

Politecnico di Milano

AA 2019/2020

DD - Design Document

Authors:

Devi Radhakrishnan,
Gayathri Prakash Menakath

Professor:

Matteo Rossi

Politecnico di Milano December 9, 2019

Version 1.0

Table of contents

1. Introduction	
1.1. Purpose	4
1.2. Scope	4
1.3. Definitions, Acronyms	
1.3.1. Definitions	6
1.3.2. Acronyms	7
1.4. Revision history	7
1.5. Reference documents	7
1.6. Document structure	8
2. Architectural Design	
2.1. Overview: High-level components and their interaction	9
2.2. Component View	11
2.3. Deployment View	14
2.4. Runtime View	
2.4.1. Reporting a violation	17
2.4.2. Area violation rates - User	19
2.4.3. Retrieving data from third-party services to cross-verify with own data	20
2.4.4. Violation stats of cities and vehicle stats - third party	21
2.5. Component Interfaces	23
2.6. Selected Architectural Styles and Patterns	27
2.7. Other Design Decisions	29
3. User Interface Design	30
4. Requirements Traceability	34
5. Implementation, Integration and Test Plan	38

5.1.	Component Integration	-----	41
6.	Effort spent	-----	45

1. Introduction

1.1 Purpose

The purpose of this document consists of giving more technical details than the RASD concerning the Safestreets application. The design document goes deeper into the detail about designing the system and giving proper guidance on the architecture of the system. The following topics are covered in this document:

- The high-level architecture
- The main components, their interfaces, and deployment
- The runtime behavior
- The architectural styles and design patterns
- User Interface - UX diagrams
- A mapping of the requirements on the architecture's components
- Implementation, integration and testing plan

1.2 Scope

Here a review of which is the scope of the application is made referring to what has been stated in the RASD document.

Safe Street being a crowd-sourced application helps to provide users with the possibility to notify authorities when traffic or parking violations occur. The application offers public or government authorities to collect the necessary data of the violators and the areas prone to violations. This will, in turn, help them to enforce more rules to prevent accidents that happen due to traffic rules violation.

The application is expected to have two types of customers, first being the users who wish to report a traffic violation or retrieve statistics on violation reports in a particular

area and third parties like authorities who can analyze and take necessary action regarding the reported violations. The S2B will allow the user to insert details of the violations like date, time, location, and the provision to send pictures of the vehicles. The user can also mention the type of violation like traffic rule violation, parking violation, etc. The user can send photos of the vehicles violating the rules. SafeStreets' NotifyViolation service will run an algorithm that will read the license plate from the received images. This will help to keep track of the vehicles that commit the most number of violations. The user can also be asked to enter the vehicle number manually so that it can be cross verified with the data received as a result of running the algorithm. The location can be determined either by requesting the user input or by retrieving the geographical position of the violation. It must be assumed that the user's device is capable of retrieving geographical data (GPS), sending images (camera) and that the images sent are of decent quality so as for the algorithm to deduce the license plate from it. The location can be verified by cross-checking the user input with that of the geographical data from GPS and also from the co-ordinates retrieved while reading the images to retrieve license plates.

Once a violation is reported by the user NotifyViolation will send a notification about the violation to the authorities who are the third parties. While reporting a violation third parties like the police, municipalities or any other government organization will be able to access the reporting user's identity and contact information so as to verify that the user is not an imposter. This will help to prevent fake notification alerts. The third parties will also be able to contact the users if needed.

In addition to notifying violations, both end-users and the authorities can mine through the information about the occurrences of the accidents in the municipality. This will help the authorities to strategize their operations to prevent further accidents. On login, a normal user will be able to see the statistics of areas with the most number of violations of all the types. The user will be able to know about the areas that have the highest

accident/violation rates. The application allows third parties like authorities to view the violations in a particular area and also the information of the vehicles that make the most number of violations.

If the municipality or other organizations offer services that will have information about the accidents or violations that occur in their territory then PreventViolations service of the SafeStreets makes it possible for it to acquire this data for further operations. PreventViolations cross-checks this data with that of the already existing data that it has from the user inputs. This will help to potentially identify areas with risk of accidents and can suggest possible interventions like signboards, barriers between a bike lane and the part of the road for motorized vehicles, etc. Here the municipality is the third party and it is assumed that municipalities will have such a service for PreventViolations to acquire data. The user will be able to know the safety level of his/her location by enabling the location service of the device. Upon retrieving the geographical data of the user's current location PreventViolation can notify the user about violations happened in that area.

1.3 Definitions, acronyms, abbreviations

1.3.1 Definitions

- **User:** The normal customer of the application that exploits the basic features of the application to notify the authorities about violations. The user can also enable PreventViolations to get the risk rate of a particular location.
- **Third-party:** The customer of the application gets notified when a normal user reports a violation. Third-party gets the details of the vehicles that commit the most number of violations.
- **Customer:** General customer of SafeStreet, can be both user or third party.

- **Traffic violation:** When a person parks the car on the wrong side of the road or whenever a person violates the traffic rules.

1.3.2 Acronyms

- **GPS:** Global Positioning System
- **API:** Application Programming Interface
- **S2B:** Software To Be
- **RASD:** Requirements Analysis and Specifications Document
- **UML:** Unified Modelling Language
- **GUI:** Graphical User Interface
- **HTTP:** HyperText Transfer Protocol
- **TSL:** Transport Layer Security
- **JSON:** JavaScript Object Notation
- **SQL:** Structured Query Language
- **AWS:** Amazon Web Services
- **EC2:** Elastic Compute Cloud
- **S3:** Simple Storage Service

1.4 Revision History

- Version 1.0
 - First release

1.5 Reference Documents

- Specification Document: “Mandatory project Assignment AY 2019 - 2020”
- IEEE Std 830--1998 IEEE Recommended Practice For Software Requirement Specifications
- UML Diagrams: <https://www.uml-diagrams.org/>

1.6 Document Structure

Chapter 1 is an introduction to the design document. It explains the goals of the project and highlights the differences with the RASD while showing the connection between them.

Chapter 2 provides a detailed description of the architectural design of the application. This section is further divided into the following sections which explain the different views, architectural designs, and patterns:

- Overview: High-level components and their interactions
- Component view
- Deployment view
- Runtime view
- Component interfaces
- Selected architectural styles and patterns
- Other design decisions

Chapter 3 specifies the user interface design. Though the RASD contains all the required mockups, this section contains UX diagrams that provide more clarity on the interaction between the application and the customers.

Chapter 4 contains the traceability of the requirements. This section explains how the requirements mentioned in the RASD are linked to the design elements mentioned in this document.

Chapter 5 explains in detail the implementation, integration and testing plan and how it is to be executed.

Chapter 6 shows the effort put in by each team member in completing this document.

2. Architectural Design

2.1 Overview: High-level components and their interaction

The application to be developed is a distributed application and the three logic software layers of Presentation, that manages the user interaction with the system, Application, that handles the business logic of the application and its functionalities, and Data Access, that manages the information with access to the database, are thought to be divided on three different hardware layers (tiers) that represent a machine (or a group of machines) so that any logic layer has its own dedicated hardware: three-tier architecture. The presentation layer is the topmost level of the application. It displays information related to the services provided by the application. It communicates with other tiers by which it puts out the results to the browser/client tier and all other tiers in the network. In simple terms, it is a layer that users can access directly (such as a web page, or an operating system's GUI). The application layer, also known as the logic tier controls an application's functionality by performing detailed processing. This layer coordinates the application, processes commands and makes logical decisions. It also moves and processes the data between the two surrounding layers. In particular, the second tier is thought to contain only the business logic to physically separate clients and data to guarantee more safety in accessing data since the system deals with sensitive data. The data access layer includes database servers where it encapsulates the servers and exposes only the data. Here information is stored and retrieved from a database server.

The following image show the high-level architecture of the system. It provides only the basic idea of the three layers.

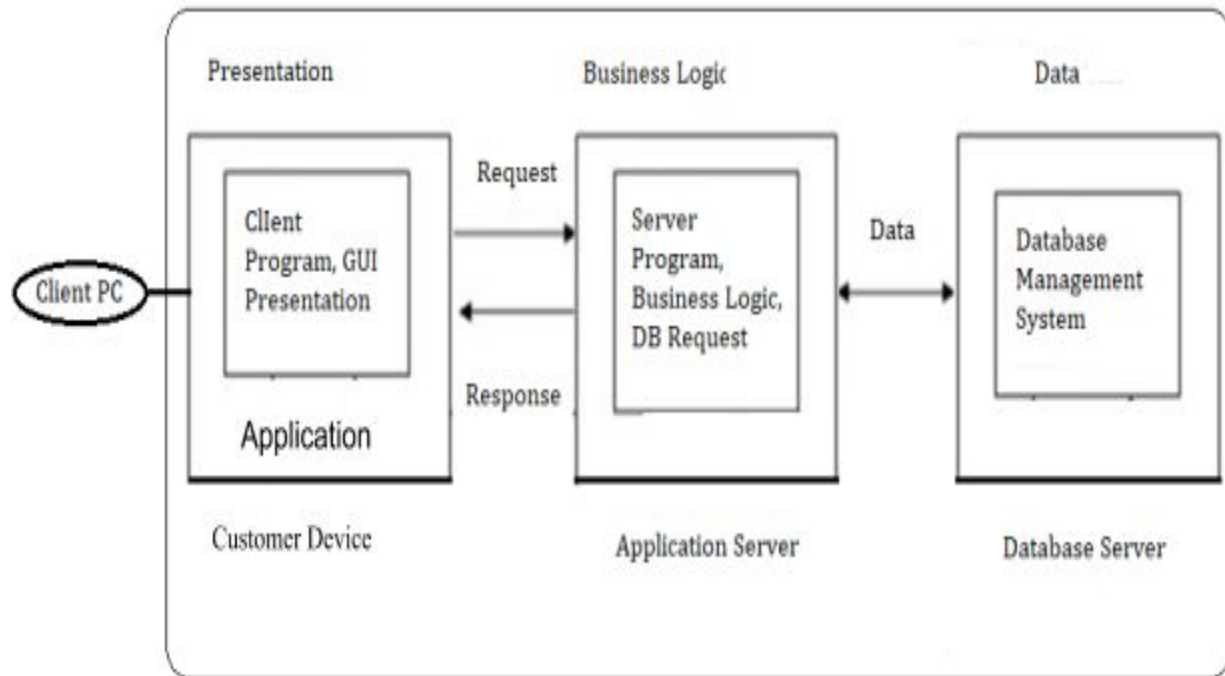


Figure 1: High - level system architecture

Third parties can communicate through their devices with the application level to make queries and to organize competitions with asynchronous message flow. The application-level server will interact with the customers when sending notifications (alerts etc) in an asynchronous way. The server in the application layer communicates synchronously with the database server (data access layer) to retrieve information or asynchronously to store information when needed.

The application server provides customers with access to the business logic which generates dynamic content. Then we have a web server for communication with the third parties' web application. The web server does not generate dynamic content through plugins for scripting languages, it will forward the requests to the application server. Caches are used to fasten and lighten the communication. The cache needs to have an appropriate knowledge of data that makes up the business logic and the transformation of the data at the user interface level. The caches for application servers can't be exploited because of node replications. It is not possible to predict what will be

requested from the application servers hence it is impossible to provide a cache that would the usage of the database server every time.

SafeStreet exploits data mining techniques to analyze the big quantity of data available and to extract data from the services offered by third parties like municipalities. It crosses verifies the data available in its system with that retrieved from the third-party services to analyze the violation rates at various locations and provides suggestions to avoid violations or accidents. The application also employs an algorithm that is run on the images uploaded by the users to figure out the vehicle registration number.

2.2 Component View

The following diagram shows all the components of the application and also their interactions with each other. For the sake of simplicity the only physical component represented is the 'ApplicationServer'. In the diagram only the application server subsystem is analyzed in detail because it is the core component of the system: it contains the business logic (Application Layer). The other components of the Presentation layer, of the Data access layer and the web server are represented (through their software components only) just to represent their interactions with the application server.

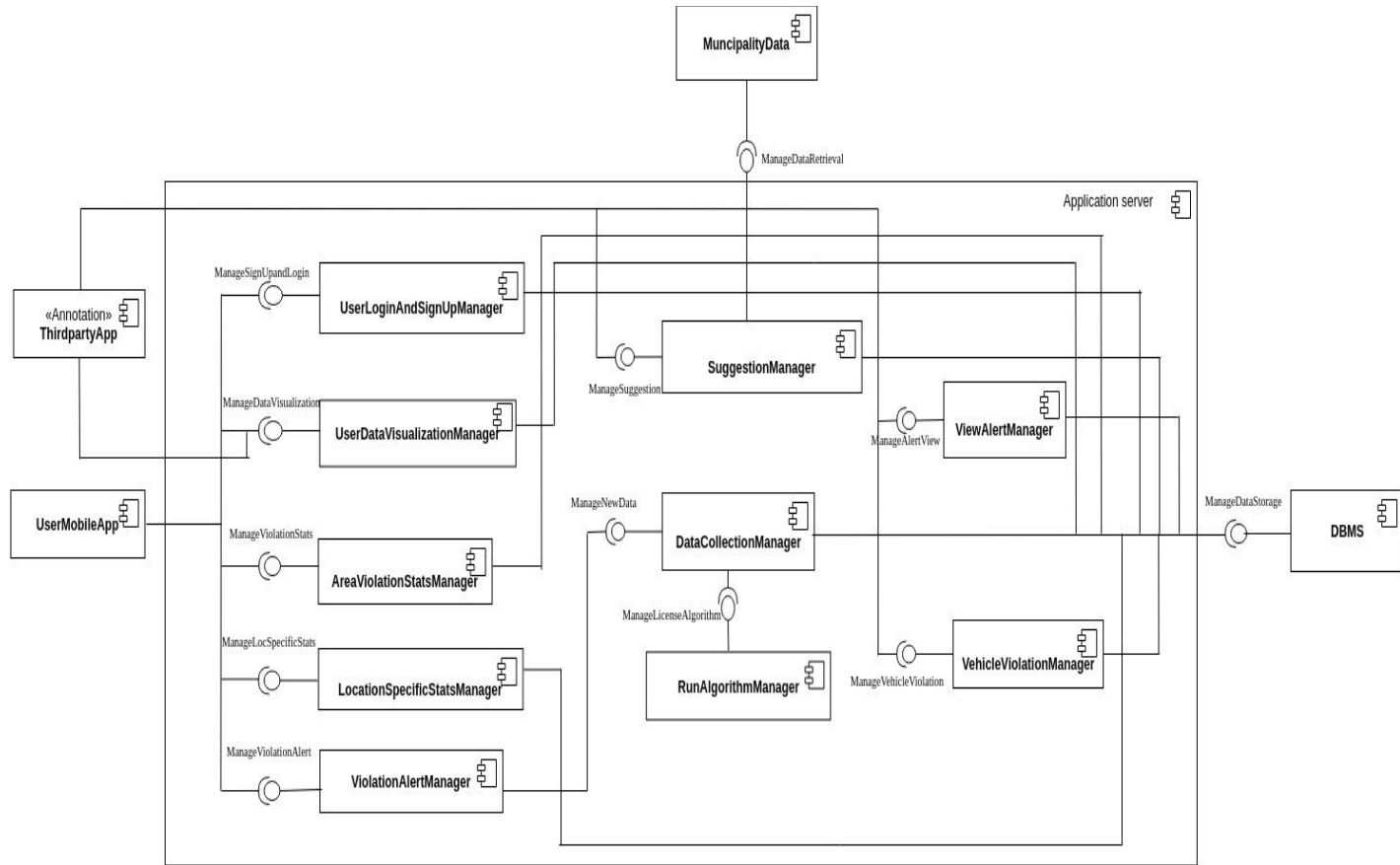


Figure 2: Component Diagram

The components' functions contained in the 'ApplicationServer' are described in the following:

- **UserLoginAndSignUpManager:** This component ensures that the customers register with the SafeStreets application. It also interacts with the DBMS to store the data collected from the customers at the time of registration. It manages the authentication of the customers to the application. It verifies the credentials with that in the DBMS.
- **UserDataVisualizationManager:** This component is used when the user wants to access his/her own personal data provided at the time of registration. It contains the applicative logic to handle the requests and invoke the DBMS to retrieve the data and provide the correct details.

- **DataCollectionManager:** This component receives all the data entered by the users and has to store them in the database. It also provides accurate information from the database to other components.
- **RunAlgorithmManager:** This component ensures the algorithm is run on the images uploaded by the user and the result is passed on to the ViolationAlertManager along with other details that the user had given on reporting the violation.
- **ViolationAlertManager:** This component receives the violation report from the user's application along with the result from the RunAlgorithmManager (the license plate number) and forwards it to the concerned third party.
- **AreaViolationStatsManager:** It provides the users and the third parties with the stats of the cities with the highest violation rates.
- **LocationSpecificStatsManager:** This component provided the violation rates of the location that the user specifies.
- **ViewAlertManager:** This component allows third parties to view the violation reports made by the users.
- **MunicipalDB:** This component provides services using which the application can retrieve violation data from third parties like municipalities.
- **SuggestionsManager:** This component verifies the data retrieved from the services offered by third parties with that of the data available with safe street to analyze the violations taking place at various locations and provides suggestions to prevent the same.

The components that majorly exploit the APIs offered by Google Maps are the AreaViolationStatsManager and the LocationSpecificStatsManager. The Google Maps APIs are also used while reporting a violation when the users wants to identify the location.

2.3 Deployment View

A 3-tier application architecture is a modular client-server architecture that consists of a presentation tier, an application tier and data tier. The data tier stores information, the application tier handles logic and the presentation tier is a graphical user interface (GUI) that communicates with the other two tiers.

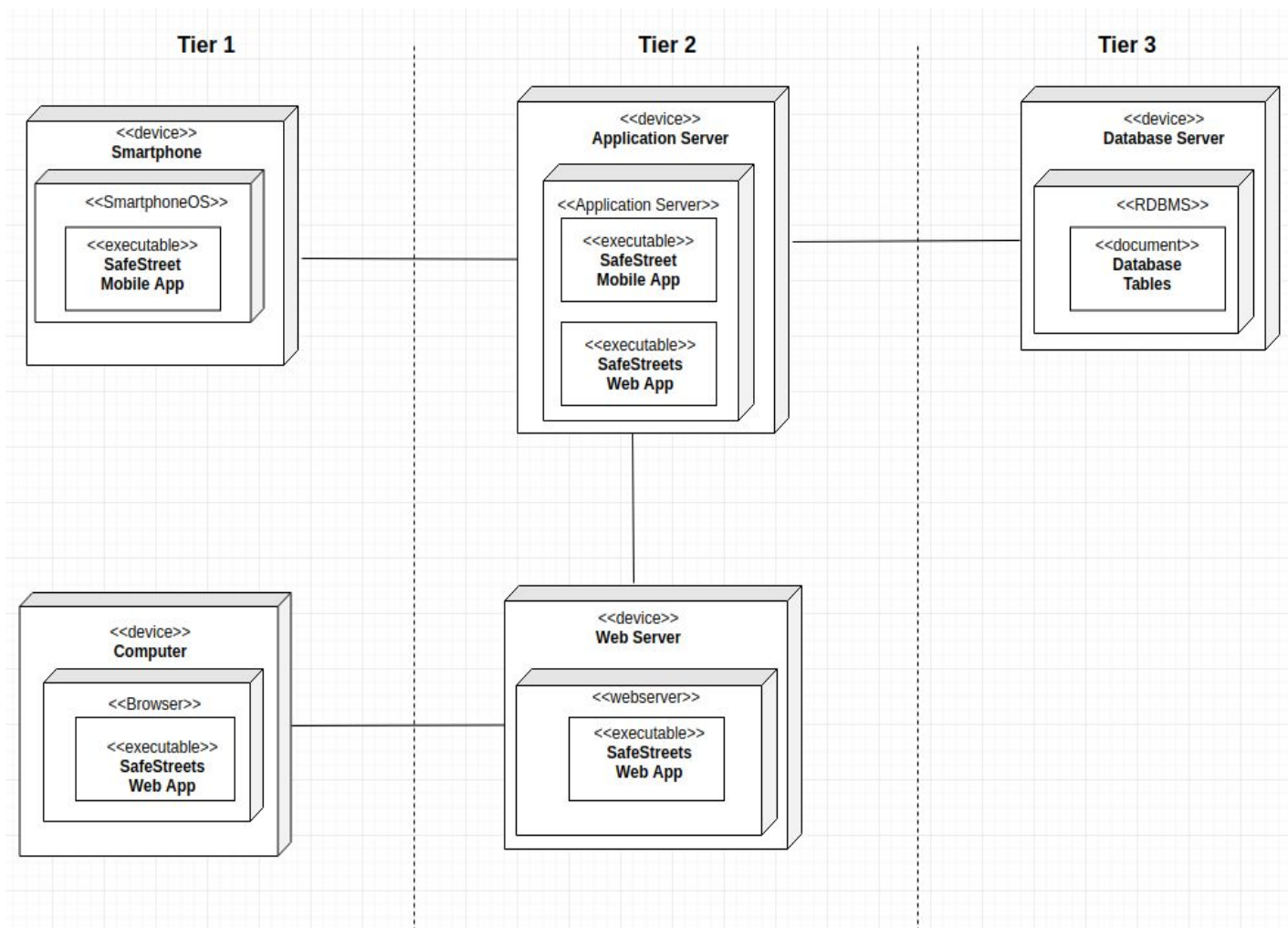


Figure 3: Deployment Diagram

'<<executable>>' and '<<document>>' are standard UML stereotypes that apply to artifacts: the former represents a program file that can be executed on a computer

system while the latter represents a generic file that is not a «source» file or «executable». The three tiers are logical, not physical, and may or may not run on the same physical server. The three tiers are:

- **Tier 1 - Presentation:** The presentation logic must be deployed here. The users must be provided with a mobile application on their smartphone and third parties with a web application accessible from the browsers. The mobile application must be available for both Android and iOS to make it available on most of the devices. The web application must be compatible with all the mainstream browsers like Google Chrome, Safari, Internet Explorer and Firefox.
- **Tier 2 - Application:** The application logic is deployed here. The application server implements all the business logic. It also handles all the requests and provides accurate data for all the services offered by the application. It is directly addressed by the mobile application and also handles some requests forwarded by the webserver.
- **Tier 3 - Data:** The data access layer is deployed here. A relational database (RDBMS) is employed. This is because the structure of a relational database will allow the linking of data from different tables with the use of foreign keys which are used to uniquely identify an atomic piece of data within that table. The application has to handle a lot of complicated querying, database transactions and routine analysis of data. For all these reasons a relational approach is best advised.

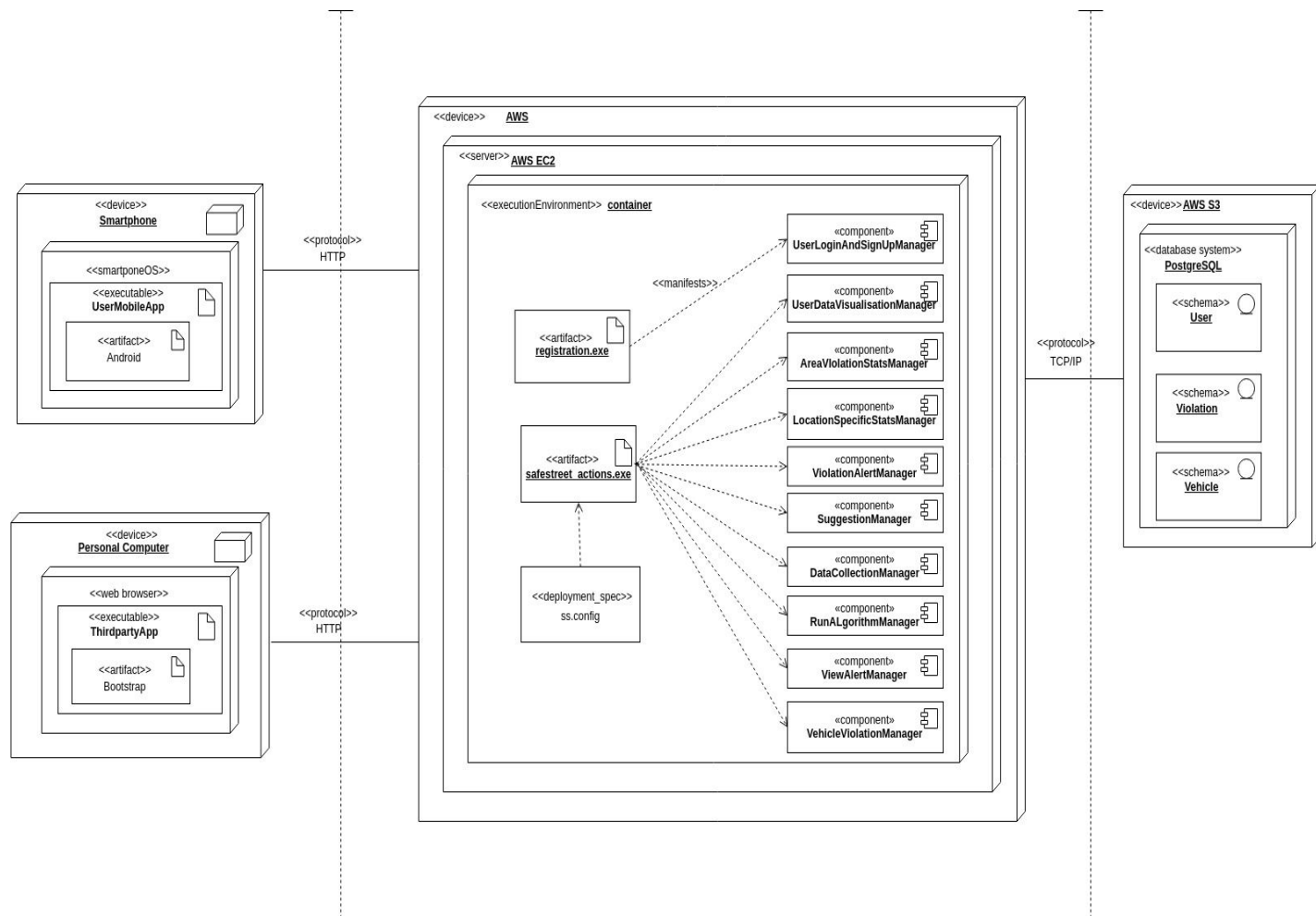


Figure 4: Deployment Diagram

AWS services like EC2 and S3 are being used. Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers. Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction. Amazon S3 is an object storage service that offers industry-leading scalability, data availability, security, and performance. This means customers of all sizes and industries can use it to store and protect any amount of data for a range of use cases, such as websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics. An EC2 instance is like a remote computer running Windows or Linux and on which you can install whatever

software you want, including a Web server running PHP code and a database server. Amazon S3 is just a storage service, typically used to store large binary files

2.4 Runtime View

2.4.1 Reporting a violation

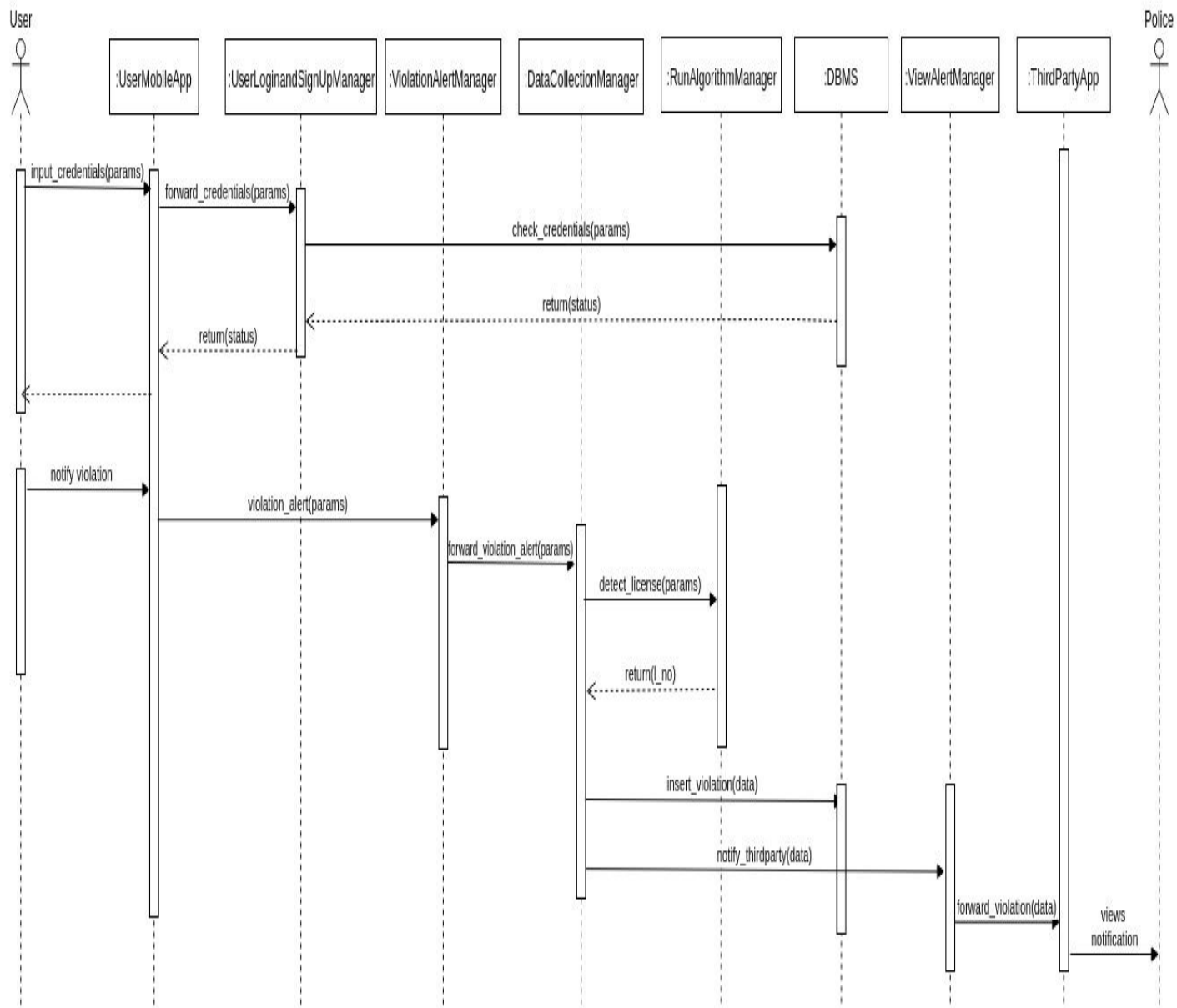


Figure 5: Reporting a violation

This sequence diagram shows the process of reporting a violation. A user has to be authenticated to the application in order to utilize its services. The UserLoginAndSignUpManager is used for this purpose which checks the credentials of the user. The next step is to report the violation by entering the necessary details. The data will be stored in the database. An Algorithm is run on the images uploaded by the user. RunAlgorithmManager is employed for this purpose and the result will be the license number of the vehicles. Once third party is notified of the violation the third party who is police can view the violations.

2.4.2 Area violation rates - User

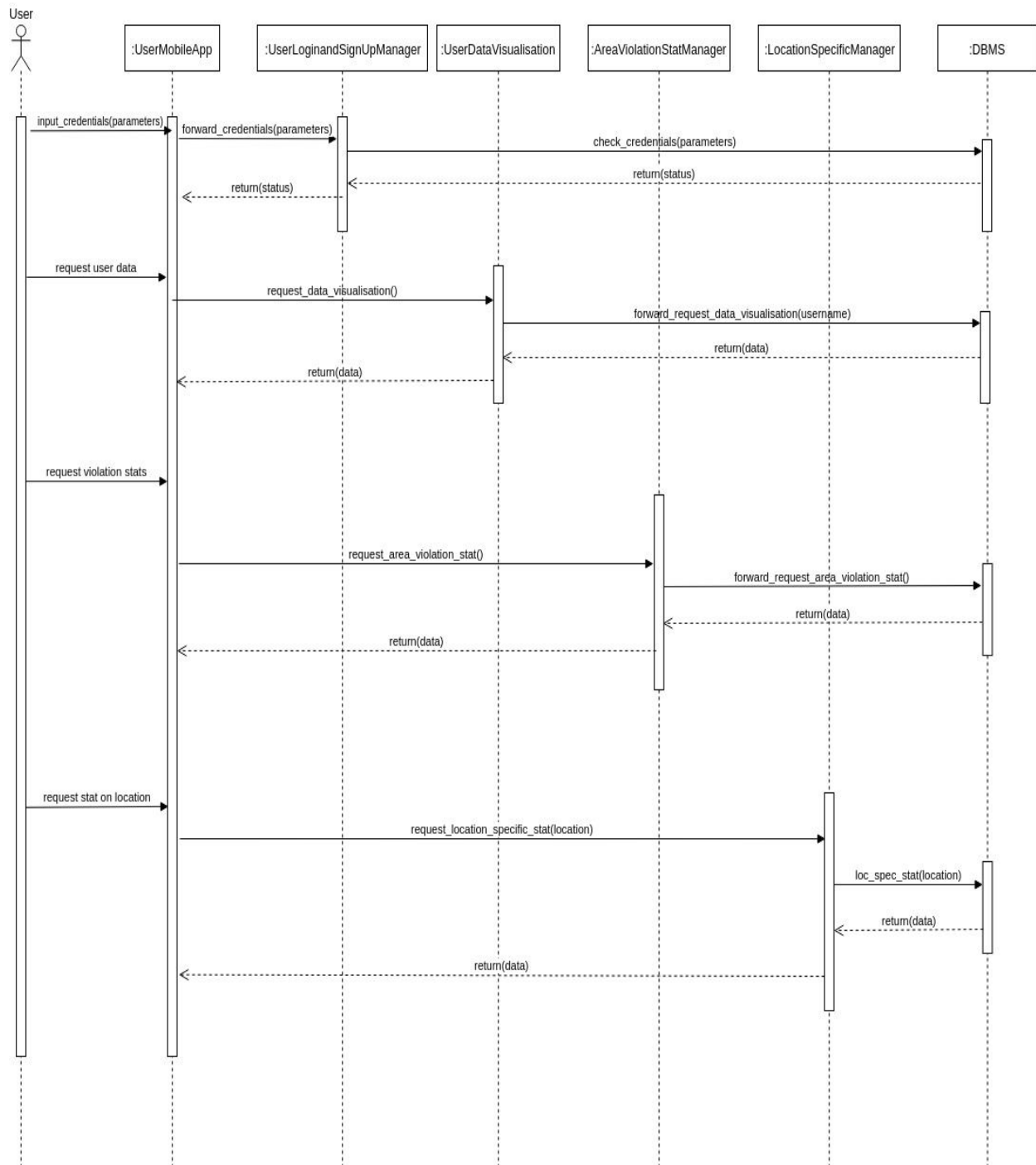


Figure 6: Area violation rates - User

The user has to be authenticated to the system in order to use the services provided by the application. In order to get the highest violation rates of cities, the AreaViolationStatsManager is used. To get the violation rates of a specific location we communicate with the Google Maps API is used in this case to get the accurate location. To get the details of vehicles that have committed the most number of violations the VehicleViolationStatManager is used which gives all the information of those vehicles.

2.4.3 Retrieving data from third-party services to cross-verify with own data

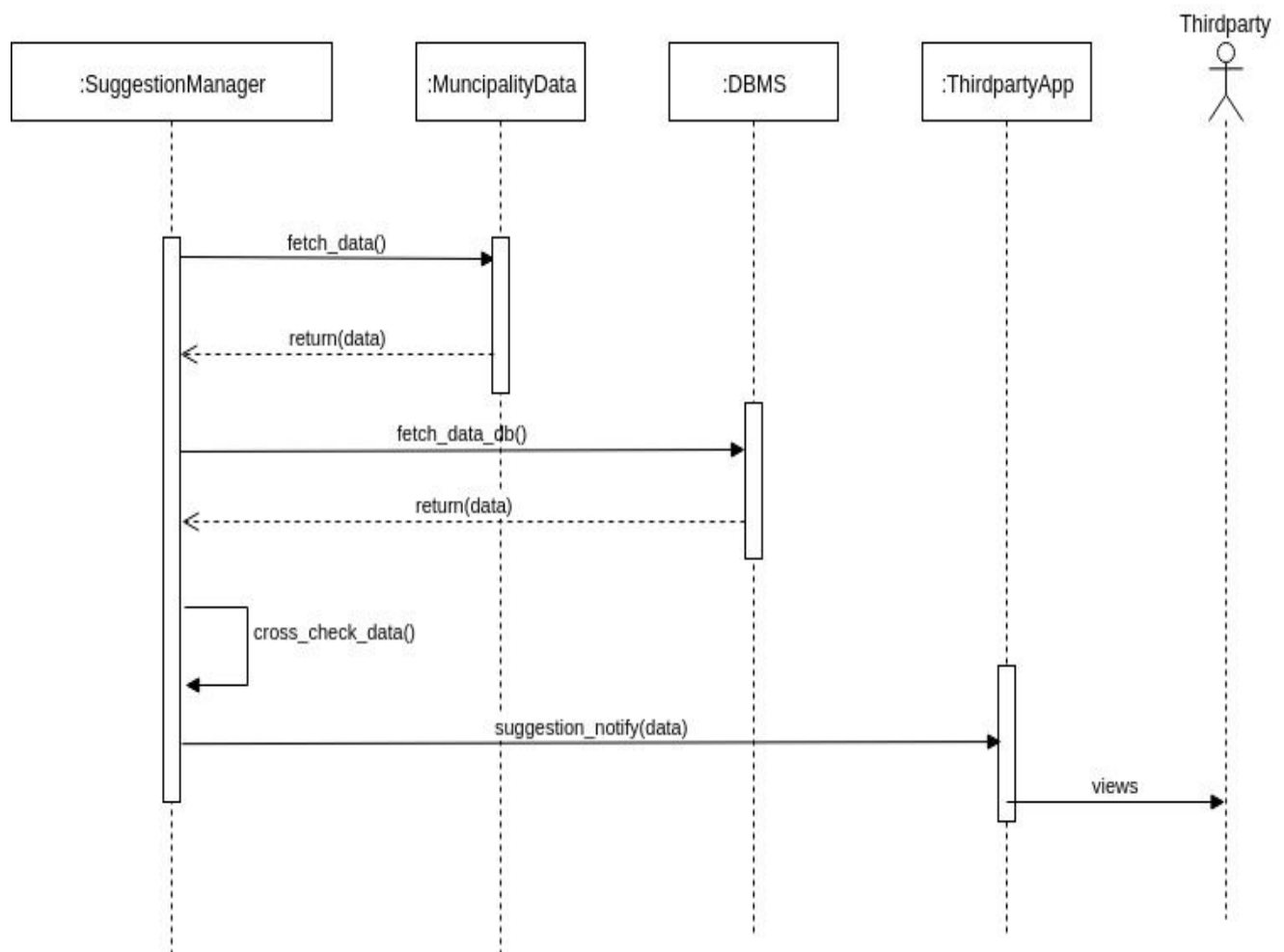


Figure 7: Retrieving data from third-party services to cross-verify with own data

The SuggestionManager will retrieve data from the data offered by the third party services by communication with the MunicipalDB. It then crosses verifies this data with that of its own data to provide suggestions on how to prevent violations.

2.4.4 Violation stats of cities and vehicle stats - third party

Third Party

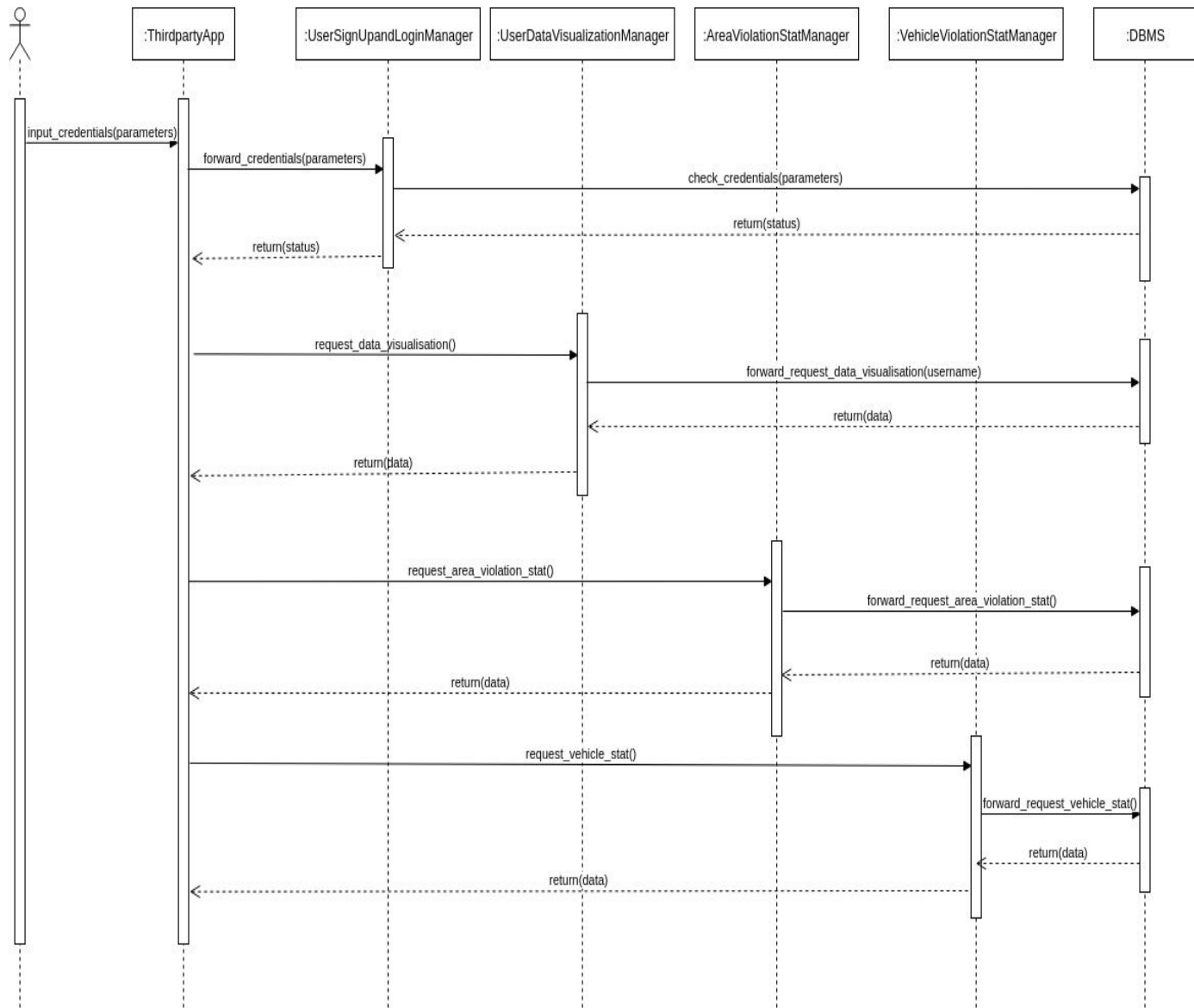


Figure 8: Violation stats of cities and vehicle stats - third party

The third-party has to be authenticated to the system in order to use the services provided by the application. In order to get the highest violation rates of cities, the AreaViolationStatsManager is used. The Google Maps API is used in this case to get an accurate location. To get the details of vehicles that have committed the most number of

violations the VehicleViolationStatManager is used which gives all the information of those vehicles.

2.5 Component Interfaces

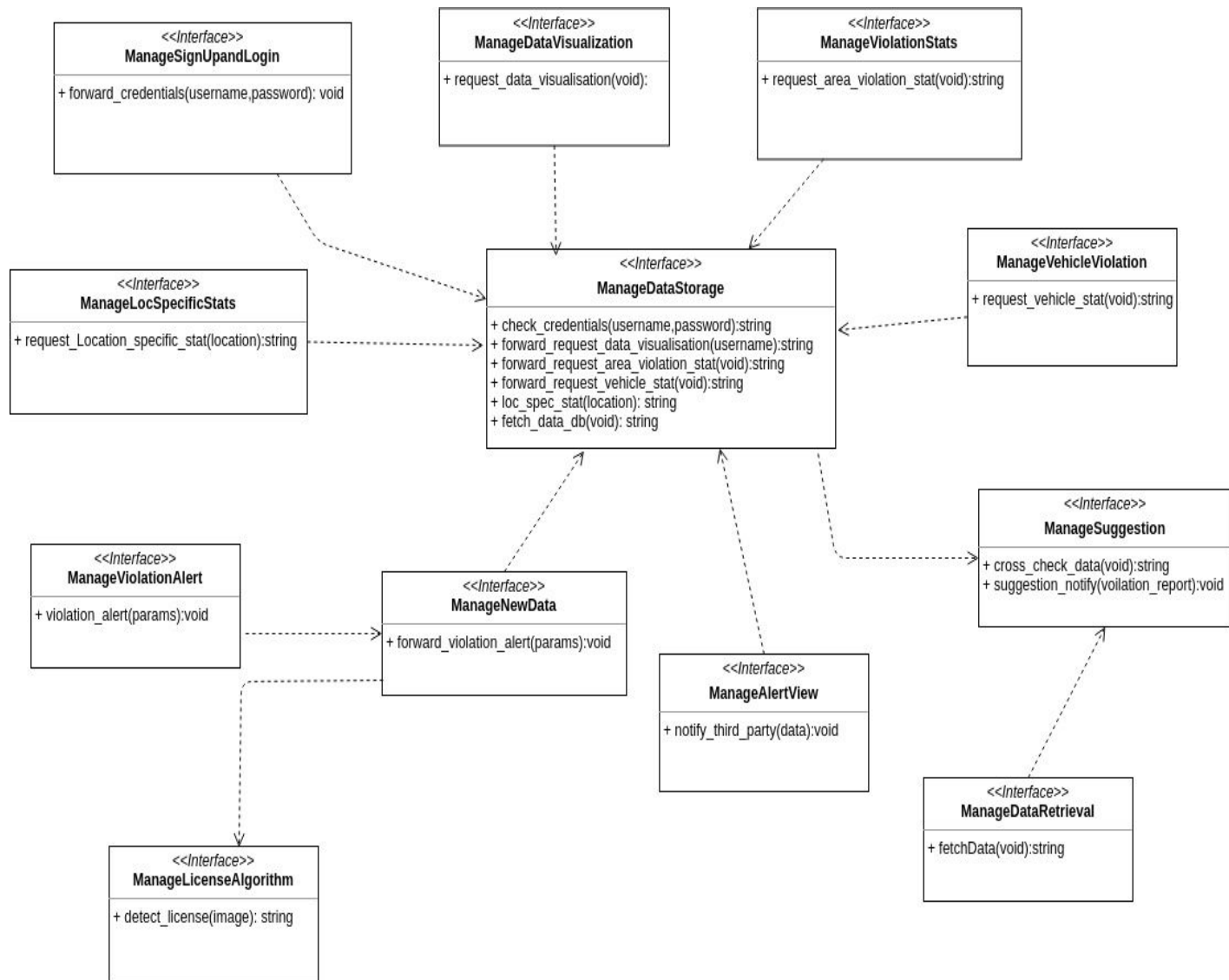


Figure 9: Component Interfaces

In the above figure the component interfaces belonging to the application server, with reference to what was shown in the component diagram, are represented. The arrows represent a dependency relation.

Certain things have to be pointed out in order to give a better understanding of how those component interfaces are intended:

- Only the interfaces offered by the application server are chosen to be represented.
- ManageLoginandSignUp forwards the credentials entered by the customer while logging in against the values present in the database. It enters the values of the customer when signing up.
- The ManageViolationStats interface has the request_area_violation_stat method which gives the information of cities with the highest violation rates.
- The ManageLocSpecificStats provides the request_location_specific_stat method which takes location as a parameter and returns the violation rates of that particular location.
- The ManageDataRetrieval interface provides the fetchData method to retrieve data from the third-parties through services offered by them. The returned data is used by the ManageSuggestions interface.
- The ManageSuggestion interface offers two methods. The cross_check_data method retrieves cross verifies the retrieved data with the data available in the application's database to analyze the violation rates at each location. The suggestion_notify method will provide suggestions to the third parties to prevent violations.
- The request_vehicle_stats method offered by the ManageVehicleViolation interface provides the details of the vehicles that have committed the most number of violations.
- The ManageNewData interface offers one method, where the new violation reported by a user is collected and an algorithm is run to detect the license plate

from the image provided of the vehicle in the violation report and then all the data of the violation is then inserted into the database.

- When a third-party sign up it can either be the police or the municipality. Only the third party who is the police will be notified in case of violation reports.
- The `ManageViolationAlert` interface offers a `violation_alert` method which helps the user to file a violation report. It takes all the details like location, color of the vehicle, description, and images as parameters. Once the images are uploaded an algorithm is run on them to detect the license numbers. This is done by the `detect_license` method offered by the `ManageLicenceAlgorithm` interface. The data is then stored in the database.
- The `ManageAlertView` interface provides the functionality of notifying the third party (police) of the violation reported by a user by fetching the violation report from the database and then forwarding it to the third-party web application.
- `ManageDataStorage` deals with all the methods to insert and retrieve violation reports for various purposes and for checking the credentials of the customer while logging in.
- `ManageDataVisualisation` enables the user to view all the details of the user and the violations reported by that user.
- `ManageLicenseAlgorithm` takes in the image of the vehicle provided by the user from the interface `ManageNewData` and then runs an image processing algorithm to detect the license plate number of the vehicle involved in the violation and then it is returned to the interface `ManageNewData` which forwards it to the database for storing.

The following class diagram represents the model of the considered system. The decision to use a class diagram instead of an Entity-Relationship model is because they can be considered pretty interchangeable in this context.

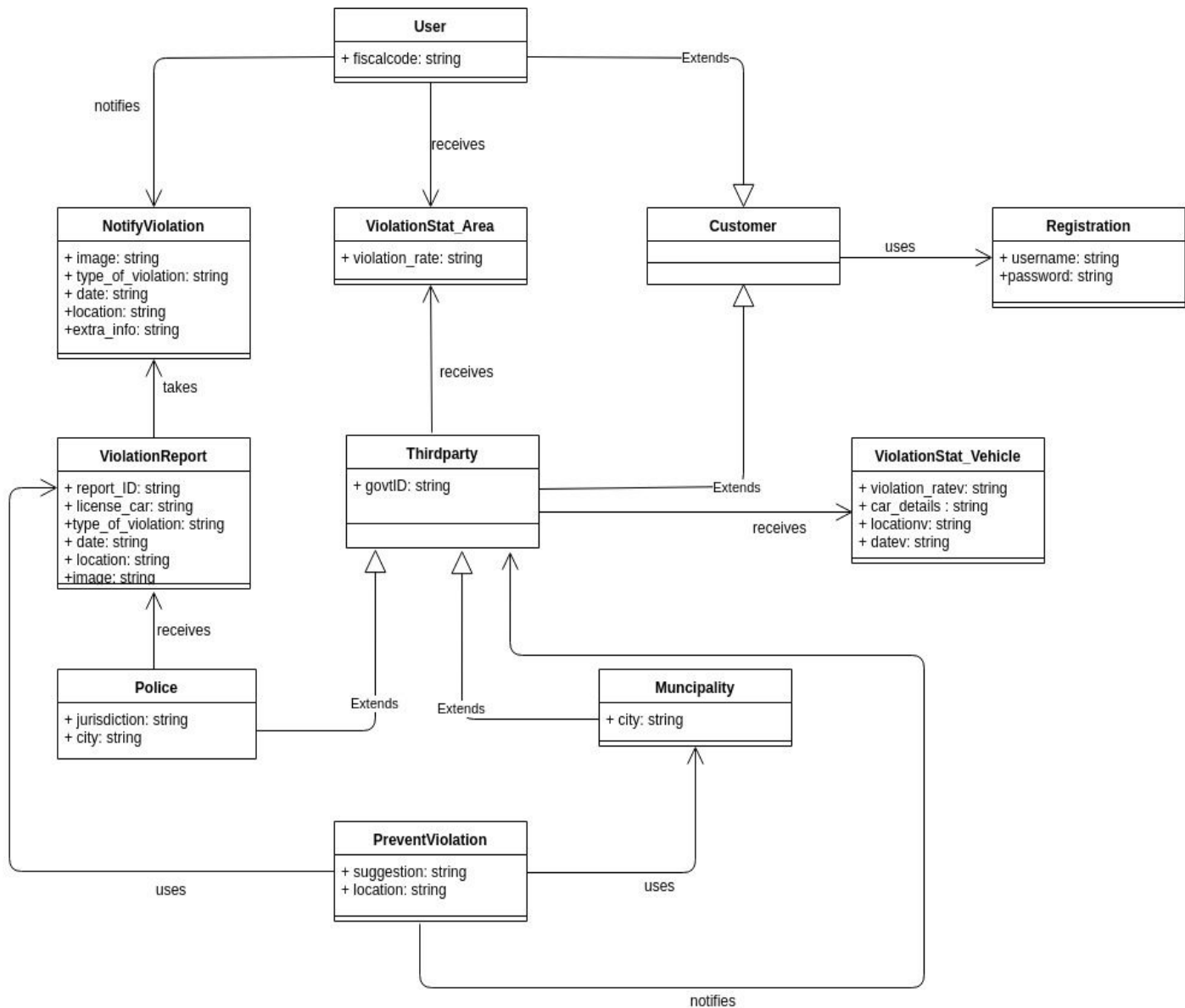


Figure 10: Class Diagram

It is to be noticed that the third party customer who is viewing the violation report can access the details of the user who reported the violation if necessary. The traffic violation reports will be forwarded only to the concerned third party like the police and not to all the third parties registered with the application. Both the third party and the user can check the cities with the highest violation rates.

2.6 Selected Architectural Styles and Patterns

There are different architectural styles and patterns employed in the designing of the application. The major styles and patterns are:

- **RESTful Architecture:** This architecture goes hand in hand with the scope of the application as it deals with many clients. The goal is to reduce the coupling between client and server components as much as possible in mind and also because the centralization of data plays an important role. All the communication exploit the HTTP protocol, using TLS when dealing with sensitive data in order to guarantee the security and the reliability of the connection. It is also used while authenticating the identity of the communicating parties. Regarding the format in which the data are transmitted, JSON is used because it is suitable for the data interchange between client and server.

By using a RESTful architecture leads to the adoption of the following constraints.

- **Uniform interface:** REST defines a set of well-defined operations that can be executed on a resource. One of the key concepts of REST and the HTTP protocol are resources. The HTTP protocol defines a set of operations that can be executed on a single or multiple resources like GET, POST, PUT, DELETE, PATCH and HEAD.
- **Client-Server:** A client-server architecture allows a clear separation of concerns. The client is responsible for requesting and displaying the data while the server is taking care of data storage and application logic. One advantage is that both sides can be developed separately as long as the agreed-upon request format is followed. It enables us to update the server on a regular basis without changing the customer software.

- **Caching:** Stateless client-server communication can increase server load since some information may have to be transferred several times so requests that only retrieve data should be cache-able.
- **Statelessness:** Communication between client and server is stateless. This means that every client request contains all the information necessary for the server to process the request. This further reduces server complexity since no global state is necessary and improves scalability since any request can be processed by any server.
- **Layered system:** It is a key feature of most networked systems. In the context of REST, this means that a client can not necessarily tell if it is directly communicating with the server or an intermediate (proxy). This allows to have a more flexible architecture and also to handle the security of the system, by inserting firewalls and proxies.
- **Three-tier client-server:** As mentioned in the deployment view section a three-tier client-server architecture is used in which an entire application is distributed across three different computing layers or tiers. It divides the presentation, application logic and data processing layers across client and server devices. This architecture provides many benefits for production and development environments by modularizing the user interface, business logic, and data storage layers. Doing so gives greater flexibility to development teams by allowing them to update a specific part of an application independently of the other parts. This added flexibility can improve overall time-to-market and decrease development cycle times by giving development teams the ability to replace or upgrade independent tiers without affecting the other parts of the system. For example, the user interface of a web application could be redeveloped or modernized without affecting the underlying functional business and data access logic underneath. This architectural system is often ideal for embedding and integrating 3rd party software into an existing application.

2.7 Other design decisions

Relational Database

The relational model means that the logical data structures — the data tables, views, and indexes, are separate from the physical storage structures. This separation means that database administrators can manage physical data storage without affecting access to that data as a logical structure. For example, renaming a database file does not rename the tables stored within it. The relational model is the best at maintaining data consistency across applications and database copies. As explained in the deployment view section our application handles a lot of querying, database transactions etc for which the ACID properties of the relational model becomes of great use.

3. User Interface Design

The mockups of the application were already exposed in the RASD document in section 3.1.1. Here we include some UX diagrams to show the flow in which the customer (user and third party) will navigate inside the application, in accordance with the mockups provided in the RASD.

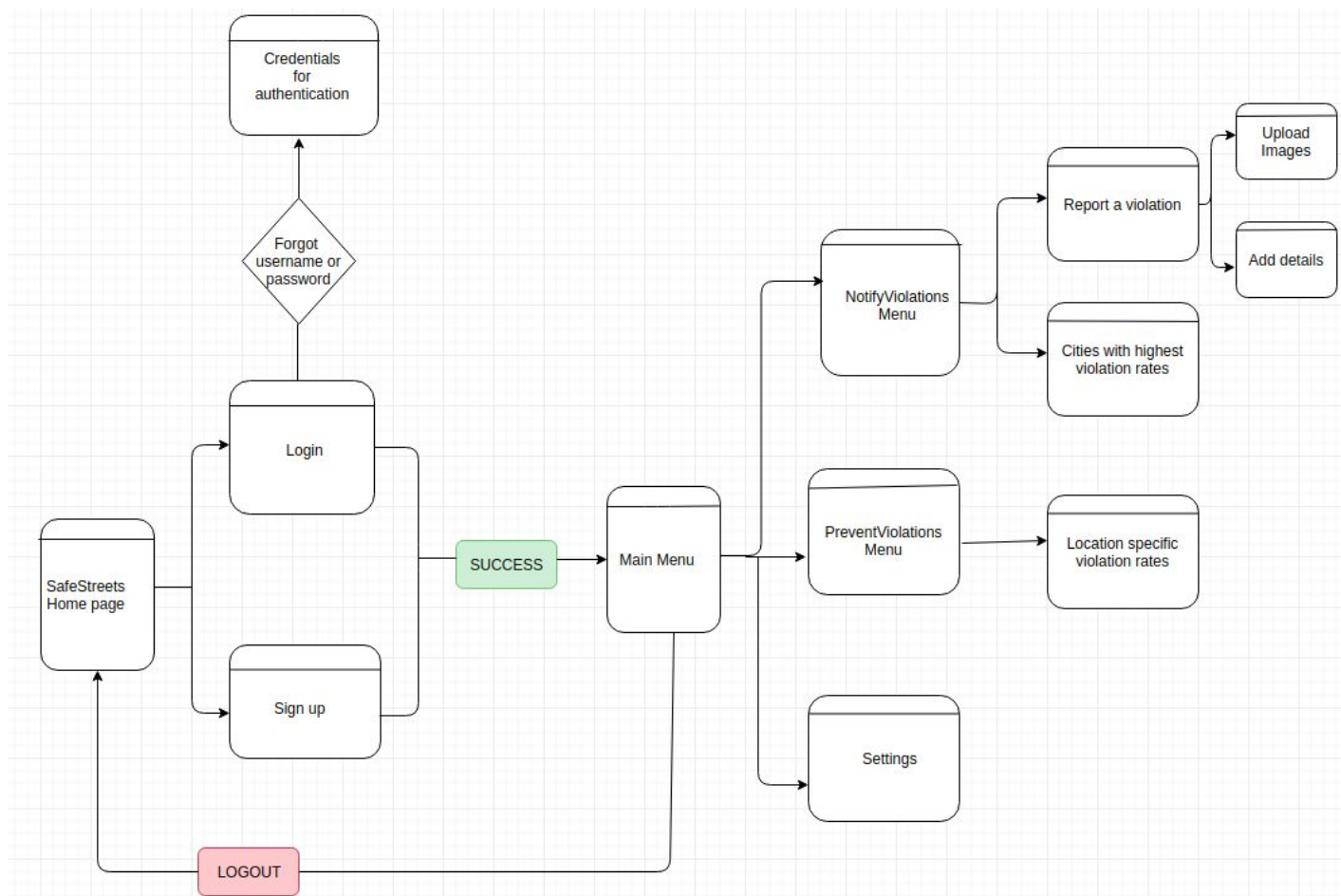


Figure 11: UX Diagram - User

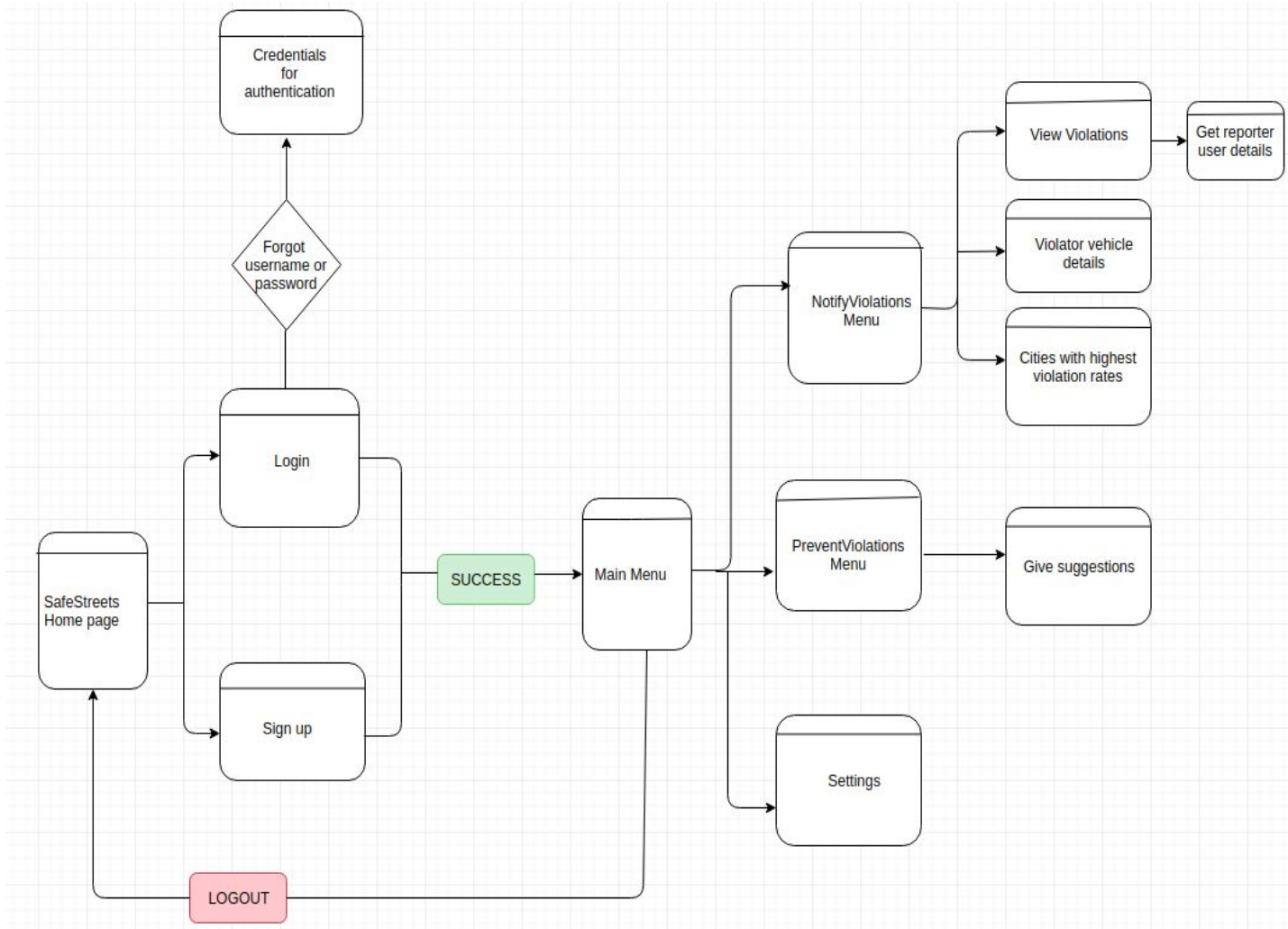


Figure 12: UX - Diagram Third Party

A few mockups that were not included in the RASD are being included here for the better understanding of the user interface designs.

John Doe

Fiscal Code:

PRK6749GTY8765

Address :

Via Rubattino, 65

Email :

johndoe@example.com

Past Reportings :

5

Images :




Figure 13: Mockups 1: Third-party screen while viewing details of violation reporter

EM 502LA

Owner :

Luca

License Number :

EM 502LA

Color :

Black

Model :

SUV

Past Violations :

5

Images :




Figure 14: Mockups 3: Third-party screen while viewing a violator vehicle's details

4. Requirements Traceability

The application is designed by ensuring that it will guarantee that all the requirements mentioned in the RASD will be enforced. This section provided a mapping between each requirement and the necessary design components used to fulfill the particular requirement.

- **R1:** The application stores the details of the violation report
 - **DataCollectionManager:** This component manages to save the data in the database.
- **R2:** The application has collected the details of the user at the time of registration
 - **DataCollectionManager:** This component manages to save the data in the database.
- **R3:** The application notifies the third parties who are concerned authorities like the local police.
 - **ViolationAlertManager:** This component receives the violation report from the user's application along with the result from the RunAlgorithmManager (the license plate number) and forwards it to the concerned third party.
 - **ViewAlertManager:** This component allows third parties to view the violation reports made by the users.
- **R4:** The application must allow the third party to access the contact details and identity documents of the user if needed.
 - **DataCollectionManager:** This component manages to save the data in the database.
- **R5:** The application runs the algorithm to retrieve the license plate number from the images uploaded by the user.

- **RunAlgorithmManager:** This component runs the algorithm on the images uploaded by the user and the result is passed on to the ViolationAlertManager along with other details that the user had given on reporting the violation.
- **R6:** The retrieved license plate number is sent as part of the violation report.
 - **ViolationAlertManager:** This component receives the violation report from the user's application along with the result from the RunAlgorithmManager (the license plate number) and forwards it to the concerned third party.
- **R7:** The application must provide the details of vehicles that have committed traffic violations.
 - **DataCollectionManager:** This component manages provides with the required data from the database.
- **R8:** The application must provide both third parties and users with the statistics of cities with the highest violation rates.
 - **AreaViolationStatsManager:** It provides the users and the third parties with the stats of the cities with the highest violation rates. The AreaViolationStatsManager uses the DataCollectionManager to get the data from the database.
- **R9:** The application must allow the users who have enabled the PreventViolation feature to get risk rates of a particular location upon request.
 - **LocationSpecificStatsManager:** This component provided the violation rates of the location that the user specifies. The AreaViolationStatsManager uses the DataCollectionManager to get the data from the database.
- **R10:** SafeStreet should cross its data with the information provided by the authorities about the accidents that occur on the territory of the municipality to give potential unsafe areas.

- **MunicipalDB:** This component provides services using which the application can retrieve violation data from third parties like municipalities.
- **SuggestionsManager:** This component verifies the data retrieved from third parties with that of the data available with the application to analyze the traffic violations happening at various locations.

It is worth noticing that all the requirements need the UserLoginAndSignupManager as all the features of the application are available only upon login. In order to be able to login to the application UserLoginAndSignupManager is required which will help the customers to register to the application.

Requirements	Components
R1	DataCollectionManager
R2	DataCollectionManager
R3	ViolationAlertManager, ViewAlertManager
R4	DataCollectionManager
R5	RunAlgorithmManager
R6	ViolationAlertManager
R7	DataCollectionManager
R8	AreaViolationStatsManager, DataCollectionManager

R9	LocationSpecificStatsManager, DataCollectionManager
R10	MunicipalDB, SuggestionsManager

5. Implementation, Integration and Test Plan

The application is divided into various subsystems:

- UserMobileApp
- ThirdPartyWebApp
- WebServerWebApp
- ApplicationServer
- External systems: DBMS, Google Maps

A bottom-up strategy is enforced to implement, test and integrate these subsystems. The components necessary for building the subsystems will be tested for each of the subsystems though it may have common components. It is worth to notice that the external systems' components need not to be implemented and tested just because they are external and they can be considered reliable. The table given below lists the major features of the application along with the importance of it for the customer and the difficulty in implementing them.

Feature	Importance to the customer	Difficulty in implementation
Signup and Login	Low	Low
Visualize personal data	High	Medium
Report a violation	High	High
Run algorithm to find the license plate number	Medium	High

View violation reports and access reporter's details	High	Medium
Get violation rates of cities	High	Medium
Get location-specific violation rates (on subscribing to PreventViolations)	High	High
Get violator vehicle details (Third Party)	High	Medium
Cross Verify Data with that offered by third party services	Medium	High

It's important to understand that the basic features of SafeStreet, which is that of NotifyViolations, has to be implemented, tested and integrated first. The other services like that of PreventViolations are available only on subscription. The table maps the features on their relevance and importance to the customer and the overall complexity. The components have to be implemented and tested (unit tests have always to be performed for each component in parallel with its implementation) in the following order to ensure the bottom-up strategy.

- **Report a violation:** For this feature, it is necessary to implement and test the components such as 'DataCollectionManager' and the 'ViolationAlertManager'. The 'DataCollectionManager' is the one that saves the data in the database and ViolationAlertManager is the one that forwards it to the concerned third party.
- **Run algorithm to find the license plate number:** The 'RunAlgorithmManager' component runs the algorithm on the images uploaded by the user and the result is passed on to the ViolationAlertManager along with other details that the user had given on reporting the violation.

- **View Violation reports and access reporter's details:** This feature requires the use of 'ViewAlertManager' and the 'DataCollectionManager'. The above features are to be integrated at this stage. It has to be tested whether the integration of the components has worked without errors.
- **Get violation rates of cities:** 'AreaViolationStatsManager' has to be implemented and tested at this point. This deals with the display of cities with that of highest violation rates. The AreaViolationStatsManager uses the DataCollectionManager to get the data from the database.
- **Get location-specific violation rates:** The LocationSpecificStatsManager is to be implemented, tested and integrated here as the customer needs location-specific violation rates. This component makes use of the DataCollectionManager to retrieve the data from the database. Both components need to be integrated and tested.
- **Get violator vehicle details:** This feature makes use of the DataCollectionManager to retrieve the data from the database. Though we test the DataCollectionManager earlier but it should again be tested while integrating here.
- **Cross Verify Data with that offered by third party services:** The MunicipalDB component has to be implemented and tested first. The SuggestionsManager is then implemented. It is to be tested here whether the cross verification of the available data with that of the data retrieved from third party services is integrated without errors.
- **Visualize personal data:** This is another feature of the system. It is the 'UserDataVisualizationManager' has to be fully implemented and tested. The data to be visualized comes from the database.
- **Sign up and log in:** The signup and login features are obviously an entry condition for the right functioning of the system, but they are not core features and they are not very complex. The UserLoginAndSignupManager can be implemented and tested in any order. It is an independent component.

It is important to start the verification and validation phases as soon as the development phase begins as it is necessary to rule out all the possible errors. As already mentioned testing should happen parallelly to the implementation. Unit test has to be performed on each of the individual components.

Whenever components are integrated, integration testing have to be done in an incremental way to facilitate the tracking of the bug. The components needed for the implementation of NotifyViolations has to be done before implementing the ones needed for PreventViolations. Once the system is completely integrated, it must be tested as a whole to verify that the functional and non-functional requirements work without errors. Load and stress testing and performance testing should also be done.

5.1 Component Integration

The following diagrams show which components will go through the process of integration for a further clarification. The arrows start from the component which uses the other one.

Integration of the frontend with the backend

Before integrating the frontend with the backend all the components used in the backend should be integrated and tested. After integrating the frontend and backend it has to be tested for all possible errors and performance issues.

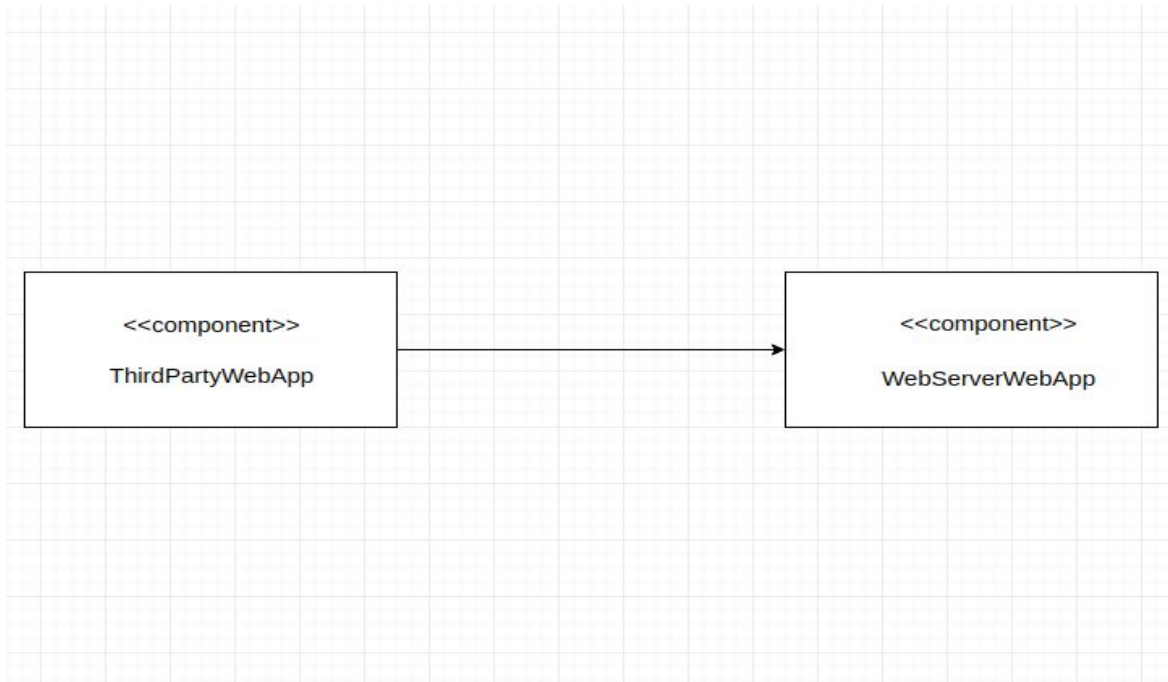


Figure 15: Third-Party Web App – WebServerWebApp integration

Integration with the external services

All the necessary components have to be integrated and tested before integrating external services.

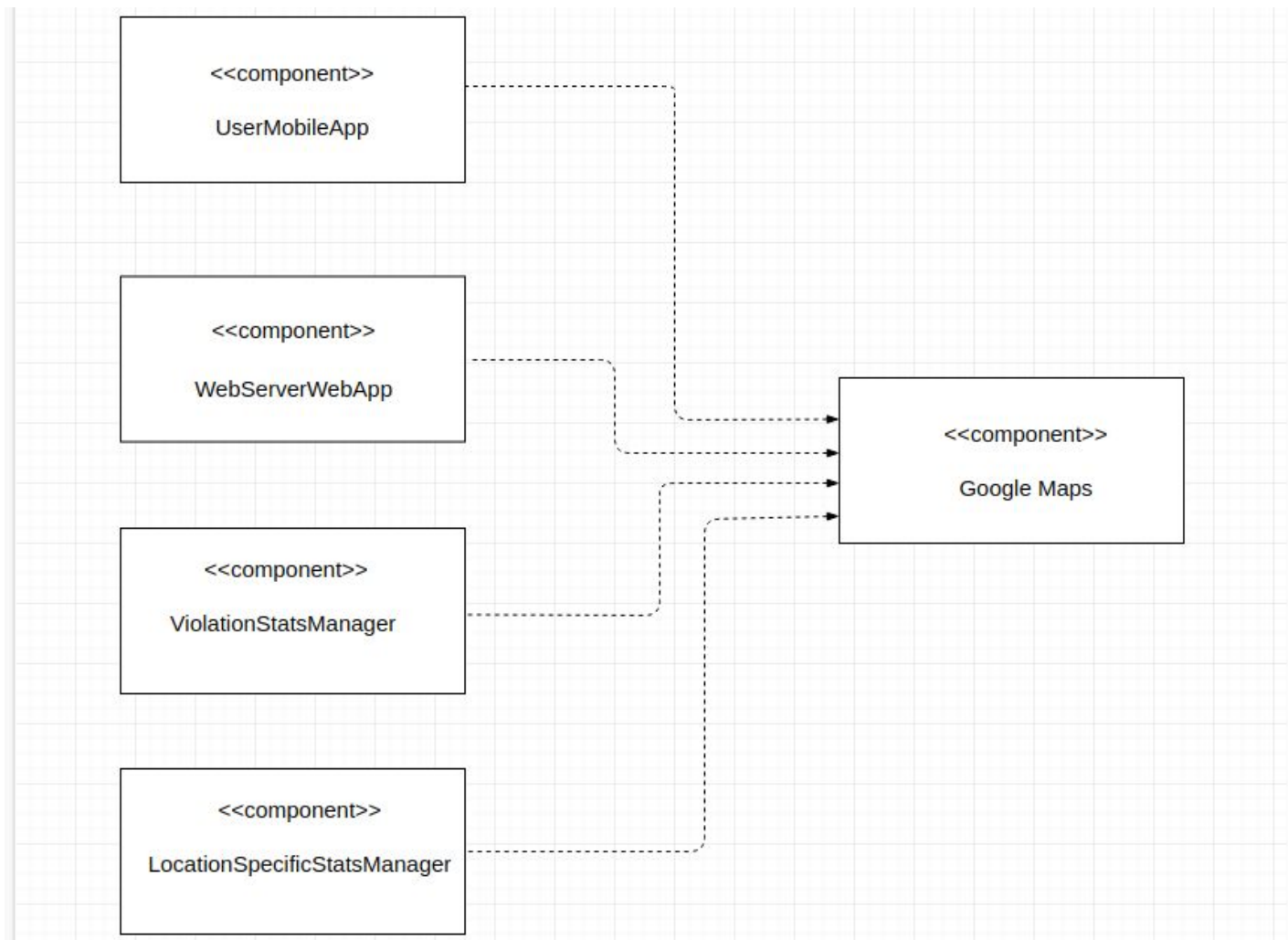


Figure 16: Google Maps Integration

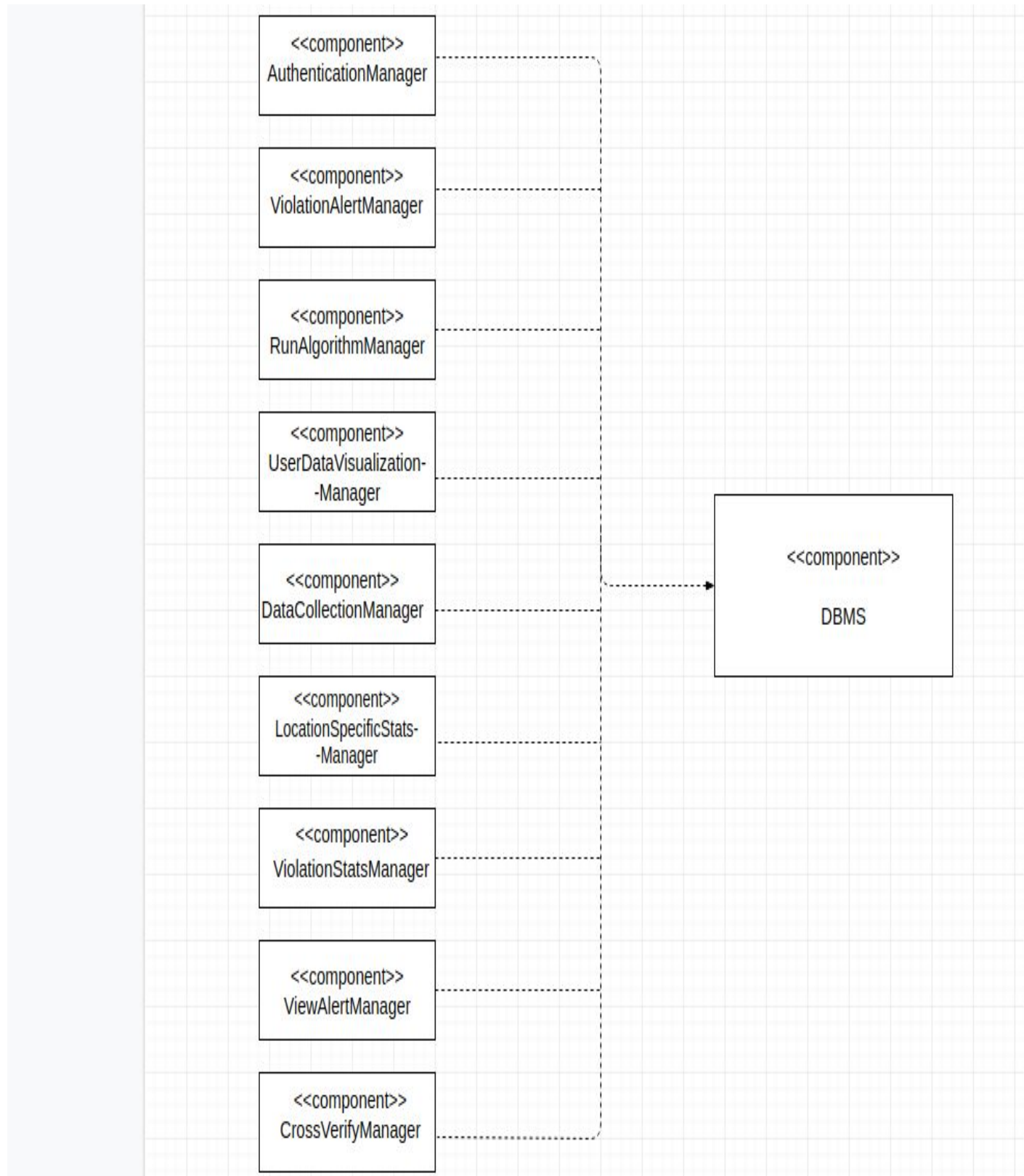


Figure 17: *DBMS Integration*

6. Effort Spent

Devi Radhakrishnan

Description of the task	Hours
Introduction	1
Architectural Design	49
Requirements Traceability	6
Implementation, Integration & Test Plans	6

Gayathri Prakash Menakath

Description of the task	Hours
Introduction	1
Architectural Design	45
Requirements Traceability	8
Implementation, Integration & Test Plans	9

