POLITECNICO
MILANO 1863

Academic Year 2019 / 2020

Safe Streets

Cappelletti Andrea, Maglione Sandro

Design Document (DD)

Deliverable:	DD
Title:	Design Document
Authors:	Cappelletti Andrea, Maglione Sandro
Version:	1.0
Date:	08 December 2019
Download page:	https://github.com/andreacappelletti97/CappellettiMaglione
Copyright:	Copyright © 2019, Cappelletti Andrea, Maglione Sandro – All rights reserved

x

Contents

1 Introduction

1.1 Purpose

The following document represents the Requirements Design Document (DD) for the service offered by the company SafeStreets. The Goals of the Design Document are:

- demonstrate a means to fulfill the requirements of the system-to-be;
- embody the essential characterstics of the software-to-be;
- serve as a basis for analysis, evaluation, integration, and testing;
- define constraints on the implementation;
- serve as a vehicle for stakeholder communication;
- guide the implementation of the software-to-be;

This document is intended for design stakeholders such as users, developers, software designers, and project managers. This document also provides an overall view of the chosen architecture.

1.2 Scope

SafeStreets is a totally new platform, so it needs to be built from scratch. SafeStreets is a service that allows Users to report violations to the municipality. Users provide a picture of the violation, the type of violation, they then confirm the license plate and the data is stored inside the System, which also adds the location of the report from the GPS of the User device. Users and Authorities can then consult those data by sending requests from the application. Authorities have access to a real-time notification service and they can also request specific recommendations about possible interventions, based on the information stored inside the System.
Some data about accidents is also integrated from the municipality, and the interface between SafeStreets and the municipality is also within the scope of the System.

1.3 Definitions, Acronyms, Abbreviations

- **API:** Application Program Interface;
- **ML:** Machine Learning;
- **UI:** User Interface;
- **GPS:** Global Positioning System.
- **UX:** User Experience
- **BLoC:** Business Logic Component
- **WBS:** Work BreakDown Structure

1.4 Revision history

DD v1.0: Document registered and delivered on 8/12/2019.

1.5 Document Structure

This specification document follows the standard suggested by IEEE and therefore is composed of different sections.

Section 1: in which the purpose of the Design Document is explained, followed by the scope of the System, which provides the description of the problem and the application domain, and a reference of Definitions, Acronyms, Abbreviations which are used throughout the document.

Section 2: in which the complete architectural design of the SafeStreets application is presented and explained. It start from an high-level overview. Then it describes the component and deployment view of the application. Following some sequence diagrams which displays the runtime view of the application, using the interfaces listen in the Component interfaces section. Finally, the selected structural patterns and other design decisions are presented.

Section 3: in which the user interface design of the application is presented. For the mockups of the application, it references back to the RASD document.

Section 4: in which each one of the requirements introduced in the RASD document is traced on the specific component that will provide that functionality.

Section 5: in which is devised the implementation, integration and test plan for the development of the application. It shows the component dependencies and the implementation strategy.

Section 6: in which the used tool and the hours of work of each team member are presented. Each working session has been registered; we provide the date, a description of the work, the total amount of time and the start and end time for each session.

2 Architectural Design

2.1 Overview: High-level components and their interactions

We decided to design the system-to-be following the client-server model, particularly with three tiers: Presentation Tier, Middle Tier (application logic) and Data Tier.

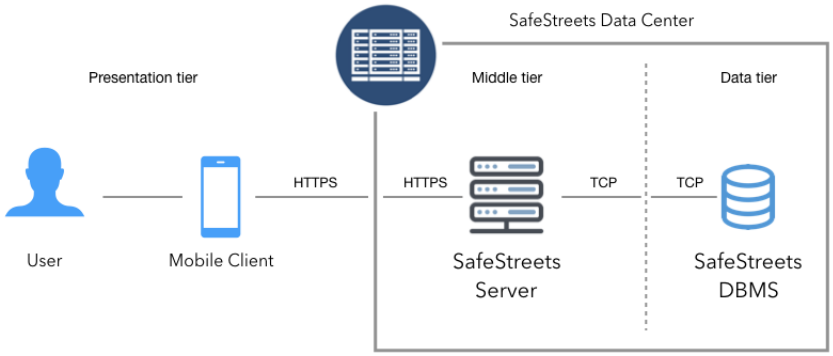

Figure 1: SafeStreets High Level Architecture

Presentation Tier
SafeStreets provide a mobile application on which all the data will be retrieved and displayed to the final User. The mobile application communicates with the Middle Tier through the protocol HTTPS and it is designed to work as a thick-client: it saves data and information offline and it updates them by esthablish a connection with the Application Server whenever internet connection is available.

Middle Tier (Application Server)
The Middle Tier contains the Application Server related to the specific features of the Mobile Application available to the User. The Application Server deploys a Web Server, that is providing the HTTPS APIs through which clients can communicate. The access to data is grant through the TCP protocol.

Data Tier
This Tier contains the databases in which data are store. The databases are managed by the DBMS. All the data that requires to be stored persistently are sent here by the Application Server.

2.2 Component view

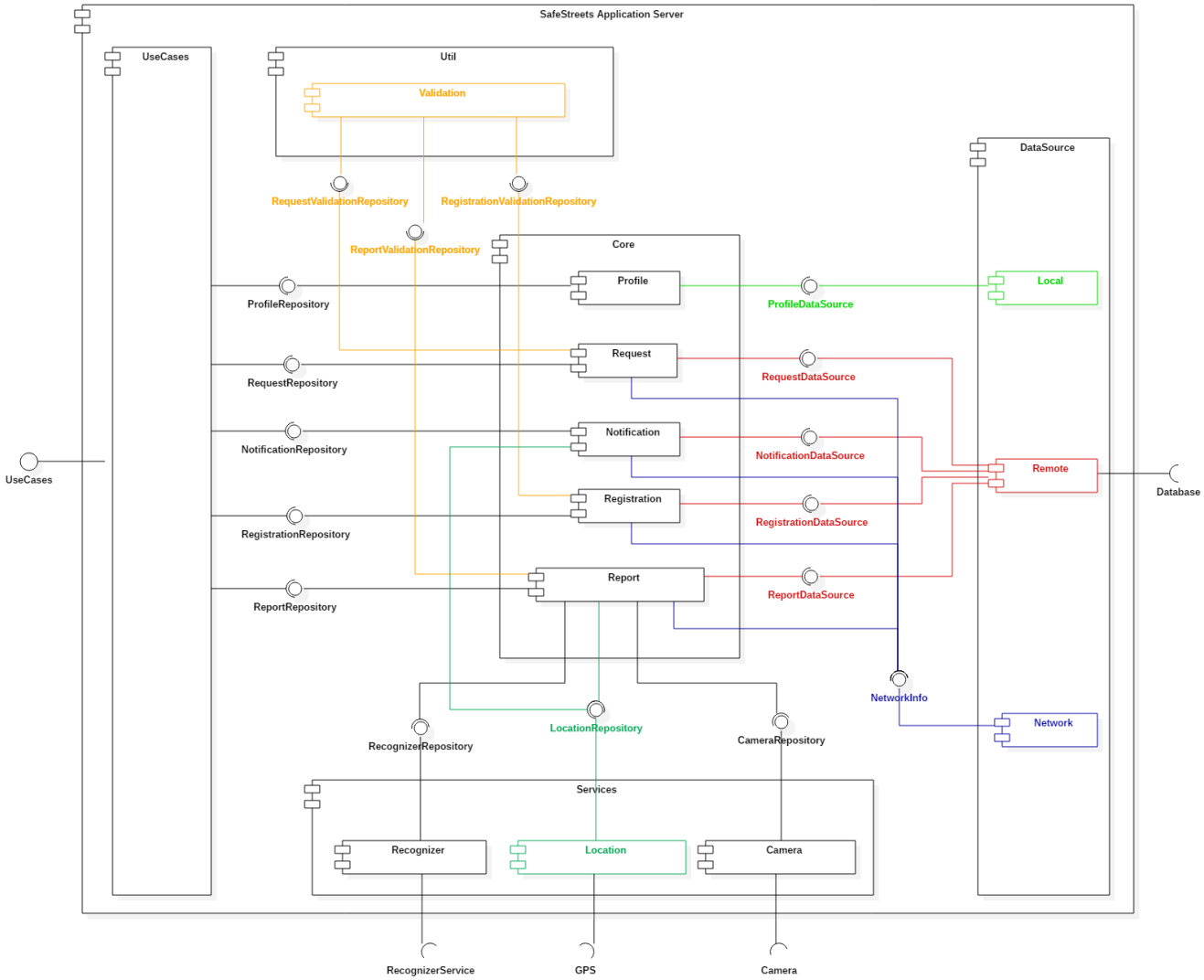

Figure 2: Component diagram overview of the complete application server of SafeStreets

This component diagram represents an overview of the application server at a high level, it shows the interfaces offered towards the clients of the system and the internal interfaces used by the system to provide its functionalities. Furthermore, the external dependencies are highlighted, pointing to the services provided by the device (GPS and Camera), the external PictureRecognizer service, and the connection with the Database layer.

The main functionalities of the application are provided through the *Core* component. All the main components of the application interact with one another by exposing utility interfaces that provide necessary functions for other components to carry out their jobs.

The system is organized in layers of abstraction. The client will only interact with the interfaces provided by the *UseCases* component. This component will then be responsible to call the correct interface from the *Core* component to fulfill the client request.

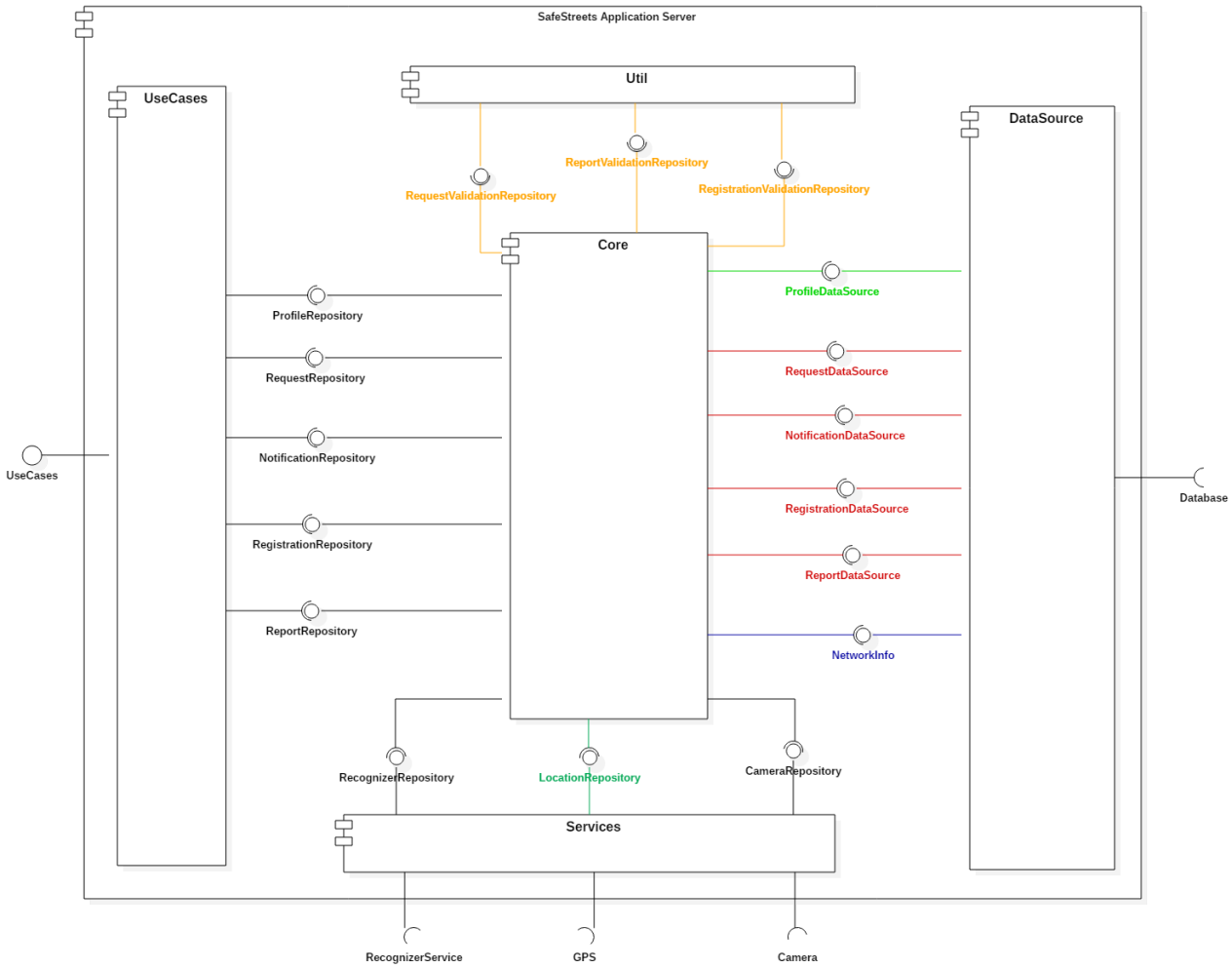

Figure 3: High level overview of the application server of SafeStreets. This Component diagram highlights the main components of the application and their interactions interfaces

The *Core* component depends on different *Util* and *Services* interfaces. Those components will be responsible to provide the necessary functionalities for which they are deisgned. Some of those functionalities will be available through third party services (Recognizer) while other will depend on the device availability (Camera, GPS).

All the data requests, either POST or GET, will pass through the *DataSources* component. This component will interact with the external Database to fetch the necessary data for the application. Those data will be collected from the web API, which will expose an endpoint to the application for each possible request.

Each main component contains some more specific sub-component, each responsible for a specific functionality:

- **Core**
  - Profile Component
  - Request Component
  - Notification Component
  - Registration Component
  - Report Component
- **Util**
  - Validation Component
- **Services**
  - Recognizer Component
  - Location Component
  - Camera Component
- **DataSource**
  - Local Component
  - Remote Component
  - Network Component

In the following part of this section we will present an overview of the functionalities provided for each component presented above. For each of them, we present a more specific Component diagram which will clarify the sub-components required by each main-component in the overview diagram, as well as their specific interactions.

2.2.1 Core Component

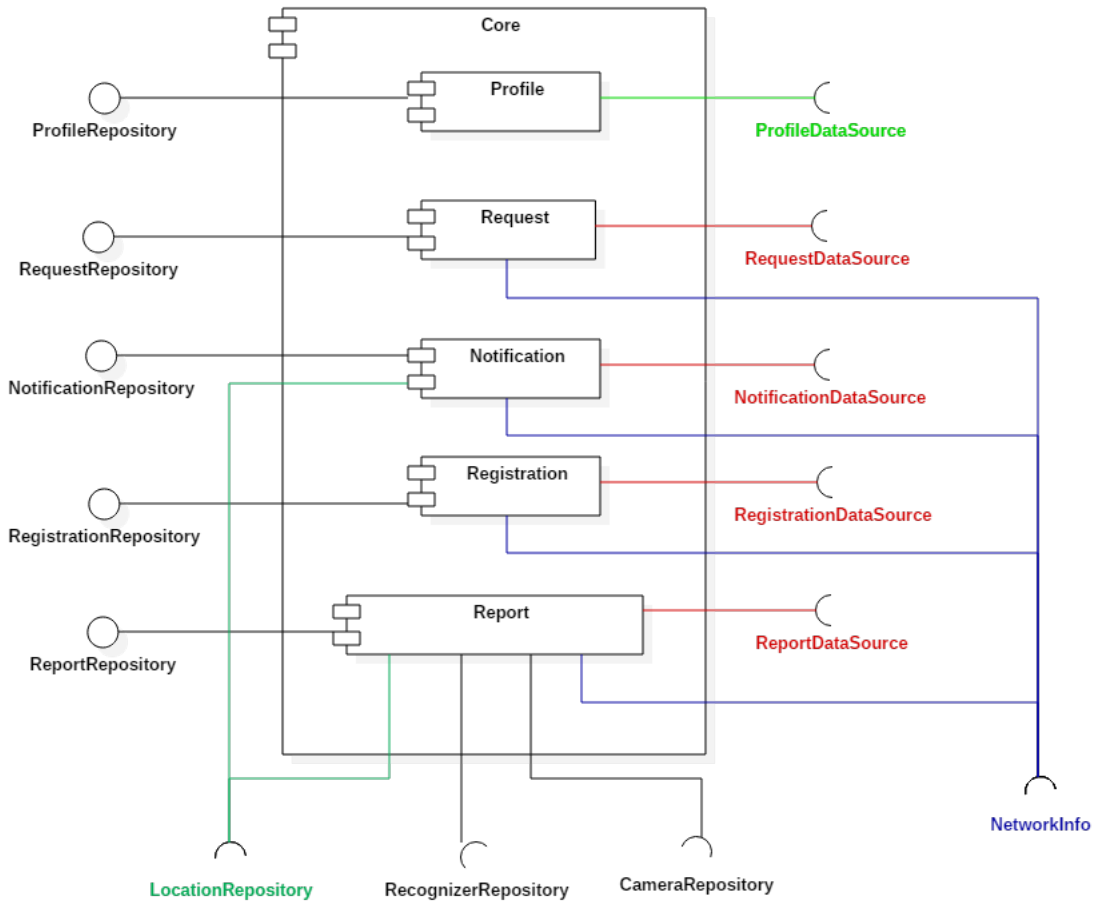

Figure 4: Component diagram for the Core main Component of the application

- *Profile:* The Profile Component handles the settings and preferences for the application. It provides the user ID, the notification service activation, and the logout functionality.
- *Request:* The Request Component will create and manage all the requests made by the User. Based on the input parameters, the component will fetch the requested data and then provide an interface to display those information to the client.
- *Notification:* The Notification Component handles all the requests for notifications. The notification service will be only available for Authorities, and will allow them to receive alerts about violations for which they are subscribed for.
- *Registration:* The Registration Component handles the initial registration process required for each User to access the application service. It manages the User data and it completes the actual registration request.
- *Report:* The Report Component will create and post the violation report filed by each User. It depends on some external services and, once all the information required will be available and validated, this component will send the actual request to the server.

2.2.2 Util Component

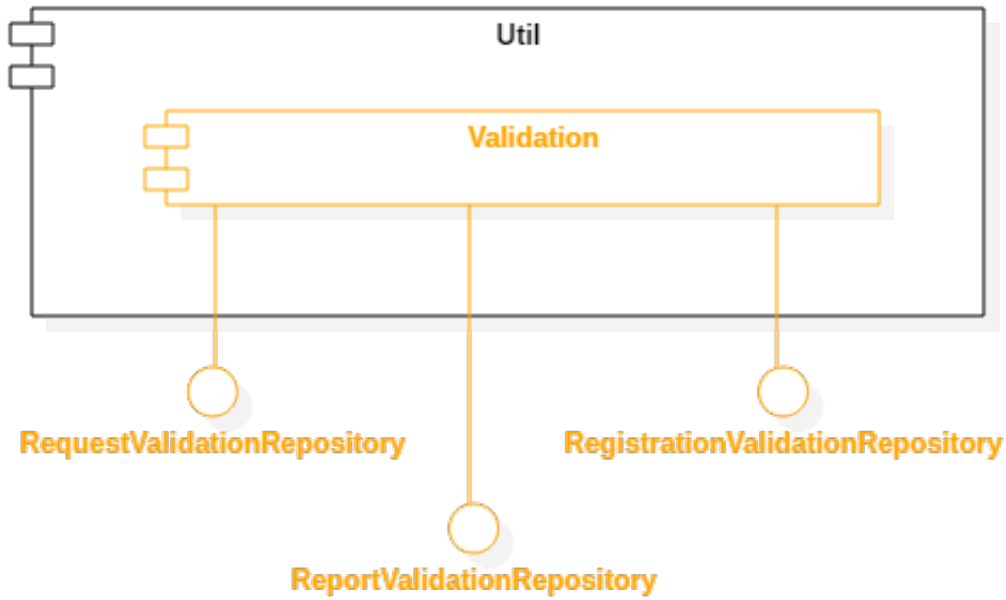

Figure 5: Component diagram for the Util main Component of the application

- *Validation:* The Validation Component will manage all the data in input from the User. It will ensure that the data is provided in the correct format and that each entity is valid and ready to be sent across the application.

2.2.3 Services Component

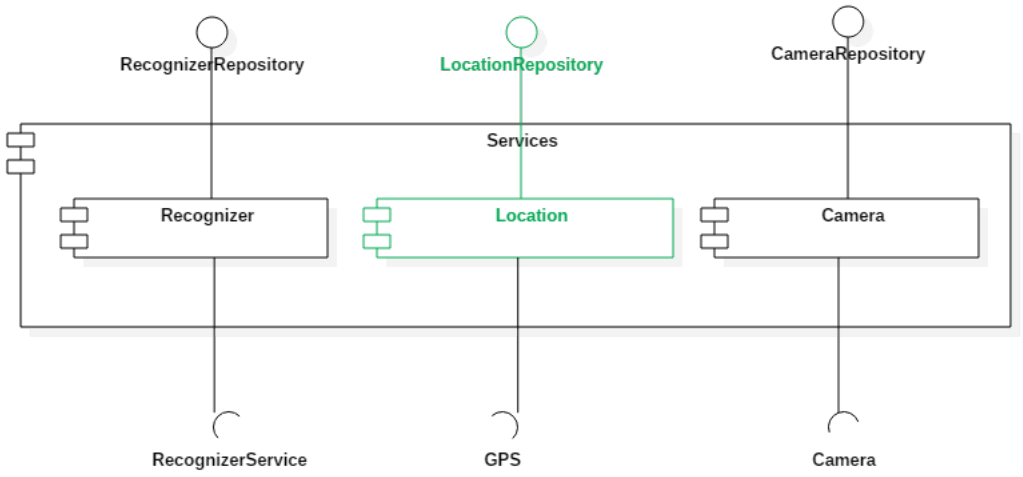

Figure 6: Component diagram for the Services main Component of the application

- *Recognizer:* The Recognizer Component will allow the application to compute the license plate code from the violation image. The recognization service will be provided by a third party service and will not be implemented by SafeStreets.

- *Location:* The Location Component interacts with the device GPS to provide the current position of the User.
- *Camera:* The Camera Component interacts witht the device Camera to provide the image of the violation.

2.2.4 DataSources Component

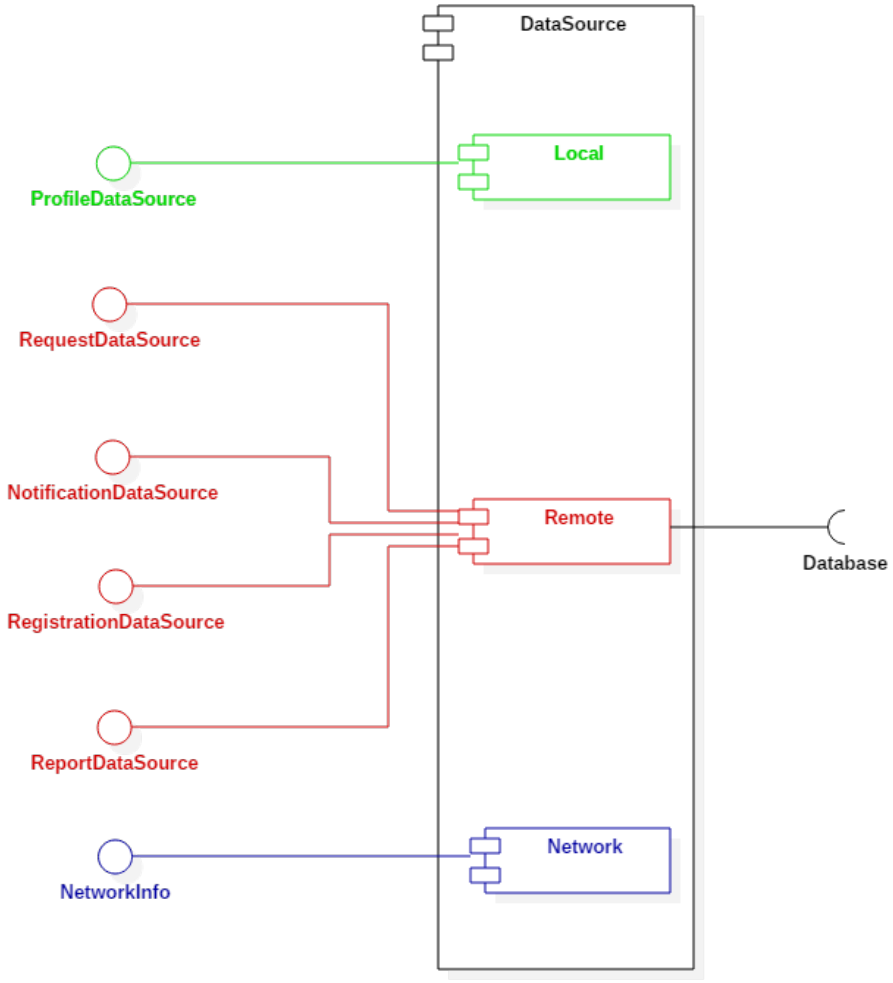

Figure 7: Component diagram for the DataSources main Component of the application

- *Local:* The Local Component handles all the data saved locally on the User device, saving information like the User preferences and the notification service activation.
- *Remote:* The Remote Component interacts with the Database to fetch the data required by the application. It manages the data in input from the external source and performs an initial validation.
- *Network:* The Network Component checks the availability of an internet connection of the device.

2.2.5 UseCases Component

The UseCases Component interact directly with the client to provide the functionalities to the User of the application. This component is a layer with separates the internal structure and functions of the

application from what concerns the User. All the client request will be forwarded to this component which will call the correct interface to fetch the requested data.

2.3 Deployment view

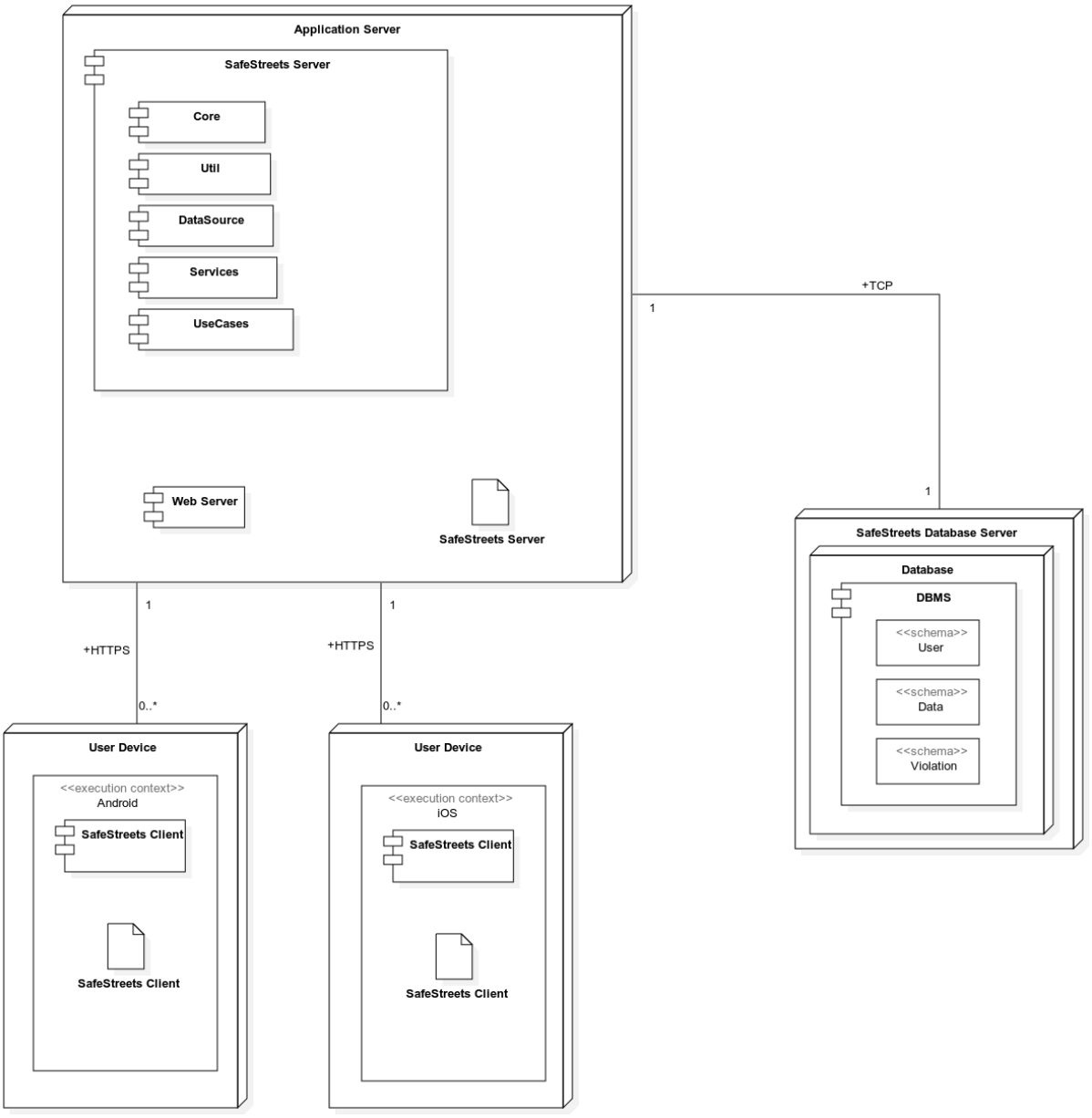

Figure 8: SafeStreets Deployment View

This deployment diagram shows how the SafeStreets system is deployed on the different machines. In order to underline the different execution contexts of the mobile application we choose to represent two User Devices; respectively one for the Android context and one for the iOS context.

User Device These nodes represent the User Devices, on which the mobile operative systems (iOS and Android) run. On the devices are deployed SafeStreets clients; the client application will not be analyzed in this document and details will not be exploited. Both iOS and Android applications can communicate through the HTTPS protocol with the Application Server.

Server This node represents the Application Server, that communicates with both the User Device and the Database Server. All the components presented in the previos section related to SafeStreets' Server

are deployed on this node. The Web Server grants the communication between nodes throught the protocols, it is used by the Application Server to receive and send data after their elaboration. Note that we are not representing the detail of every component, this representation is exploited in fig.2.

SafeStreets Database Server This node contains the databases and the DBMS. To grant performance, security, scalability we deploy the database on a different server machine. The Database Server is isolated from the Application Server. This improves the performance because of the dedicated Server and it will be more efficient to recover informations in case of loss. Finally the scalability will be very handle.

2.4 Runtime view

2.4.1 User Registration

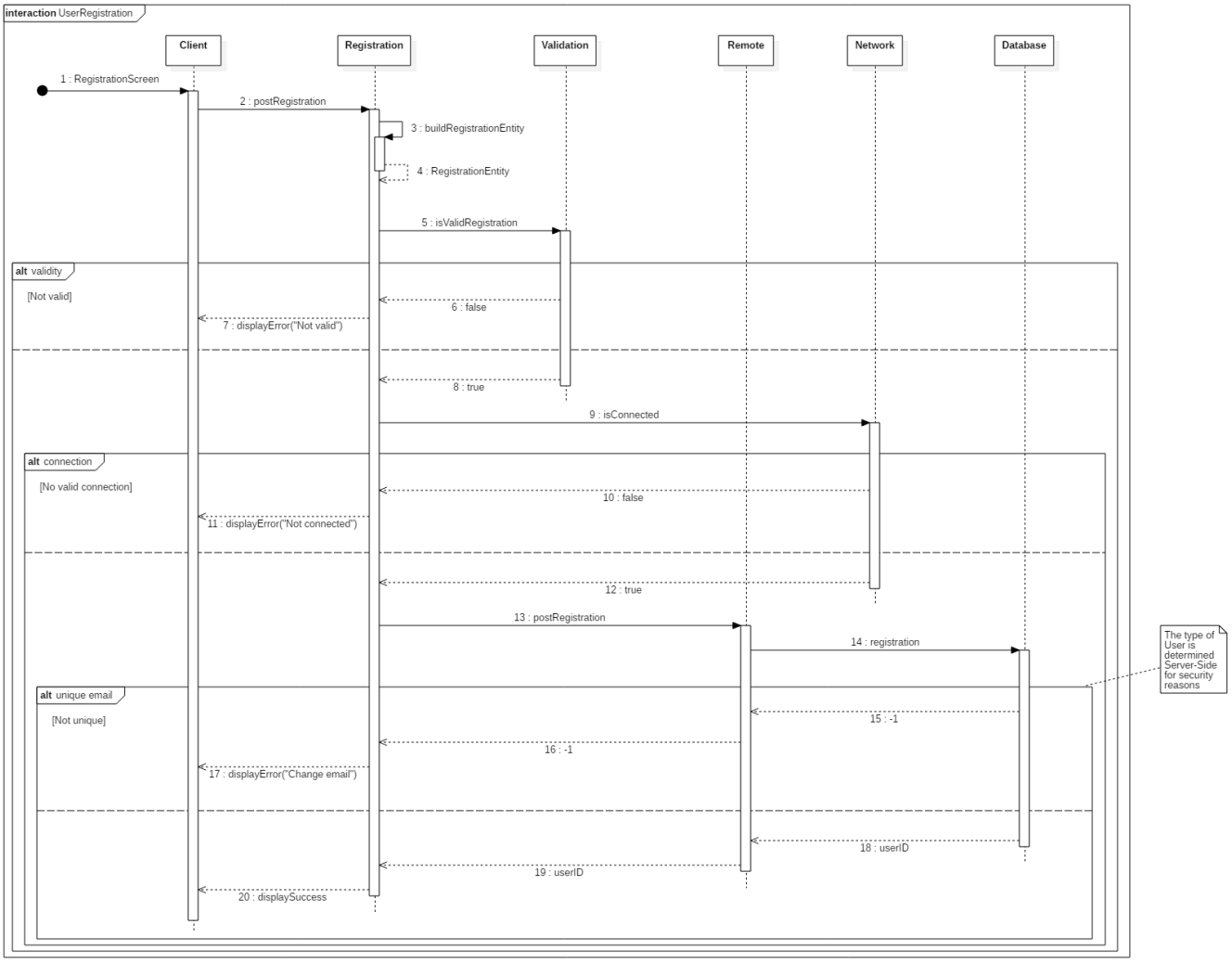

Figure 9: Sequence diagram representing the registration process. It models all the steps and calls made by each component of the application before completing the actual registration of the user.

The diagram models the registration process from the point of view of the SafeStreets application. The process start when the User, after inserting all the input required (name, surname, email, password), sends a request to be registered into the System. The Registration component takes all the data from the User and builds a RegistrationEntity. Then it calls the Validation component which returns true if the input data is valid, false otherwise. If the data is valid the Network component checks the internet connection, which is required to complete the process. If also the connection is present, then the Remote component is invoked to complete the actual registration. This component posts the data on the Database through an API call. The Database returns the new User ID. Then application displays a success message at the end of the process.

If any of the calls returns an error, the application displays an error message to the User.

2.4.2 User sending a Report

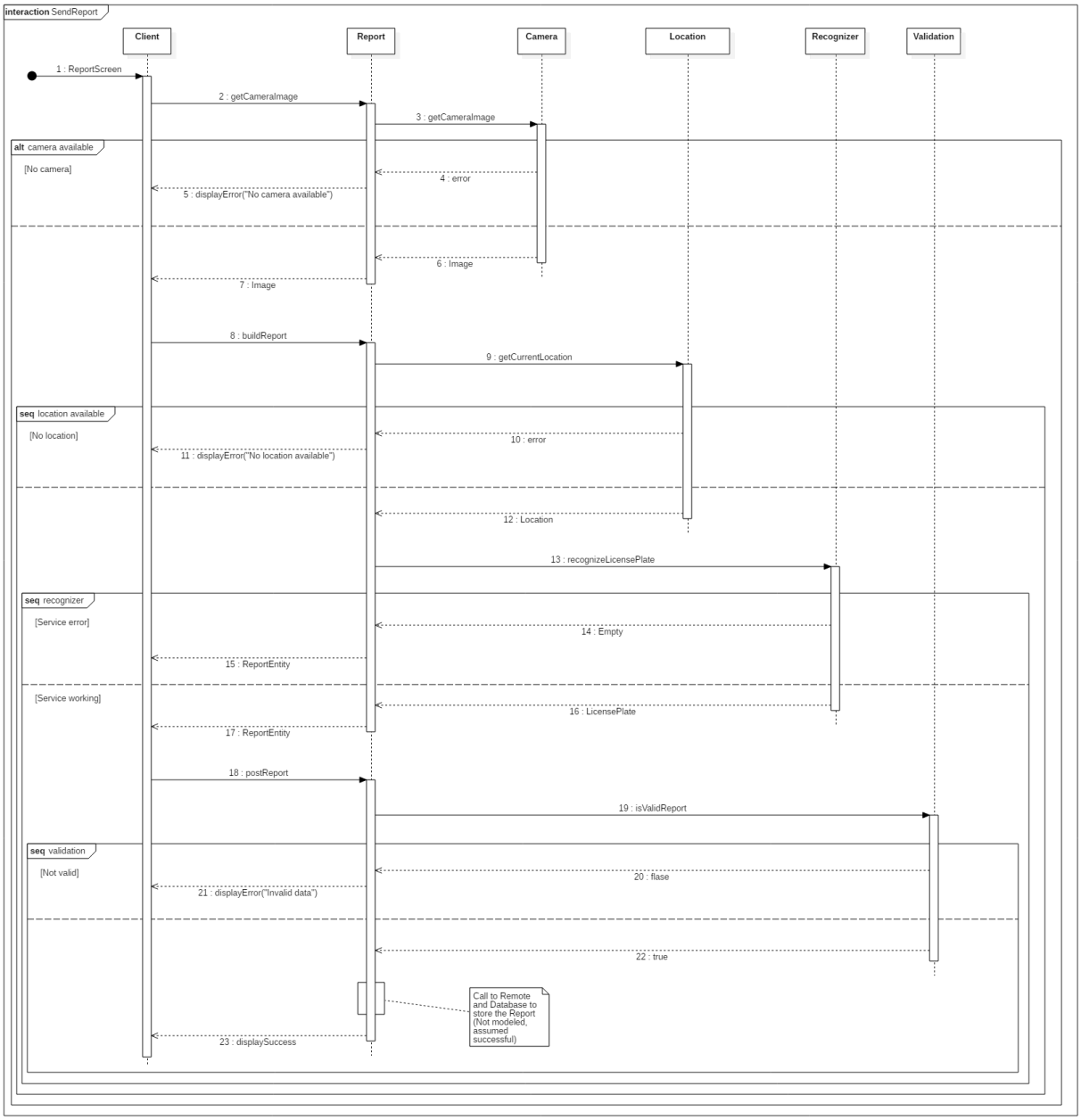

Figure 10: Sequence diagram representing the process of sending a report. It models all the steps and calls made by each component of the application before completing the actual storing of the report.

The diagram models the sending of the violation report from the User. The process start when the User invokes the Camera to take a picture of the violation. After taking the picture, the User is redirected to another screen to insert the type of violation, and then the Report component is called again to build the actual ReportEntity. The Location component is called to get the current location of the User and the Recognizer component is also called to recognize the LicensePlate from the picture taken previously. If the recognization service is not available, an empty LicensePlate is returned. In this case the User will be required to input the LicensePlate code manually. After all the data is in place and the User and the Validation component confirm all the parameters, the violation report is stored into the Database (the Database call is not modeled in the diagram, which focuses mainly on the Report creation and validation

process). At the end of the process, a success message is displayed to the User. Some components could return one of the following errors:

- Camera not available or missing permission
- Location not available or missing permission
- Validation component returns false, which means that some input parameter (type, picture, license plate) is not valid

In any of those cases, the application displays an error message to the User.

2.4.3 NormalUser sending a Request

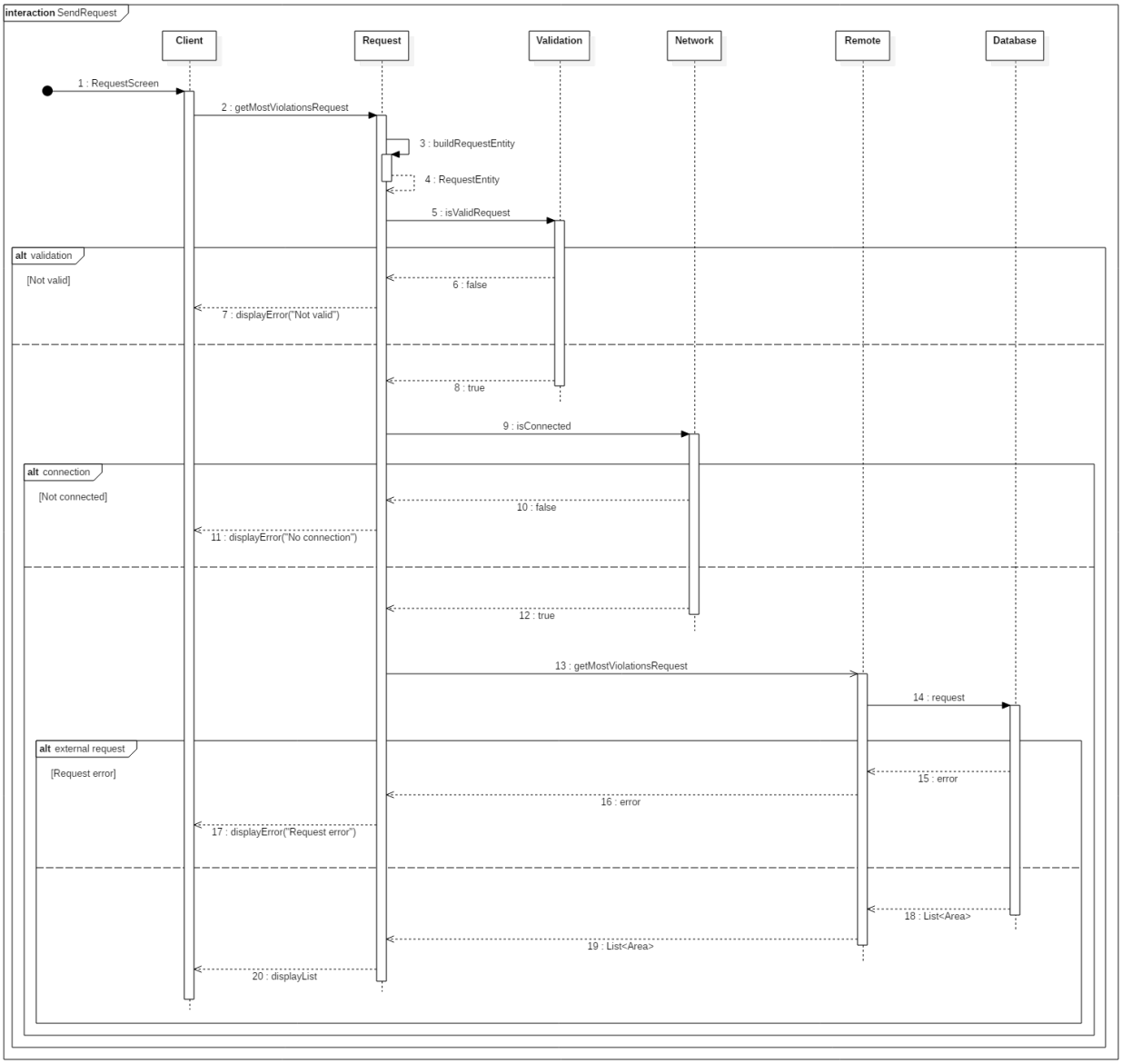

Figure 11: Sequence diagram representing the process of sending a request for the area with most violations. It models all the steps and calls made by each component of the application before completing the actual request of the user.

The diagram models the process related to the request of the list of area with most violation from a NormalUser. The process starts when the User sends the actual request. The application checks the validity of the input data and the connection availability. The Request component then send the request to the Remote component which calls the Database to retrieve the requested data. If no error occurs, the list of area requested is returned and displayed to the User.

2.4.4 User login into the application

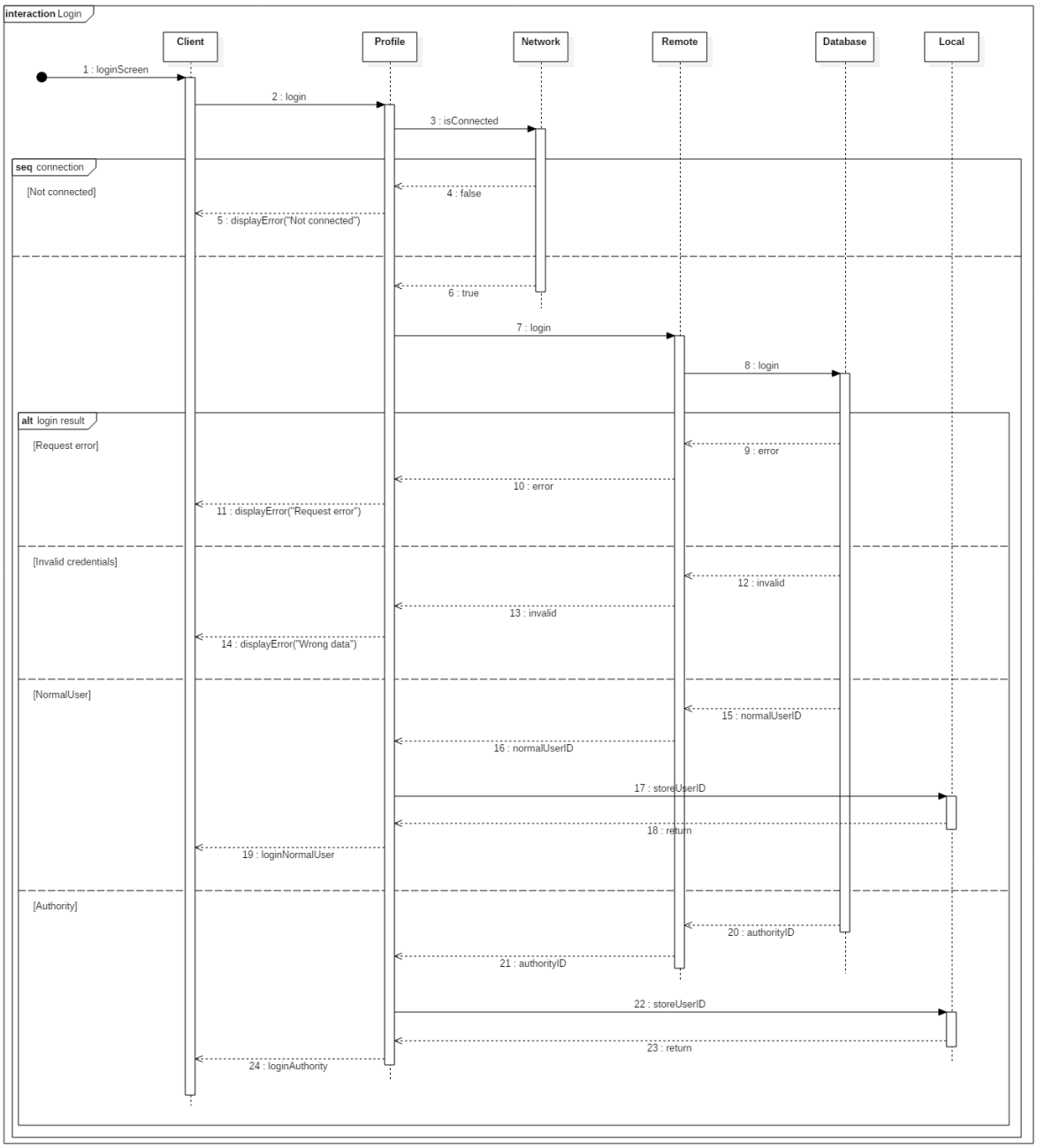

Figure 12: Sequence diagram representing the login process from a generic User. It models all the steps and calls made by each component of the application before completing the login of the user.

The diagram models all the step involved in the login process of a generic User. When the user sends his/her login credentials (email, password), the Profile component, after checking the internet connection

availability, passes the request to the Remote component. The Remote component calls the Database. Four outcomes are then possible:

- The Database responds with an error
- The Database signals that the credentials are not correct
- The Database recognizes a NormalUser and sends back its userID
- The Database recognizes an Authority and sends back its userID

If the login is successful, the userID is stored locally and the user has access to the application services. The services available will be activated based on the specific user type.

2.4.5 Application checks for notifications nearby

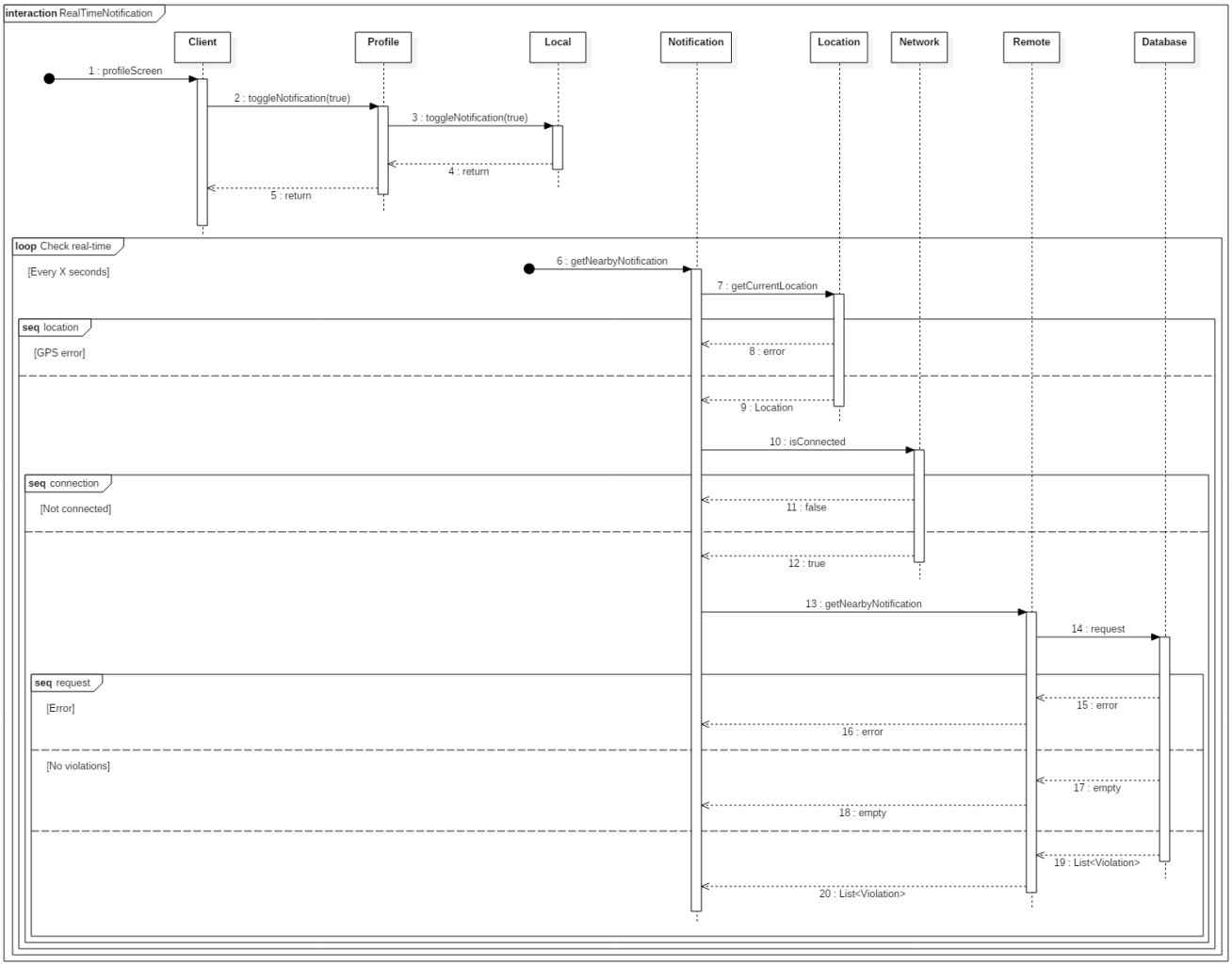

Figure 13: Sequence diagram representing the process of checking for a notification nearby. It models all the steps and calls made by each component of the application before completing the notification process.

The diagram models the steps for the activation of the notification service and the process of the application to retrieve nearby violations. The Authority activates the notification service by making

a specific request to the application. Once the service is activated, the application starts listening for nearby real-time violation updates, indicatively every X minutes based on the device availability. The Notification request requires the Authority location and an internet connection. The application then retrieves the violations from the Database and, in case some violation is available nearby, it send a notification to the Authority.

2.4.6 Authority License Plate Violations Retrieval

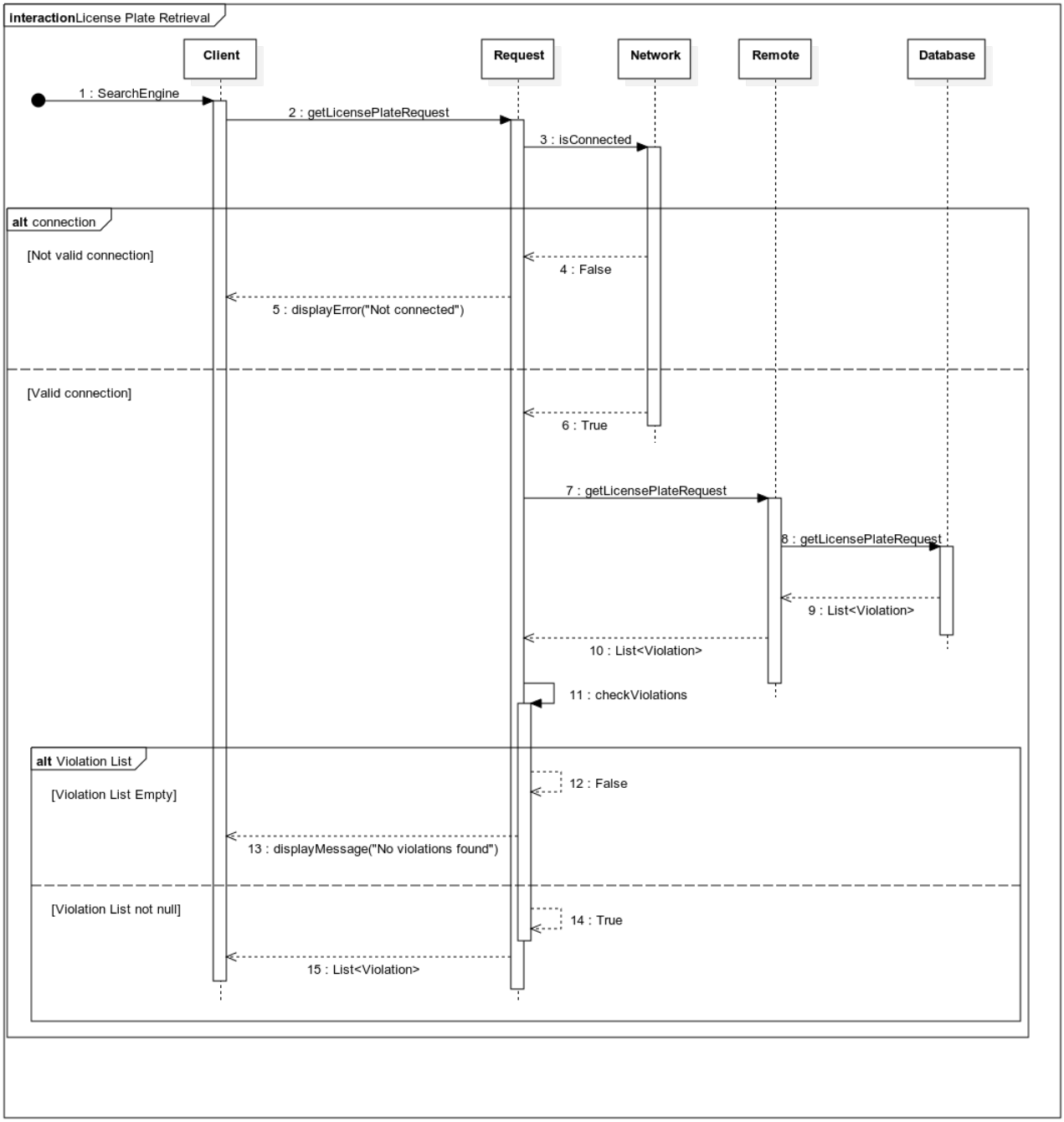

Figure 14: Sequence diagram representing the process of retrieve violations of a specific license plate.

The diagram describes the process of retrieving information about a specific licese plate from an Authority. The Authority inputs the license plate on the search engine and send a specific request to the application. The application check the network status through the dedicated function, if internet is available it can retrieve the data from the database. Particularly it retrieves all the violations related to the submitted license plate. Once retrieved the list, the application checks if it is actually empty or not and returns it to the Authority.

2.5 Component interfaces

- **Core**
  - *ReportRepository*
    - postReport(ReportEntity) : bool
    - buildReport(Image, type : int, userID : int, LicensePlate, Location) : ReportEntity
    - getCameraImage() : Image
  - *RegistrationRepository*
    - postRegistration(RegistrationEntity) : int
  - *NotificationRepository*
    - getAreaNotification(Area, userID : int) : List<Violation>
    - getLicensePlateNotification(LicensePlate: userID : int) : List<Violation>
    - getNearbyNotification(userID : int) : List<Violation>
    - getToValidateNotification(userID : int) : List<Violation>
    - subscribeLicensePlateNotification(userID : int, LicensePlate) : bool
    - subscribeAreaNotification(userID : int, Area) : bool
  - *RequestRepository*
    - getAreaSafetyRequestResult(userID : int) : List<Area>
    - getMostViolationsRequest(userID : int) : List<Area>
    - getAreaViolationsRequest(Area, userID : int) : List<Violation>
    - getLicensePlateRequest(LicensePlate, userID : int) : List<Violation>
    - getRecommendation(userID : int) : List<Recommendation>
    - getMyReport(userID : int) : List<Violation>
  - *ProfileRepository*
    - toggleNotification(activate : bool) : void
    - logout(userID: int) : bool
    - login(email : String, password : String) : int
    - getUserID() : int
- **Services**
  - *RecognizerRepository*
    - recognizeLicensePlate(Image) : LicensePlate
  - *LocationRepository*
    - getCurrentLocation() : Location
  - *CameraRepository*
    - getCameraImage() : Image
- **Util**
  - *RequestValidationRepository*
    - isValidRequest(RequestEntity) : bool
  - *ReportValidationRepository*

    - isValidReport(ReportEntity) : bool

  - *RegistrationValidationRepository*

    - isValidRegistration(RegistrationEntity) : bool

- **DataSource**

  - *NetworkInfo*

    - isConnected() : bool

  - *ReportDataSource*

    - postReport(ReportEntity) : bool

  - *RegistrationDataSource*

    - postRegistration(RegistrationEntity) : int

  - *NotificationDataSource*

    - getAreaNotification(Area, userID : int) : List<Violation>
    - getLicensePlateNotification(LicensePlate: userID : int) : List<Violation>
    - getNearbyNotification(Location, userID : int) : List<Violation>
    - getToValidateNotification(userID : int) : List<Violation>

  - *RequestDataSource*

    - getAreaSafetyRequestResult(userID : int) : List<Area>
    - getMostViolationsRequest(userID : int) : List<Area>
    - getLicensePlateRequest(LicensePlate, userID : int) : List<Violation>
    - getAreaViolationsRequest(Area, userID : int) : List<Violation>
    - getRecommendation(userID : int) : List<Recommendation>
    - getMyReport(userID : int) : List<Violation>

  - *ProfileDataSource*

    - toggleNotification(activate : bool) : void
    - logout(userID : int) : bool
    - login(email : String, password : String) : int
    - storeUserID() : void
    - getUserID() : int

2.6 Selected Architectural Styles and Patterns

In this section we present the choosen architectural styles and patterns, explain with respect to the system-to-be.

2.6.1 Client-Server: three tier architecture

The entire system is based on the client-server architecture. The client interacts with the server to exchange data. An entire representation is provided in Figure 1 (SafeStreets High Level Architecture), in the Overview section.

2.6.2 Layered Style

The application is organized through abstraction levels. Each level depends only on the one below, which makes the system solid and flexible, since failures in one part of the system will not be propagated to the entire application.

2.6.3 Repository Pattern

The system is designed and will be implemented using the Repository Pattern. The use of this pattern provides an abstraction of data between the application and the data sources. The system will call the methods of the repository abstract layer and the job of fetching the data, either from external (Remote) or internal (Local) data sources will be indipendent from the call itself.

2.6.4 BLoC Pattern (Event-driven)

BLoC (Business Logic Component) is a reactive programming pattern. The client will not deal with any Business Logic which will be all implemented inside bloc classes. The client will send events to the bloc that will react on each specific event by changing the state of the application. The client will then listen to the change in the state and rebuild itself accordingly.

2.6.5 Service Locator

The Service Locator is a way to decouple the interface (abstract base class) from a concrete implementation and at the same time allows to access the concrete implementation from everywhere in the application over the interface.

3 User Interface Design

The user interface is described in the "User Interface" section in the RASD document.

3.1 UX diagrams

The figure below represents the UX diagram of the Mobile Client with the main features and functionality of the system-to-be. We distinguish two flows: one for the Normal User and the other for the Authority. We represent both in the same diagram, please refer to the legend below.

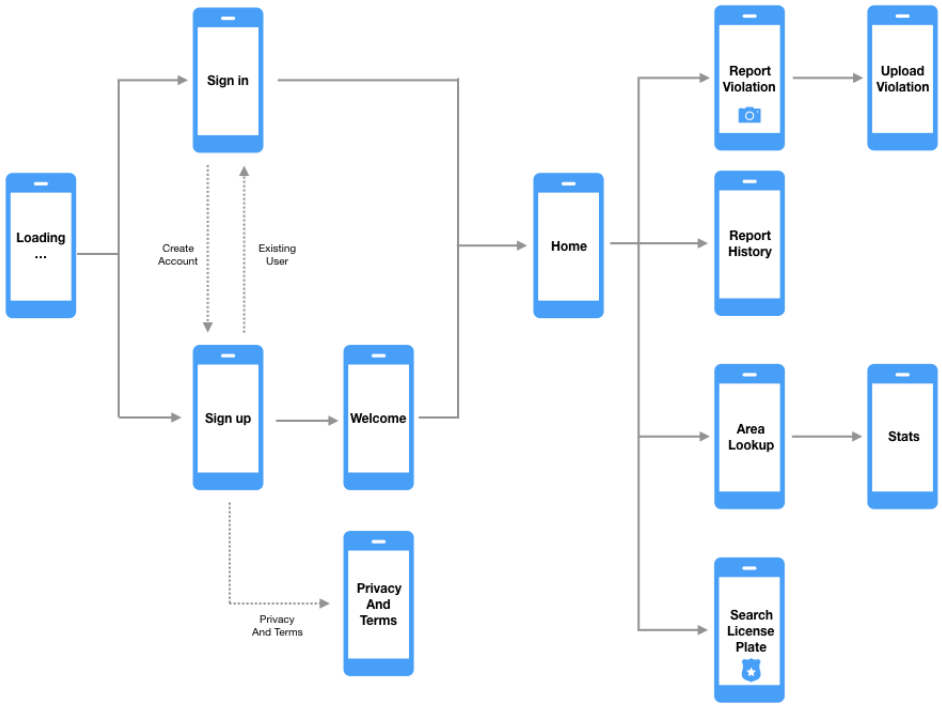

Figure 15: SafeStreets UX Diagram

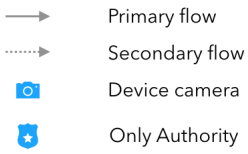

Figure 16: UX Diagram Legend

The primary flow is the ordinary flow of the execution of the Application, it is intended to provide all the features of the system. The secondary flow is designed to improve the usability of the system-to-be, it

also includes some special pages for example the Privacy and Terms page. The Device Camera indicates that the built in camera of the User device is involved. The Authority symbol underlines that this page is only accessible by a User with Authority privileges.

4 Requirements Tracebility

This section is intended to map the requirements of the system-to-be defined in the RASD document on the components of the designed software.

Following is the list of the requirements, each associated with the component which is responsible to manage that specific functionality. It is important to note that, because of the continuous interaction between the application and the client, the component UseCases is involved in satisfing every listed requirement. Therefore, we avoid to trace the dependency of this component with the requirements.

- **[R1]** After a Visitor correctly fills and sends the registration form, he/she is allowed to sign in to the System as a User
  - Registration Component
  - Validation Component
  - Remote Component
  - Network Component
- **[R2]** A User is upgraded to Authority if the System recognizes a PEC email address
  - Registration Component
  - Validation Component
- **[R3]** If the User credentials are correct, he/she is logged in by the System
  - Profile Component
  - Remote Component
  - Network Component
- **[R4]** The System visualizes the user's previous sent reports when requested
  - Request Component
  - Remote Component
  - Network Component
  - Validation Component
- **[R5]** A User can access the camera to take a picture of the violation
  - Camera Component
- **[R6]** After a picture has been taken, the System provides a form to the user to insert the type of violation
  - Report Component
- **[R7]** The System displays the safetiness of an area when requested
  - Request Component
  - Remote Component
  - Network Component
  - Validation Component

- **[R8]** The System displays the frequency of the violations of an area when requested
  - Request Component
  - Remote Component
  - Network Component
  - Validation Component
- **[R9]** When a report has been stored, the System associates the report to the User who sent it
  - Report Component
  - Remote Component
  - Network Component
  - Validation Component
- **[R10]** A User can select the area that he/she is interested in
  - Request Component
- **[R11]** An Authority can input a license plate
  - Request Component
- **[R12]** The System attaches the GPS position when a report is sent
  - Location Component
- **[R13]** The Authority can toggle the notification service
  - Profile Component
  - Local Component
- **[R14]** The System periodically checks the GPS position of an Authority if he/she has the notification service enabled
  - Notification Component
- **[R15]** If a violation is detect nearby the Authority position and he/she has the notification service activated, the System sends a notification
  - Notification Component
  - Profile Component
  - Local Component
  - Remote Component
  - Network Component
  - Location Component
- **[R16]** An Authority is periodically updated about violation in an area if he/she has activated the notification service in that area
  - Notification Component
  - Profile Component
  - Local Component
  - Remote Component

  - Network Component
  - Location Component

- **[R17]** When information about a license plate is available, the System returns it
  - Request Component
  - Remote Component
  - Network Component

- **[R18]** An account is registered if and only if the email provided is not already present into the System
  - Registration Component
  - Remote Component
  - Network Component

- **[R19]** The System suggests possible interventions in an area when requested by an Authority
  - Request Component
  - Remote Component
  - Network Component

- **[R20]** The User must insert the license plate if the System is unable to recognize it from the picture
  - Report Component
  - Recognizer Component
  - Camera Component

- **[R21]** If a User reports more than one violation of the same license plate in the space of 24 hours, only one violation will be considered valid, thus uploaded.
  - Validation Component
  - Report Component

- **[R22]** A notification is sent to an authority every time a license plate reaches ten not yet validated violation reports.
  - Notification Component
  - Remote Component
  - Network Component

5 Implementation, Integration and Test Plan

5.1 Scope

In this section we define an implementation and integration plan for the designed system SafeStreets. Particularly we exploit the order and the methods with which the components are to be implemented, integrated and tested also according to their dependencies. The plan follow a risk-oriented approach. This means that all the components that are likely to cause issues or are hard to implement should be tested as early as possible.

5.2 Component dependencies

The SafeStreets application, as shown in the component diagrams in Figure 2 and Figure 3, retrieves the data from the Database layer, computes this data internally and then it displays it to the client. The application is composed of 5 main high-level components:

- Core component
- DataSources component
- Services component
- Util component
- UseCases component

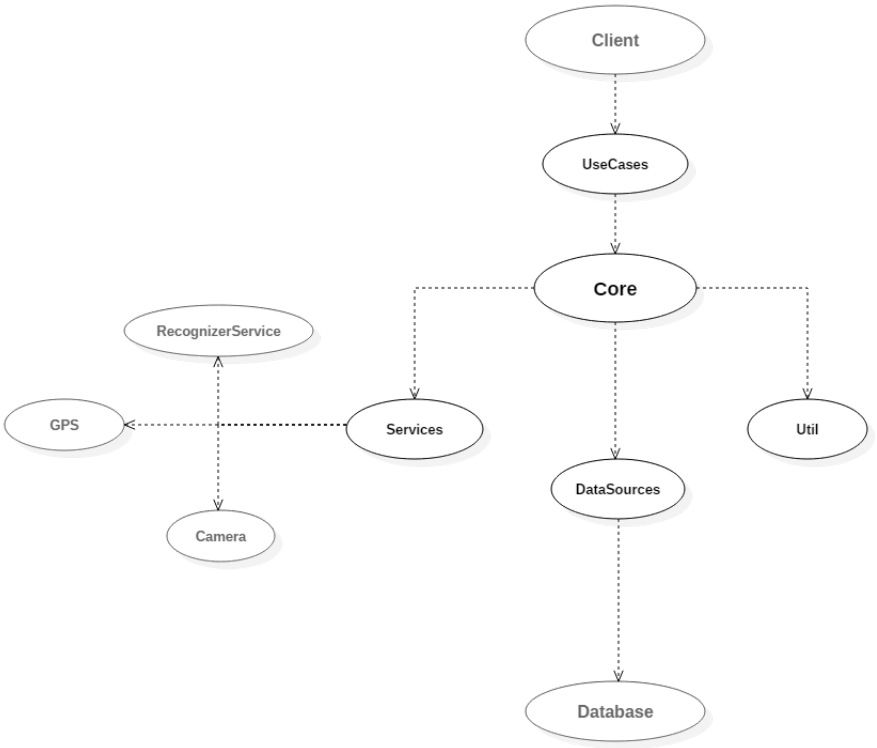

Figure 17: Diagram representing the dependencies between each high level component of the SafeStreets application

The graph in Figure 17 shows the dependencies between each component of the application. More formally, it shows the transition relation "component X uses interface offered by component Y". We can

notice that the graph shows no circular dependency. The application is in fact structured in layers (layered style), and each component, starting from the UseCases, makes request 'top-down'.

Moreover, identifying families of sets of components independent from each other's it's possible to define an implementation schedule that parallelizes the development as well as an estimation of the minimal duration of the project. Components can be divided in three sets, as shown in the figure below:

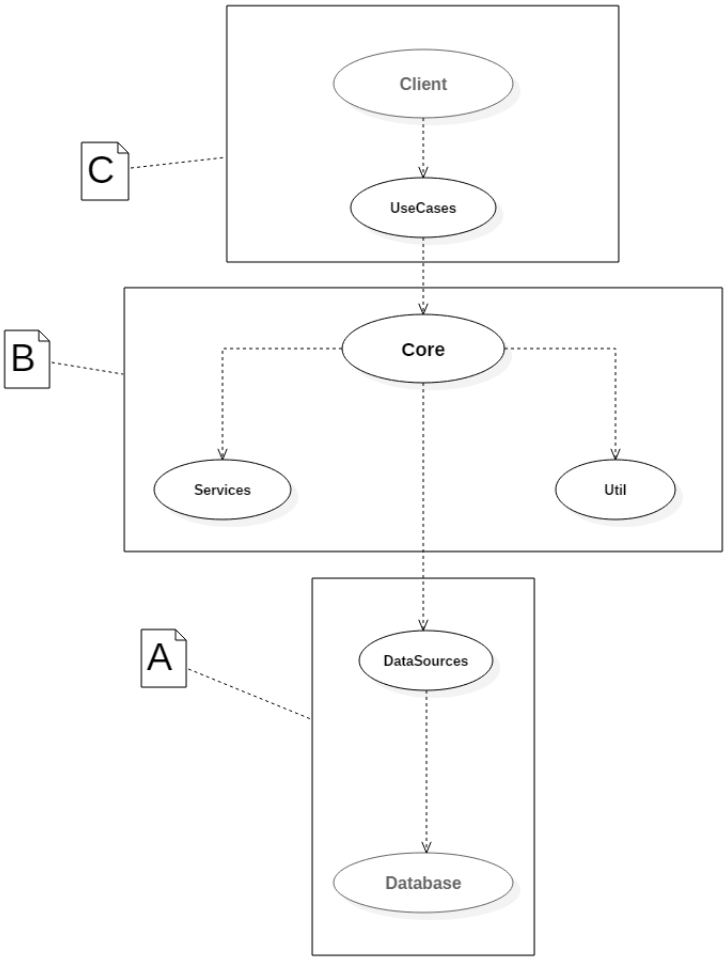

Figure 18: Diagram representing the three sets of component of the SafeStreets application

- **A**: Is the part of the system concerned with the storage of the data. All the components of the application relays on it to fetch and post the data. Therefore, this part needs to be implemented first.

- **B**: This part is the core of the application, in which all the data from the storage is computed and then exposed to the client. The Core component depends on the Services and Util components, therefore those are needed and should be implemented first.

- **C**: The last part is the front-end, which will display all the information to the client. Since this part relays on the internal computation of the system, its implementation can be dealyed util the end of the development process.

5.3 Implementation and integration strategy

According to the dependency and risk analysis previously preformed we can define the following strategy:

- Develop bottom-up module A. DataSources play a very important role on the system-to-be, therefor the first step is to implement Database component and DataSources component and unit test them, then integrate the two.
- Develop top-down module B. This is the core of the application and it has to be developed top-down in order to improve the generalization and the inheritance of the classes. Starting from the core, we can develop Util and Services in parallel. Both the components rely on the module A so they have to be integrated. Testing has to be done in order to validate the right behaviour, expecially for the Services component that communicates with GPS and Camera.
- Develop module C. This is the last part to be developed because it includes all the UseCases that rely on the proper behaviour of module B and A. It is basically the Client presentation of all the functionalities of the system. A proper integration have to be performed.

All the operations linked above can be performed in parallel thanks to the clean-architecture: this approach allows to develop the modules indipendently one from the others. Then a proper testing and integration has to be performed. The clean-architecture grants scalability and flexibility on the system-to-be.

5.4 Work Breakdown Structure

We provide a Work Breakdown Structure starting from the development phase. The Work Breakdown Structure presents a process-oriented top-down approach because we think it better suits the project. Note that the order is relevant and the Testing section presents different kind of testing to be performed after the Development phase, this is strictly different from the Unit Testing. Both have to be performed.

1. **Development**
    - 1.1 Environment setup
    - 1.2 Version control setup
    - 1.3 Coding phase
2. **Testing**
    - 2.1 Alpha-Beta Testing
    - 2.2 Security Testing
    - 2.3 Regression Testing
    - 2.4 User Acceptance Test
    - 2.5 Analyze reports
3. **Deployment**
    - 3.1 Setup servers
    - 3.2 Store licensing
    - 3.3 Release

6 Appendix

6.1 Used tools

- Visual Studio Code: Editing LaTeX files, pdf generation, and Github repository management - https://code.visualstudio.com/
- LaTeX: For the production of technical and scientific documentation - https://www.latex-project.org/

6.2 Hours of work

6.2.1 Cappelletti Andrea

Date	Task	Hours	Start-End time
16/11/2019	Setup DD file structure	1h 0min	16:00 - 17:00
23/11/2019	Review DD class discussion	1h 0min	8:00 - 9:00
23/11/2019	Deployment diagram	1h 0min	11:00 - 12:00
25/11/2019	Deployment diagram Introduction	1h 30min	17:30 - 19:00
26/11/2019	Deployment diagram, High level components and requirements trace.	1h 30min	11:00 - 12:30
02/12/2019	Deployment Update, High level description	1h 00min	11:00 - 12:00
02/12/2019	UX Diagram	1h 00min	21:00 - 22:00
03/12/2019	UX Diagram and sequence diagram	2h 00min	10:00 - 12:00
06/12/2019	Sequence diagram and document review	2h 00min	10:00 - 12:00
07/12/2019	Implementation and Testing, Work Break-Down Structure	2h 30min	10:00 - 12:30
08/12/2019	Final Review	1h 00min	11:00 - 12:00
08/12/2019	Pdf style and final review	1h 30min	16:30 - 18:00
Total		**18h 00min**	16/11 to 09/12

6.2.2 Maglione Sandro

Date	Task	Hours	Start-End time
12/11/2019	DD theory and structure review	1h 30min	11:00 - 12:30
16/11/2019	Setup DD file structure	1h 0min	16:00 - 17:00
18/11/2019	Review DD class discussion	1h 0min	12:00 - 13:00
19/11/2019	Introduction and component diagram	1h 45min	11:00 - 12:45
20/11/2019	Component diagrams	1h 15min	10:45 - 12:00
24/11/2019	Component diagrams	1h 00min	11:00 - 12:00
25/11/2019	Component diagram descriptions	1h 30min	11:00 - 12:30
26/11/2019	Component view completed	30min	12:00 - 12:30
27/11/2019	Requirements tracebility, component interfaces	1h 0min	11:30 - 12:30
28/11/2019	Component interfaces, sequence diagram	1h 45min	11:30 - 13:15
30/11/2019	Component diagram review	1h 45min	14:30 - 16:15
01/12/2019	Updated Component diagram	2h 30min	15:00 - 17:30
02/12/2019	Sequence diagrams runtime view	2h 00min	14:30 - 16:30
03/12/2019	Sequence diagrams runtime view	1h 45min	11:00 - 12:45
05/12/2019	Component dependencies	1h 15min	10:45 - 12:00
08/12/2019	Other patterns	30min	10:15 - 10:45
08/12/2019	Final Review	1h 00min	11:00 - 12:00
Total		**23h 00min**	16/11 to 09/12

References

The following resources were read before the writing of the DD document.

- Specification document "SafeStreets Mandatory Project Assignment"
- https://www.uml-diagrams.org
- Slide "6.a Design.pdf"
- https://resocoder.com/flutter-clean-architecture-tdd/

