Politecnico

di Milano

2019/2020

POLITECNICO
MILANO 1863

DD – Design Document

Version 1.0 && 09/12/2019

Author:

938653 – Berk MORKOÇ

Professor:

Matteo Rossi

Table of Contents

1) Introduction

1.1. Purpose

The purpose of this document is to provide more systemic knowledge by describing technical terms and the technical aspects of the system. In other words, this document goes deeper into detail about the design, providing overall guidance to the architecture of the system whereas, the requirements document has abstractly the information of the system.

In particular, the following topics are touched in the document:

- The high-level architecture.
- The main components, their interfaces, and deployment.
- The runtime behavior.
- The design patterns.
- Additional diagrams and information about the user interface.
- A mapping of the requirements on the architecture's components.
- Implementation, integration and test plan.

1.2. Scope

The SafeStreets system has been mainly designed to allow the users to report the violations to the authorities. Therefore, the system easily guides the users on how to report the violation. Firstly, the system offers the user to take or upload the picture of the violation that he/she wants to report. If the user attempts to take the picture through the application, the system collects all the necessary information such as the location, date and time (GPS) and directly attaches these pieces of information to the picture, asks the user to confirm the truthiness of the collected data and transfers (notify) the violation to the authorities.

The system has also the ability to highlight the streets where the most violation occurs. The recorded violations are shown on the map with a different level of frequencies regarding the occurrences rate of the violations. This service can also be used by the authorities.

1.3. Definitions, acronyms, and abbreviations

1.3.1. Definitions

- **User:** The person who has the SafeStreets application and is able to report traffic violations.
- **Authorities:** Institution or institutions responsible for traffic safety.
- **Violation:** It refers to any breach made in traffic.
- **Crowd-source:** It is a sourcing model in which individuals or organizations obtain goods and services, including ideas and finances, from a large, relatively open and often rapidly-evolving group of internet users; it divides work between participants to achieve a cumulative result.
- **Library:** The source codes that have been prepared and merged in order to be used for software developments.
- **Illegal:** The operations or actions which do not follow the rule/rules determined by the government.
- **Illegal Parking:** Parking that does not follow rules.

1.3.2. Acronyms

- GPS = Global Positioning System
- DBMS = Database Management System
- UI = User Interface
- RASD = Requirement Analysis and Specification Document
- DD = Design Document
- DB = Database
- API = Application Programming Interface
- MVC = Model-View-Controller

1.3.3. Abbreviations

- [Rn]: n-functional requirement

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 Requirements Specifications
- Alloy doc: *http://alloy.lcs.mit.edu/alloy/documentation/quickguide/seq.html*

1.6. Document Structure

The DD document consists of six parts (except for the appendix) as indicated below:

➢ **Chapter 1:** This is the first chapter of the document and it refers to the introduction. In this section, it is aimed to describe what information this document covers. Also, it is roughly given how the system works and it is mentioned some definitions by extending definitions described in the RASD in order to provide a better understanding.

➢ **Chapter 2:** This chapter is the main chapter of the DD document. It basically contains the descriptions of the components and their relations, dependencies, interfaces as well as their diagrams separately and/or undividedly. In particular, this chapter has been divided into several sub-chapters as follows;

  - Overview: High-level components and their interaction
  - Component view
  - Deployment view
  - Runtime view
  - Component interfaces
  - Selected architectural styles and patterns

- Other design decisions

- **Chapter 3:** In this chapter, the requirements traceability has been provided. In other words, the requirements given in the RASD document are addressed to the components in order to satisfy the goals of the system.
- **Chapter 4:** This section has the information on implementation, integration, and test plan about the SafeStreets system. All components' implementation, integration and test plan are tackled in this chapter.
- **Chapter 5:** This chapter shows the effort spent to reveal the project.
- **Chapter 6:** This is the last chapter which refers to the references of the document.

2) Architectural Design

2.1. Overview: High-level components and their interaction

The SafeStreet system has been divided into three-tier in order to provide more flexibility, readability, and consistency for both the users and developers. In this layer-based architecture, there are 3 tiers (*Figure 1*) which are presentation, logic and data tier as it is mentioned earlier.

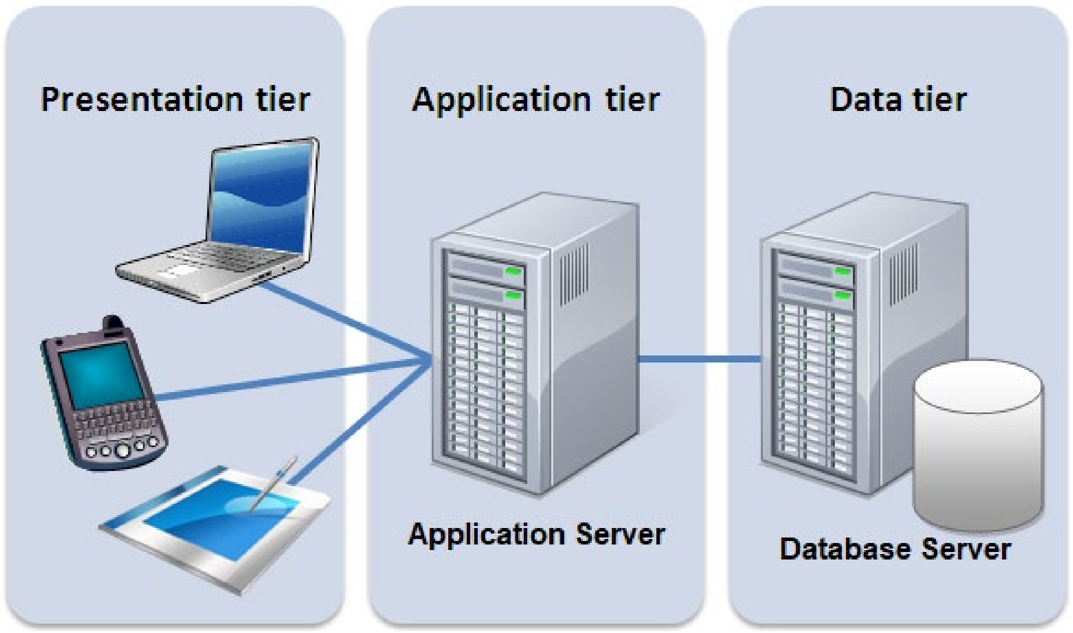


Figure 1 - Three-tier architecture

More precisely;

- **Presentation Tier:** This is the topmost level of the application and it basically includes the UI. The main function of this tier is to translate tasks and results in something that the user can understand. In simple terms, it is a layer which users can access directly.
- **Logic Tier:** This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.
- **Data Tier:** Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.

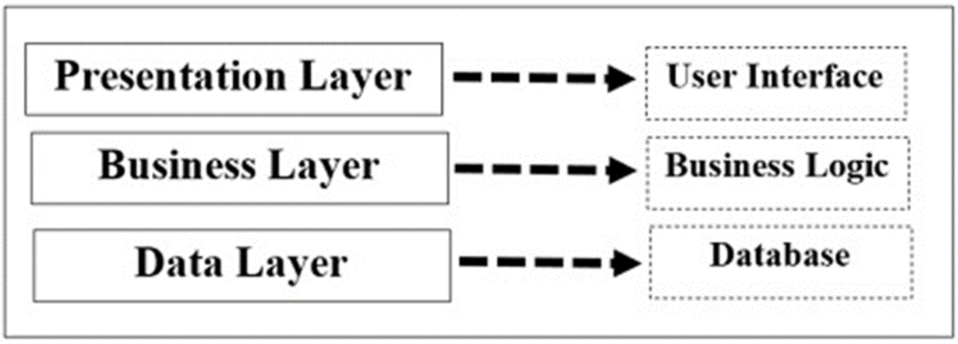


Figure 2 - Layers correspondences

In the data tier, the SafeStreet system has aimed to provide an API to the application tier that exposes methods of managing the stored data without exposing or creating dependencies on the data storage mechanisms. Avoiding dependencies on the storage mechanisms allows for updates or changes without the application tier clients being affected by or even aware of the change. Thus, the system is going to service the users more effectively. The system could be divided into another extra tier which is called entity tier (the tier that includes the main classes of the application.). However, as with the separation of any tier, there are costs for implementation and often costs to performance in exchange for improved scalability and maintainability. That's why restriction the number of tiers three is thought to be adequate.

2.2. Component view

The figure below (*Figure 3*) shows the components and their interactions. The figure contains only the components in the application layer in detail. However, other components belong to the presentation and data tier are provided just to present their interactions to the application tier. Nevertheless, the description of the other components which are not located in the application server will be given in order to provide a complete frame of the system.

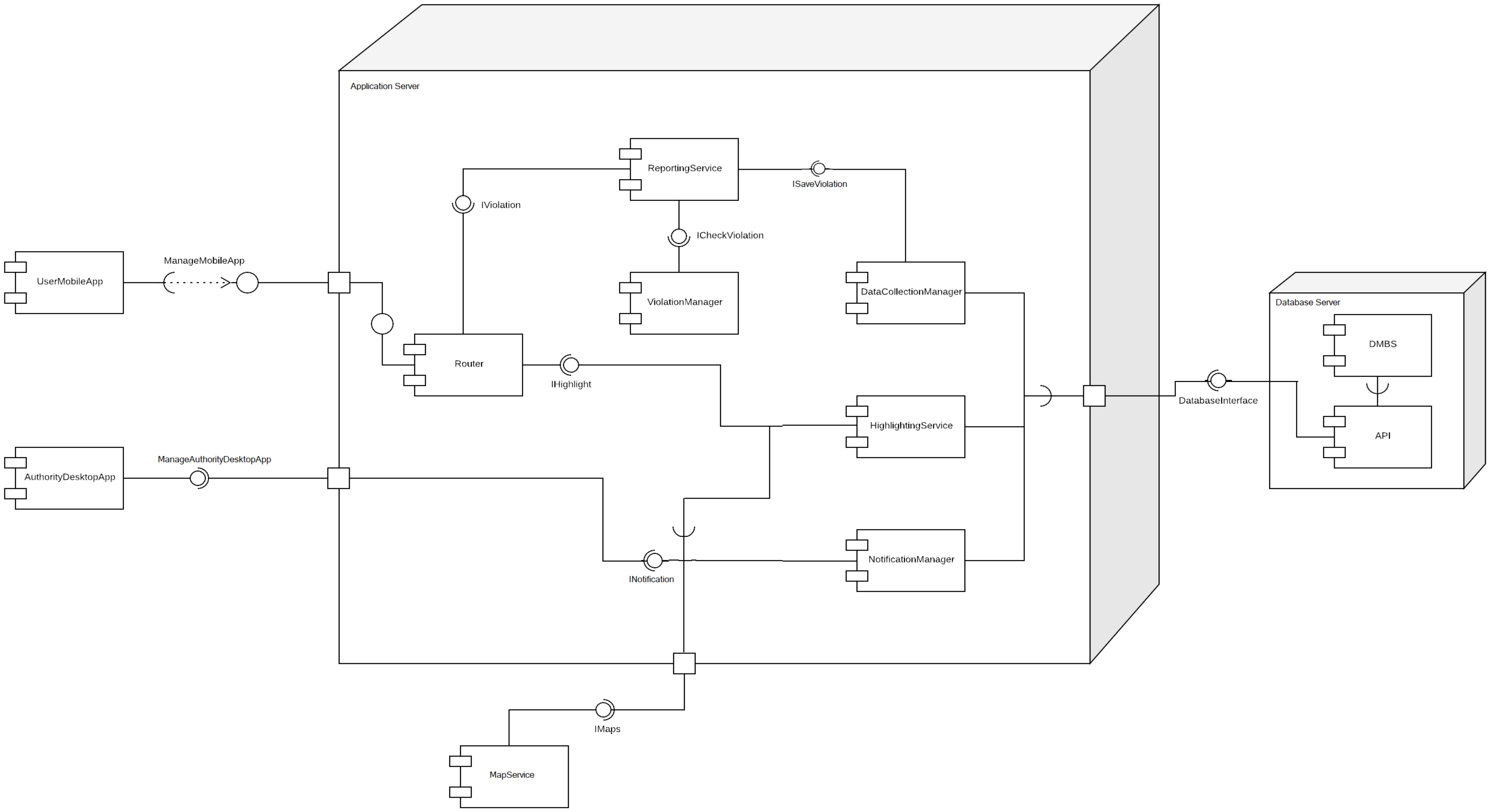


Figure 3 - Component Diagram

The description of the components (presentation tier (Client), application server, and database server, respectively) given in Figure 3 are as follows;

- **UserMobileApp:** This is one of the main components of the presentation tier. It makes sense of the operations processed in the application tier and data tier in a way that the users can understand. In other words, it contains the user interfaces that have already been provided in the previous document, RASD.
- **AuthorityDesktopApp:** This component is exactly the same as the UserMobileApp component. However, it contains the interfaces provided for the authorities, instead. Those interfaces are also given in the previous document.
- **Router:** It is the component in charge of directing the requests coming from the presentation tier to the necessary component. Since the only one component was identified in the component diagram for the AuthorityDesktopApp, the Router component has not been used for the AuthorityDesktopApp component.
- **ReportingService:** This component is actually the main component of the SafeStreet system. It directly answers the necessities and the main purpose of the system. The function of this component is to provide the users to reporting service. More precisely, it is the component that is responsible for the chain of the reporting processes. It also provides an interface to the ViolationManager component in order to make the ViolationManager component do its' job. Additionally, the ReportingService component transfers the data to the DataCollectionManager.
- **ViolationManager:** It is the component which is in a relationship with the ReportingService component. The main purpose of this component is to check whether the same violation has already been reported and recorded by the ReportingService component.
- **HighlightingService:** This component provides users to see the frequencies of the violations by highlighting the streets where the violations occur. In other words, it collects the recorded violations and transfers this information to the presentation tier. However, it gives a complete frame by taking support of the external component of the system which is MapService.

- **DataCollectionManager:** This component is responsible to collect the provided data that belongs to the violation and send these to the database server. These processes are done in real-time.
- **NotificationManager:** This component reveals the main functions of the AuthorityDesktopApp by providing a service in order to notify the authorities about the violation sent.
- **MapService:** This component is the only external component of the system. It basically supports the system by presenting the mapping service. The main function of this component is to provide a map that the users are going to be able to see the frequencies and locations of the violations on the map.
- **DMBS:** This is one of the most essential components in the system. It provides the management of the data collected and going to be provided.
- **API:** This is the component provided by the DMBS component. More precisely, DBMS gets and posts the data through this component. Additionally, this component supplies managing the stored data without exposing or creating dependencies on the data storage mechanisms. Therefore, the end-users will not be affected by the updates and changes as far as possible.

2.3. Deployment view

The deployment diagram is provided *(Figure 4)* in order to show the execution architecture of the SafeStreets system, including nodes as hardware and software. Therefore, it is aimed to put across how the system will be physically deployed on the hardware.

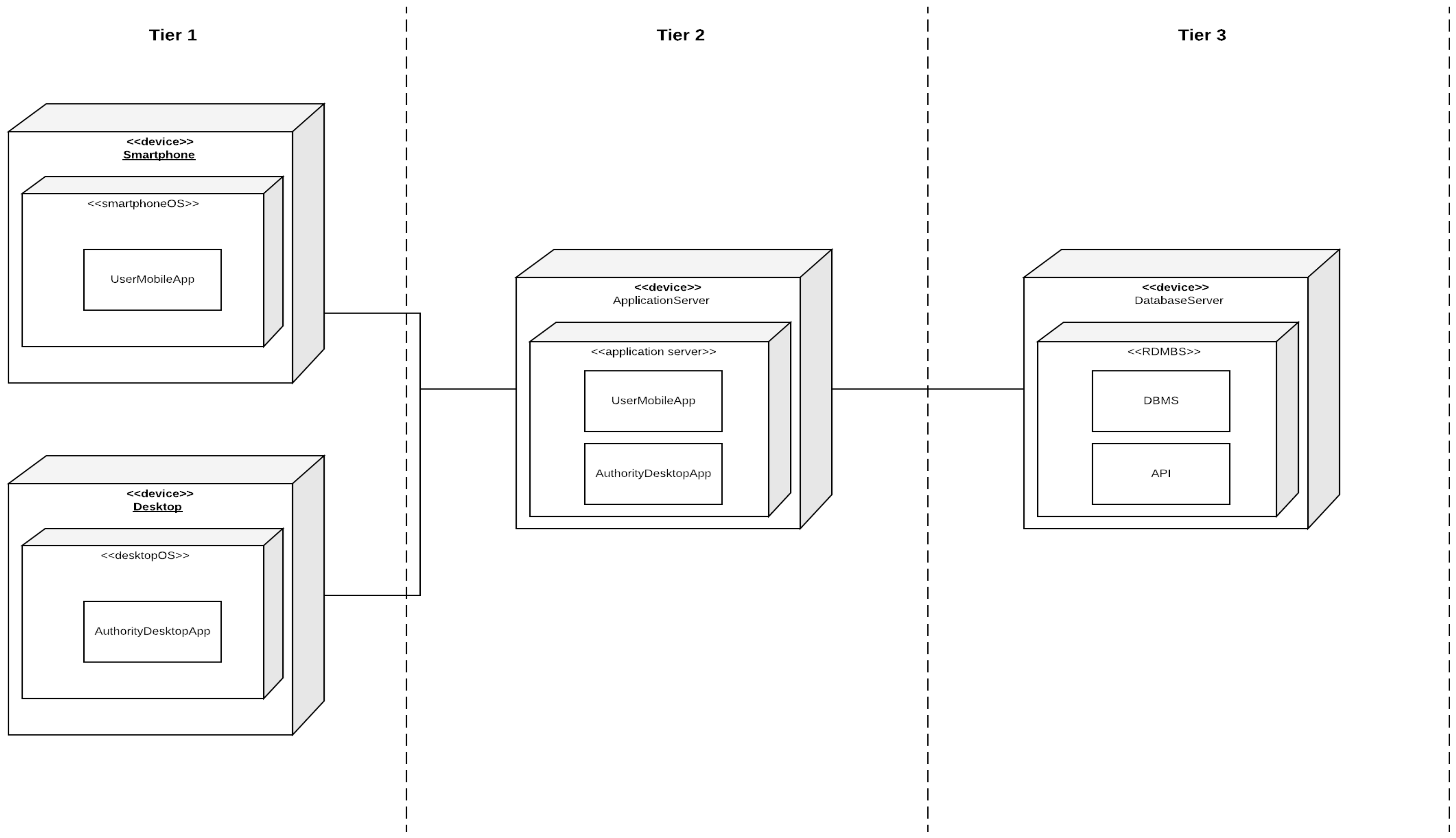


Figure 4 - Deployment Diagram

Figure 4 can be described in-depth as follows;

- **Tier 1:** This tier refers to the presentation layer that directly interactives with the users. Explicitly, the users and authority must be provided an application that they can access to the system. For that reason, users are going to be provided a mobile application and the authority with a desktop application. The mobile application is the most suitable choice to let the users access to the services provided by the SafeStreets system. The mobile application is going to support the operating systems of Android and ios. On the other hand, the reason why making the choice of desktop application for the authority is that they are most likely an institution and they usually manage their works on computers. The application provided for the authority is going to be able to run on both the operating systems of windows and mac.
- **Tier 2:** This tier corresponds to the application server. The application server implements all the business logic, handles all the requests and provide the appropriate answers for all the offered services. It is directly addressed to the user mobile application and authority desktop application in order to answer the requests coming from the end-users. It is also in relation to the external components.
- **Tier 3:** This tier corresponds to the storage (Database server) of the SafeStreets system. It includes DBMS and its' dependencies. There is existentially identified an artifact API that manages the data flow in the system.

2.4. Runtime view

2.4.1. Reporting service and authority notification

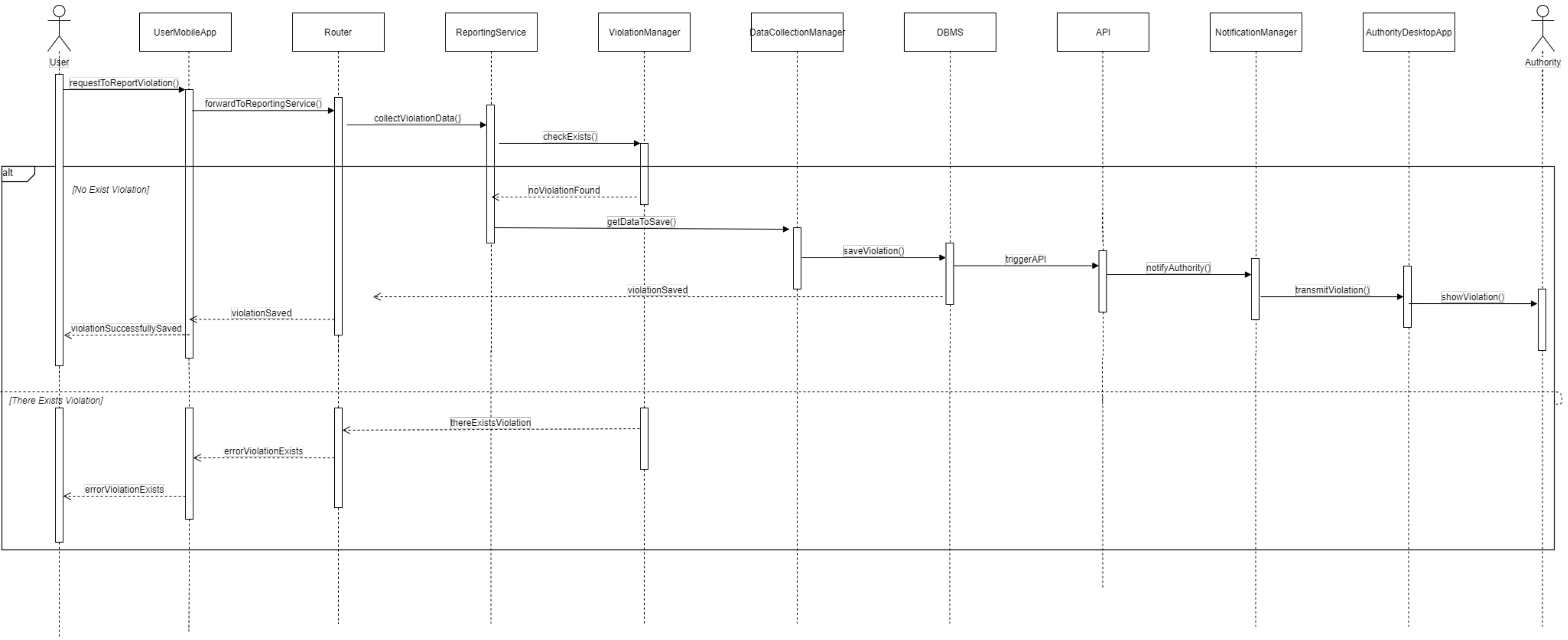


Figure 5 - Sequence diagram of reporting and notifying services

Figure 5 shows the sequence of reporting a violation and notifying authority. First, the user access to the application and starts the chain of reporting the violation. The router forwards the requested operation to the related component which is ReportingService. Afterward, the component of ReportingService forwards the collected data to the ViolationManager. The ViolationManager checks whether if the same violation has been already recorded and forwards the confirmation to the ReportingService component if the violation that the user wants to report does not exist in the database. Otherwise, related error that indicates the state is directed to the router in order to notify the user about the circumstance. In other words, the reporting process is canceled. After the confirmation sent to the ReportingService component, the information of the violation is transferred to the DataCollectionManager in order to store the violation and notify the authority about the violation. Once, the violation's saved to the DBMS, the chain of the authority notification begins. The API component triggers the NotificationManager component to inform the authority. Eventually, the violation and its' collected data are sent to the AuthorityDesktopApplication. Thus, the authority is notified.

2.4.2. Highlighting service

Figure 6 below shows the sequence of the highlighting service. The user accesses the application in order to see the violation map and implicitly starts the chain of the highlighting service. The router component forwards the requested action to the related component which is HighlightingService. The HighlightingService component gets the recorded violations from the database and also send a request to the MapService component in order to start the mapping service. Although fetching the violations from the database is quicker than starting the mapping service, the processes are waited to be directed to the application until both actions revealed and accessed to the Router component. Finally, the results are forwarded to the application through the Router component and violations with frequencies are showed on the map.

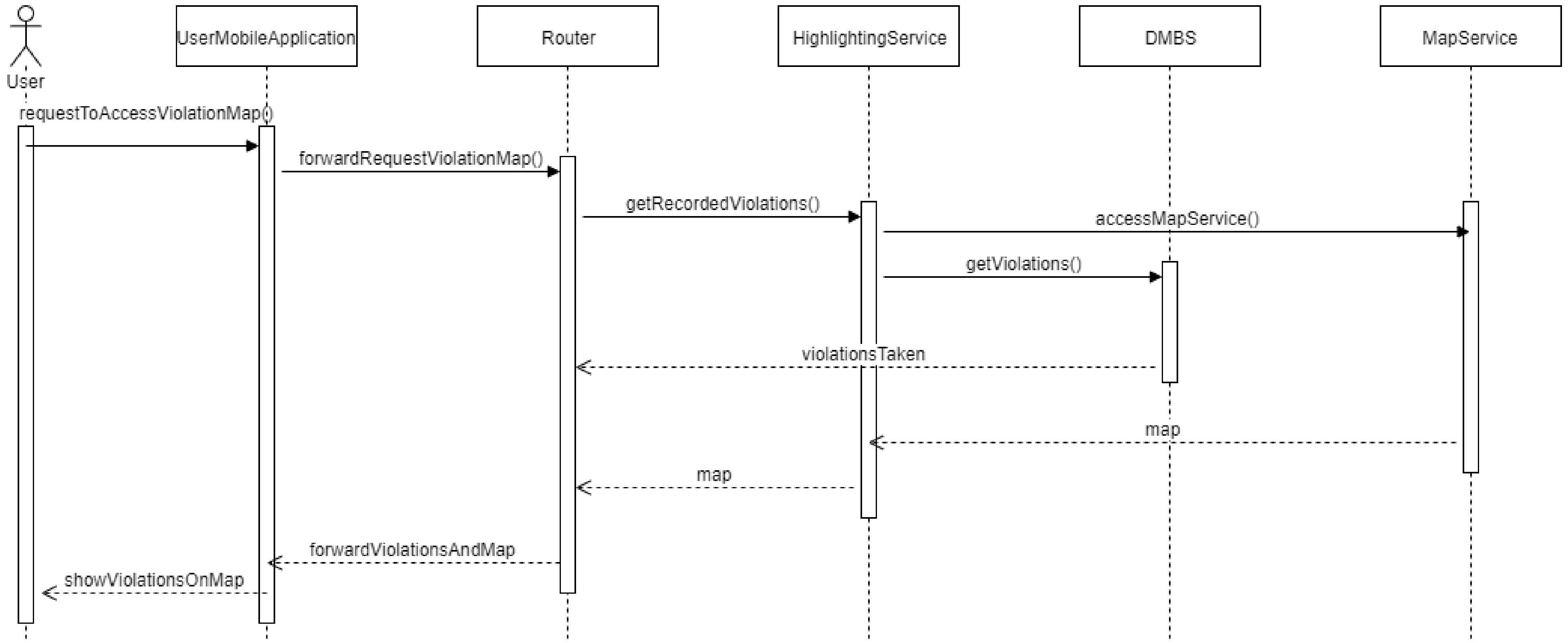
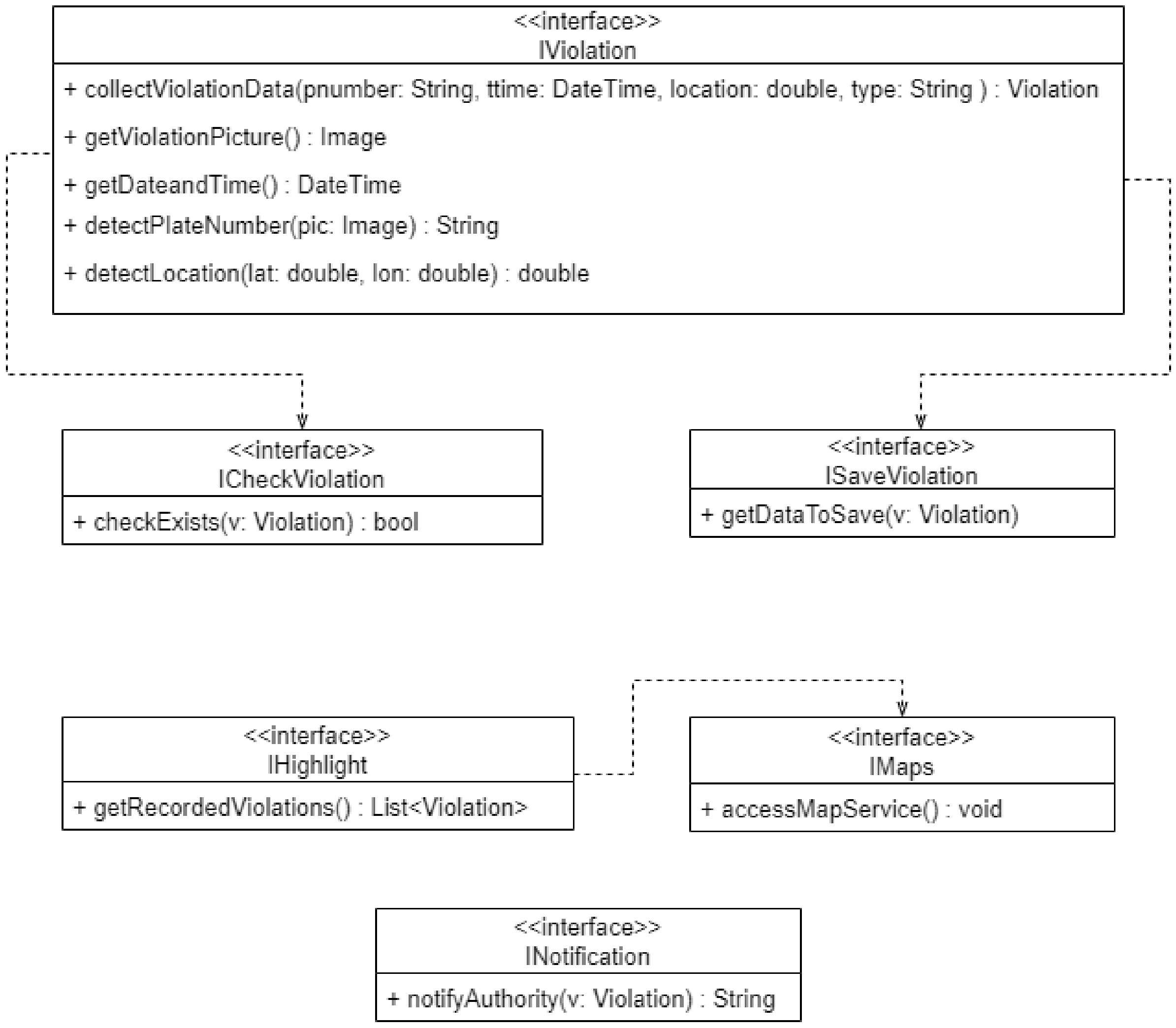


Figure 6 - Sequence diagram of highlighting service

2.5. Component interfaces



<<interface>>
IViolation
+ collectViolationData(pnumber: String, ttime: DateTime, location: double, type: String) : Violation
+ getViolationPicture() : Image
+ getDateandTime() : DateTime
+ detectPlateNumber(pic: Image) : String
+ detectLocation(lat: double, lon: double) : double
<<interface>>
ICheckViolation
+ checkExists(v: Violation) : bool
<<interface>>
ISaveViolation
+ getDataToSave(v: Violation)
<<interface>>
IHighlight
+ getRecordedViolations() : List<Violation>
<<interface>>
IMaps
+ accessMapService() : void
<<interface>>
INotification
+ notifyAuthority(v: Violation) : String

Figure 7 - Component interfaces

In the diagram *(Figure 7)* above, the component interfaces are presented and the dependencies between the parts of the application server are shown. The diagram includes the interfaces and its' functions/methods. IViolation interface is responsible to collect and complete the necessary information related to the violation. That's why it has the main functions that process the violation reporting chain. On the other hand, the other most important service that the SafeStreets system provides is the highlighting service which occurs in the diagram as IHighlight. It gets the recorded violations from the database and triggers the map service called IMaps.

2.6. Selected architectural styles and patterns

RESTFUL architecture style

The reason why choosing RESTFUL architecture is the most suitable architecture style since the SafeStreets system is aimed to provide a smooth service to the users. It is also able to reduce the shuttling between the client and server by not carrying extra headers of the data. In other words, it is a stateless architecture. On the other hand, the RESTFUL architecture provides us to make changes and updates without affecting the client-side application. Last but not least, since the SafeStreets system deals with internal and external APIs, it fits the requirements of the system such as publishing an internal API to the authority desktop application and getting the results in the JSON format in order to commit it to post the reported violations to the authorities through the desktop application. More precisely, the violations are going to be returned in JSON format by the API, then the data that includes the violations in JSON format is going to be commented and brought a type that the authority can understand.

Three Tier architectural pattern

A multilayered architecture is a client-server architecture in which presentation, application, and data access functions are physically separated.

It is chosen the multitier architecture because it allows decoupling the complexity of the system, making it more flexible and reusable. Indeed, the developers acquire the power of modifying or adding specific layers without disrupting the entire application. Whereas separating the system into sub-systems gets more flexibility to reveal the system, it also brings more cost for the implementation. That's why, the SafeStreets system consists of three main tiers which presentation, application and data tier.

2.7. Other design decisions

The SafeStreets system offers a highlighting service in which the users are able to see the violations on the map. That's why the system has to deal with external API provided by the map provider which is going to be implemented and integrated GoogleMaps. GoogleMaps is the most suitable map provider as it also helps with location detection.

3) Requirements Traceability

The design of SafeStreets system aims to meet all the goals and requirements that have been previously specified in RASD. Below are listed the design components to which SafeStreets requirements and goals are mapped:

- ❖ **[G1]** Allow users to report traffic violations.
    - **[R1]** The system must save violations in real-time.
        - **DataCollectionManager:** This component manages the data insertion to the database and collection from the other components.
    - **[R2]** The system must be able to process the reporting chain all the time.
        - **ReportingService:** This component manages the collection of the violation from the users.
    - **[R3]** The system must have a DB that can sufficiently correspond to the needs to store the information provided by the users.
        - **DataCollectionManager:** This component is responsible for all the necessities about the violation data coming from the users.
    - **[R4]** The system must be able to prevent the submission of the same violation more than one time.
        - **ViolationManager:** This component checks whether there exists the same violation that the user wants to report once again.
    - **[R5]** The system must take the user back to the first step of the violation submission to make him/her provide a new picture if the license plate of the vehicle cannot be detected by the application.

        - **ReportingService:** This component includes all the necessary functions and methods in order to check whether the license plate number can be detectable from the picture provided by the user. If the detection cannot be done, then the request goes back to the initial step.

- ❖ **[G2]** Highlight the streets that the most violation occurs and provide this information to the users as well as the authorities.
    - **[R6]** The system must be able to show the violations in real-time. So, if a new violation is saved to the database, then it must be shown on the violation map real-timely.
        - **DataCollectionManager:** This component is responsible for the data related to the violations.
        - **HighlightingService:** This component is the main component that satisfies the given goal. The essential function of this component is to show the violations on the map for the users. It also triggers the MapService component in order to start the mapping service provided externally.
        - **MapService:** This component is externally provided in order to let the users access the map. This service is used through an external API.
- ❖ **[G3]** The system must be able to notify the authorities about the violation sent by the users.
    - **[R7]** The system must have an API service to provide and notify the authorities about the violations.
        - **NotificationManager:** This component is identified to notify the authority when a violation is saved to the database. (This component works with an internal API (the NotificationManager covers the API.) encoded by the SafeStreets system. Since the application which is going to be provided for the authorities is different from what is going to be provided for the users, the documentation only gives the interfaces of the authority application.)

4) Implementation, Integration and Test Plan

The system is going to be divided into sub-systems and bottom-up strategy will be followed during the improvement of the SafeStreets system. For that purpose, the system has been divided into sub-systems as follows;

- UserMobileApp
- AuthorityDesktopApp
- Application Server
- External systems: MapService (Google Maps)
- DBMS and API

Essential sub-systems as it's seen above. Those all sub-systems are going to be implemented, integrated and tested. The table that refers to the importance of the features (as services that the SafeStreets system provides) for the users and difficulties levels of implementation is provided below in order for a better understanding about the implementation, integration and test plan priorities of the sub-systems and their components.

Feature	**Importance for the users**	**Difficulty of implementation**
Reporting Service (including authority notification)	High	High
Highlighting Service	Medium	Medium

Since the reporting service is the most important service that the SafeStreets provides for the users to let them report the violations, it has to be implemented, integrated and tested at the first step. The other service which is highlighting relies on the reporting service since the violations are collected through the reporting service.

All components (the external component MapService is not going to be implemented since it has been availably provided by GoogleMaps.) have to be implemented and tested (unit tests have always to be performed for each component in parallel with its implementation) with the following order regarding the table given above.

Reporting Service: This feature is exactly the main the function and service of the system. There are several components and existing features that rely on this feature. Therefore, the components which bring out the reporting service must be implemented and unit-tested. So, those components *ReportingService* and *ViolationManager* are going to be implemented and unit-tested following the bottom-up approach. (The *ViolationManager* component has meaning once the *ReportingService* has been revealed. In other words, the *ReportingService* component uses the *ViolationManager* component in order to present its' function. Therefore, the *ViolationManager* component is implemented and unit-tested before the *ReportingService* component.). Then, *DataCollectionManager* must be partially implemented and unit-tested (*DataCollectionManager* is partially implemented and tested after the implementation of some components used by the *DataCollectionManager* by following the bottom-up approach.). Once the *DataCollectionManager* (partially), *ReportingService* and *ViolationManager* components are implemented and unit-tested, they have to be tested and integrated all together to understand the first chain of the reporting service has been successfully identified. In the second stage following the bottom-up approach, the *NotificationManager* has to be implemented and unit-tested. The *NotificationManager* component is not actually directly connected to the other reporting service components, however, it is indirectly connected since the violations are collected through the reporting service components (In other words, the *NotificationManager* component uses the components that we identified as the first stage.). That's why it has to be tested and integrated with the other existing components that create the reporting service.

Highlighting Service: The *HighlightingService* component is going to be implemented by following the bottom-up approach (*HighlightingService* component indirectly uses all the components that present the reporting service except *NotificationManager*.). In order to test this component, the external component *MapService* has to be integrated and tested and this is going to be mentioned in the integration chapter. To guarantee this feature, the *DataCollectionManager* has to be fully implemented and unit-tested (before the

HighlightingService component has been implemented and unit-tested) since the violations are collected from the database for this feature.

Finally, the *Router* component must be implemented and unit-tested then its' integration must be performed with the other components in the application server. The *Router* component only manages the users' requests and it is not heavily related to the business logic but, it is a fundamental component for the system to be worked correctly. For that reason, it is going to be implemented and tested at the end (In terms of the bottom-up approach, the *Router* component is used by all the existential components.).

The verification and validation of the systems' components must be done in order to find bugs and mistakes as soon as the components are revealed. Finding the bugs will be done in parallel to the implementation in order to detect the bugs if there exist. The components that build up the service are going to be implemented, then they are going to be integrated and finally the integration is going to be tested altogether. Eventually, the integrated components of the system are going to be tested to determine the functional and non-functional requirements are held.

4.1. Component integration

The integration of the components are provided by giving diagrams below for further clarification. The direction of the arrows show the component being used. (For example, the ReportingService component uses the ViolationManager component. So, the direction of the arrow will be from ReportingService to ViolationManager.)

4.1.1. Integration of application server components

All the components are implemented and unit-tested. However, some components are integrated and the integration is tested as well.

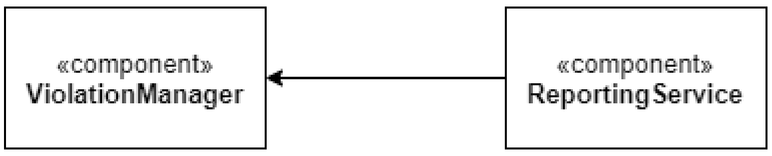


Figure 8 - ReportingService component integration

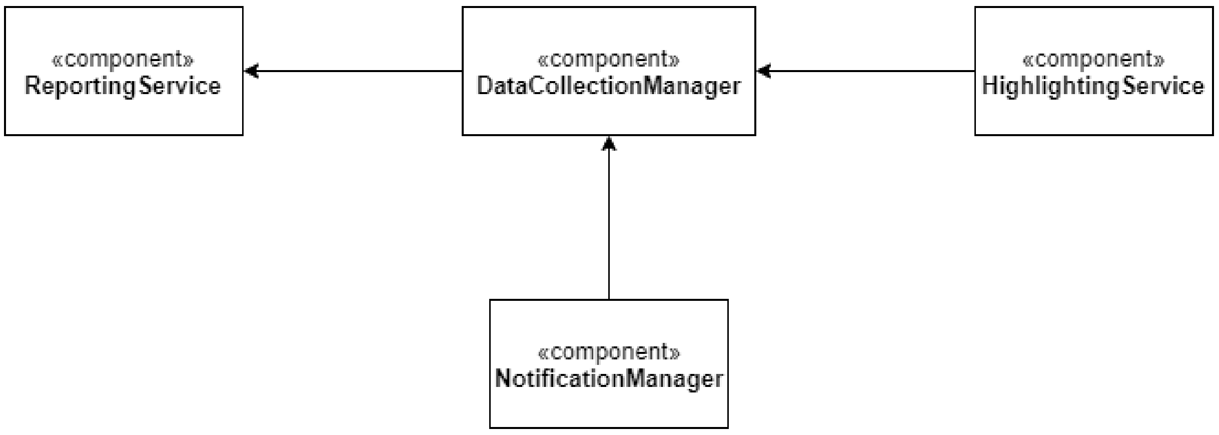


Figure 9 – DataCollectionManager, NotificationManager and HighlightingService integration

4.1.2. Integration of front-end and back-end

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

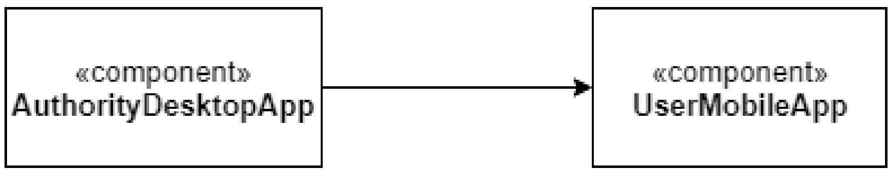


Figure 10 - AuthortityDesktopApp - UserMobileApp integration

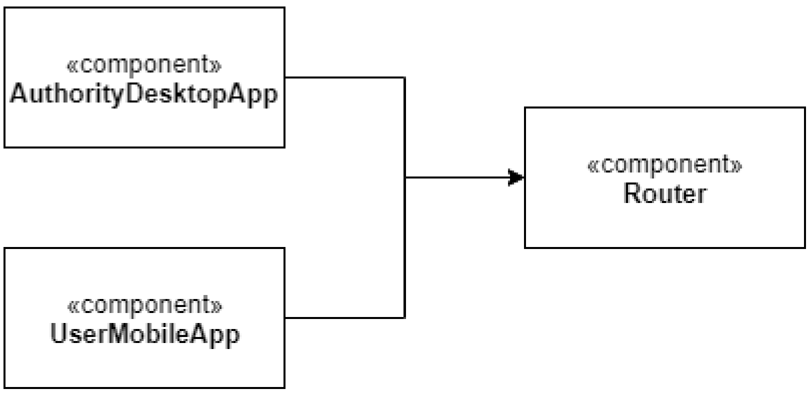


Figure 11 - Router integration

4.1.3. Integration of external services

The integration of external services is going to be done after all the components have been implemented and unit-tested (The integration and testing of the *MapService* component will be done before the *HighlightingService*).

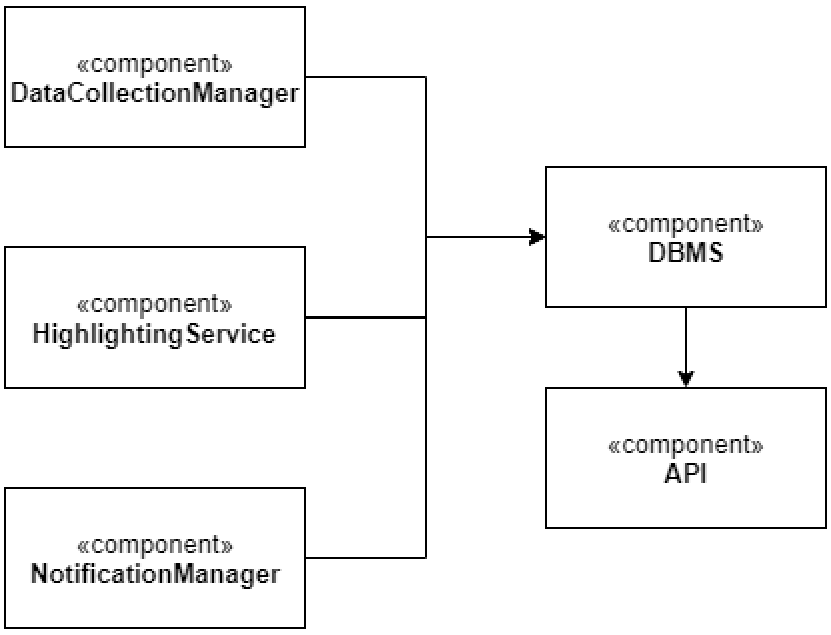


Figure 12 - DBMS and API integration

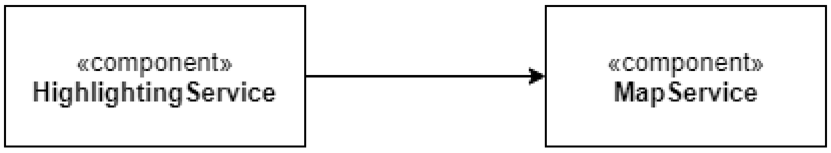


Figure 13 - HighlightingService - MapService integration

5) Effort Spent

Chapters of the Document	Hour(s) Spent
Introduction	3
Architectural Design	13
Requirements Traceability	4
Implementation, integration and test plan	8
TOTAL HOURS SPENT	**28**

6) References

1) https://en.wikipedia.org/wiki/Top-down_and_bottom-up_design
2) https://www.smartdraw.com/component-diagram/
3) https://herbertograca.com/2017/07/28/architectural-styles-vs-architectural-patterns-vs-design-patterns/