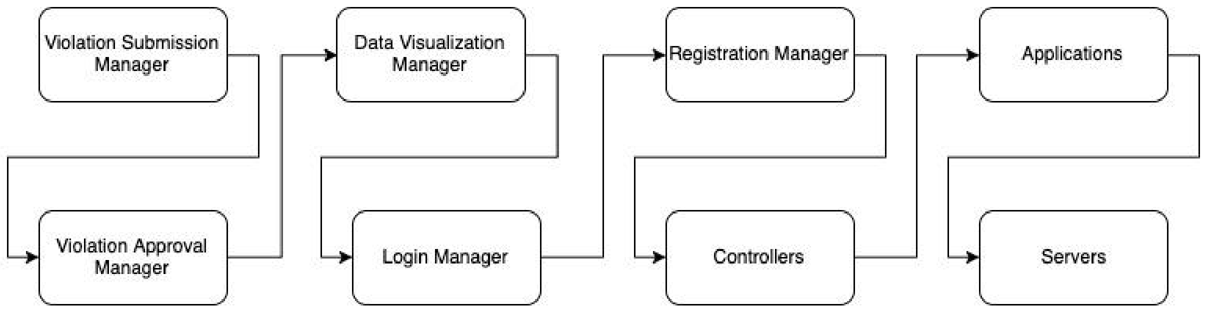
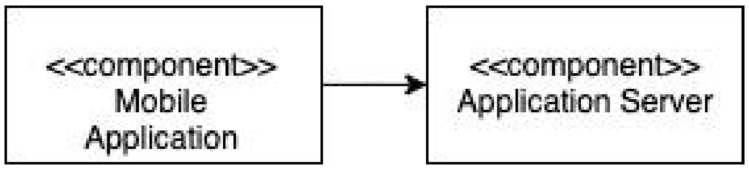

Violation Submission Manager
Violation Approval Manager
Data Visualization Manager
Login Manager
Registration Manager
Controllers
Applications
Servers

Figure 28: Implementation Flow

After the necessary implementation of these components are done. Implemented and external components will be integrated. Subsystems such as Mobile Application, Application Server, Application Controller, Web Application, Web Server, Web Controller will be complete and integrated.

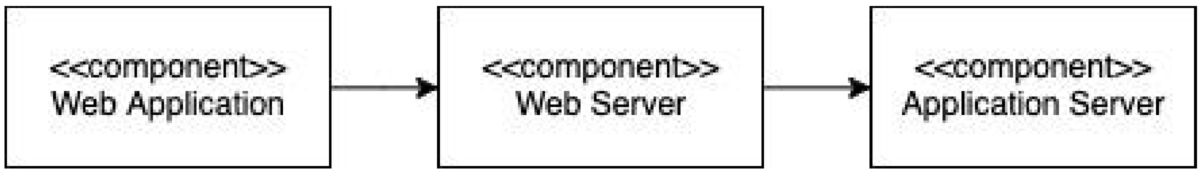
5.2. Integration

5.2.1. Application – Server Integration



<<component>> Mobile Application
<<component>> Application Server

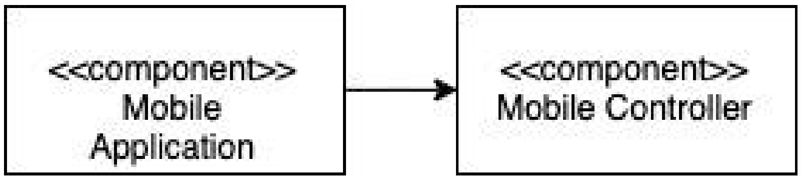
Figure 29: Mobile Application – Application Server Integration



<<component>> Web Application
<<component>> Web Server
<<component>> Application Server

Figure 30: Web Application – Application Server Integration

5.2.2. Application – Controller Integrations



<<component>> Mobile Application
<<component>> Mobile Controller

Figure 31: Mobile Application – Controller Integration

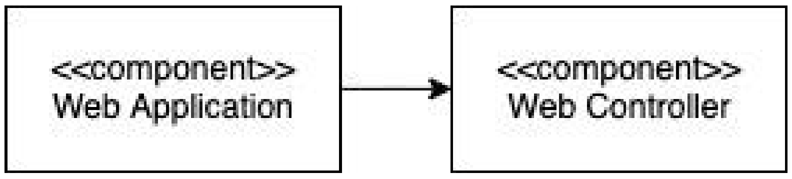
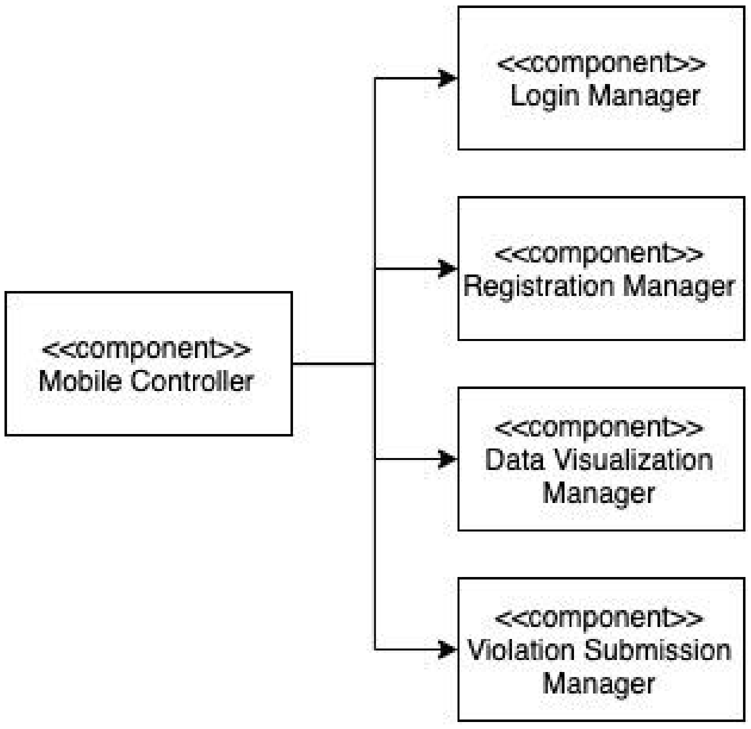


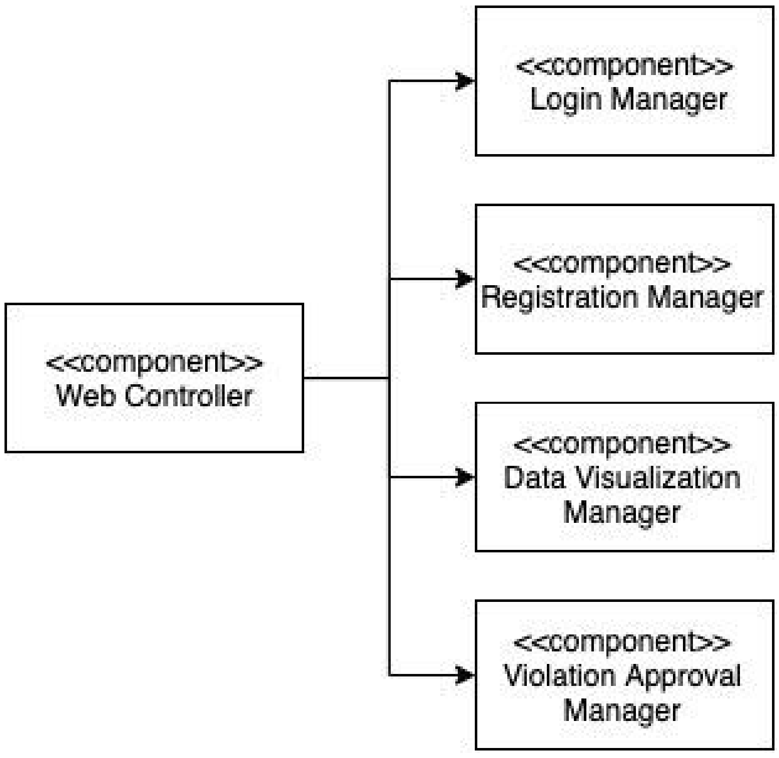
Figure 32: Web Application – Controller Integration

5.2.3. Controller – Component Integration



<<component>> Mobile Controller
<<component>> Login Manager
<<component>> Registration Manager
<<component>> Data Visualization Manager
<<component>> Violation Submission Manager

Figure 33: Mobile Controller – Functionality Components Integration



<<component>> Web Controller
<<component>> Login Manager
<<component>> Registration Manager
<<component>> Data Visualization Manager
<<component>> Violation Approval Manager

Figure 34: Web Controller – Functionality Components Integration

5.2.4. External Component Integration

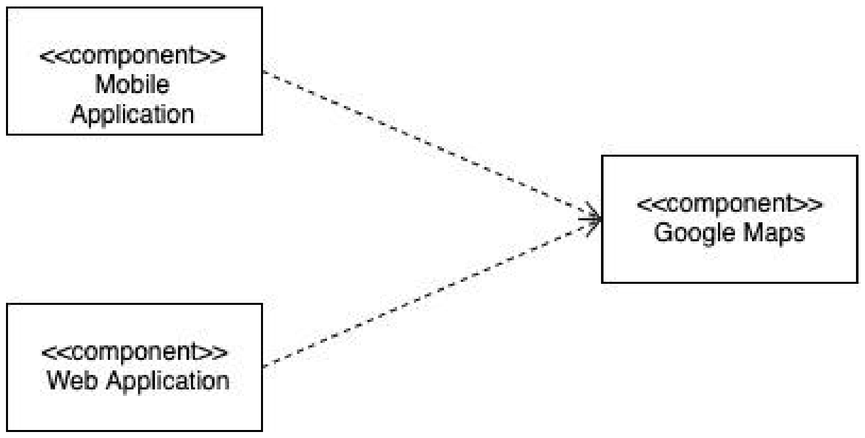


Figure 35: Google Maps Integration

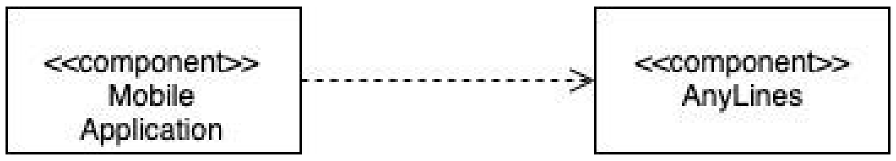


Figure 36: AnyLines Integration

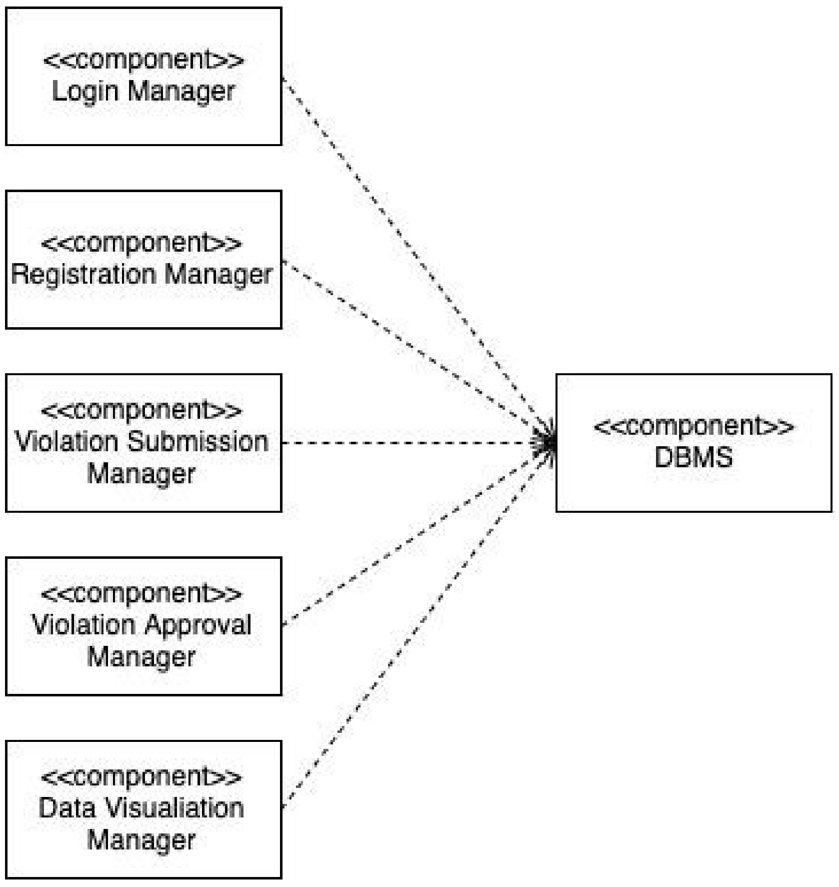


Figure 37: Database Management System Integration

5.3. Testing

The development of the whole system will follow a bottom-up approach. In other words, each component will be implemented within itself and build bigger components until the whole system is implemented. During this procedure, each component will be unit tested within itself. Then it will be tested with connection to other components until the whole system is connected/integrated and complete.

6. Effort Spent

ST1

Subject	Hours Spent
Introduction	0.5
Architectural Design	18
User Interface Design	4
Requirements & Traceability	3
Implementation, Integration & Testing	4

ST2

Subject	Hours Spent
Introduction	0.5
Architectural Design	17
User Interface Design	4
Requirements & Traceability	3
Implementation, Integration & Testing	4

ST3

Subject	Hours Spent
Introduction	0.5
Architectural Design	16
User Interface Design	6
Requirements & Traceability	3.5
Implementation, Integration & Testing	4