Politecnico di Milano

AA 2019/2020

POLITECNICO
MILANO 1863

Safe Streets

DD- Design Document

Version 1.1 - 15/12/2019

Authors:

Priyanka Rajendran

Shara Priyanka Vakati

Professor:

Matteo Rossi

Table of contents

1. Introduction

1.1 Purpose

The purpose of this Design Document is to provide a detailed description of the system design fully such that it gives a better understanding for developing the software. It focuses mainly on what is to be built and how it is expected to be built. In the RASD document we have described the requirements for Safe Streets.

In this document we are going to explain especially about the following things in detail.

- Architecture chosen.
- Components involved in this system and their respective process.
- Interaction among the components.
- The design patterns selected.
- User interface design.
- Mapping the requirements to the components.

1.2 Scope

The following presents a rehash of the scope of Safe Streets that is stated in the RASD document.

Safe Streets is aimed in reducing the traffic violations especially parking violations. The users registered with this application will be able to send the details of the violation by filling out some details to proceed with reporting the violation. The application after receiving the information validates, process and store the data. These information will be sent to the authorities who are registered with Safe Streets. Both the users and the authorities can access the data stored in Safe streets. The authorities can access all the data like the areas where the number of violations is high and the vehicles committing a greater number of violations whereas the users will have only limited access.

Safe Streets will be able to access the data from the municipality services. Using these data the safe streets will identify the unsafe areas. It will also provide some suggestions for reducing the violations which when implemented by the authorities will reduce the number of violations. It will create a vigilance among the citizens about traffic and parking violation.

It will help the authorities to get know about almost all the violations occurring in the city with the help of the public people who acts as the source in providing these information that they come across in their day-to-day life. This will also help the user to know about the incidents in the city by accessing the services provided by the municipality to retrieve the accidents in the selected area. This will make the users stay alert in the areas that are marked unsafe. On the whole Safe Streets acts as an intermediator between the user and authorities by facilitating some useful services.

Some of the machine phenomena that needs to be considered are as follows:

- The system will not highlight an area where the violations are more if no user reports the incidents in that area.
- The authority will not be able to find the vehicle violating the rules if the user did not take a picture covering the license plate or if the image quality is too low or shaky.
- The device that the user using should have a GPS with high accuracy.

1.3 Definitions, Acronyms, and Abbreviations.

1.3.1 Definitions

User: The customer of the application who provides information about the traffic violations, retrieve information from Safe Streets about the accidents occurring in the unsafe areas.

Authorities: Traffic officials who has the power or right to give orders, make decisions, and enforce obedience.

Violation: A violation is any act that fails to abide by the existing law.

Meta-data: Data that provides information about other data.

Algorithm: A process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.

Unsafe areas: The areas where large number of accidents occur and the area where the number of violations reported is high.

1.3.2 Acronyms

API: Application Programming Interface.

GPS: Global Positioning System.

UI: User Interface.

SS: Safe Streets.

ID: Identification number.

GDPR: General Data Protection Regulation.

ELB: Elastic Load Balancer.

SSL: Secure Socket Layer.

1.3.3 Abbreviations

no. –number

1.4 Revision History

Version 1.0: First release.

Version 1.1: Added UI diagrams.

1.5 Reference documents

1. **Specification document:** "Safe Streets Mandatory Project Assignment A.Y. 2019-2020".
2. **Requirement Analysis and Specification Document:**
3. IEEE standard for Information Technology- Systems Design- Software Design Decriptions.
4. **UML diagrams:**
   https://www.uml-diagrams.org/
5. **Traffic rules:**
   http://www.poliziamunicipale-online.it/?l=eng#/Legislation
6. http://www.cloudcomputingpatterns.org/

1.6 Document Structure

Chapter 1: This chapter contains the purpose of the design document and a rehash of what is written in the RASD document with some more information being added.

Chapter 2: This chapter gives the overall description of the architecture of the system and the design patterns chosen. It introduces all the components of the system and their

interactions. It also contains sequence diagrams to provide the run time picture of the components and a diagram to describe the interfaces.

Chapter 3: This chapter provides a better understanding of the application design through user interface mockups. It picturizes almost all the important features of the application.

Chapter 4: This chapter describes the mapping between the requirements and the design components identified and how these components help in achieving all the requirements that are formulated in the RASD.

Chapter 5: This chapter contains the integration plan for the components that is the order in which it is going to be implemented.

Chapter 6: This chapter shows the effort spent by each member of the group spent on working on this design document.

2. Architectural Design

2.1 Overview: High--level components and their interaction

Safe Streets will be a crowdsourced application where all the data is maintained in the Microsoft Azure Cloud. It is supposed to have a layered architecture where components are separated in layers. Also it inherits some of the concepts from cloud and distributed computing. Presentation layer (P) that contains all the user interface elements, the Application layer (A) that contains all the business or domain logic of the application and finally the Data access (D) that contains all the components to access the database. Basically, it is a three-tier architecture where it can be extended into n- number of layers based on the requirement. Layered architecture is very flexible to add or remove components and is the most successful model used to implement many applications.

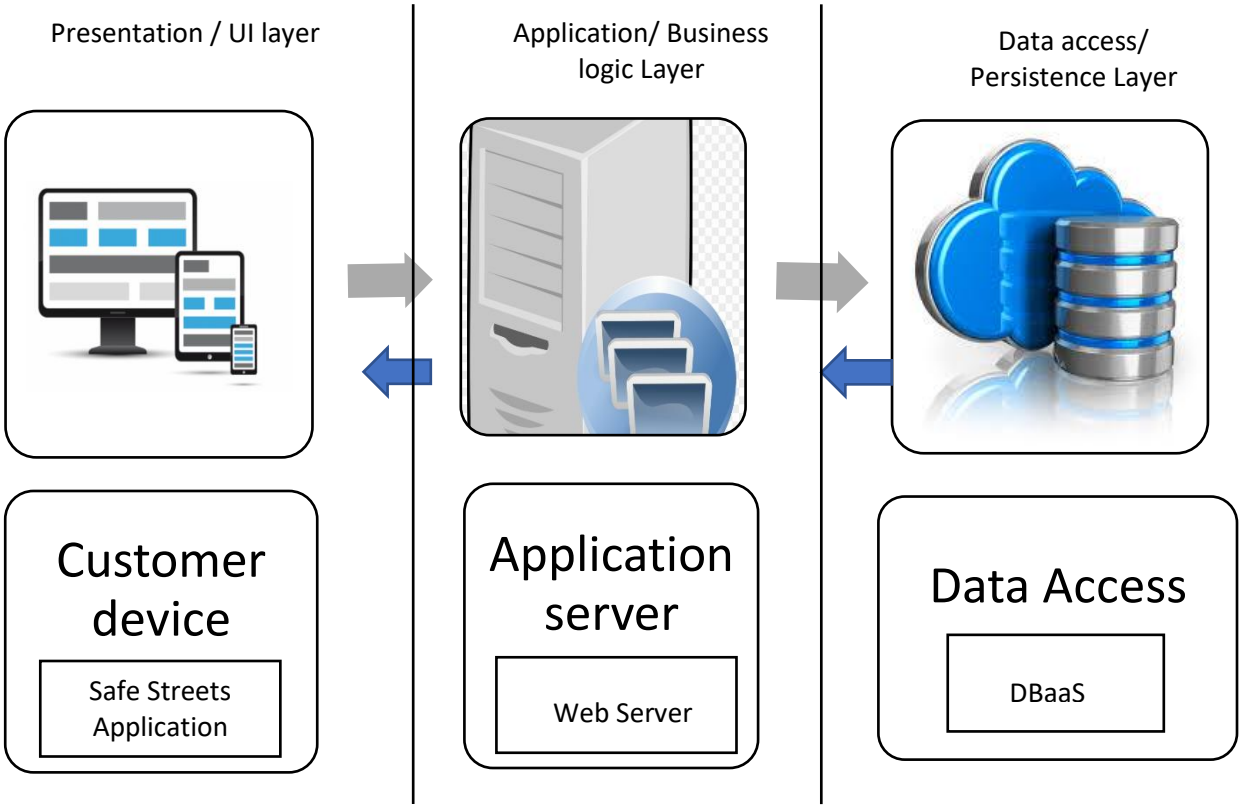

Figure 1: High Level System Architecture of Safe Streets.

By adopting a layered architecture there is a lot of advantages and some of them are listed below.

1. This architecture will provide elastic scalability and flexibility.
2. It will give the ability to update the technology stack of one tier, without impacting other areas of the application.
3. It will allow different development teams to work independently on their own areas of expertise.
4. It adds reliability and more independence of the underlying servers or services.
5. It provides an ease of maintenance of the code base, managing presentation code and business logic separately, so that a change to business logic, for example, does not impact the presentation layer.

A Layered architecture provides the ability to utilize new technologies as they become available. It also ensures the product is ready to adapt; ready for the future and will provide the opportunity to redesign the product or application and actually look not only to today's needs but into the future. It stays ahead of the game and maintains a competitive advantage.

The main intent of choosing a cloud based data storage is to reduce the cost, offer a strategic edge, high speed, backup and restoration of data, automatic software integration, reliability, mobility, unlimited storage capacity, collaboration, quick deployment and offers resilient computing.

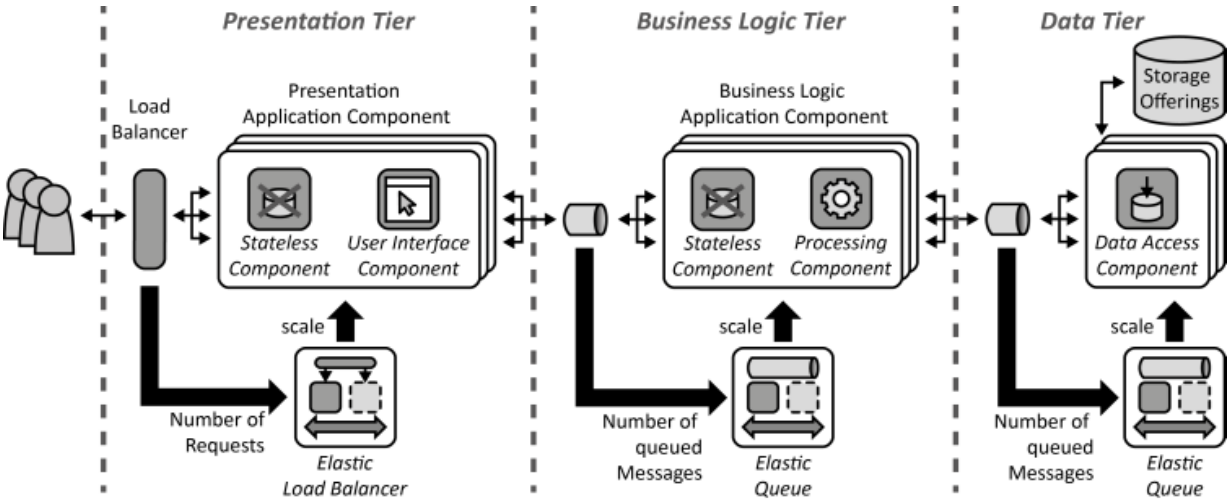

Figure 2: Layered architecture with high level components in each tier.

The Presentation tier contains the load balancer that distributes the workload across multiple computing resources and a user interface component. This tier contains all the UI components necessary for the user to interact with the system. The user reports the violation and the authorities can view the same using these UI components. It also contains a caching layer that is distributed. A distributed data cache, also called a distributed data grid, is a storage layer that sits between a database server and the in-memory of an application. It is believed that it will speed up an application's performance. It contains all the data so that it can be accessed very quickly, much more quickly than if it were kept just in the database server.

The internal state of the components is not maintained in the component itself instead it is stored in an external storage to ensure fault tolerance and recovery. Hence each component is represented to contain a Stateless component. Load balancers will be explained in the deployment diagram section along with router.

MapReduce, a method of analysis that divides a computation among several servers and then combines the results, can be more easily deployed using distributed data grids. The primary use of distributed data caches is to store fast changing data that is accessed by multiple servers and the distributed data caches will grow over time. It will continue to provide a platform for performing parallel data analysis. Map reduce technique is used

everywhere in the architecture where there is a need to scale up the components for distributed computing.

The Business logic layer contains all the processing components. This is the layer where all the business logic is implemented like finding the license number from the picture provided, analyze the data for identifying the unsafe areas, highlighting the areas with large number of violations and the vehicles committing more violations, finding the most common type of violations in a given areas, identifying no. of accidents in each area. It generates dynamic content based on the user request.

This component splits the processes into separate function blocks and assigns it to independent processing components. Since each processing component will be scaled out independently and will be implemented in a stateless fashion as described in the Stateless Component pattern the Scaling will be handled by an Elastic Queue. The data required for processing will be provided with requests or by Storage Offerings.

The Data access layer will contain the components to access the data from the Storage offering. This component will be responsible for maintaining all the data manipulation. This layer maintains all the data regarding the violation. In case of replacing a storage offering interface or a storage offering, only the data access component need to be modified. This layer will also have the Elastic Queue which scales to assign process to independent components.

2.2 Component View:

Component diagram describes the organisation and the interaction between the physical components of the system. In this diagram the business logic is described in detail since it is the core element of the system. Even though some of the components in the other layers are also shown in the component diagram in order to show their interaction with the application layer. The components and their interactions are described below.

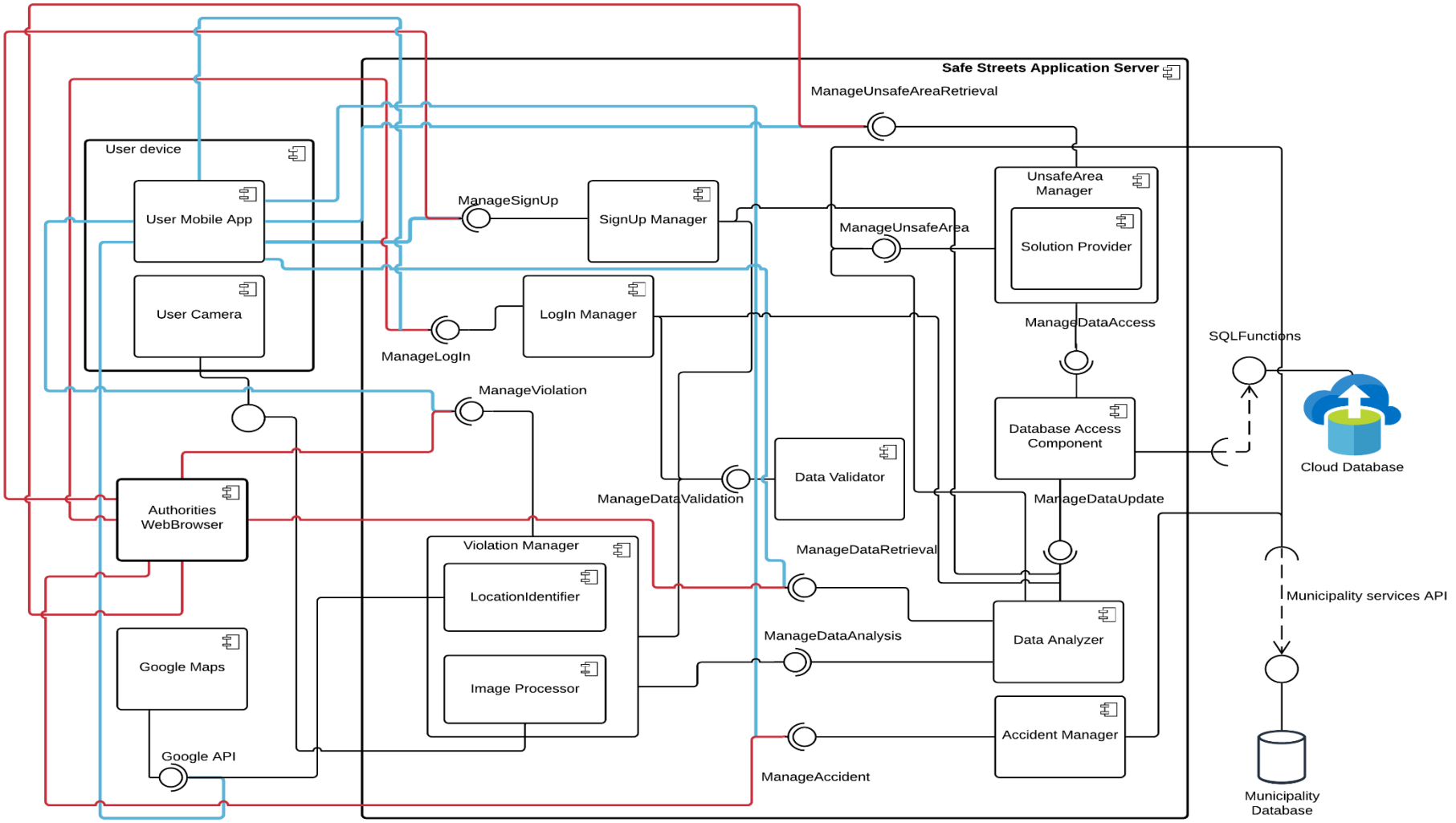

Figure 3: Component Diagram

The client side contains mobile app and authorities web browser where the user's mobile device also contains a camera.

- **User Mobile App:** This component contains all the user interface elements for the user to access the application server.
- **Authorities Web Browser:** This component contains all the user interface elements for the authorities to access the application server.

The server side contains all the domain logic and the components that are required to accomplish all the functionalities of the system. Cryptographic techniques like MD5 hash can be used to check the integrity while sending and receiving data across the Internet.

- **SignUp Manager:** This component contains all the procedures to allow the customers to register to Safe Streets. It is connected to the database component to store and retrieve data from the cloud database. It is also connected to the Data Validator component which verifies and validates all the data entered in the SignUp page before storing the data in the cloud.
- **LogIn Manager:** This component contains all the routines to allow the customers to login to the SafeStreets. It is connected to the database component and Data Validator component to check the username and password entered to ensure authentication.
- **Data Validator:** This component is used to validate login, signup and violation data. It checks whether all the constraints and requirements specified for each field is met. In violation data it mainly checks the mandatory field type of violation is specified or not and also checks for the image of the violation.
- **Violation Manager:** It handles all the data regarding the violation like type of violation, its description, image, date and time and the location where the violation is identified. It automatically retrieves the date and time from the user's mobile device. It is also connected to a data validator to check the data. It is responsible for all the actions that are performed on the violation data. This component contains another two sub components called Location Identifier and Image processor.
    - **Location Identifier:** It uses the Google API to find the location of the user. Using which it finds and returns the area where the violation has taken place.
    - **Image Processor:** It runs Image processing algorithm over the image provided by the user to find the license number of the vehicle that is violating the rules.

- **Accident Manager:** This component retrieves all the information regarding the accidents in a particular area specified by the customer from the municipality database through municipality services API.
- **Data Analyzer:** This component performs operations over the data by accessing the violation manager component and stores the results in the cloud database through the database access component. It has routines that identifies the area where the number of violations are higher comparatively and the vehicle committing the highest number of violations. It also identifies the most common type of violation in each area. It is used by the User Mobile App and the Authorities Web Browser components to retrieve analysed data where the authorities are allowed to access all the analysed data and the user is allowed to access only the data regarding the highlighted areas. It is another main component containing the business logic.
- **Unsafe Area Manager:** This component identifies the potentially unsafe areas in the city by comparing the accident data from the municipality database and the violation data from the Safe Streets database that is stored in the cloud. It also provides some solutions using its inbuilt Solution Manager to the authorities to curtail the violations and the accidents happening in the city. It provides the list of the unsafe areas to the user and with solutions to the authorities.
- **Database Access Component:** This component acts a middleware between the application layer and the data layer. All the actions either read or write passes through the database component to the cloud database.

2.3 Deployment View:

A deployment diagram, models the run-time architecture of a system. It shows the components that is deployed in each layer. It shows the configuration of the hardware elements and shows how software elements and artifacts are mapped onto those nodes.

Generally, a node has two stereotypes as follows:

<< device >>: It is a node that represents a physical machine capable of performing computations. A device can be a router or a server PC. It is represented using a node with stereotype <<device>>.

<< execution environment >>: It is a node that represents an environment in which software is going to execute.

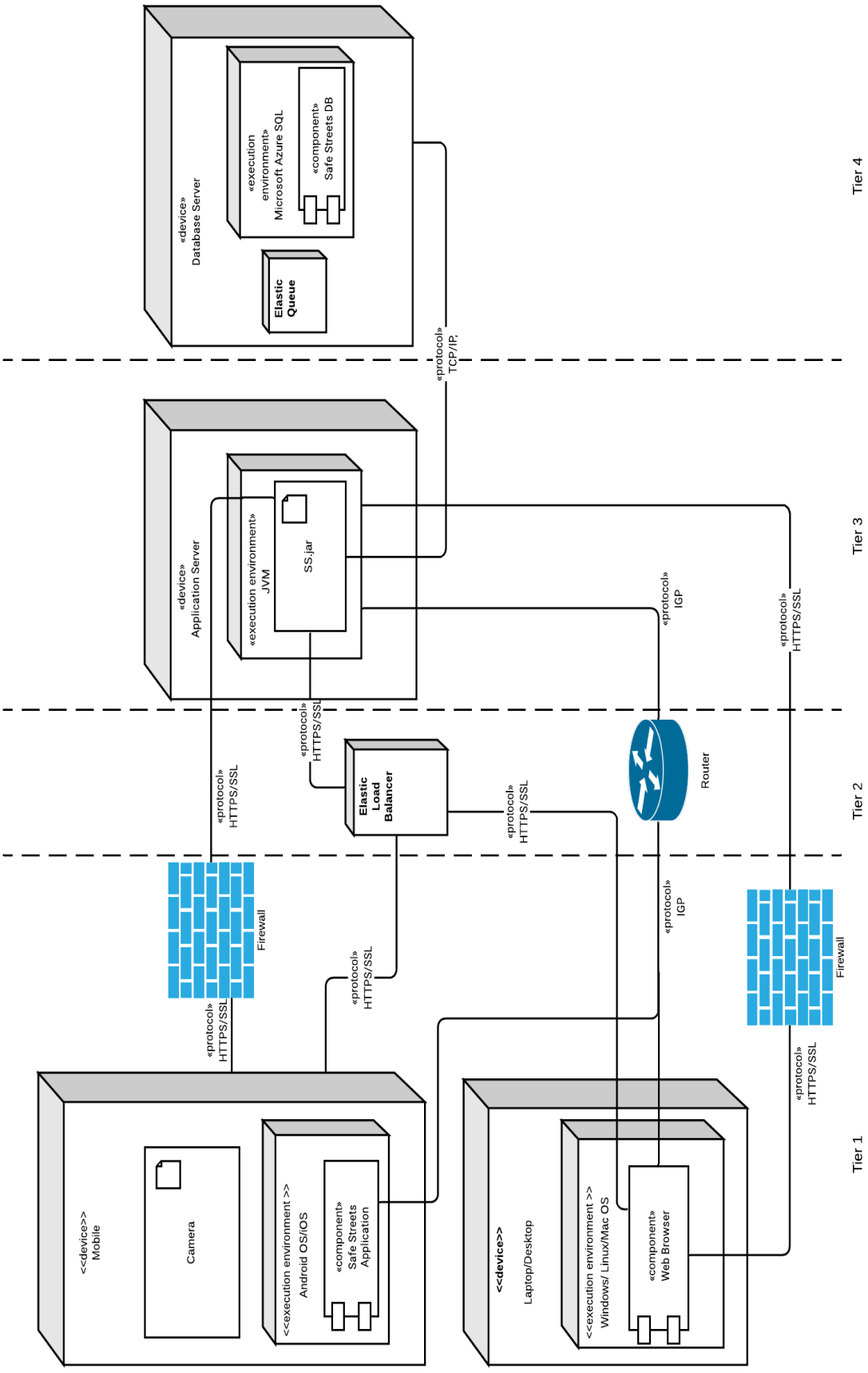

Figure 4: Deployment diagram

Tier 1: This tier has presentation logic. It has the user devices through which the customers can access the SS. The user is provided with a mobile phone with a camera. The user can have a mobile phone with either an android OS or with an iOS. This device should be installed with SS mobile app in order to interact with the system. On the other hand authorities are provided with a web application that runs on any web browser. They can simply access the system by hitting the URL. The desktop that the authorities are using can have either a Windows OS or Linux OS or Mac OS. This layer also has a firewall which restricts the unauthorized access and also monitoring the network traffic.

Tier 2: This tier has the components required for transporting data across the network. It is provided with an Elastic Load Balancer and a router. The ELB automatically distributes incoming application traffic and scales resources to meet traffic demands. It also detects the unhealthy Elastic Compute Cloud instances and spreads the instances only through healthy channels. It offers a flexible cipher support and manages the SSL certificates. It supports both IPv4 and IPv6. Router is another main component in an architecture of the system since it is the component that connects devices and favors a better transmission of data. A router is a physical or virtual appliance that passes information between two or more packet-switched computer networks. A router inspects a given data packet's destination Internet Protocol address (IP address), calculates the best way for it to reach its destination and then forwards it accordingly. A router is a common type of gateway. It is positioned where two or more networks meet at each point of presence on the internet.

Tier 3: This tier contains the application server which is the core element of the system. This tier contains all the components required to run the system. It handles all the request from the client and provides the desired responses in return. All the components are implemented using Java. So, it contains the jar file being deployed in it. It executes all the commands using Java Virtual Machine which acts as the execution environment for the Java classes.

Tier 4: This tier contains the database which acts as the store for all the data. It has a Microsoft Azure Database which is a cloud storage offering provided by Microsoft. It manages all the storage requirements. It has an Elastic Queue that is used to distribute asynchronous requests among multiple application components instances. Based on the number of enqueued messages the Elastic Queue adjusts the number of application component instances for handling these requests.

2.4 Runtime View:

2.4.1 Reporting a Violation:

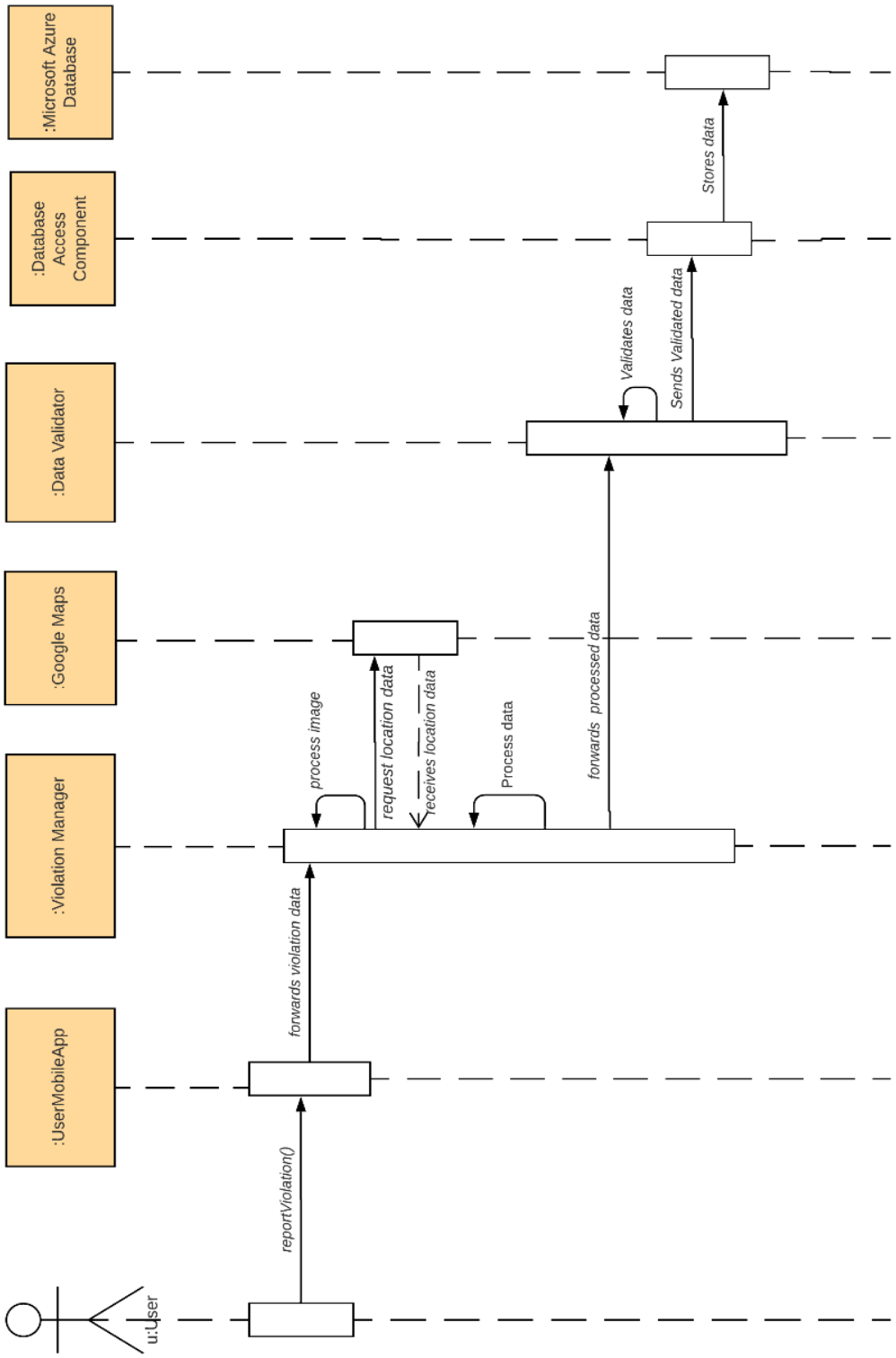

Figure 5: Reporting a violation

..

In this sequence diagram, the flow of the messages from the user to the database for reporting a violation is presented. The user with the mobile app installed in his/her mobile phone log in and take a picture of the violation and provides some more details regarding the violation and then submits it. This data is transferred from the UserMobileApp to Violation Manager which process the image for getting the license number of the violated vehicle. It also finds the area in which the violation has taken place using the Google Maps. It also receives the date and time of the user's mobile and then send all the data that is received and processed to the data validator. This again checks the data and forwards to the database access component. This then stores the violation data in the Microsoft Azure database.

2.4.2 View Violation:

In this sequence diagram, the process through which the authorities are able to view the violation data is presented. The authorities after logging into the SS application then clicks on the View Violation button which triggers the following actions. The web browser interacts with the Violation Manager by forwarding the request. This in turn request the data from the database component which retrieves the requested data from the cloud database and then sends it to the Violation Manager. This is forwarded to the web browser that displays the authorities with the list of violations and its details.

The authorities will be provided with the image of the violation, type of the violation, description if it is provided by the user, date and time of the violation and the area where the violation took place.

The authorities after viewing the violation details takes the required action and then closes the violation by providing the comments. The violation has to be closed after taking action since other authorities can know that action has been taken on the particular violation. This update is progressively passed to the database through the same set of components that are previously used by the application to retrieve violation data. This data update is done each time after closing or entering comments by the authorities.

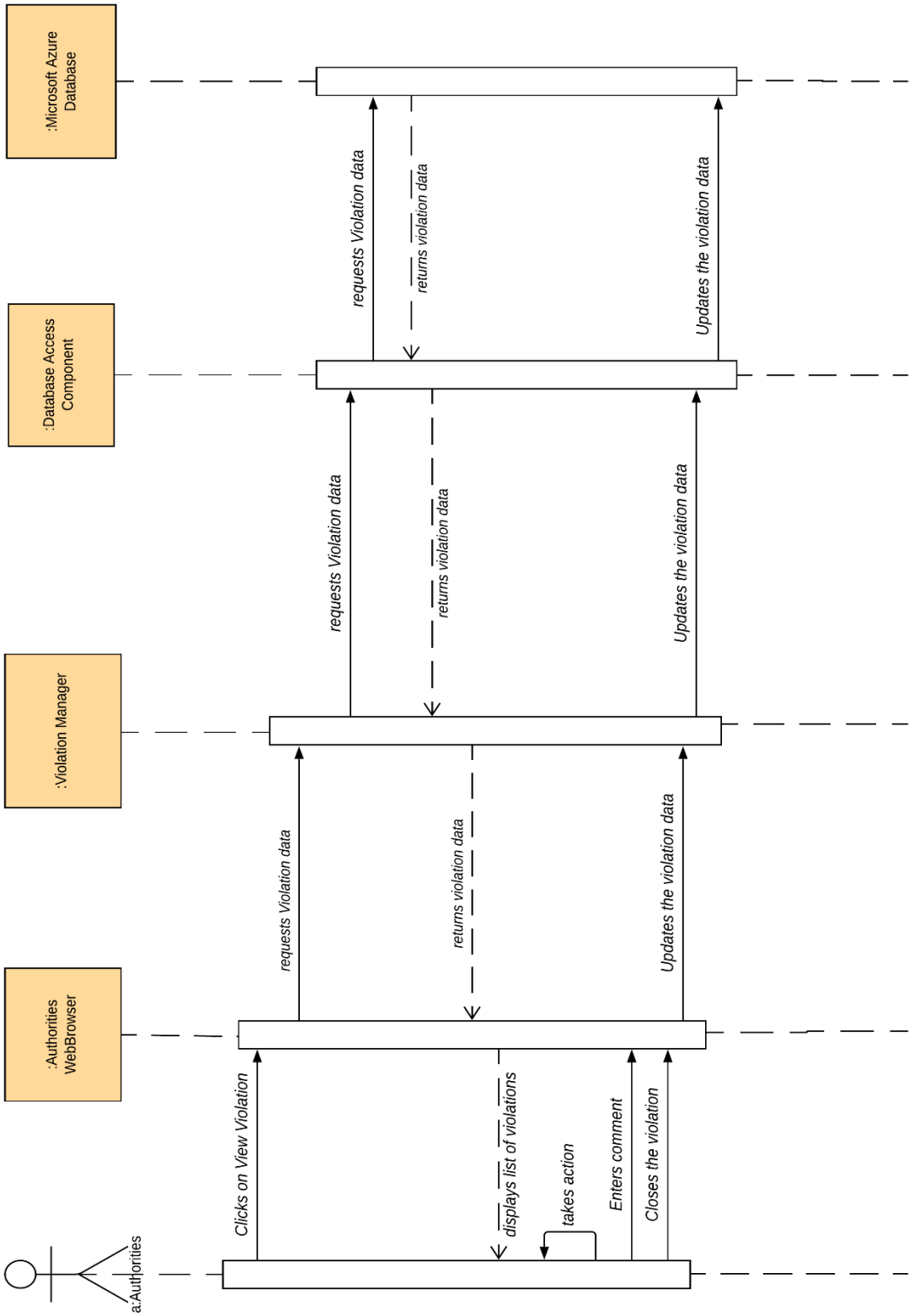

Figure 5: View Violation

2.4.3 Retrieving Analyzed Data:

In this sequence diagram, the process in the retrieval of the information from SS.

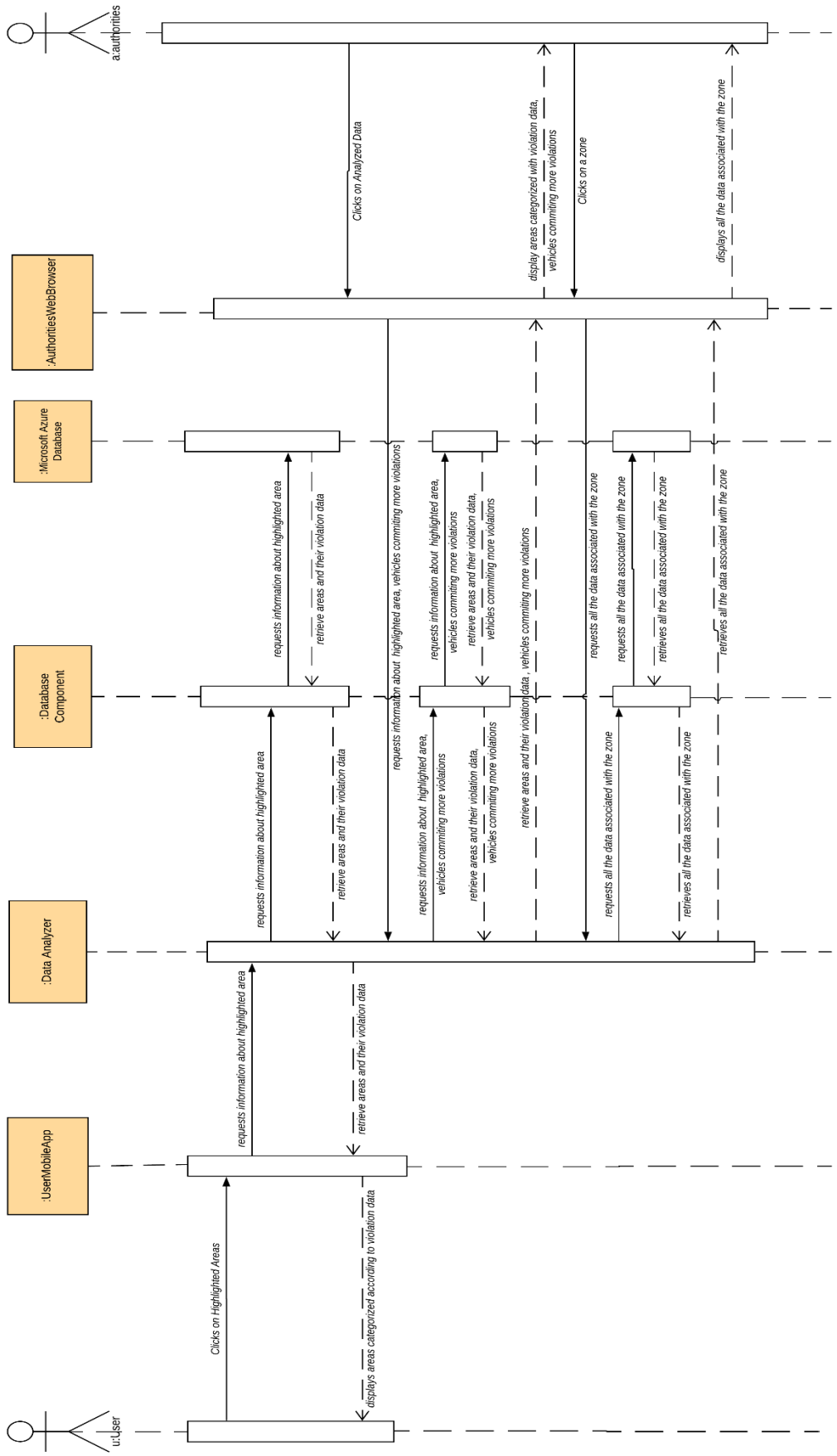

Figure 6: Retrieve Analyzed Data

The user clicks on Highlighted Data. This request is passed through the User Mobile App to the Data Analyzer which fetches the analyzed data from the database through database access component.

The authorities click on Analyzed Data and this request is passed through the web browser then to the data analyzer and then this component fetches the data from database thorough database access component. The web browser displays a pie chart with all the zones categorized as highly violated areas and less violated areas. It also displays the license number associated with the vehicle committing more violations.

The authorities clicks on any of the zone and the web browser fetches the information regarding that zone from again through the same set of components and provides the authorities with all the data associated with that zone including the most common type of violation in that zone.

2.4.4 View Accident Information:

The user through the Mobile app clicks on Accident Information and enters the area then clicks on search. The request is transferred to the Accident Manager which fetches the information from the Municipality database through Municipality services. The requested information is passed again through the same set of components to the User mobile app which displays the retrieved information to the user.

The authorities through the web browser clicks on Accident Information and enters the area then clicks on search. The request is transferred to the Accident Manager which fetches the information from the Municipality database through Municipality services. The requested information is passed again through the same set of components to the web browser which displays the retrieved information to the authorities.

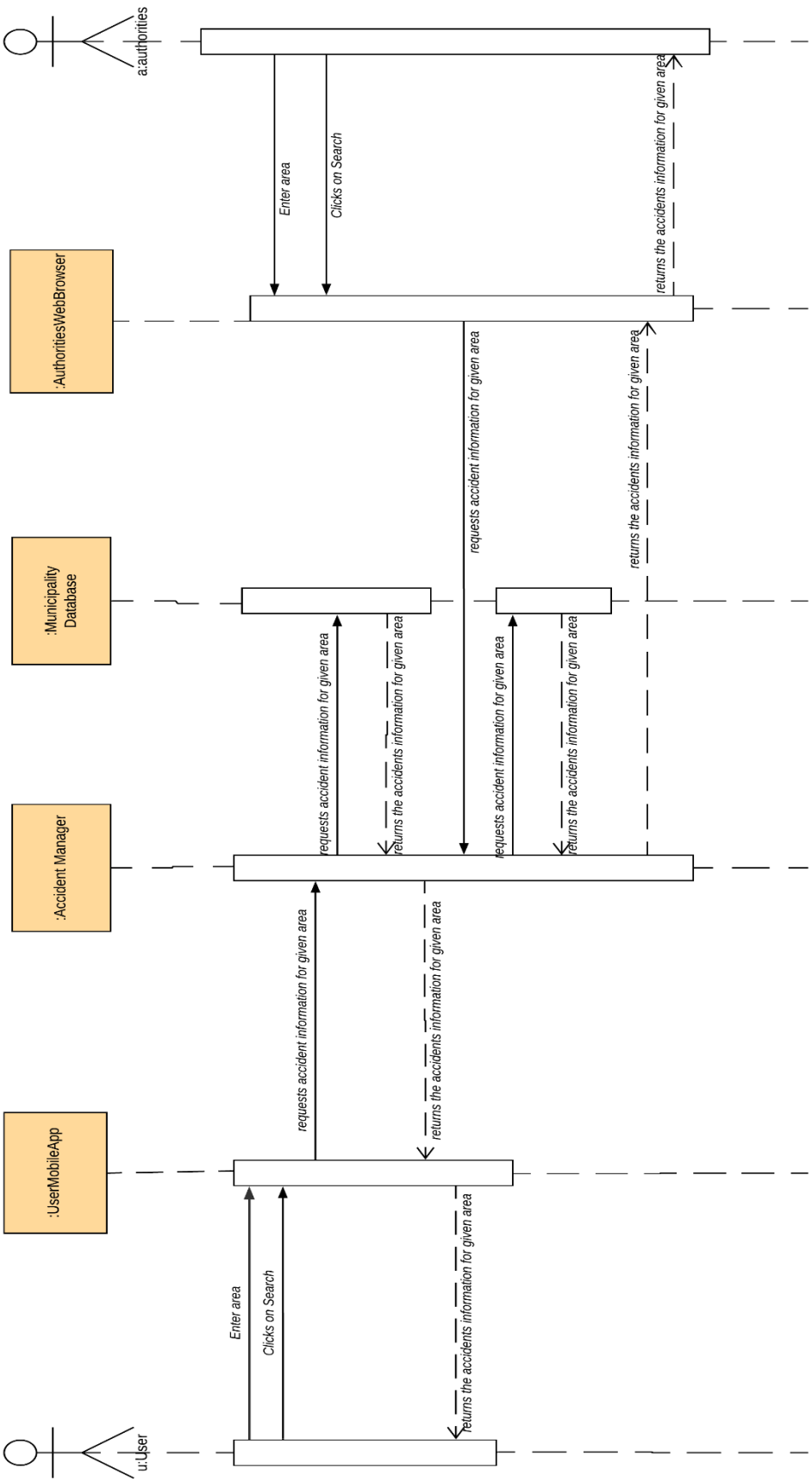

Figure 7: Accident Information

2.4.5 View Unsafe Areas:

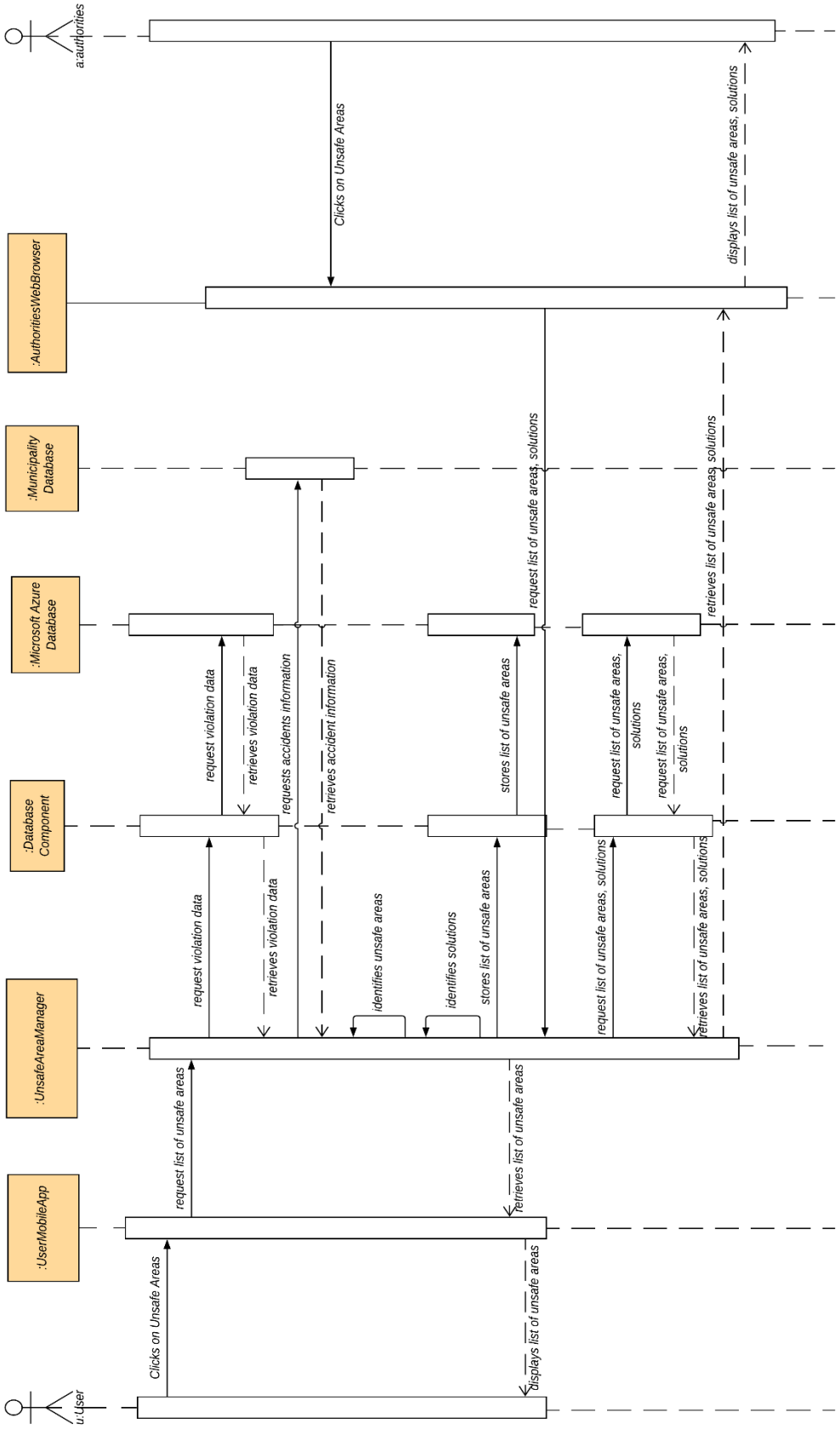

Figure 8: Unsafe Area

In this sequence diagram, the process involved in finding and retrieving the unsafe areas is illustrated. The user clicks on the unsafe area in the Mobile app, this request is redirected to the unsafe area manager which request the accident information from the municipality database and the violation data from the database through database access component. It then uses the information retrieved to identify potentially unsafe areas. It also uses the data to identify the solutions in order to reduce the number of violation and accidents in the specified unsafe areas thus by converting it into a Safe Street. The list of unsafe areas are then passed to the mobile app by the Unsafe Area Manager and displayed to the user. Solutions will not be displayed to the users.

The authorities clicks on the unsafe area in the web browser, this request is redirected to the unsafe area manager which request the accident information from the municipality database and the violation data from the database through database access component. It then uses the information retrieved to identify potentially unsafe areas. It also uses the data to identify the solutions in order to reduce the number of violation and accidents in the specified unsafe areas thus by converting it into a Safe Street. The list of unsafe areas and the identified solutions are then passed to the web browser by the Unsafe Area Manager. These details are displayed to the authorities by the web browser.

2.5 Component Interfaces:

The component interface diagram displays the interaction between the interfaces in the domain layer. The dependency relationships are shown using the arrows.

- The interfaces ManageSignUp, ManageLogIn allows the customer to register with the application. These interfaces can be implemented for both the user and the authorities. It interacts with database to store and retrieve data.
- ManageViolation is the main interface that is very much essential for the business logic of the application. When the user submits a violation data the data needs to be processed which is taken care of the functions ProcessImage(Image:BufferedImage):String to retrieve the license number of the vehicle and the FindStreet(Position: List): String finds the area where the violation has taken place. It also has a method UpdateViolation(status:Boolean):boolean which sets the violation open or

closed. This method is called when the authority takes action and closes the violation. ManageDataUpdate interface uses this function to perform changes to the database.

- When the user or the authorities requests for accident information the method RetrieveAccidents(area: String): Int in ManageAccident interface retrieves the data from Municipality database through the APIs provided by municipality services.

- ManageDataAnalysis interface has all the methods that are required to provide information about the area with highest violation, vehicle with more violations and the most common type of violations. This data will be retrieved by methods in ManageDataRetrieval.

- When the user wants to view the list of unsafe areas then the UnsafeAreaManager uses the methods RetrievAccidents(area: String): Int and RetrieveViolationData():String from the interfaces ManageAccident and ManageDataAccess to find the unsafe areas and it is accessed by the method RetrieveUnsafeArea():String in UnsafeAreaRetrieval interface and then displays to the user. If the authorities request data about unsafe areas the solution returned by the method IdentifySolutions(TypeOfViolation: String, NoOfViolations;Int, NoOfAccidents:Int): String will also be displayed.

- ManageDataValidation has methods ValidateLogInData(username: String, password: String): boolean, ValidateSignUpData(u:User): boolean, UserVerification(fa:fiscalcode:String): boolean, VerifyAuthority(aid:authorityId:String): Boolean, ValidateViolationData(): Boolean to validate the data handled by almost all the interfaces to ensure there is no invalid data.

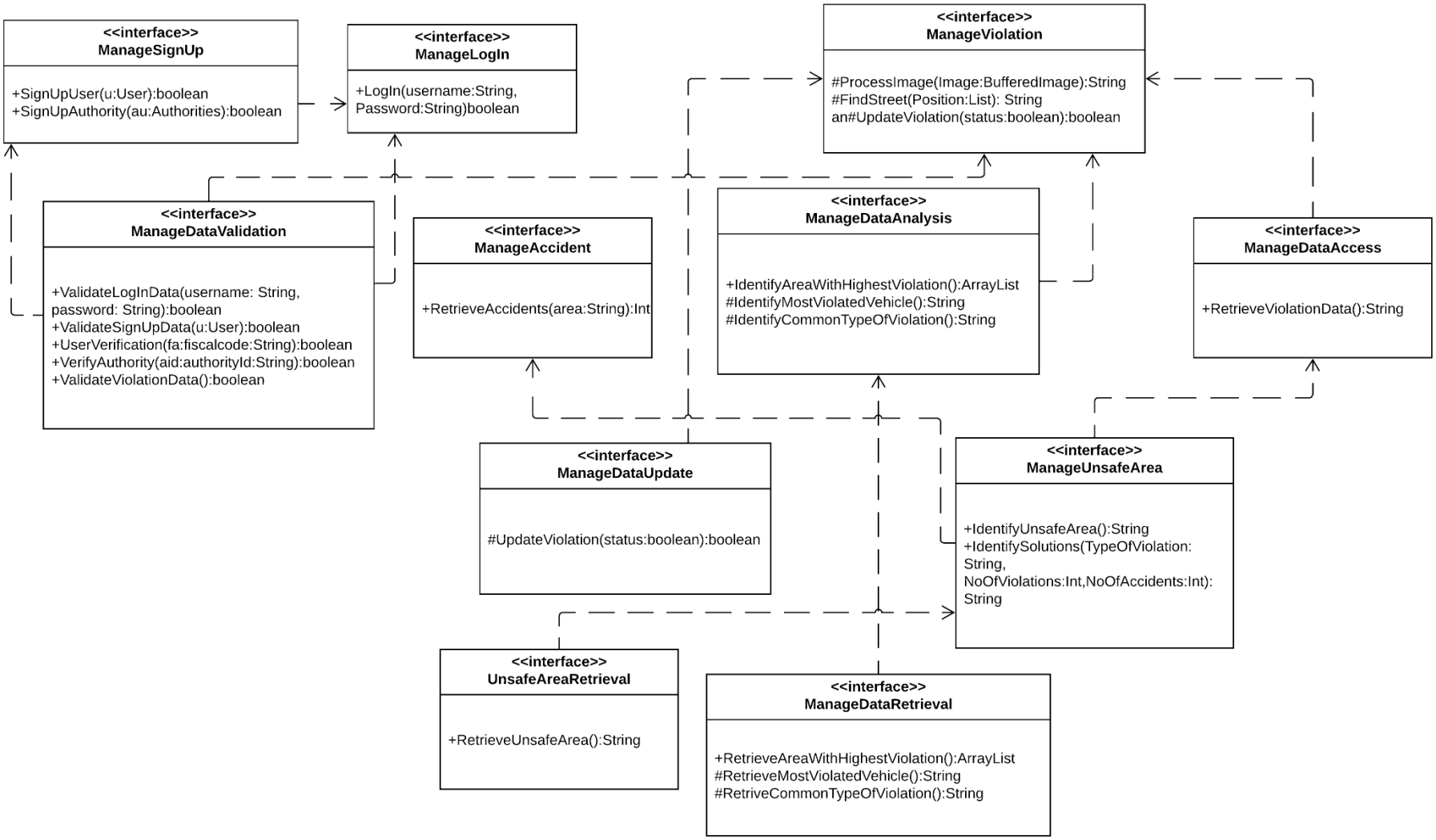

Figure 9: Component Interfaces

2.6 Selected Architectural Styles and Patterns:

Selected Architecture:

Layered Architecture:

Layered architecture patterns are n-tiered patterns where the components are organized in horizontal layers. This is the traditional method for designing most software and is meant to be self-independent. The reasons for selecting this architecture is already detailed in section 2.1.

Relational Database:

A relational database is a set of formally described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables. The standard user and application programming interface (API) of a relational database is the Structured Query Language (SQL). SQL statements are used both for interactive queries for information from a relational database and for gathering data for reports.

Azure SQL Database is a fully managed relational database with built-in intelligence supporting self-driving features such as performance tuning and threat alerts. Explore all SQL Database pricing options and find the performance that fits your workload.

Selected design pattern:

Creational Design Pattern:

Creational patterns provide various object creation mechanisms, which increase flexibility and reuse of existing code.

Factory Method is a creational design pattern that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.

Abstract Factory is a creational design pattern that lets you produce families of related objects without specifying their concrete classes.

Builder is a creational design pattern that lets you construct complex objects step by step. The pattern allows you to produce different types and representations of an object using the same construction code.

Prototype is a creational design pattern that lets you copy existing objects without making your code dependent on their classes.

Singleton is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.

2.7 Other Design Decisions for implementation:

RESTFul WebServices:

REST is used to build Web services that are lightweight, maintainable, and scalable in nature. A service which is built on the REST architecture is called a RESTful service. The underlying protocol for REST is HTTP, which is the basic web protocol. REST stands for REpresentational State Transfer. REST is easier to use for the most part and is more flexible. It has the following advantages when compared to SOAP:

No expensive tools require to interact with the Web service

Smaller learning curve

Efficient (SOAP uses XML for all messages, REST can use smaller message formats)

Fast (no extensive processing required)

Closer to other Web technologies in design philosophy.

MVC design Pattern:

Model-View-Controller (MVC) framework is an architectural pattern that separates an application into three main logical components Model, View, and Controller.

Model − the lowest level of the pattern which is responsible for maintaining data.

View − this is responsible for displaying all or a portion of the data to the user.

Controller − Software Code that controls the interactions between the Model and View.

MVC is popular as it isolates the application logic from the user interface layer and supports separation of concerns. Here the Controller receives all requests for the application and then works with the Model to prepare any data needed by the View. The View then uses the data prepared by the Controller to generate a final presentable response.

Bridge is a structural design pattern that lets you split a large class or a set of closely related classes into two separate hierarchies—abstraction and implementation—which can be developed independently of each other.

Facade is a structural design pattern that provides a simplified interface to a library, a framework, or any other complex set of classes.

3. User Interface Design

The user interface for the authorities is the web browser and mobile application for the users. This section provides a detailed mockup of the Safe Streets application as specified in the section 3.1.1 in RASD document. Some pages are represented in mobile view and some in webpage. However the functionality will be the same for both user and authorities except view violation and report violation, viewing highlighted data.

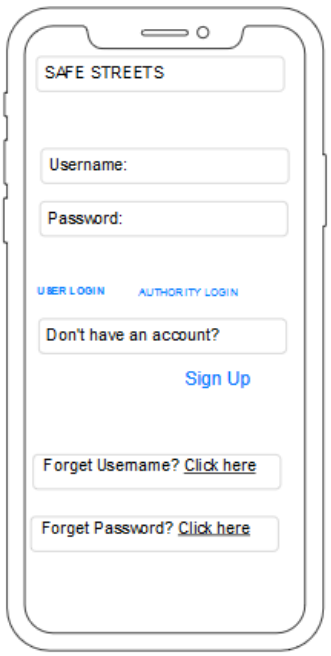

Figure 10: LogIn Page

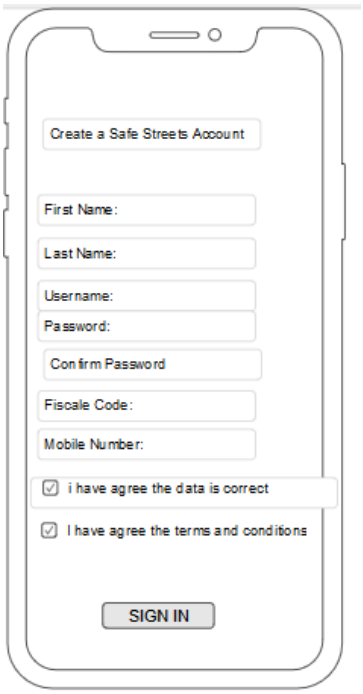

Figure 11: SignUp Page.

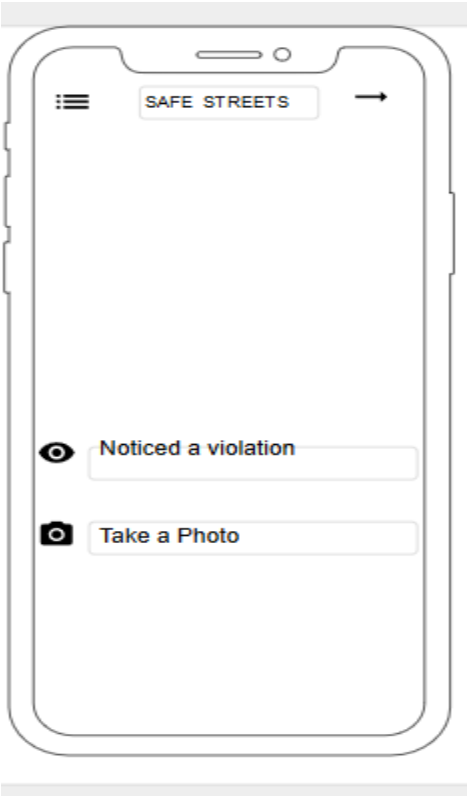

Figure 12: Upload Violation Image

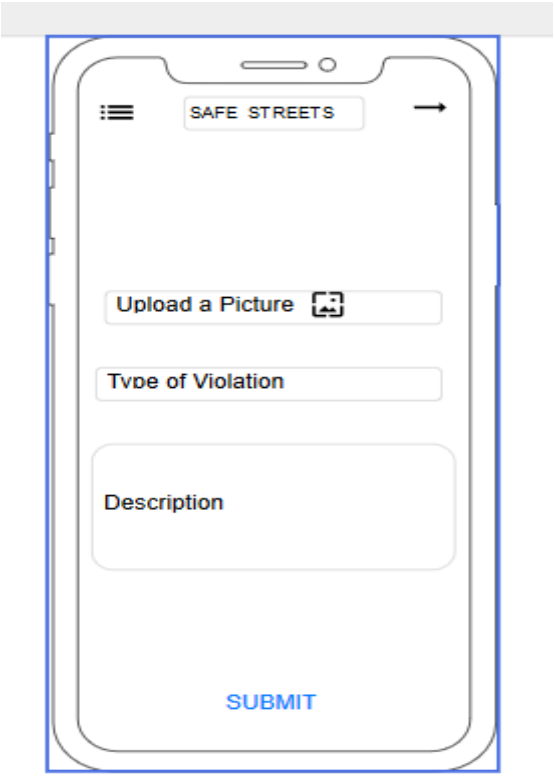

Figure 13: Violation Data Entry

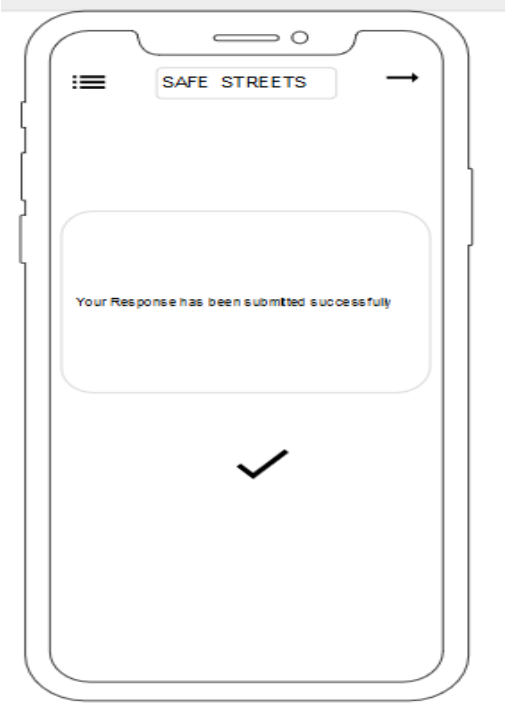

Figure 14: Success Message

Figure 15: Menu Options

Figure 16: Accident Information

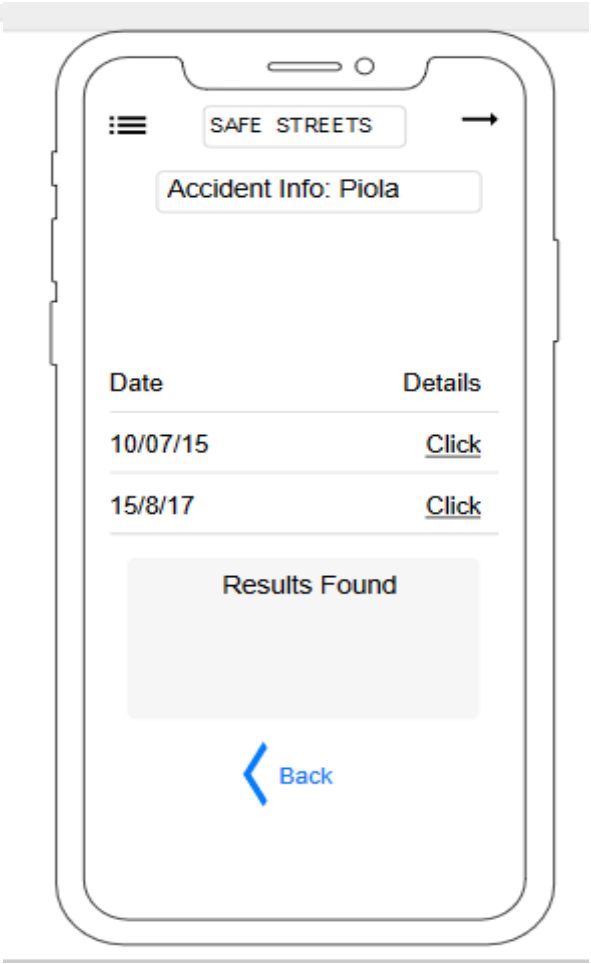

Figure 17: Accident Information

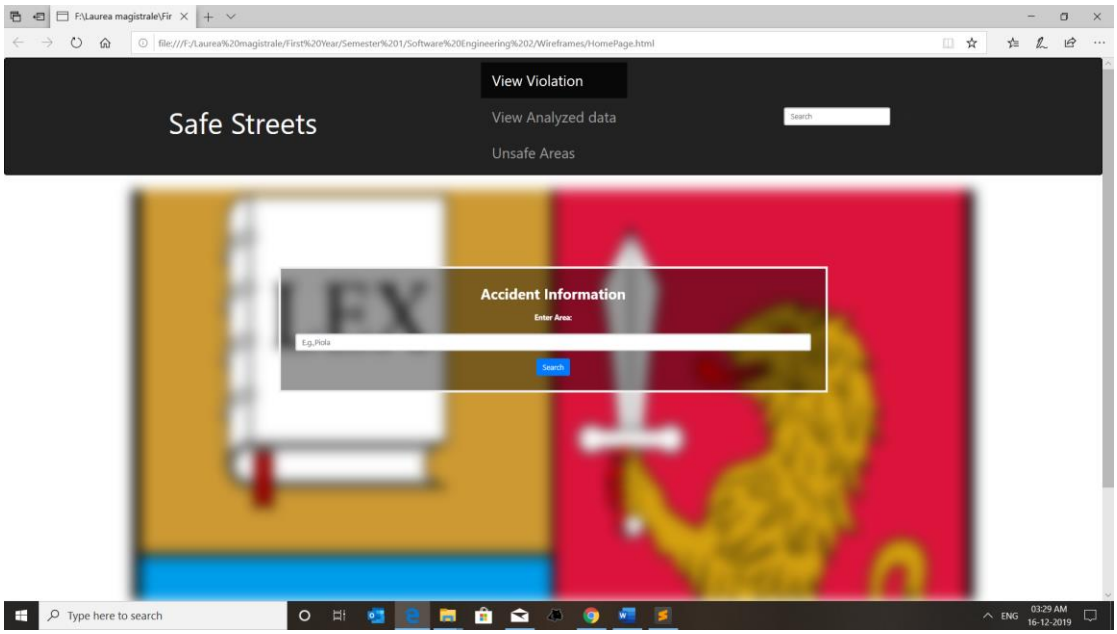

Figure 18: Accident Information_Webpage for Authorities

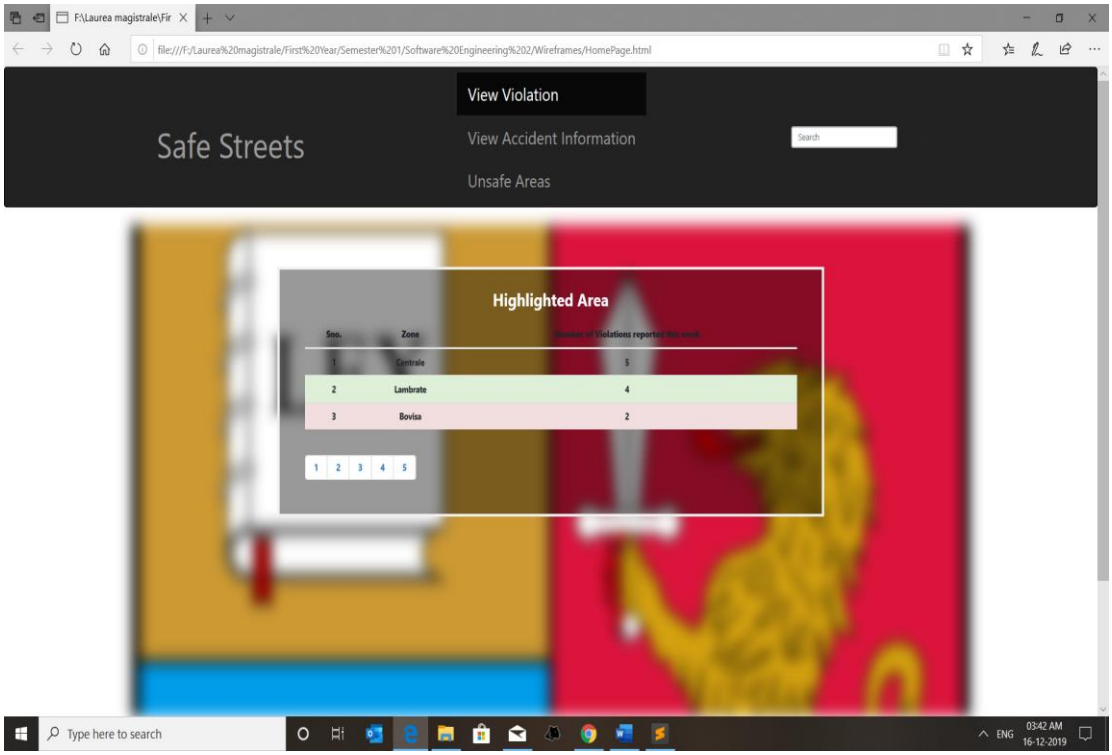

Figure 19: Accident Information_Webpage for Authorities

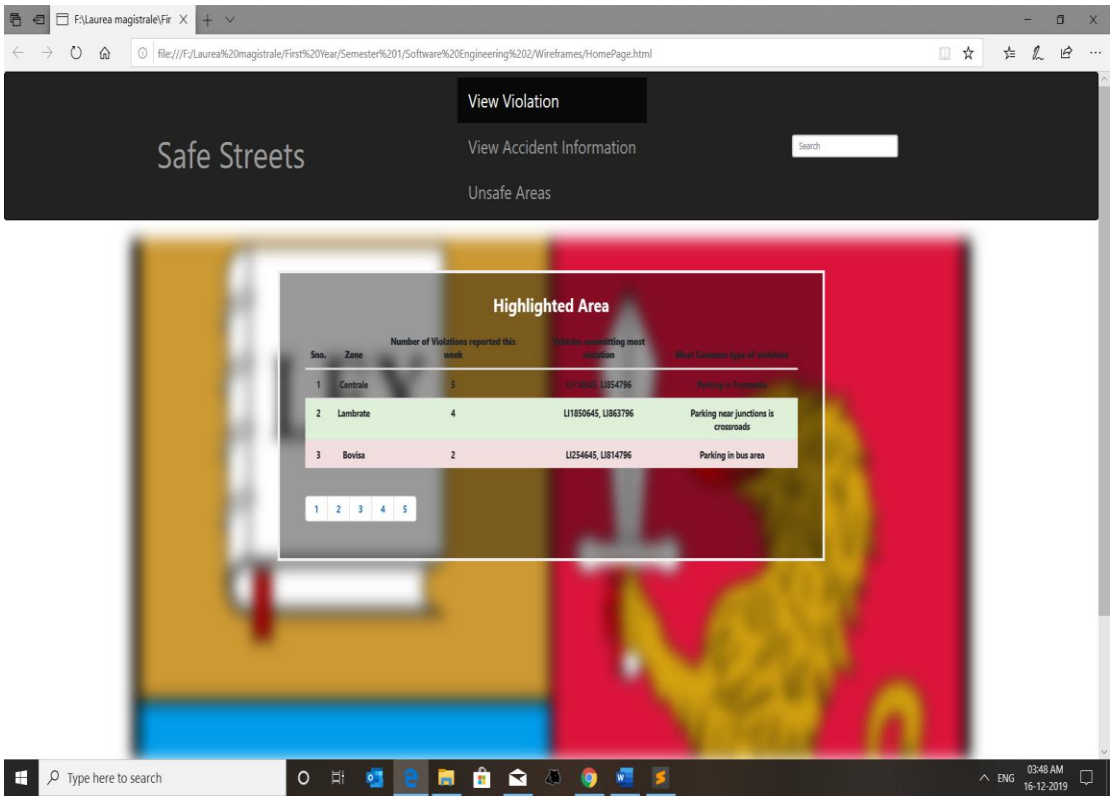

Figure 20: Analyzed Data_Webpage for Authorities

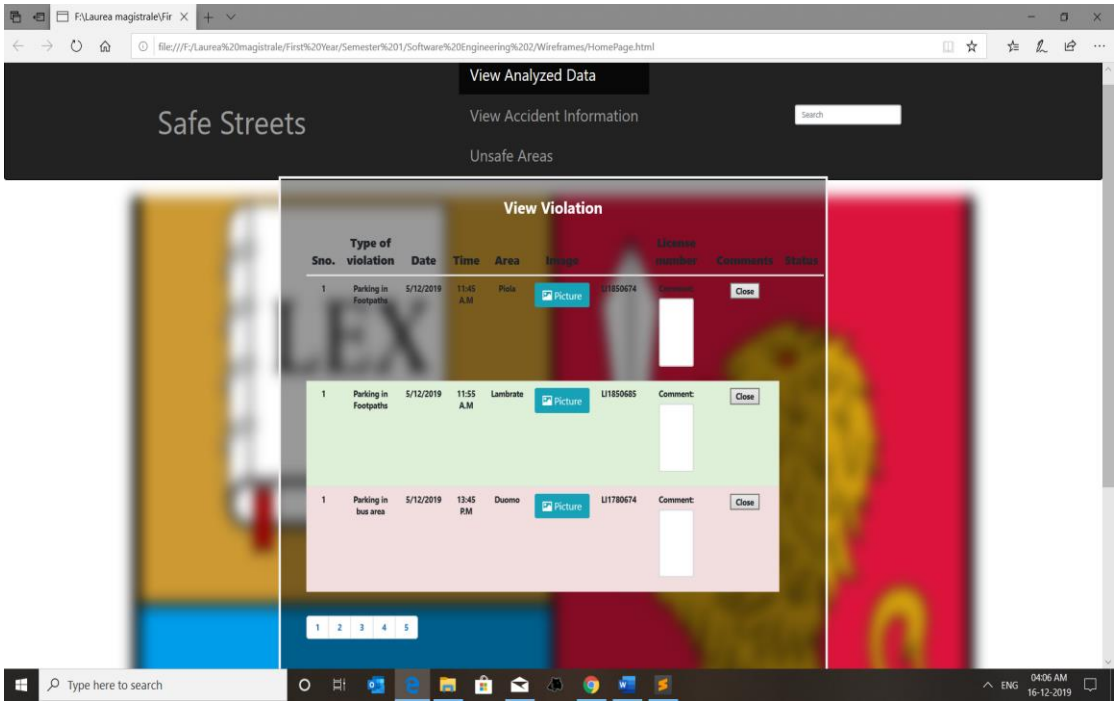

Figure 21: View Violation_Webpage for Authorities

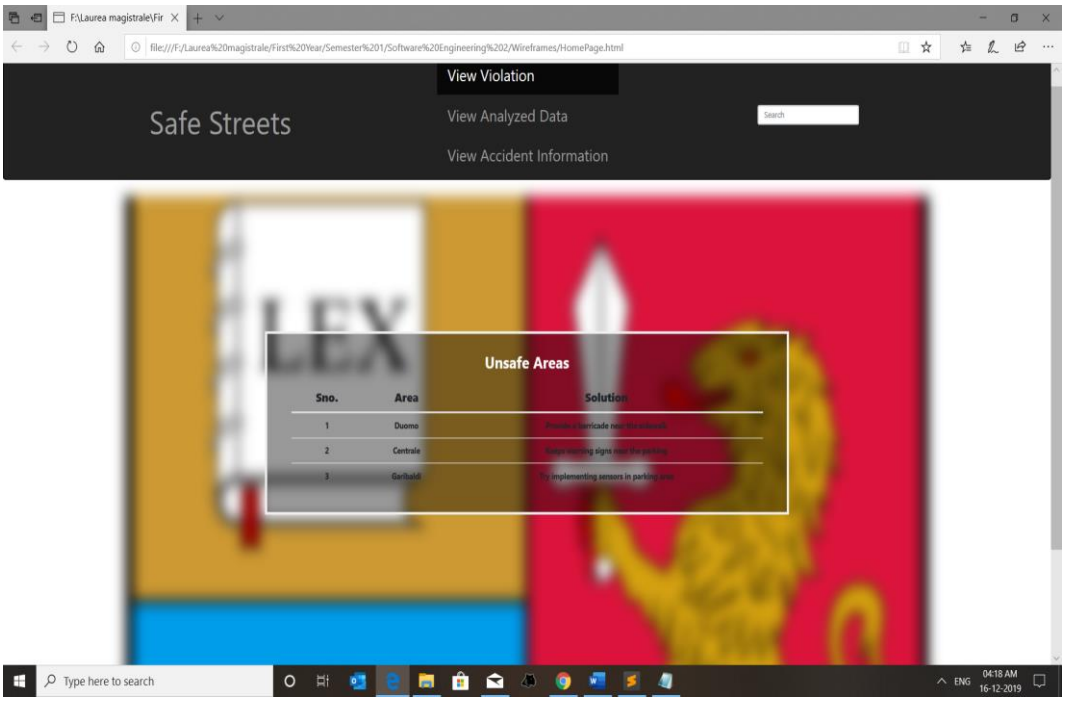

Figure 22: UnsafeAreas_Webpage for Authorities

4. Requirements Traceability

In the below Requirements Traceability the design components are mapped with its corresponding requirements in the RASD.

R1: The system should allow the user to select the type of violation and upload an image of the violation.
Violation Manager: manages all the process related to violation data like storing the data received from the user, updating the data, validating and processing the data.

R2: The system should not allow users to upload images from Gallery to prevent data manipulation rather it should have an inbuilt camera in the application.
Data Validator: Validates the type of image received.
Image Processor: This does not allow the user to upload images from their gallery since it may lead to uploading edited images.

R3: The system should be able to access the location and date and time of the user's data automatically.
Location Identifier: This component access the device location and uses google maps to identify the area of the violation.
Violation Manager: It receives the date and time of the user's device while submitting the violation data.

R4: Authorities should be able to access all the data regarding the violation like type of violation, location and date and time.
Violation Manager: This component provides all the information regarding the violation to the authorities.

R5: Authorities should be able to access the processed data such as the license no. from the image submitted by the user and the area where the violation occurred.
Violation Manager: This component provides all the information regarding the violation to the authorities.
Image Processor: This component provides the license number of the violated vehicle.
Location Identifier: This component fetches the location details and using Google

maps returns the area of the violation.

R6: After taking actions, authorities should close the issue. Such that investing time on the same violation that has been reported and solved already is reduced.

Violation Manager: This component manages all the interaction with the data regarding to violation. As soon as the violation is closed it will update the information in database.

R7: The application should allow both the users and the authorities to retrieve information regarding the area with highest violation recorded.

Data Analyzer: This component allows both the user and the authorities to access data regarding the information about highest violated areas.

R8: The application should allow only the authorities to retrieve data regarding the vehicle that is committing most violation and the most common type of violation.

Data Analyzer: This component allows the authorities to access data regarding all the information about highest violated areas like the vehicle that is committing most violation and the most common type of violation.

R9: The user and the authorities should be able to access the data regarding the accidents from municipality services through Safe Streets.

Accident Manager: This component allows both the user and the authorities to retrieve accident data from the municipality database using municipality services.

R10: The municipality services should provide the data regarding the no. of accidents in each area when queried.

Accident Manager: This component allows both the user and the authorities to retrieve accident data for an area by entering it in the application and searching it. The municipality database is accessed through the municipality services.

R11: Safe Streets should be able to compare the data from the municipality services with its own data.

R12: An area will be characterized as an unsafe area if the frequency of accidents and the no. of violations reported in that particular area is higher than other corresponding areas.

R13: The threshold for an area to be considered safe keeps changing according to the new data.

R14: After each update of the data, it will be compared with the area labelled unsafe. If it is likely to have at least half of the parameters of the unsafe area then it will also be marked as an unsafe area.

R15: Information regarding the road in which most the violations or accidents occurred, the time in which the frequency is higher should also be provided by SS.

Unsafe Area Manager: The unsafe area manager request the accident information from the municipality database and the violation data from the database through database access component. It then uses the information retrieved to identify potentially unsafe areas.

R16: Safe Streets should suggest solutions based on the analyzed data.

R17: The solutions formulated will be sent to the authorities who will be considered responsible for implementing the solutions provided.

R18: Safe Streets should also provide expected results when the solutions are implemented.

Solution Manager: This component provides the solution to reduce the number of violations in that area. It is a subcomponent of Unsafe Area Manager.

5. Implementation, Integration and Test Plan

5.1 General Idea:

The system is divided into various sub systems.

- User Mobile App
- Authority Web App
- Application Server
- External systems: Microsoft Azure Database, Google Maps, Municipality database.

These subsystems will be implemented, integrated and tested by using an incremental approach i.e., top down approach. Incremental Approach is carried out by using dummy programs called **Stubs and Drivers**. Stubs and Drivers do not implement the entire

programming logic of the software module but just simulate data communication with the calling module.

Stub: Is called by the Module under Test.

Driver: Calls the Module to be tested.

Each module is implemented then integrated with the other then it is tested with the stubs of the later modules that are still in the development phase.

Top-down Integration: Testing takes place from top to bottom, following the control flow or architectural structure (e.g. starting from the GUI or main menu). Components or systems are substituted by stubs.

Advantages:

- Fault Localization is easier.
- Possibility to obtain an early prototype.
- Critical Modules are tested on priority; major design flaws could be found and fixed first.
- The tested product is very consistent because the integration testing is basically performed in an environment that almost similar to that of reality
- Stubs can be written with lesser time because when compared to the drivers then Stubs are simpler to author.

Disadvantages:

- Needs many Stubs.
- Basic functionality is tested at the end of cycle

Diagrammatic Representation:

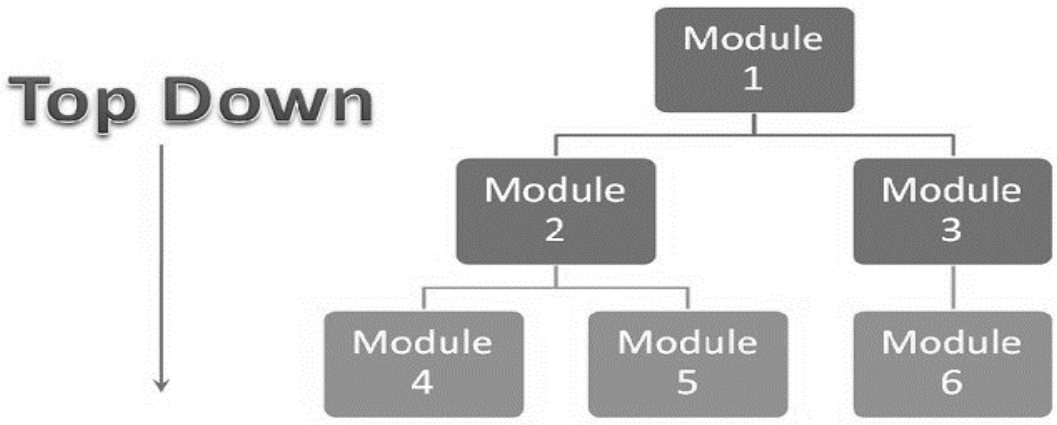

Figure 18: Representation of a Top Down Implementation.

Feature	Business Priority	Difficulty of implementation
Sign up and logIn	*Low*	*Low*
Report violation	*High*	*High*
Visualize Analyzed Data	*High*	*Medium*
Visualize Accident Information	*Medium*	*Medium*
Visualize Unsafe Areas	*High*	*High*

Figure 19: Table representing the priority and difficulty of features.

5.2 Component Integration:

Testing Strategy: Each component should be unit tested before integration. After the integration of each component Regression testing should be done in order to verify there is no breakages in the functionalities that are implemented already. Functional testing should be done after implementing all the modules of the functionality.

After implementing all the components system testing and performance testing should be done to ensure the quality of the system.

Test Coverage: This is essentially required and it will do conformance mapping of the business needs and the test cases so that one can ensure if the entire software has been tested or not.

Integration of Internal Components:

- The feature **Report Violation** is implemented first since it is the core functionality of the system. To implement this functionality the components **Violation Manager, Location Identifier** and **Image Processor** will be implemented and integrated. While testing these modules a stub for Data validation can be introduced.

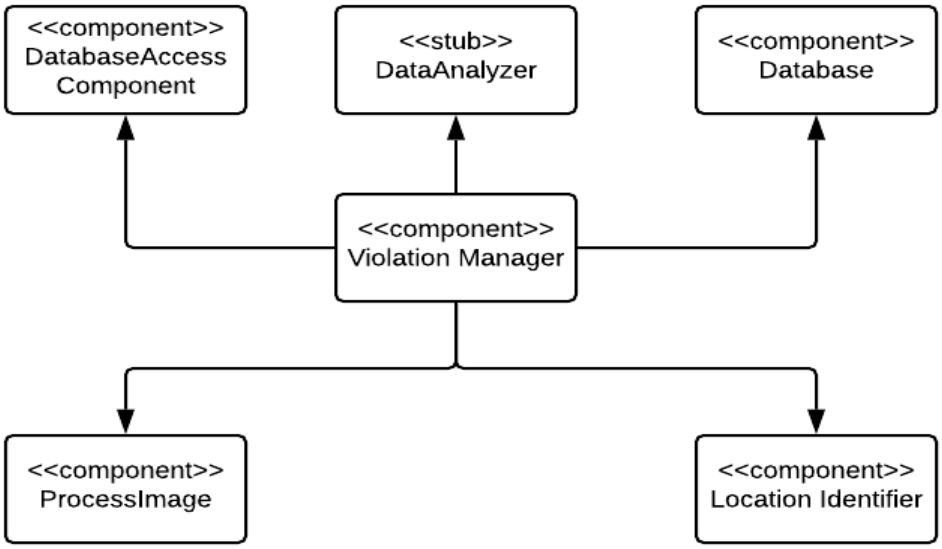

Figure 20: Level 1 Implementation and Integration of components

- **Visualize Analyzed Data** feature is implemented next in the pipeline. The components **Data Analyzer, Database access component and Cloud Database** are implemented. **Data Validator** component is introduced as a stub for testing the data validations.

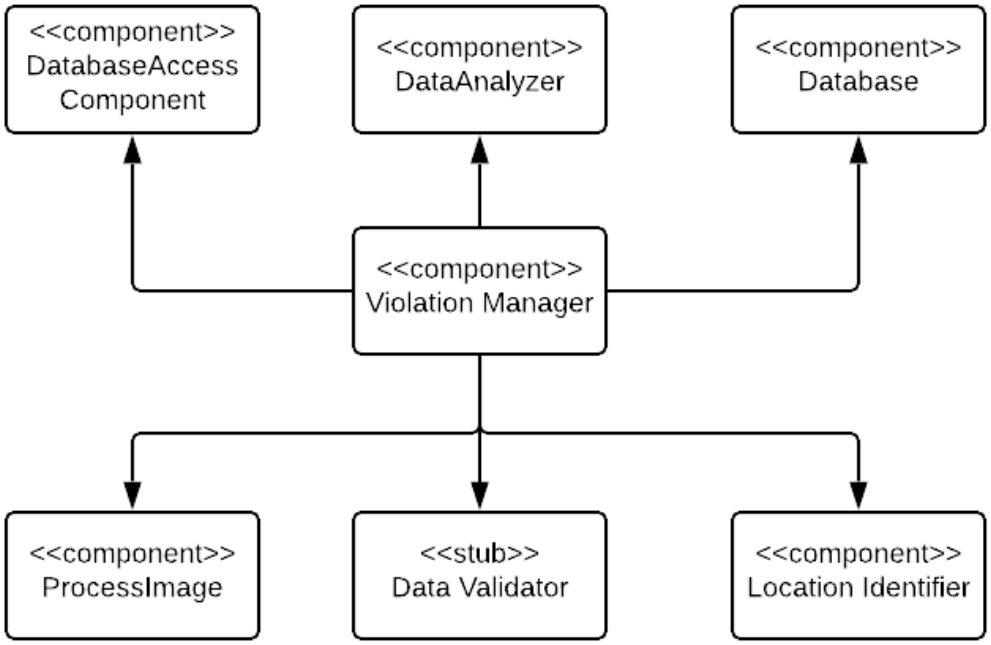

Figure 21: Level 2 Implementation and Integration of components

- **Visualize Accident Information** feature will be implemented using the component **Accident Manager** and **Municipality services API.**

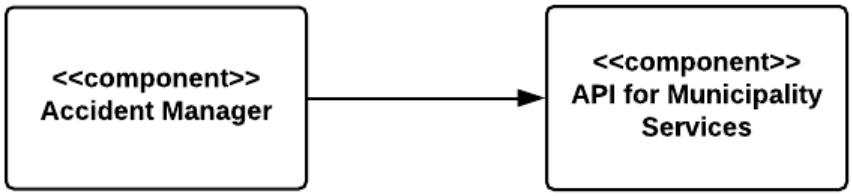

Figure 22: Level 3 Implementation and Integration of components

- **Visualize Unsafe Area feature** will be implemented using the components **Data Analyzer and Municipality services API.**

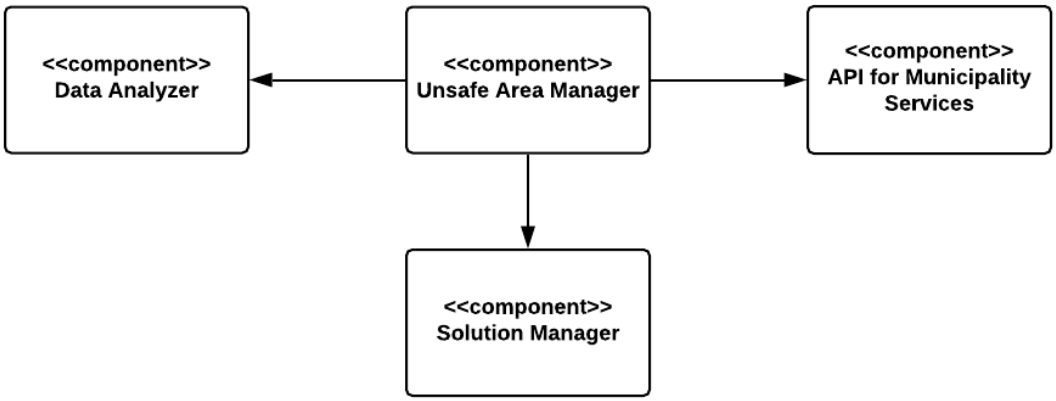

Figure 23: Level 4 Implementation and Integration of components

- SignUp and LogIn are implemented using the components **LogInManager, SignUpManager and Data Validator.**

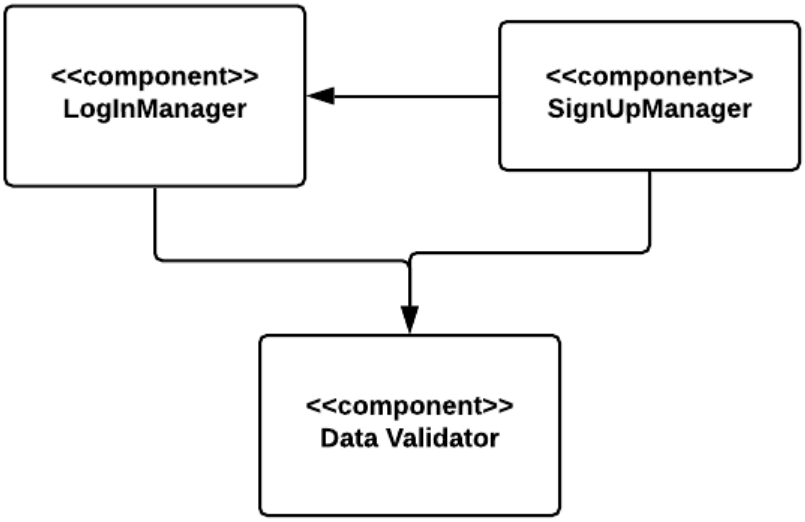

Figure 24: Level 5 Implementation and Integration of components

Integration of the front end with the back end:

The integration and testing between the front end and the back end happens only once all the components of the respective parts have been implemented and unit tested.

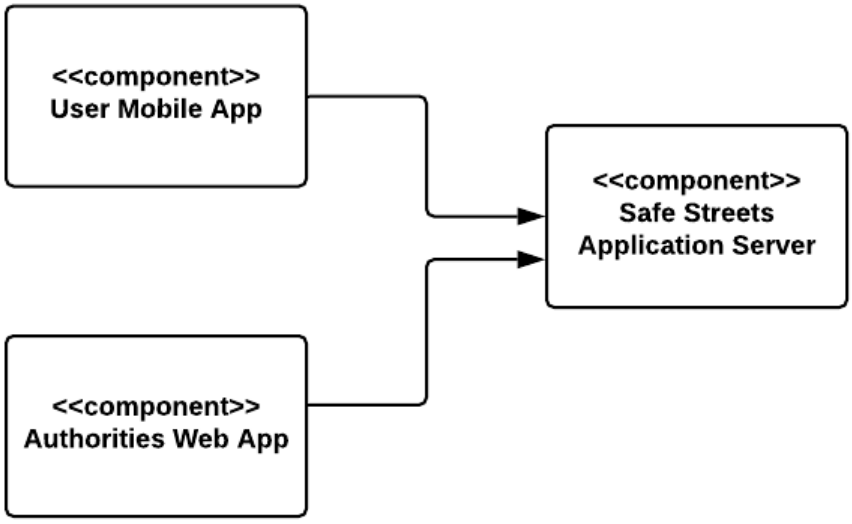

Figure 25: Integration of Front end and Backend

Integration with transport layer:

The integration between the front end and the transport layer happens only after all the components of the respective parts have been implemented and unit tested.

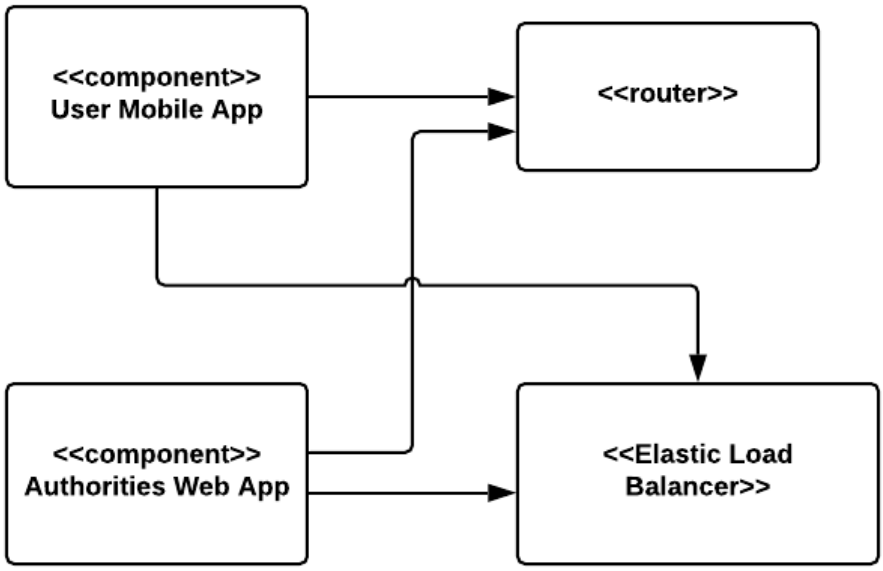

Figure 26: Integration of Front end with transport Layer

Integration with the external services:

The integration with the external services is done after all the needed components have been implemented and unit tested accordingly.

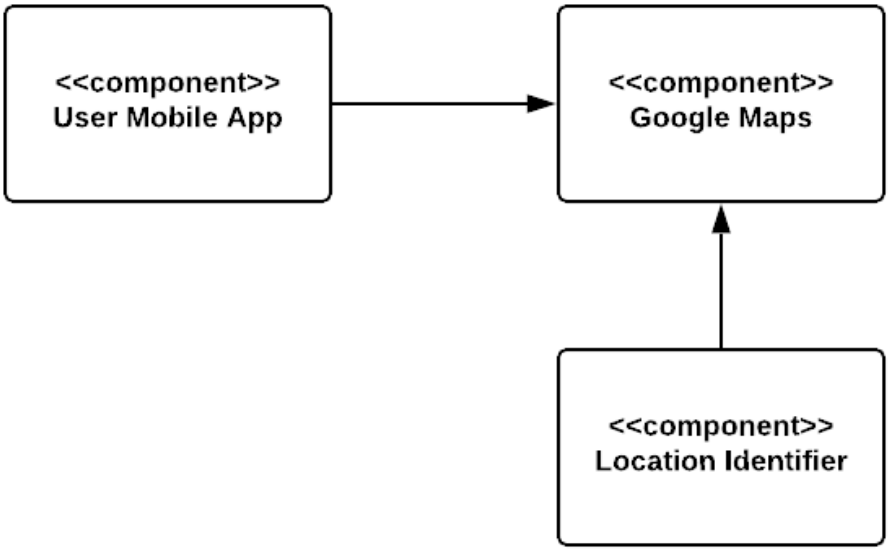

Figure 27: Integration with external services -Google Maps

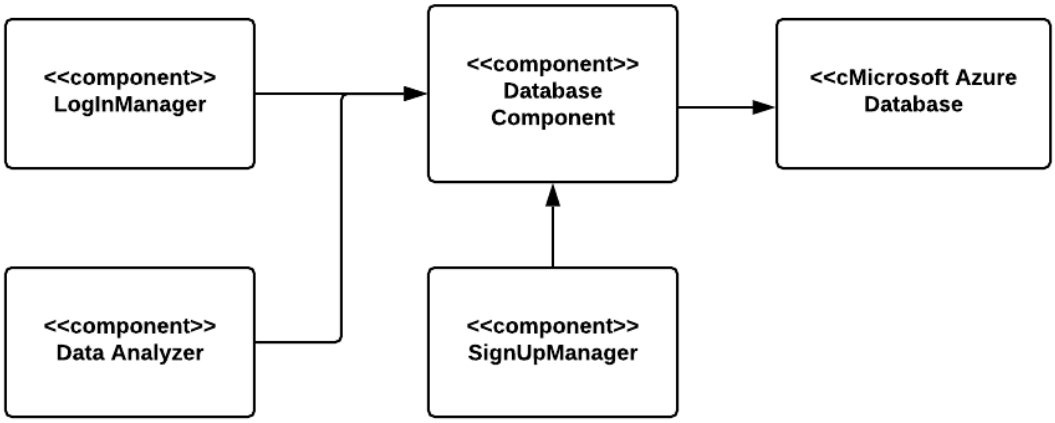

Figure 28: Integration with external services –Database

Integration with the municipality services is already implemented during Visualize Accident Information feature.

6. Effort Spent

ST1

Description of the task	**Hours**
Introduction	2
Architectural design	48
User Interface Design	3
Requirements Traceability	2
Implementation, Integration and test plan	5

ST2:

Description of the task	**Hours**
Introduction	0.5
Architectural design	1.5
User Interface Design	15
Requirements Traceability	2
Implementation, Integration and test plan	7