POLITECNICO
MILANO 1863

SafeStreets project

Manuel Pedrozo, Tomás Perez Molina

Design Document

Deliverable:	DD
Title:	Design Document
Authors:	Manuel Pedrozo, Tomás Perez Molina
Version:	1.0
Date:	December 7, 2019
Download page:	https://github.com/lethanity/PedrozoPerez
Copyright:	Copyright © 2019, Manuel Pedrozo, Tomás Perez Molina – All rights reserved

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

The purpose of this document is to dive into technical details concerning the SafeStreets system. In the RASD, the system and its functionalities were introduced in an abstract manner. The Design Document focuses more on implementation, explaining the decisions that had to be made and the reasoning behind them. The topics covered in this document are the following:

- An overview of the proposed architecture
- The different system components and their interactions
- The deployment plan
- Utilized design patterns
- User interface design and flow
- Requirement traceability
- The implementation, integration and testing plan

1.2 Scope

The SafeStreets system is designed to provide users with the ability to report and get information of reported traffic violations through an application. Any user with a device capable of running the application can sign up to the system, which enables them to access its functionalities. In order to submit a report, the user needs to fill a form. In it, they have to enter the license plate number of the vehicle committing the violation, the type of violation and at least one photo of the scene, where the license plate of the vehicle can be easily recognized. This data, along with metadata retrieved from the user's device (geographical position, date and time) is then sent to the system. The system is responsible for analysing the validity of the report. To achieve this, a license plate recognition algorithm is utilized. When faced with difficulties in the detection, the photo must pass through community review, in which users reach a consensus on the validity of the report photo. The data collected by the system in relation to reports is to be queried by its users. There are two distinct targets of this functionality: standard users and the municipality. The main difference between the two is that the municipality can access information that should not be freely accessible to everyone because of security and privacy concerns. Through the application, users are capable of visualizing a city map showing where the violations happened. Furthermore, a public API is made available, facilitating data analysis and system integration.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- Traffic violation: An action performed by a driver of a vehicle which is against the local traffic regulations.
- Report: Information submitted by a user to notify the system and, by extension, authorities of a traffic violation.
- Compromised report: A report that has been modified by an unauthorized agent outside the system boundaries.
- Authority: A local agency whose purpose is, as indicated by the current law, to enforce traffic rules. For example: the police.

- Ticketing system: A government database containing information about issued traffic tickets.
- License plate registry: A government database connecting a license plate with the car registered to it and information about it such as the make, model and color.
- False-positive report: An invalid report that is considered valid by the system.
- False-negative report: A valid report that is considered invalid by the system.

1.3.2 Acronyms

- GPS: Global Positioning System
- API: Application Programming Interface

1.3.3 Abbreviations

Gn : n-th goal

Dn : n-th domain assumption

FRn : n-th functional requirement

SRn : n-th security requirement

1.4 Revision history

1.5 Reference Documents

1.6 Document Structure

2 Architectural design

2.1 Overview

The SafeStreets system is structured in a three tier architecture. As shown in [Figure x], these are the presentation, business and persistence tiers. This architecture allows for modularization by splitting the system into multiple parts. Because of this, the scalability and flexibility of the system is highly increased.

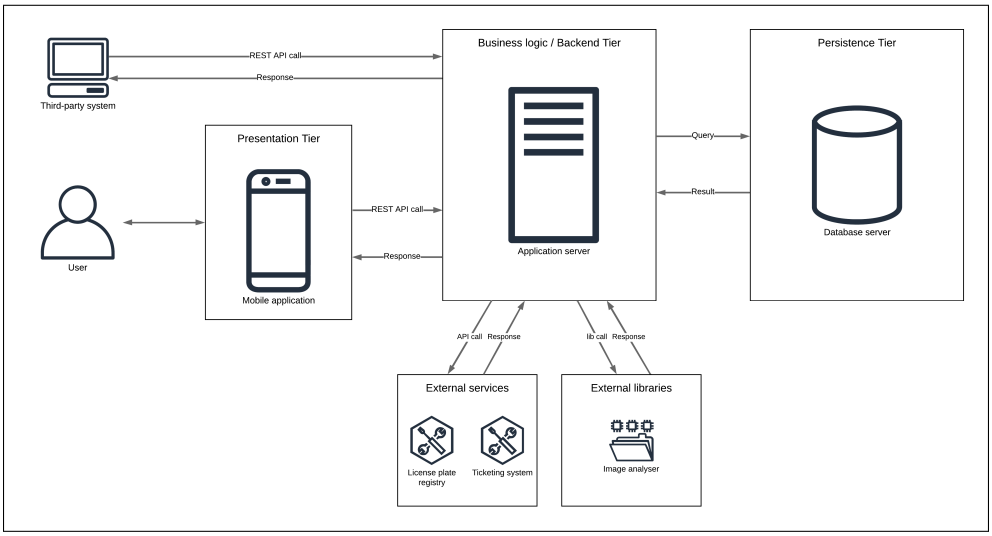

Figure 1: Three tier architecture diagram.

The presentation tier is the only part of the system the user interacts with. Here is where the SafeStreets mobile application is located. The app provides a GUI for every user to interact with. An alternative is for users to interact with the SafeStreets system through a third-party system, not developed by the team, which is capable of communicating with the application server via REST api calls. Next is the business logic tier, where the core functionality of the system is provided. The backend presents a RESTful API, through which the presentation tier can communicate by making requests. This tier depends on external services that help in providing different functionalities, like the image analysis. Last but not least, the persistence tier focuses on saving and managing the information gathered by the business logic. For this, a database management system is utilized. The only tier capable of accessing the persistence tier is the business logic, providing higher security for any sensitive data.

2.2 Component view

The following diagrams describe the internal structure of the application server, which contains the business logic of the system. Users will communicate with it via its exposed interfaces, either with the mobile application or any system capable of performing HTTP requests. Figure 2 shows a complete overview of the components to get a general idea of the structure. Following it, Figures 4 and 3 zoom into parts of the diagram to appreciate the finer details.

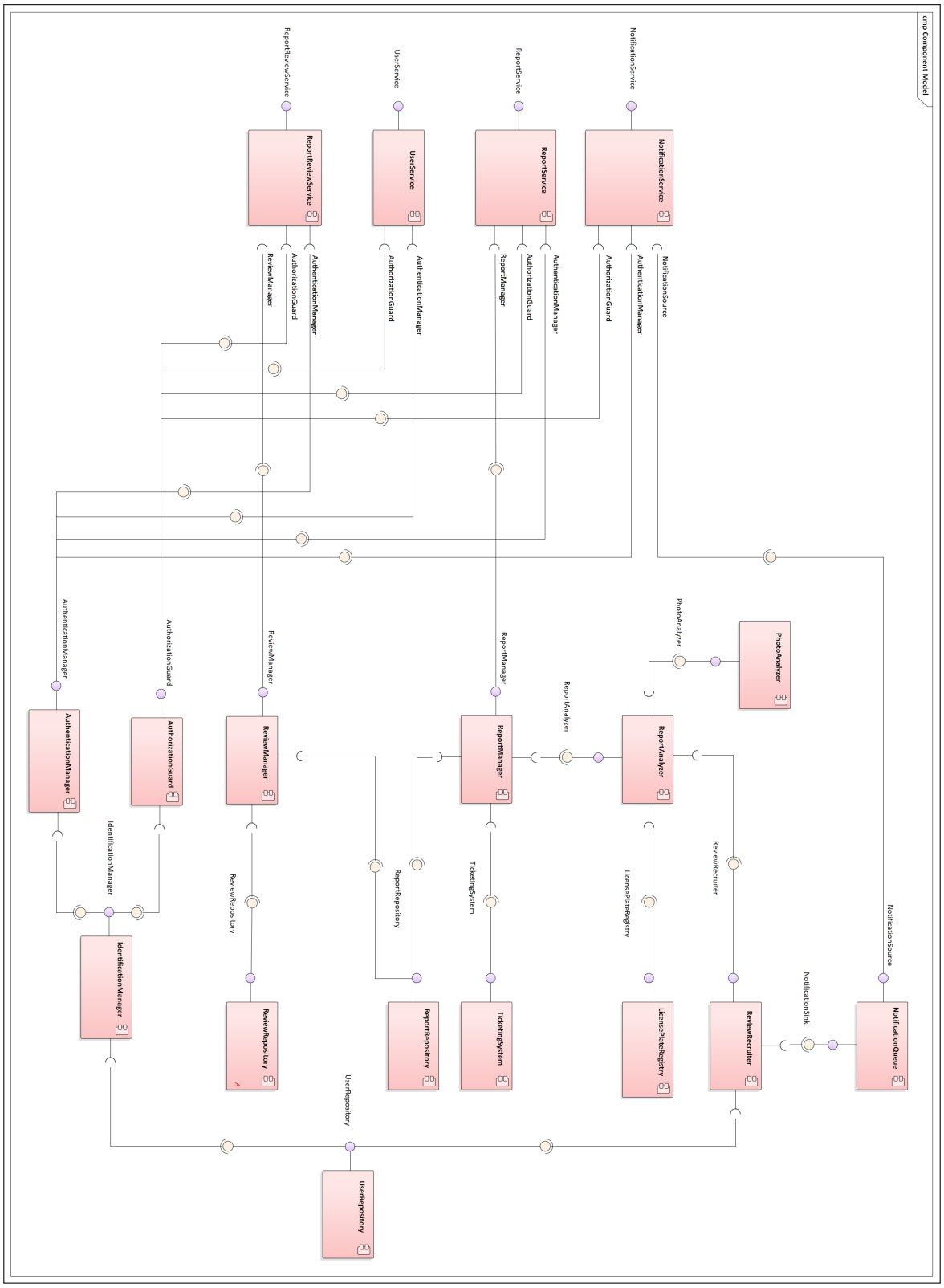

Figure 2: Component model (Overview).

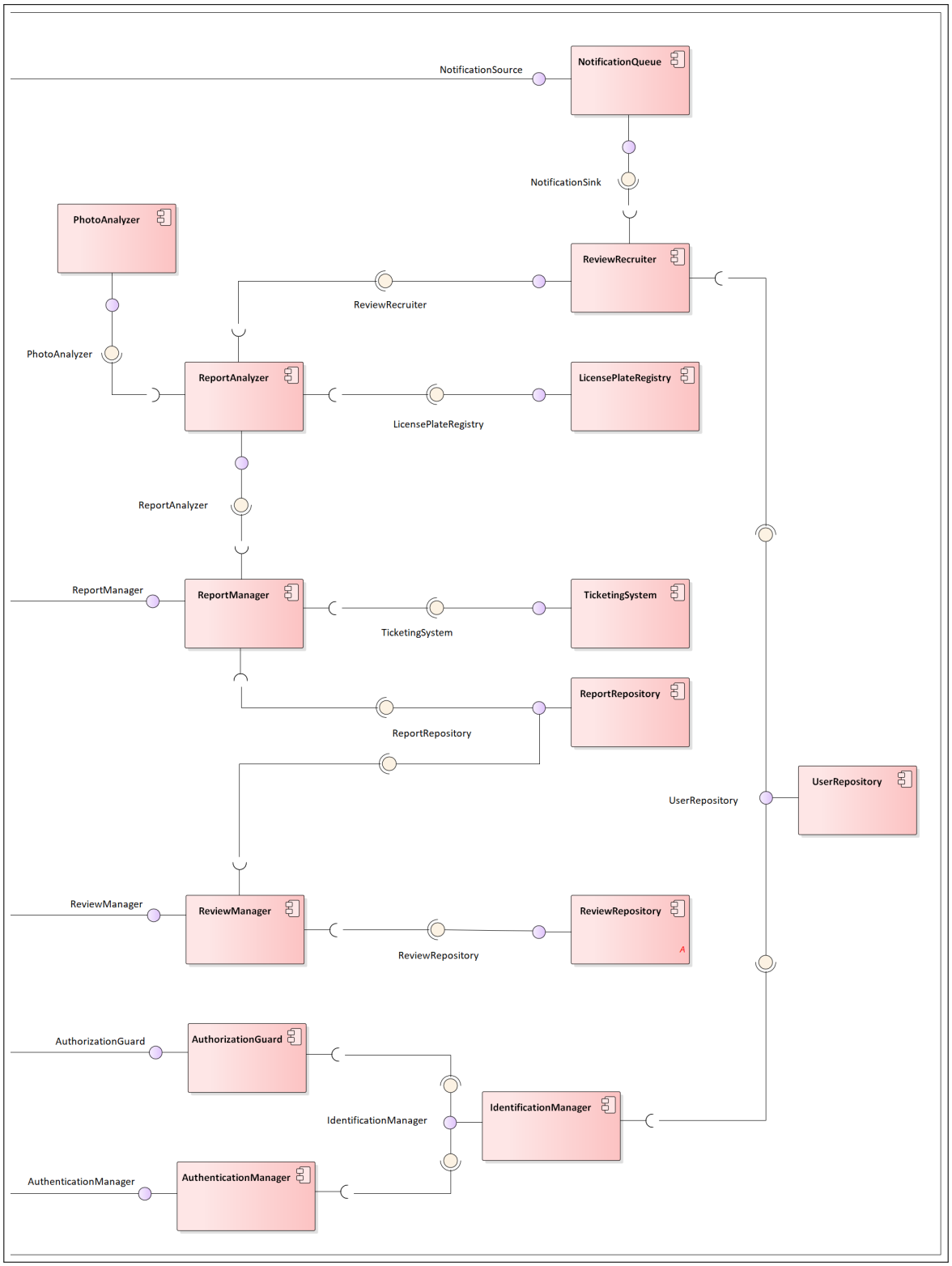

Figure 3: Component model (Application server internals zoom-in).

Figure 3 shows the components responsible for the application business logic, mainly report submission and analysis, but also authorization and authentication. The purpose of each component is the following (going top-down, left-right):

- NotificationQueue: queues notifications to users so they can be received in the order they were

generated when the user requests them.

- PhotoAnalyzer: detects license plate in a given photograph along with the car it is attached to.
- ReviewRecruiter: recruits users to review a given photograph with poor license plate detection by sending them notifications.
- ReportAnalyzer: takes a submitted report and performs an analysis on it to determine its validity and confidence in its veracity.
- LicensePlateRegistry: communicates with the external license plate registry to obtain information about a car given a license plate.
- ReportManager: contains the logic needed to satisfy report submissions and queries.
- TicketingSystem: communicates with the external ticketing system to obtain ticket information about a given license plate.
- ReportRepository: handles interaction with the DBMS regarding reports.
- UserRepository: handles interaction with the DBMS regarding users.
- ReviewManager: manages review submissions and updates the status of a reviewed report accordingly.
- ReviewRepository: handles interaction with the DBMS regarding reviews
- AuthorizationGuard: checks whether a given token or API key is authorized to perform a certain request.
- IdentificationManager: assigns identification tokens and can extract information from them, used for authentication purposes.
- AuthenticationManager: responsible for the authentication flow needed to use the application, registering as a new user, logging in and obtaining profile information.

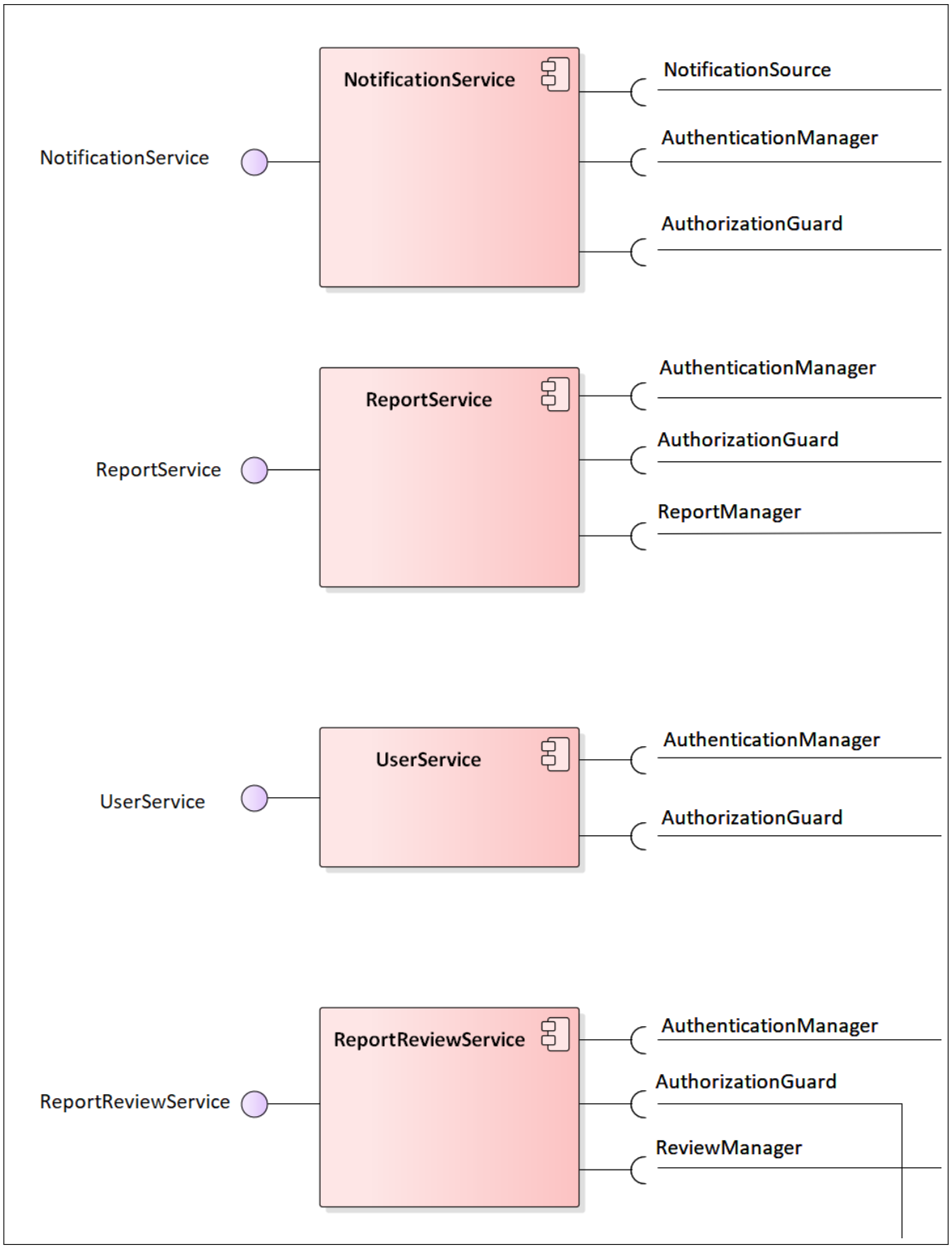

Figure 4: Component model (Services zoom-in).

Figure 4 goes into the four services exposed by the application server. These provide the functionality needed to the mobile application and allow other systems to integrate to SafeStreets. To identify who is performing the request and maintain security, all services are connected to and use the AuthorizationGuard and AuthenticationManager. The services (from top to bottom) provide the following functionality:

- NotificationService: allows a user to obtain their pending notifications.
- ReportService: report submissions and queries of report data.
- UserService: authentication flow to identify a user of the mobile application.
- ReportReviewService: submissions of report reviews.

2.3 Deployment view

As stated previously, the system presents a three tier architecture.

- Tier 1 is the presentation tier, where the mobile app is installed on compatible Android or iOS phones. Distribution of the appropriate executable would be handled by the corresponding app stores. Deployment on both platforms is important to be able to reach the greatest amount of possible users.
- Tier 2 contains the business logic and is the home of the application server. To facilitate deployment and testing under controlled conditions, the executable itself is isolated from the underlying server OS through Docker. This minimizes the setup needed after a container is built and avoids any possible conflict with the OS.
- Tier 3, persistence, then includes the database management system, which is again deployed using a docker container because of all the advantages stated before.

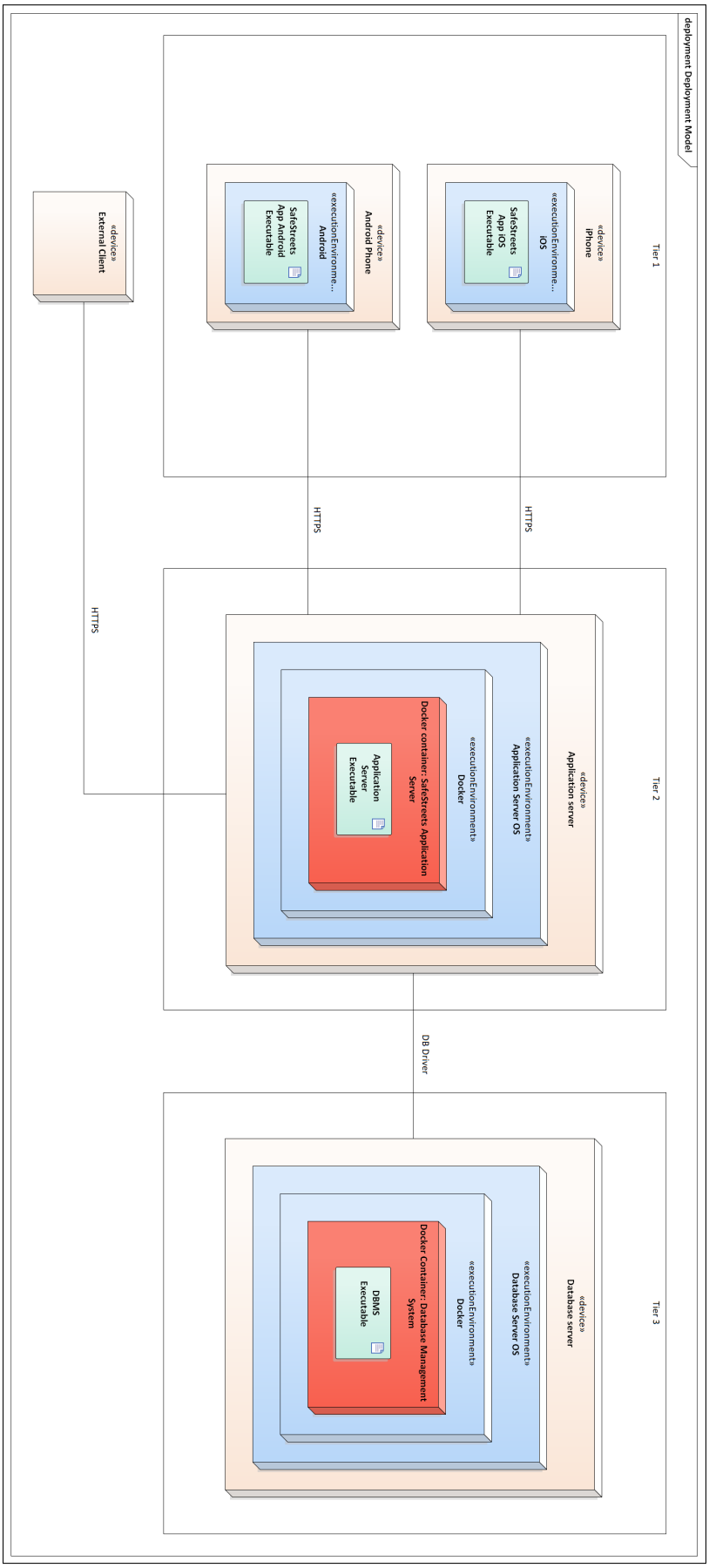

Figure 5: Deployment diagram.

Figure 5 shows a diagram of the deployment of all three tiers, along with their form of communication, which consists of HTTPS from the mobile application to the business logic tier, and the corresponding database driver from the business logic tier to the persistence tier.

External to the system a separate device is depicted, indicating third party system or some other kind of API user, communicating directly with the application server via HTTPS.

2.4 Runtime view

To avoid repetition and explain each process more clearly, the first two sequence diagrams present the authorization procedures that are executed by all services on each request. These are needed to guarantee that whoever is performing a request has the proper authorization and the system can answer it. As this process is not tied to a particular component, a generic "Service" component represents the corresponding service answering the request and a "SystemComponent" is what the service will be using to generate the response.

2.4.1 Token authorization

In this sequence, the general process of authorizing a user request through a token is shown. This process is executed by each and every one of the services that provide a token interface to the user. Any particular action has an access level that needs to be equal or lesser than the token access level, this is checked by the AuthorizationGuard, which asks the IdentificationManager for information about the token. If this is correct and the user is found, then the action is executed by the corresponding system component.

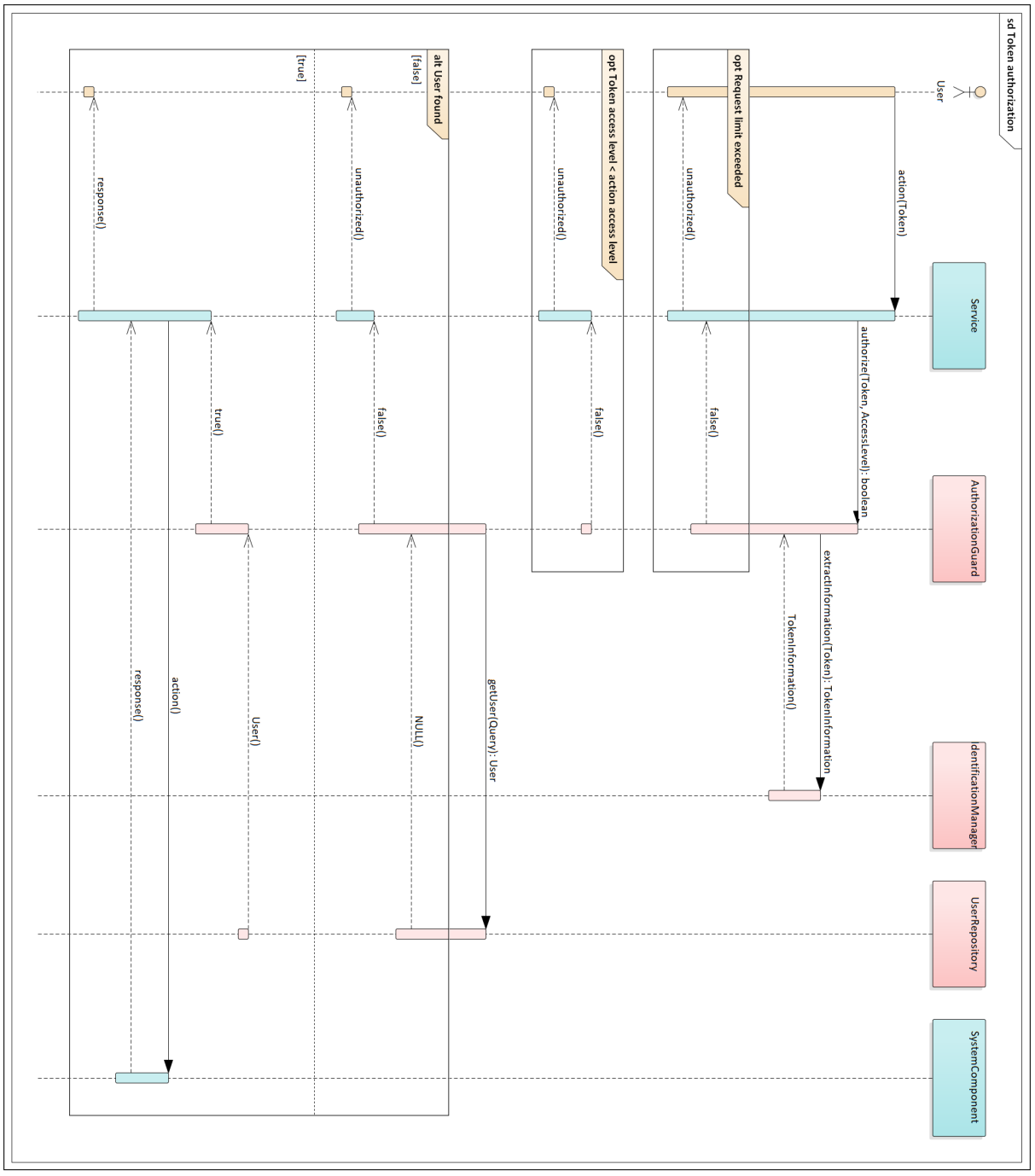

Figure 6: Sequence diagram - Token authorization.

2.4.2 Key authorization

If the user wants to make use of the public API, then instead of a token, an API key is provided. As with the token authorization, this is performed by every service that provides an interface with an API key. The process is the same as before with the difference that since the user is not logged into the application, it does not have to be retrieved from the system. Also, the possible actions performed through the public API do not need to identify the user performing them as they consist of queries only.

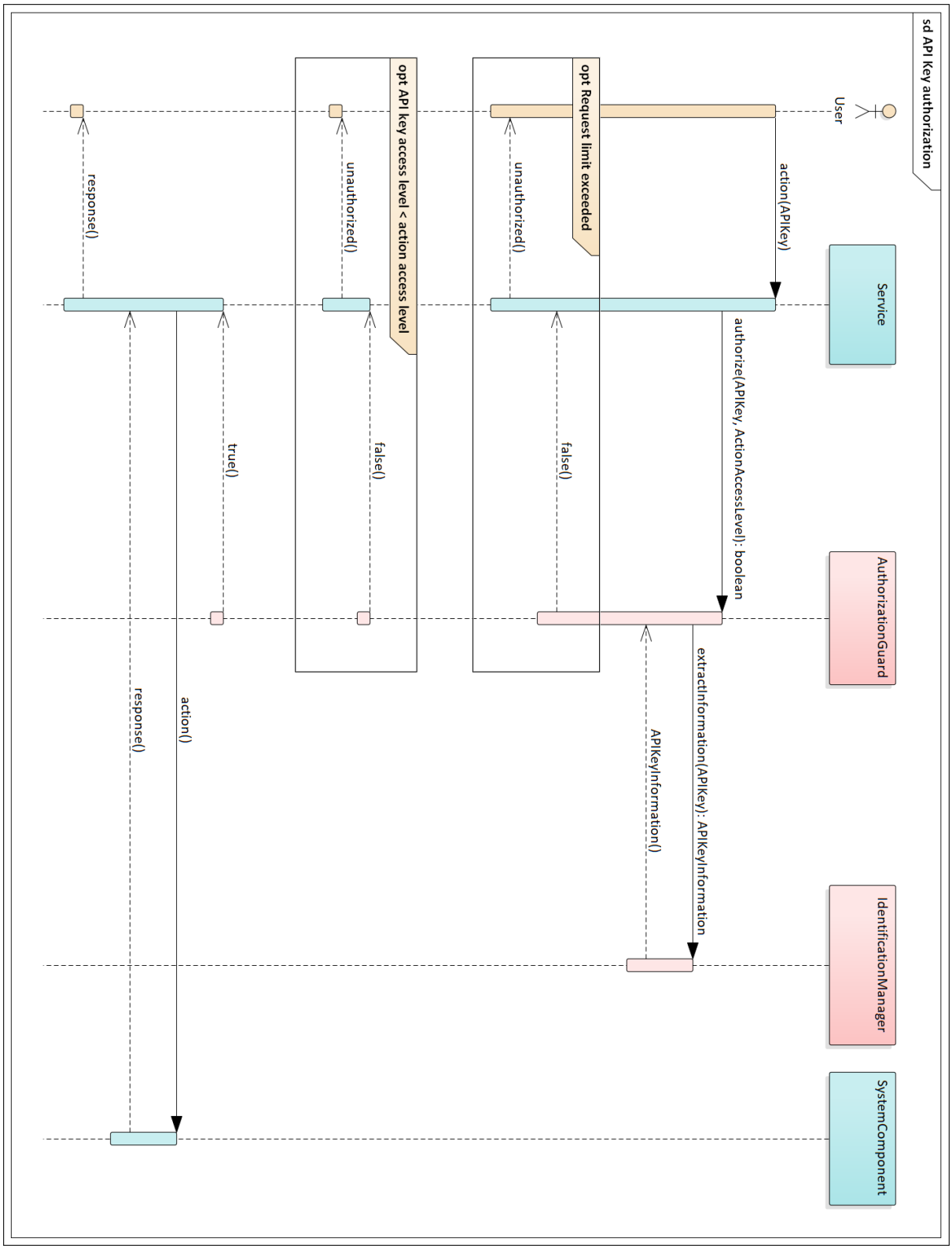

Figure 7: Sequence diagram - API Key authorization.

2.4.3 Report submission

The following sequence describes the process of submitting a report. The ReportService is in charge of executing the action. First of all, it gets the user from the AuthenticationManager through the token and

gives the order to the ReportManager to submit the report. The ReportManager gets the ReportAnalyzer to analyse the report. If the report is invalid, then it is saved through the ReportRepository with an explanation of the error and an error message is returned to the user. If the report is valid, then it is saved along with the analysis result and an ok code is returned to the user.

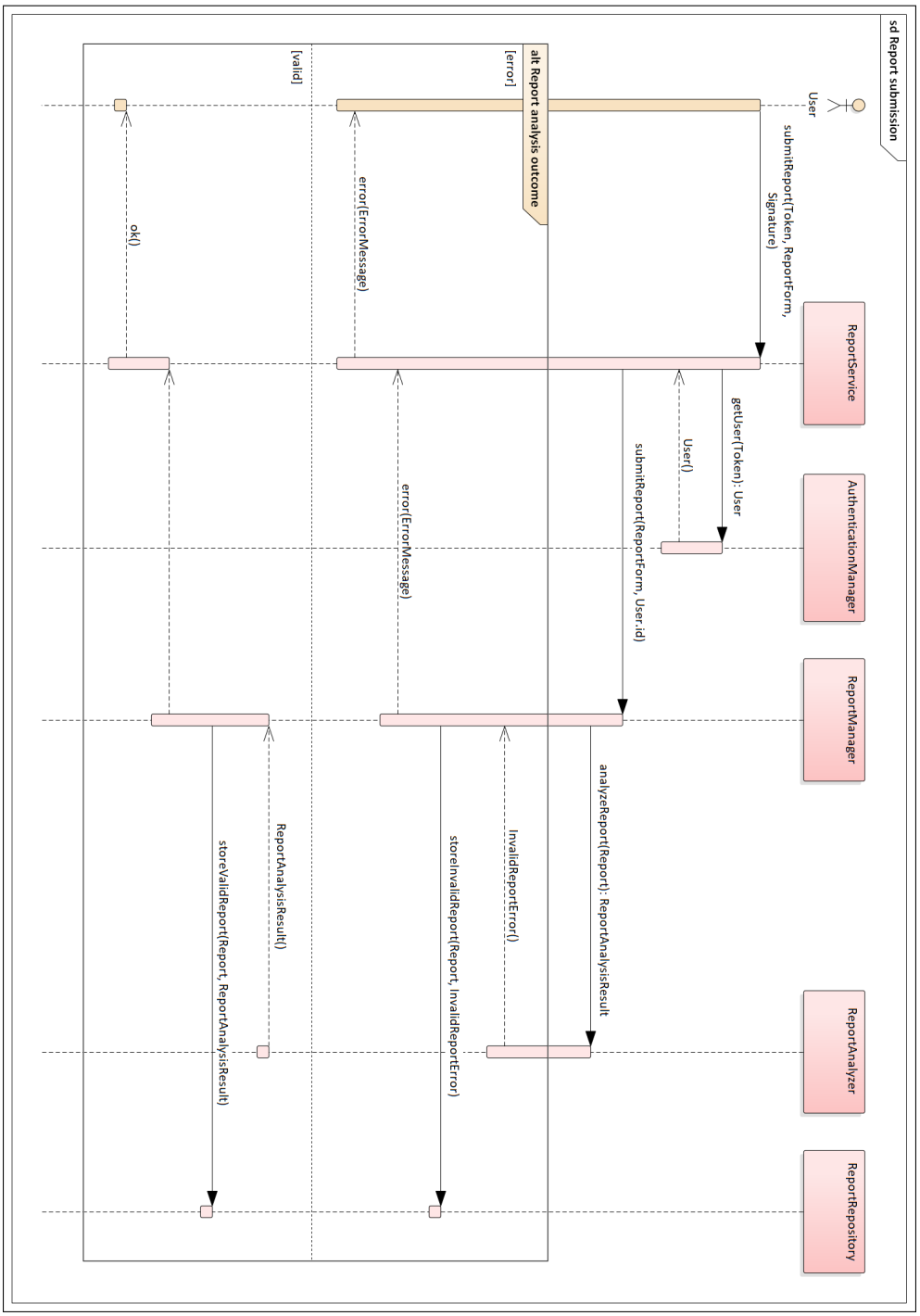

Figure 8: Sequence diagram - Report submission.

2.4.4 Report analysis

Here the previously mentioned report analysis is explained with more detail. The ReportManager gives a report to the ReportAnalyser for its analysis. The ReportAnalyser uses the PhotoAnalyser to get car and

license plate detections from the report photo. The report is considered invalid and an error is returned if no license plates are detected, if none of them match with the one provided in the report or if no cars are detected. If a licence plate matches but the confidence of the PhotoAnalyser analysis is below 80%, then the report is sent to the ReviewRecruiter for user review and an in-review result is returned to the ReportManager. The car detection is also required to have a confidence of at least 80%, or the analysis result is low-confidence. If the car detection is ok, then the LicensePlateRegistry is consulted for information about the car with the detected license plate. This information is compared with the detected car; if they match, the result is a high-confidence report, if they do not, the result is a low-confidence report.

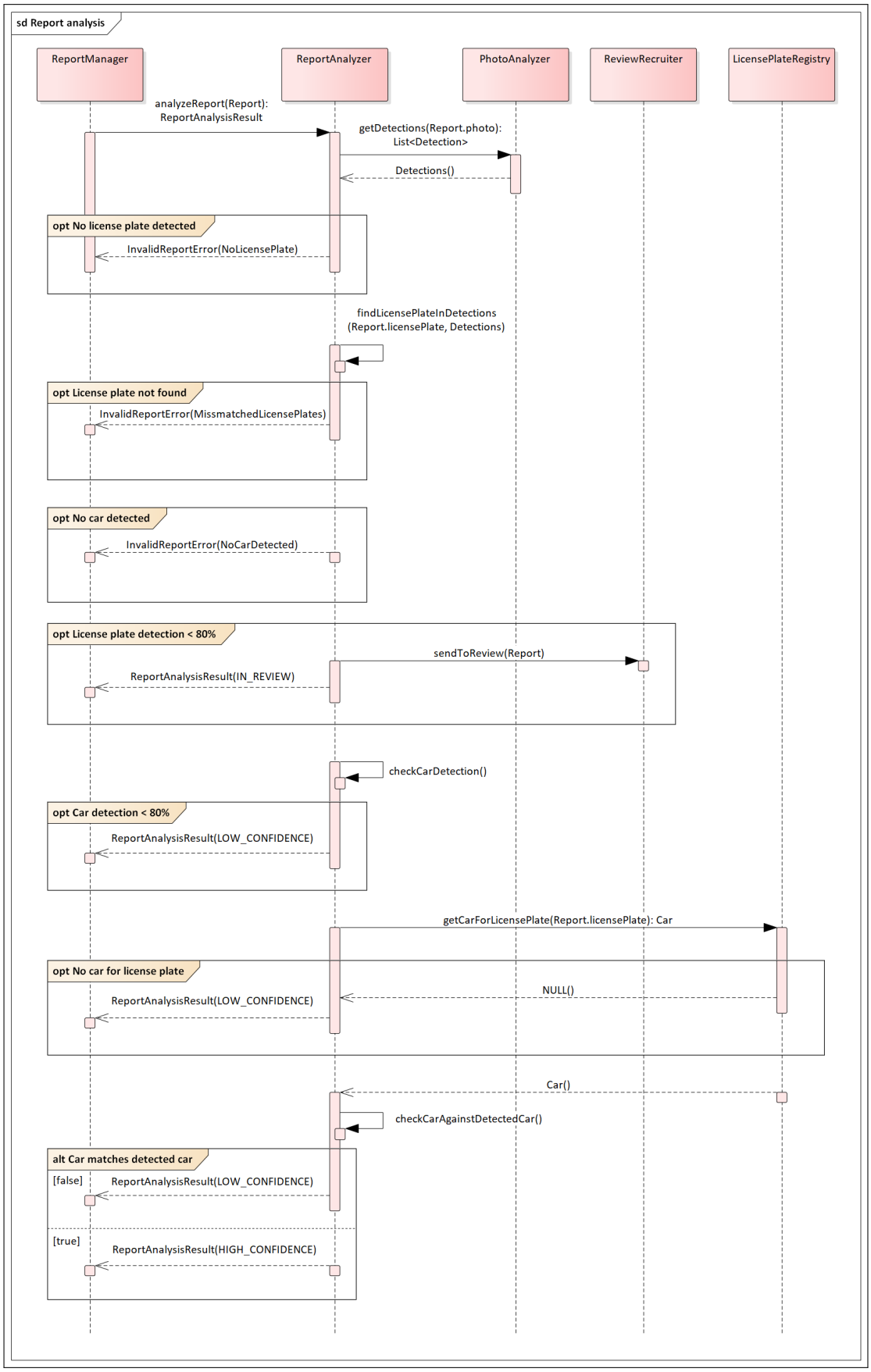

Figure 9: Sequence diagram - Report analysis.

2.4.5 Review recruitment

The recruitment process is simple. The ReviewRecruiter gets the list of users from the UserRepository, then selects the suitable users from this list. A notification is enqueued for each user in the NotificationQueue.

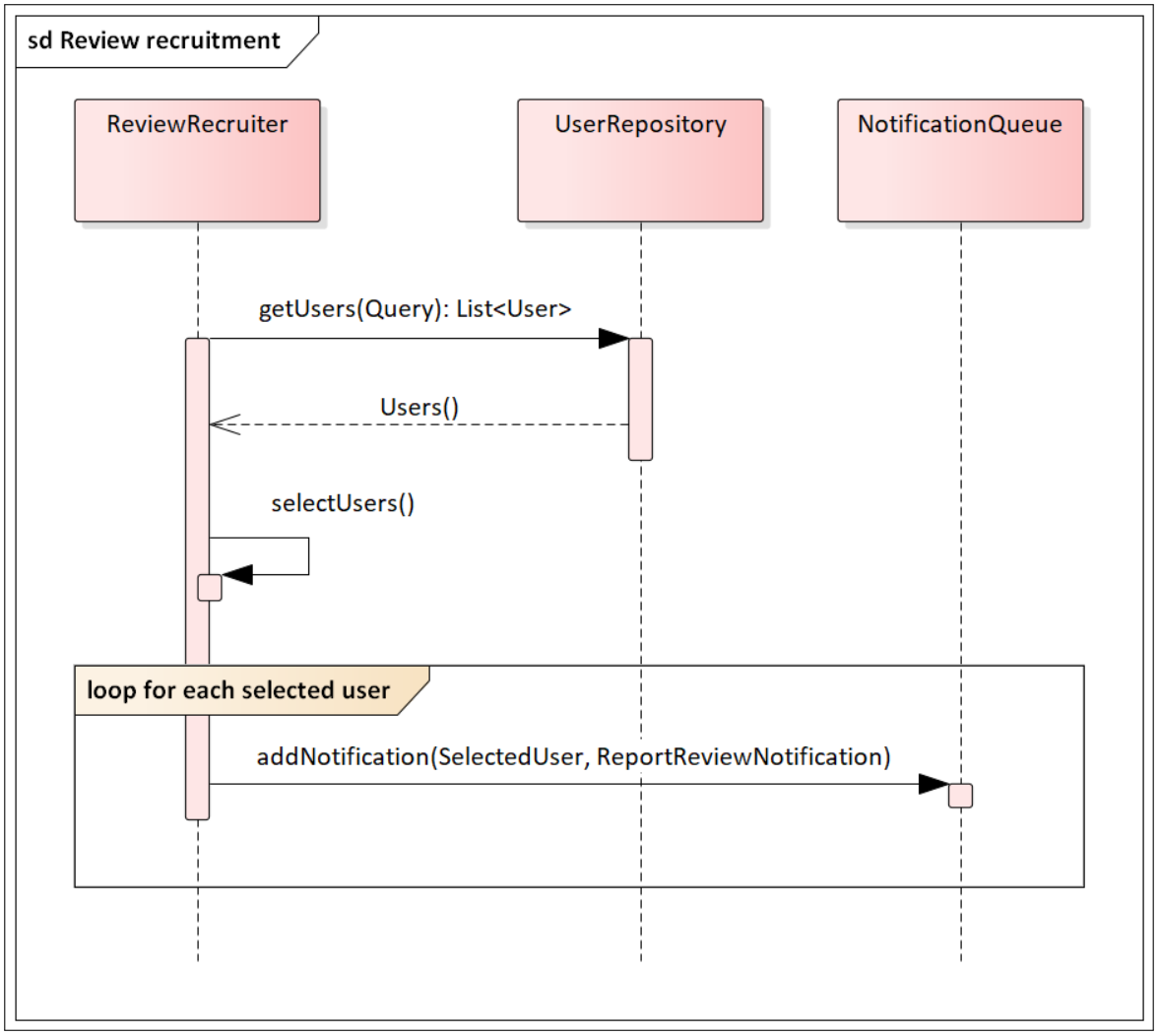

Figure 10: Sequence diagram - Review recruitment.

2.4.6 Get notifications

In this sequence, the user asks for their pending notifications. The NotificationService first gets the user from the AuthenticationManager and then retrieves their pending notifications from the NotificationQueue.

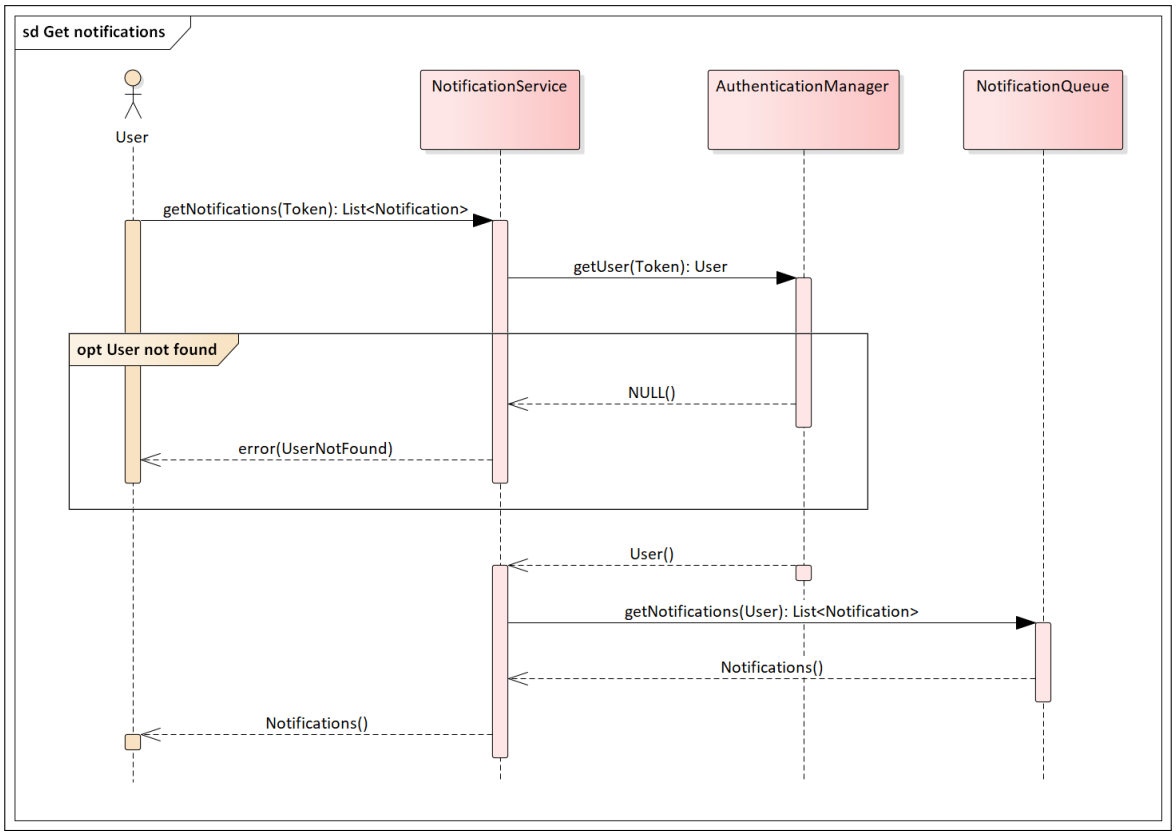

Figure 11: Sequence diagram - Get notifications.

2.4.7 Review submission

When the user submits a review, the ReportReviewService receives the request and passes the review to the ReviewManager. The ReviewManager stores the review via the ReviewRepository and checks whether the amount of reviews is higher or equal to the decision threshold. If this is not the case then the sequence ends returning an ok code to the user. Otherwise, the report is retrieved from the ReportRepository and is submitted to the ReviewManager along with its review match percentage. Then the ReviewManager interacts with the ReportAnalyzer and the ReportRepository to classify the report as valid or invalid and save the result. An ok code is returned to the user.

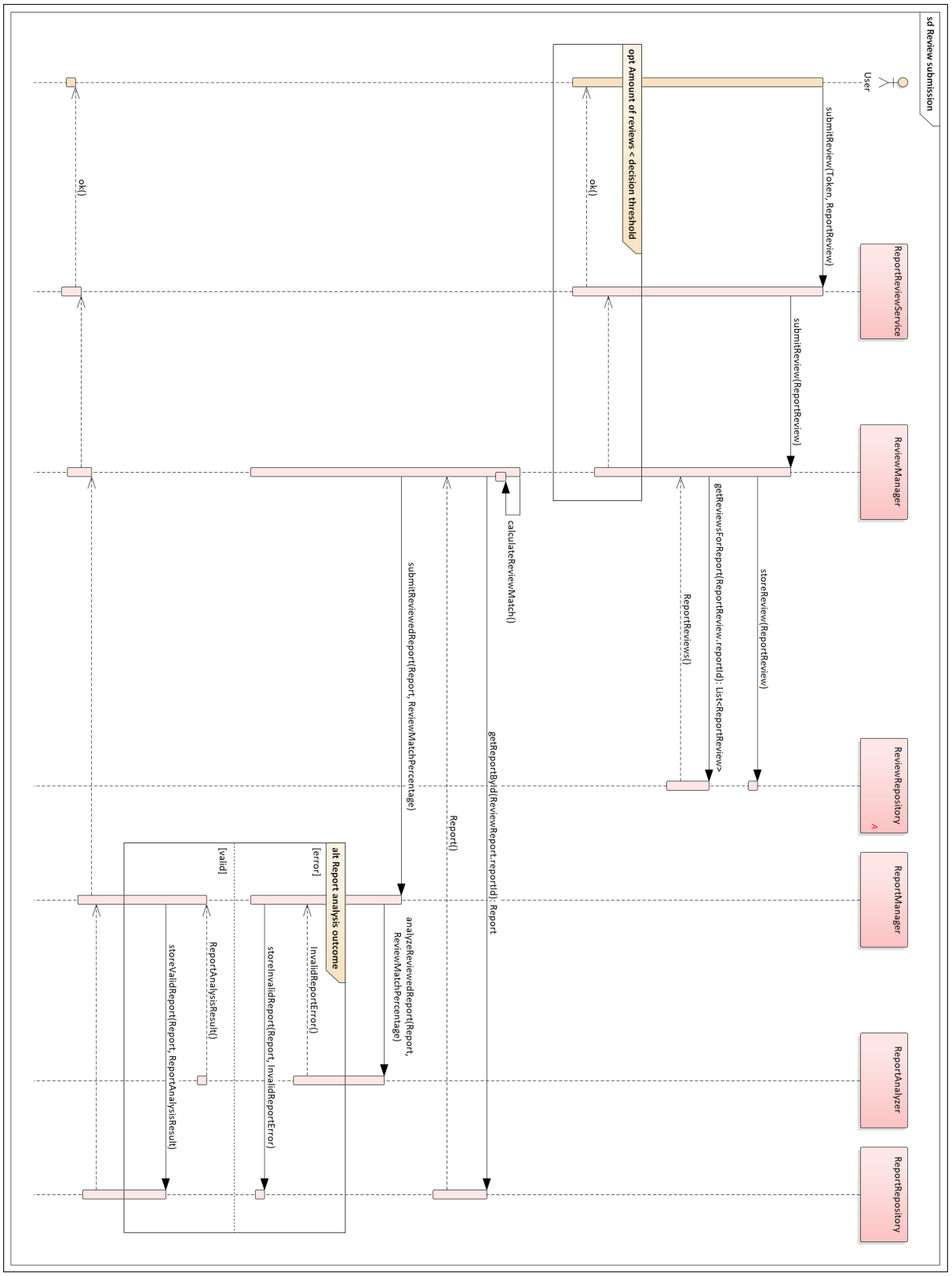

Figure 12: Sequence diagram - Review submission.

2.4.8 Reviewed report analysis

This sequence is similar to the report analysis, but the review matching percentage is utilized to decide whether the report is invalid or low-confidence at the beginning of the process. If it's considered neither then it proceeds like the normal analysis.

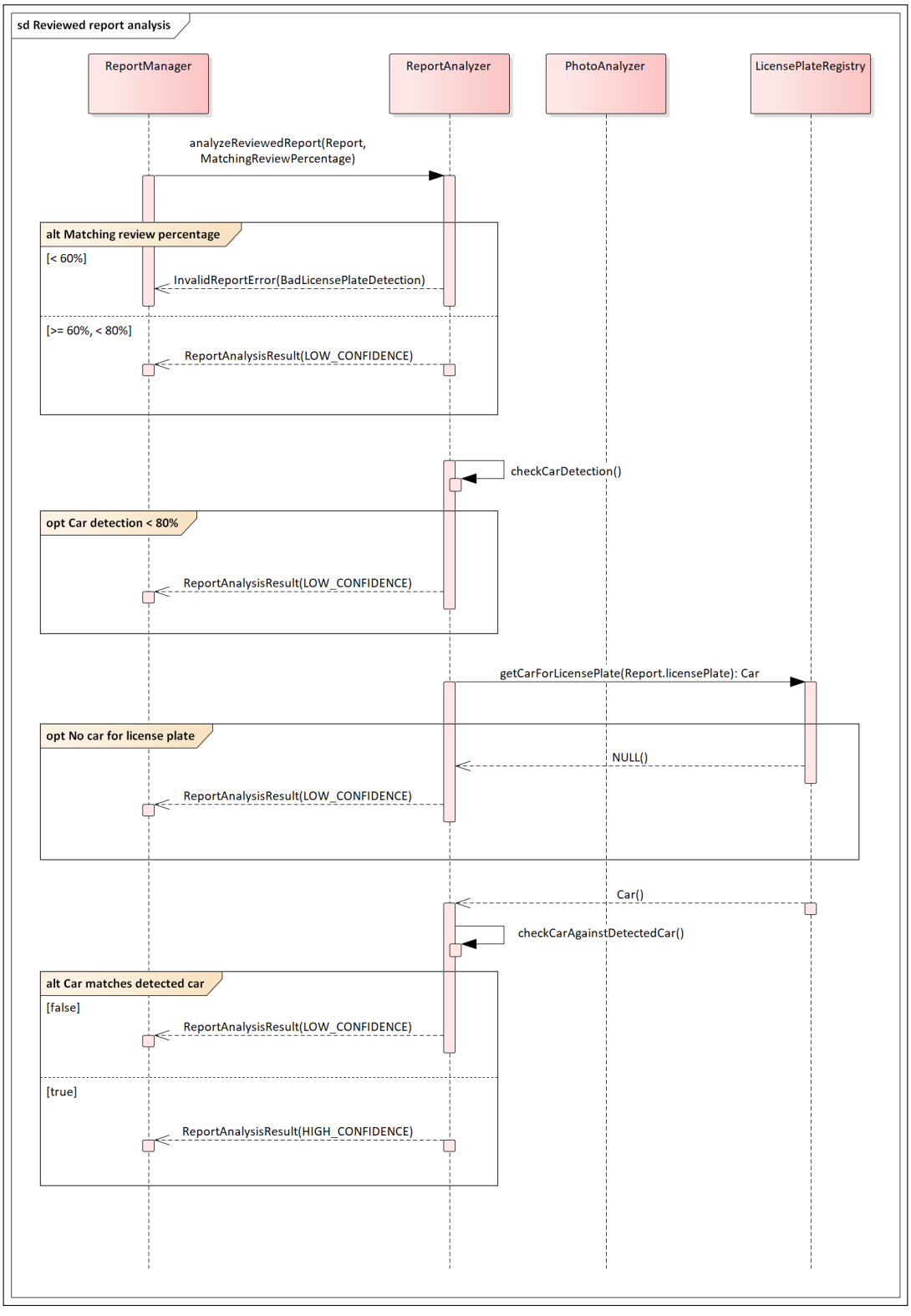

Figure 13: Sequence diagram - Reviewed report analysis.

2.4.9 Get reports

The ReportService handles the request and asks the ReportsManager for the reports. The ReportManager retrieves the reports from the ReportRepository and interacts with the TicketingSystem to check if the reports have related tickets. Afterwards, sensitive information is removed from the report and a list of reports with partial information is returned to the user.

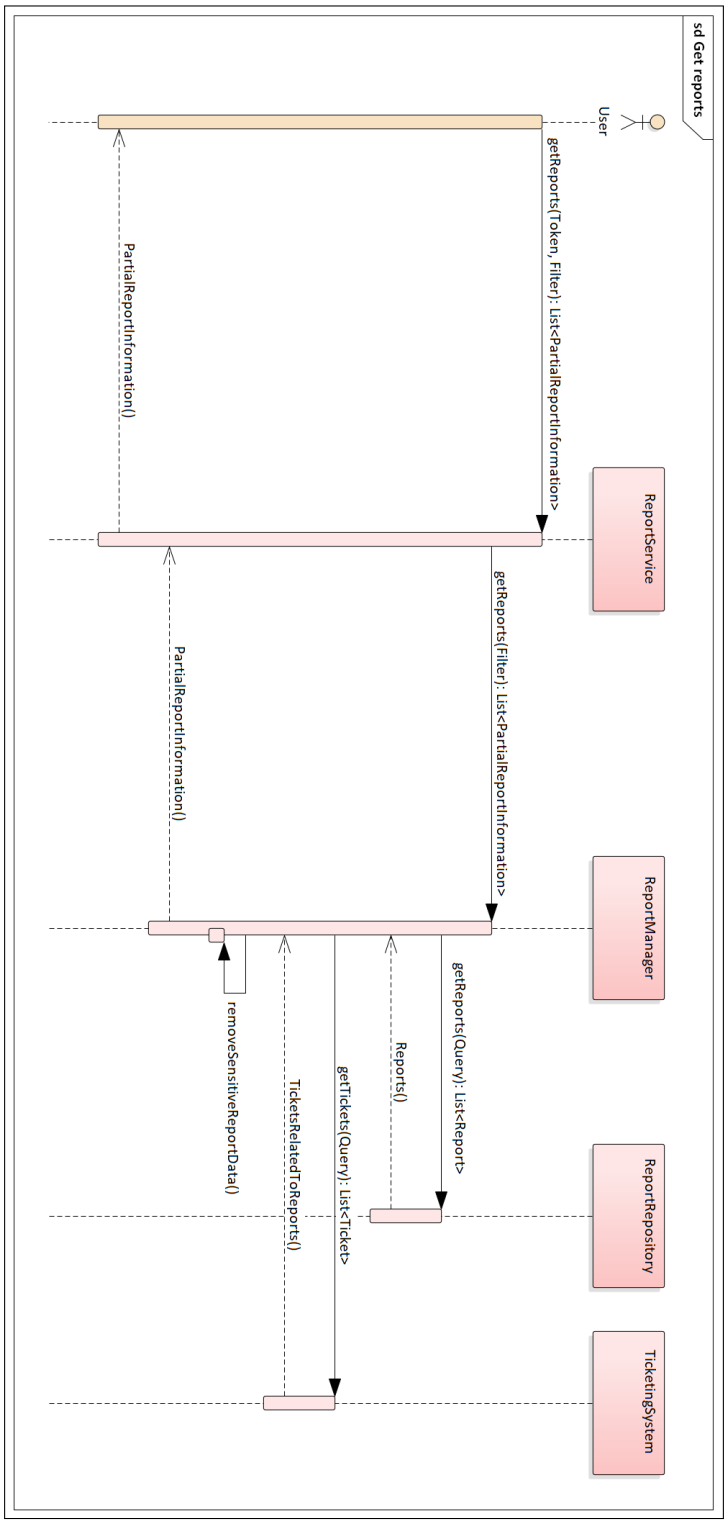

Figure 14: Sequence diagram - Get reports.

2.4.10 Get full reports

This process is similar to the previous one, but no information is removed from the report, thus including all photos in the report and the target license plate.

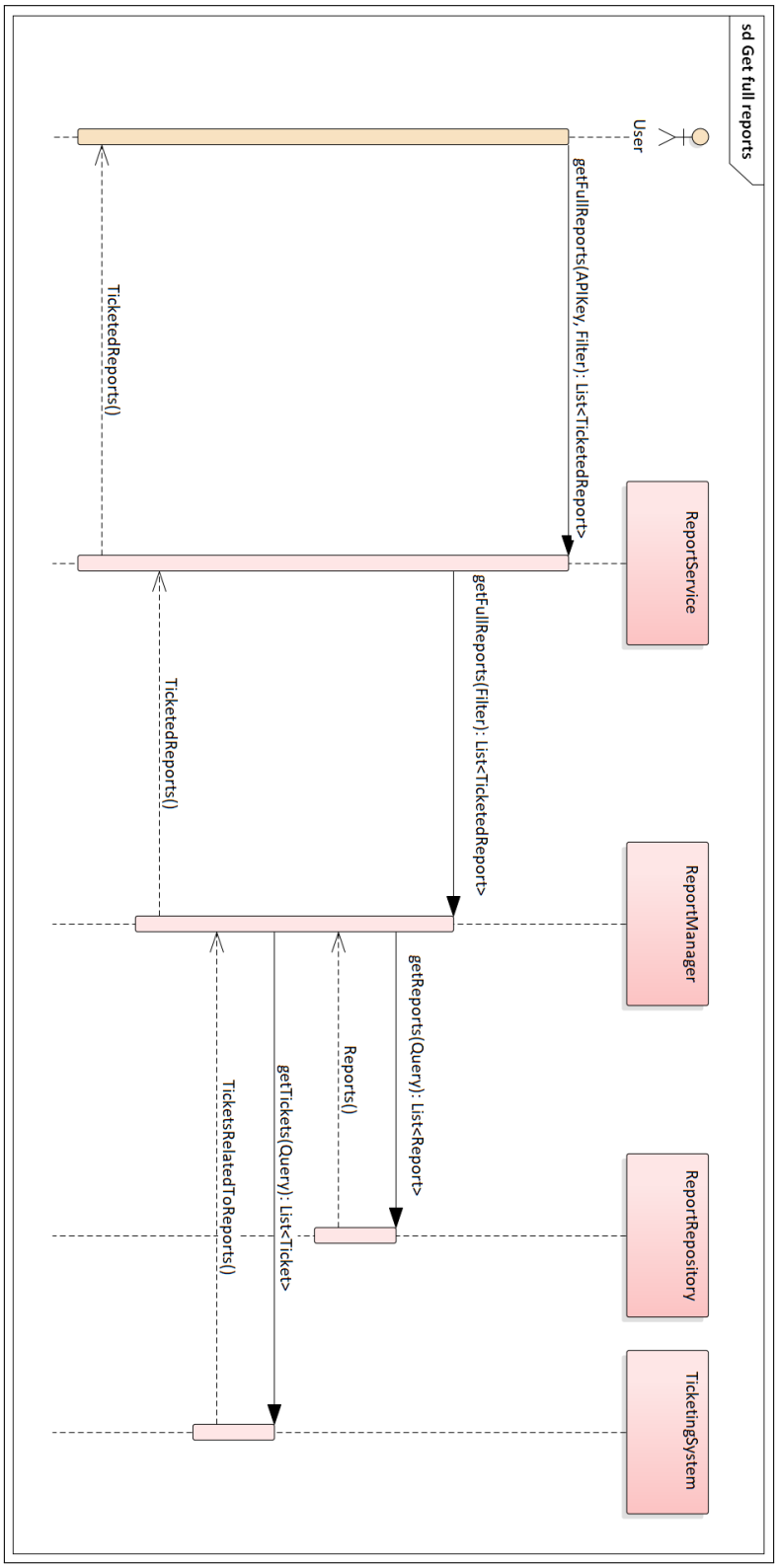

Figure 15: Sequence diagram - Get full reports.

2.4.11 Registration

This is the standard sign up procedure needed for the user to use the mobile application and be able to get a token when logging in. The RegistrationForm is submitted to the server, where the Authentication-Manager validates it (valid, non repeated email, secure password) and, if this validation is successful, stores the new user date.

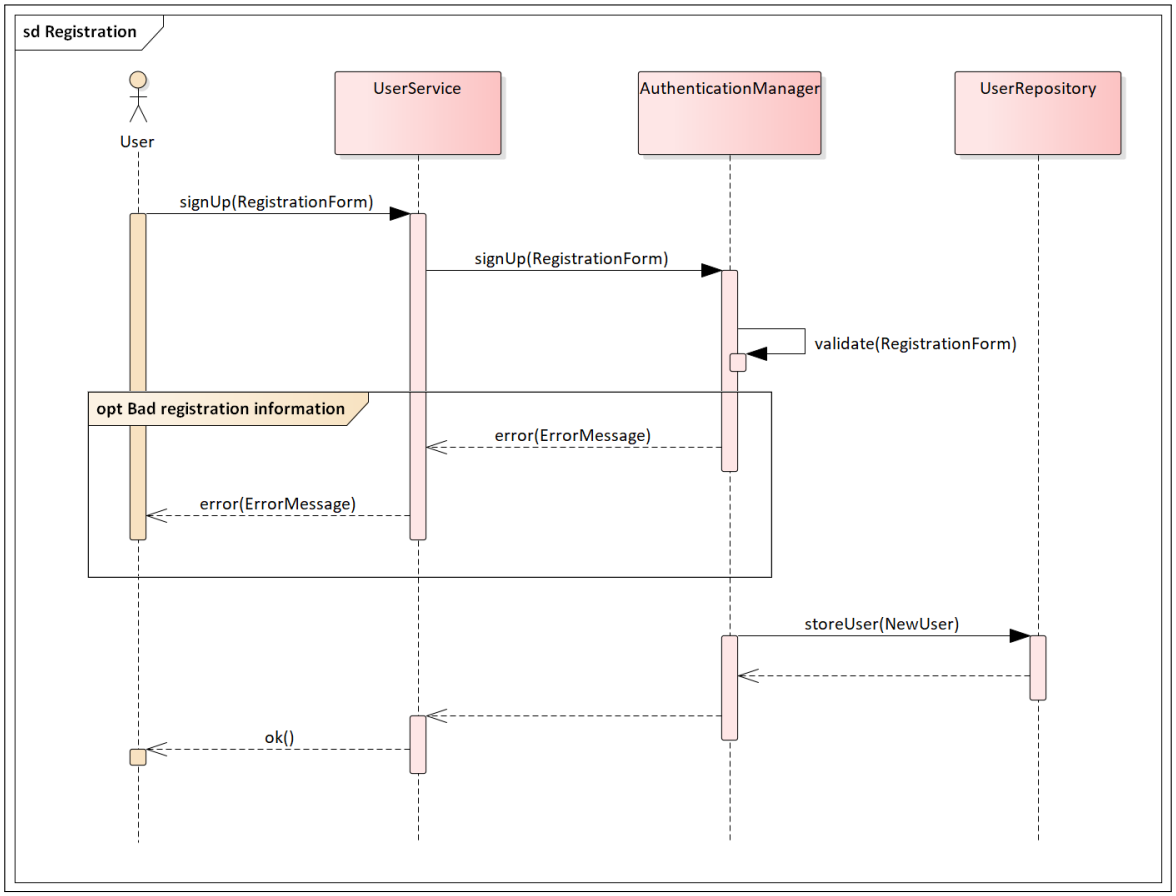

Figure 16: Sequence diagram - Registration.

2.4.12 Login

The user enters their email and password, then the server looks for a registered user with that information, if it is found, a token is generated and returned, otherwise an error message is sent.

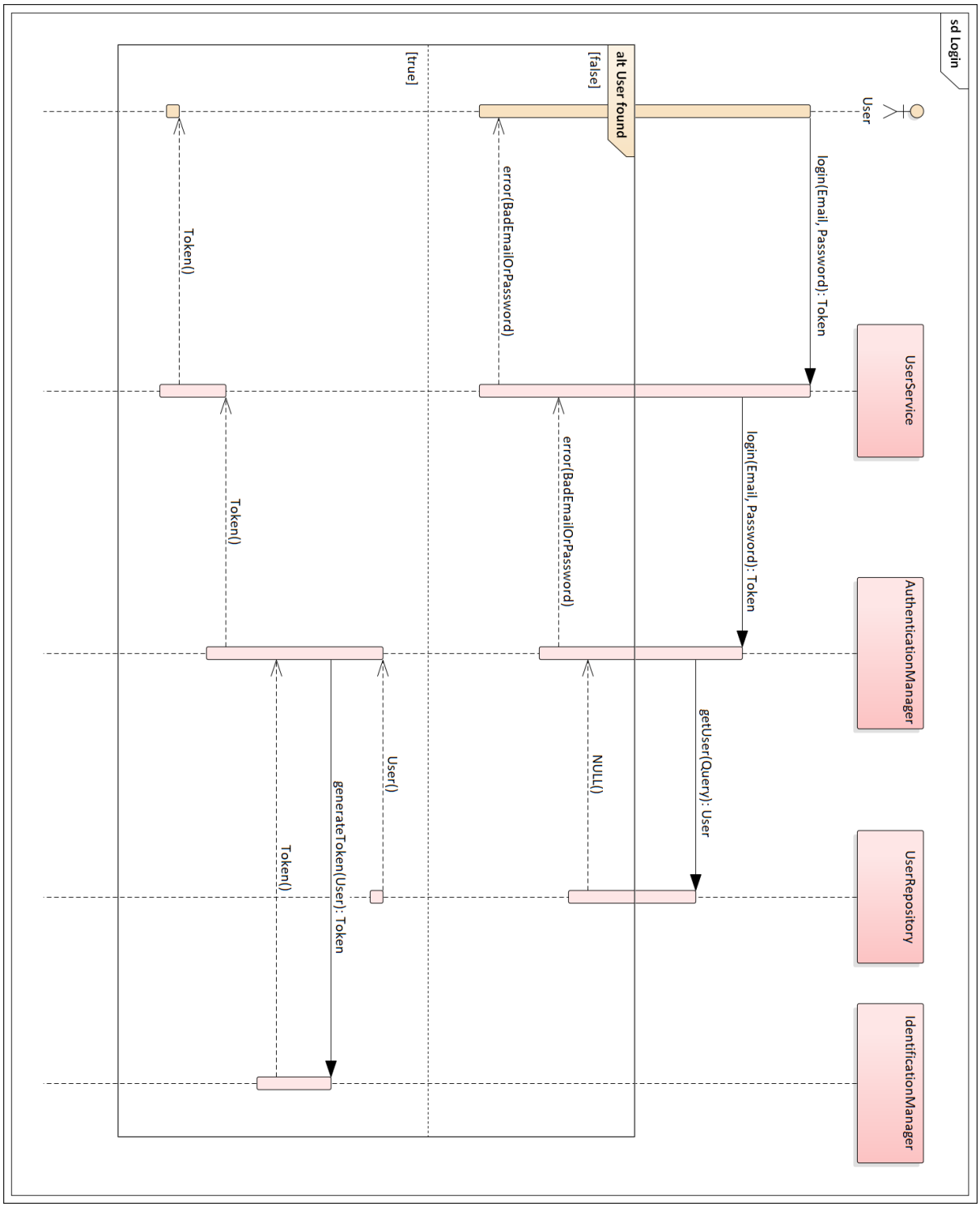

Figure 17: Sequence diagram - Login.

2.4.13 Get user information

The user solicits to obtain their stored information in the system. By looking at the token, the AuthenticationManager can query the UserRepository and provide this data if a user is found, otherwise an error message is sent (realistically speaking, this will never fail as the token has to first be verified by the AuthGuard for authorization purposes).

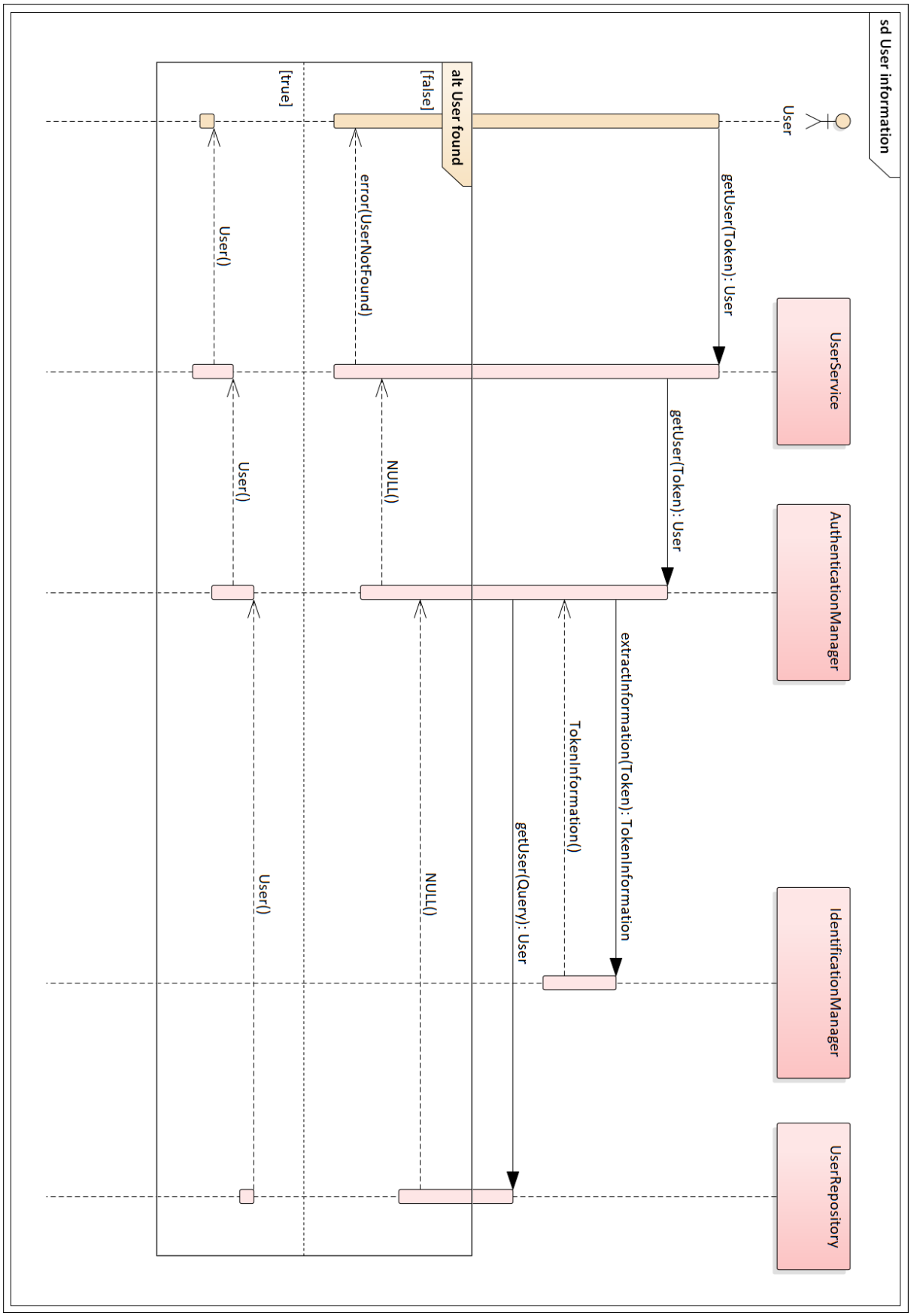

Figure 18: Sequence diagram - User information.

2.5 Component interfaces

The interfaces shown in Section 2.2 which connect the components internally and expose the public API are detailed here. Each component introduced there is included here, along with the interface it implements. Connections between the interfaces and the component that uses them are not included, as they are already present in the component diagram of Section 2.2.

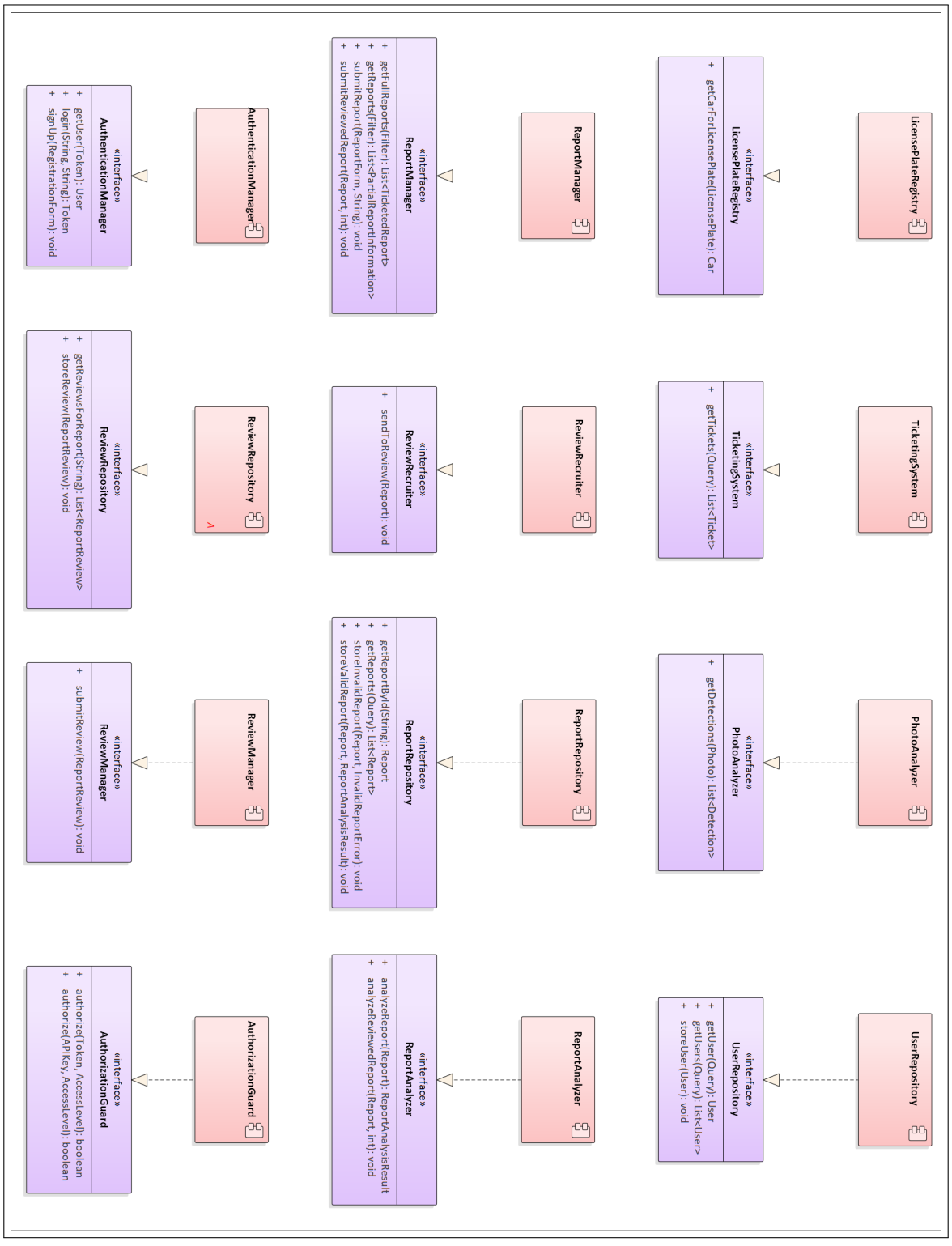

Figure 19: Component interfaces (1/2).

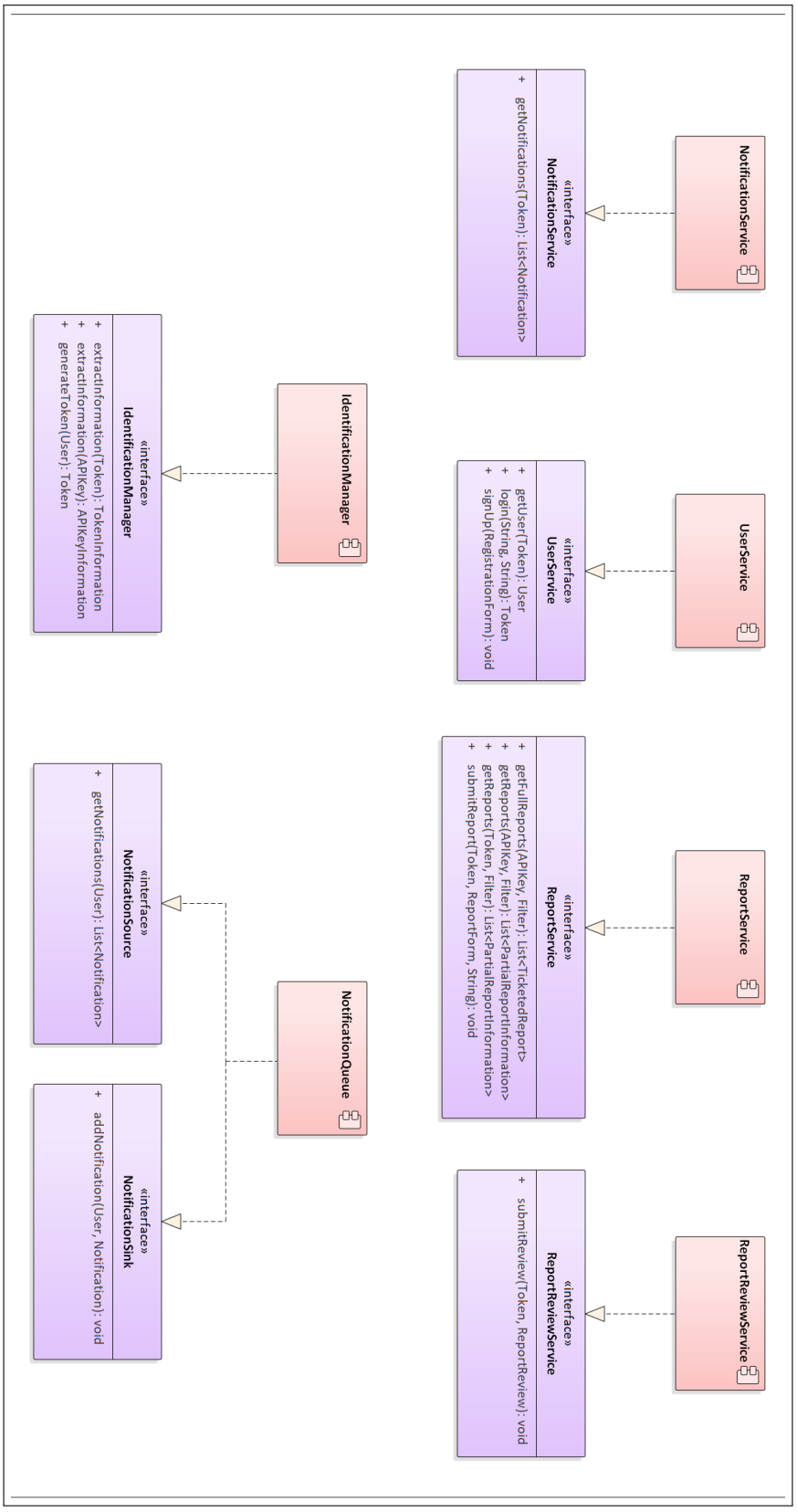

Figure 20: Component interfaces (2/2).

Data model The following diagram shows the crucial information contained in the data structures used by the component interfaces.

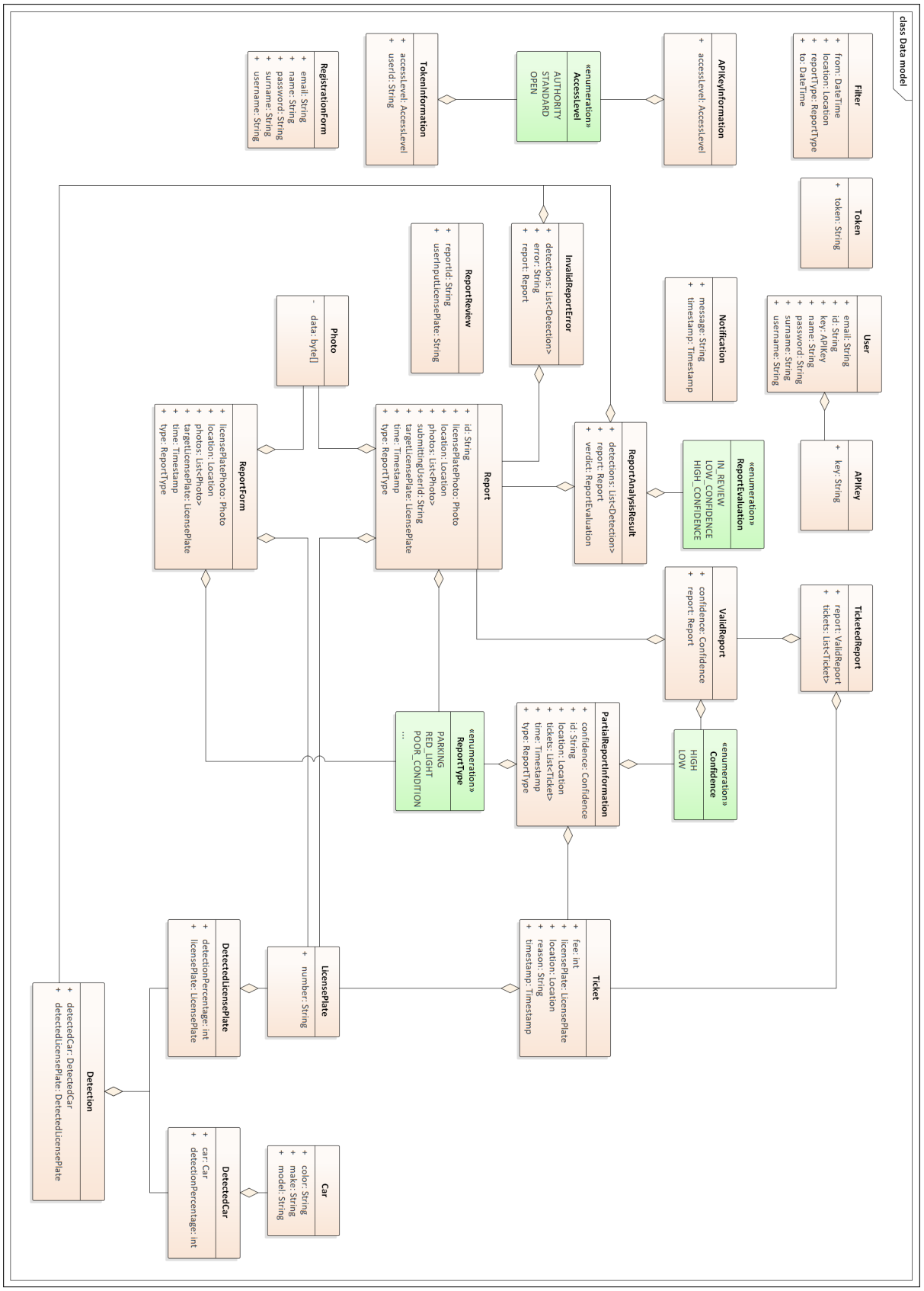

Figure 21: Data model diagram.

Note that the Token and APIKey are simply strings, how the information provided by them is ob-

tained depends on the implementation. For example, a JSON Web Token would include all relevant data encrypted in the string itself, but the token could also be used as a key to search in the database. How this is done is not relevant, as long as the needed information can be obtained for authentication purposes.

2.6 Selected architectural styles and patters

2.6.1 Architecture patterns

- **Three tier architecture:** As already mentioned, the SafeStreets system is divided into three tiers: Presentation, Business Logic and Persistence tier.
- **Client-Server architecture:** Computing model in which the server hosts, delivers and manages most of the resources and services that are consumed by the clients.

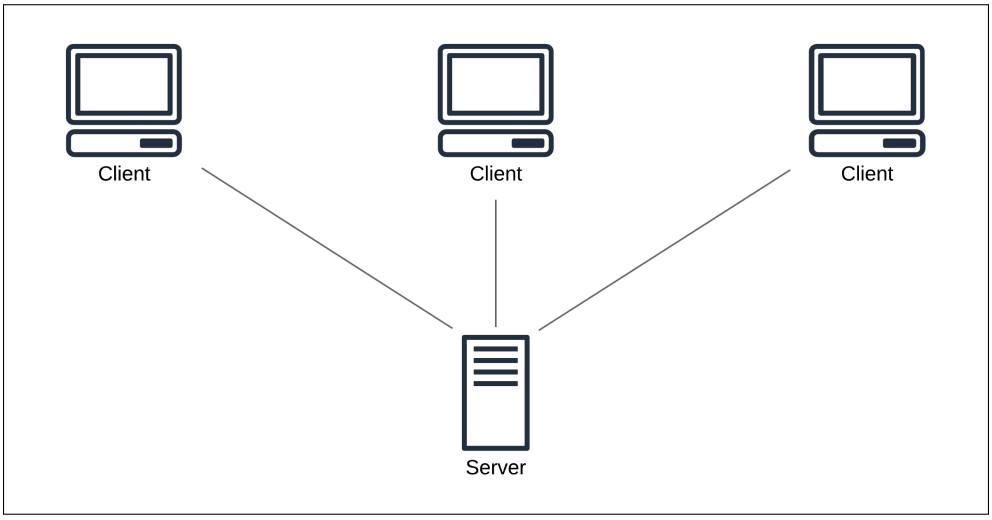

Figure 22: Client-Server architecture.

The mobile devices running the application are the clients, which interact with the business logic tier (Application server); it is here that the heavy computation of the system is done. The “Thin Client Server architecture” can also be referenced.

2.6.2 Design patterns

- **Model-View-Controller:** Divides the program logic into three interconnected elements.
  - Model: Central component. Manages the data, logic and rules of the application.
  - View: Visual representation of the model.
  - Controller: Accepts inputs and converts it to commands for the model or view.

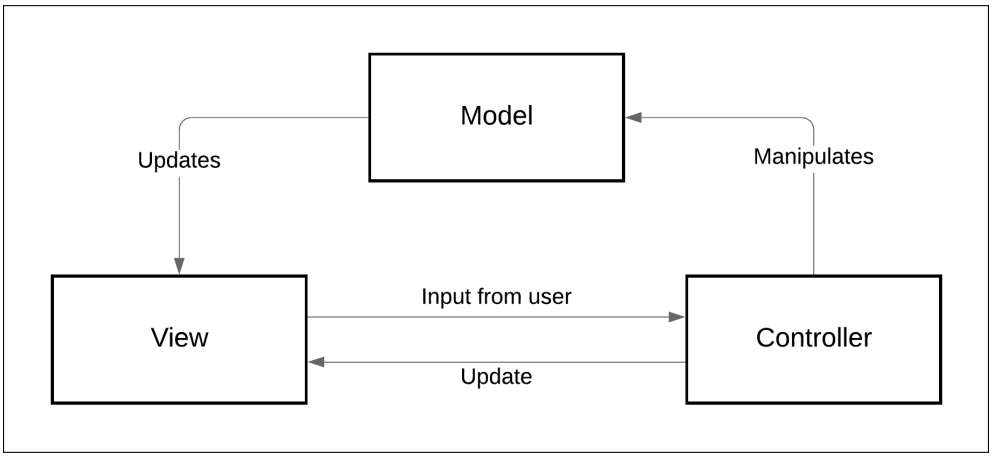

Figure 23: Model View Controller pattern.

This pattern is used in the development of the mobile application, which is the part of the system the user directly interacts with.

- **Facade pattern:** Provides a simple interface to a larger body of code. This is used in the backend to expose the API to external users. The router component acts as the facade, interacting with the lower level components and exposing the appropriate endpoints.

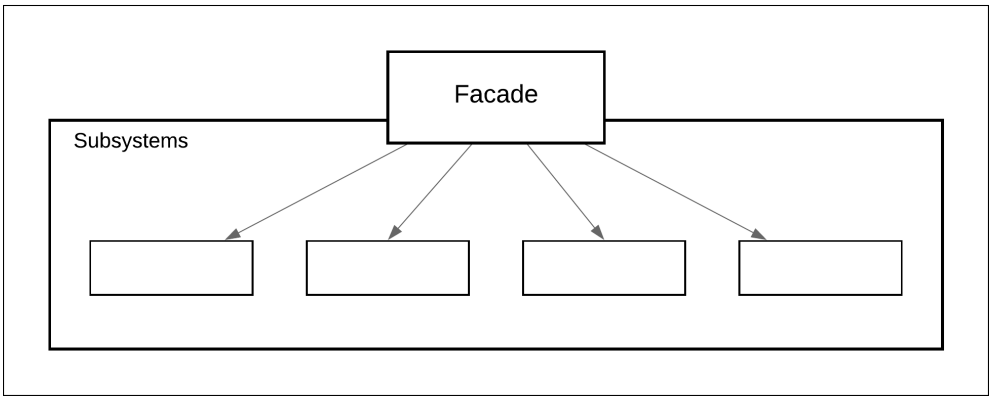

Figure 24: Facade pattern.

- **Dependency injection pattern:** A technique whereby one object supplies the dependencies of another object.
  Utilized in the backend to solve the dependencies between services.

2.7 Other desing decisions

3 User interface design

The following mockups represent the final look of the application in great detail. They are based on the sketches provided in the RASD. Material design principles were used as inspiration in the design process.

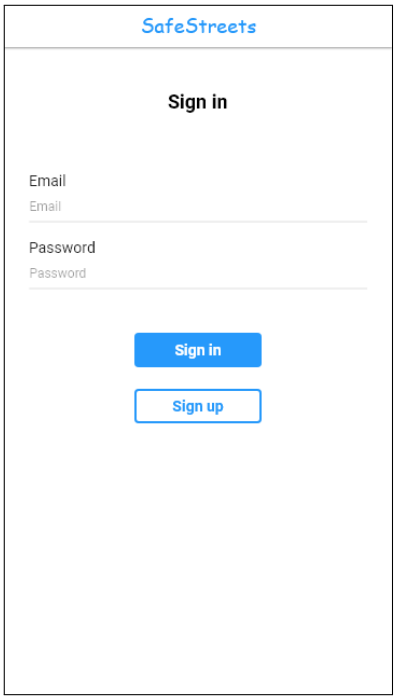

Figure 25: Mockup - Sign in.

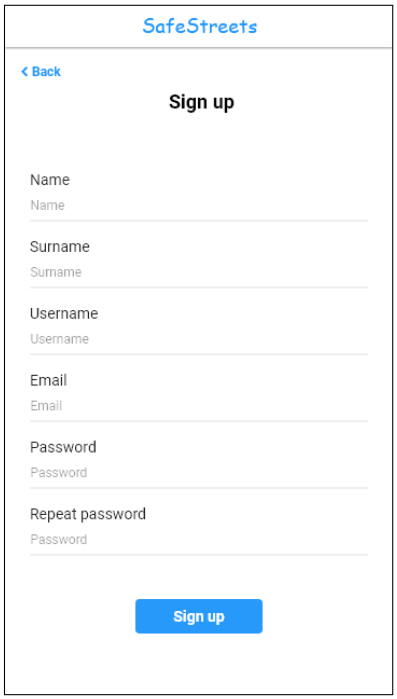

Figure 26: Mockup - Sign up.

Figure 27: Mockup - Home.

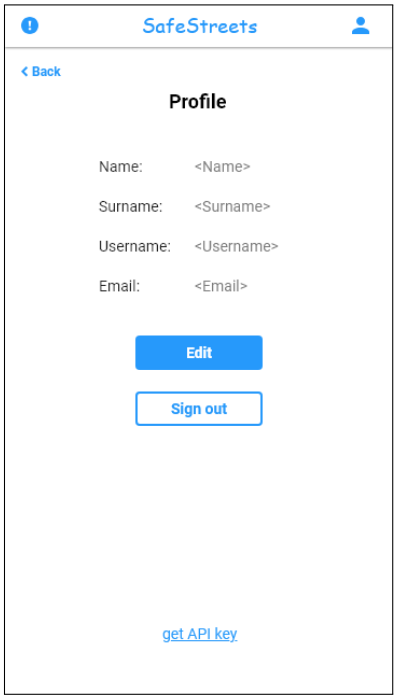

Figure 28: Mockup - Profile.

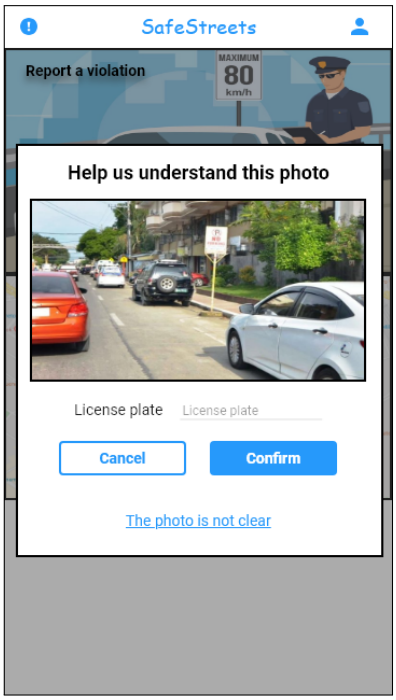

Figure 29: Mockup - Photo review.

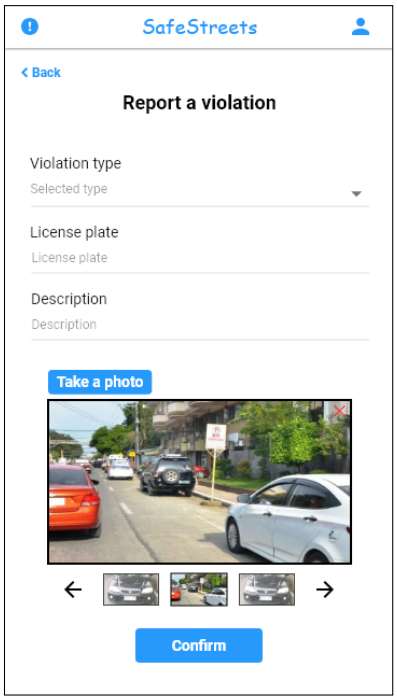

Figure 30: Mockup - Report violation.

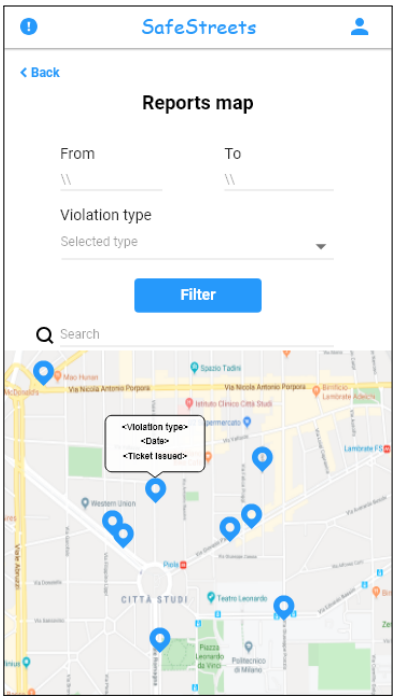

Figure 31: Mockup - Reports map.

Here is a description of each screen and its functionalities:

- Sign in: Form for the user to access the system.
- Sign up: Form for the user to register in the system.
- Home: Main screen of the application, from here the user can access every functionality.

- Profile: Shows the user's personal information and allows them to modify it. Also allows the user to request an API key and sign out.
- Photo review: The user is required to input the license plate being shown in the photo, or specify that is not clear enough.
- Report violation: Form to submit a report violation. Includes a preview of the photos taken where they can be discarded.
- Reports map: A map with the reports is shown, the user is capable of filtering the results by date, type and location. When pressing on a report, a popup with details appears.

The flow of the application can be seen in [figure n], where the arrows represent a transition between two screens. As it can be appreciated, every functionality can be accessed from the home screen.

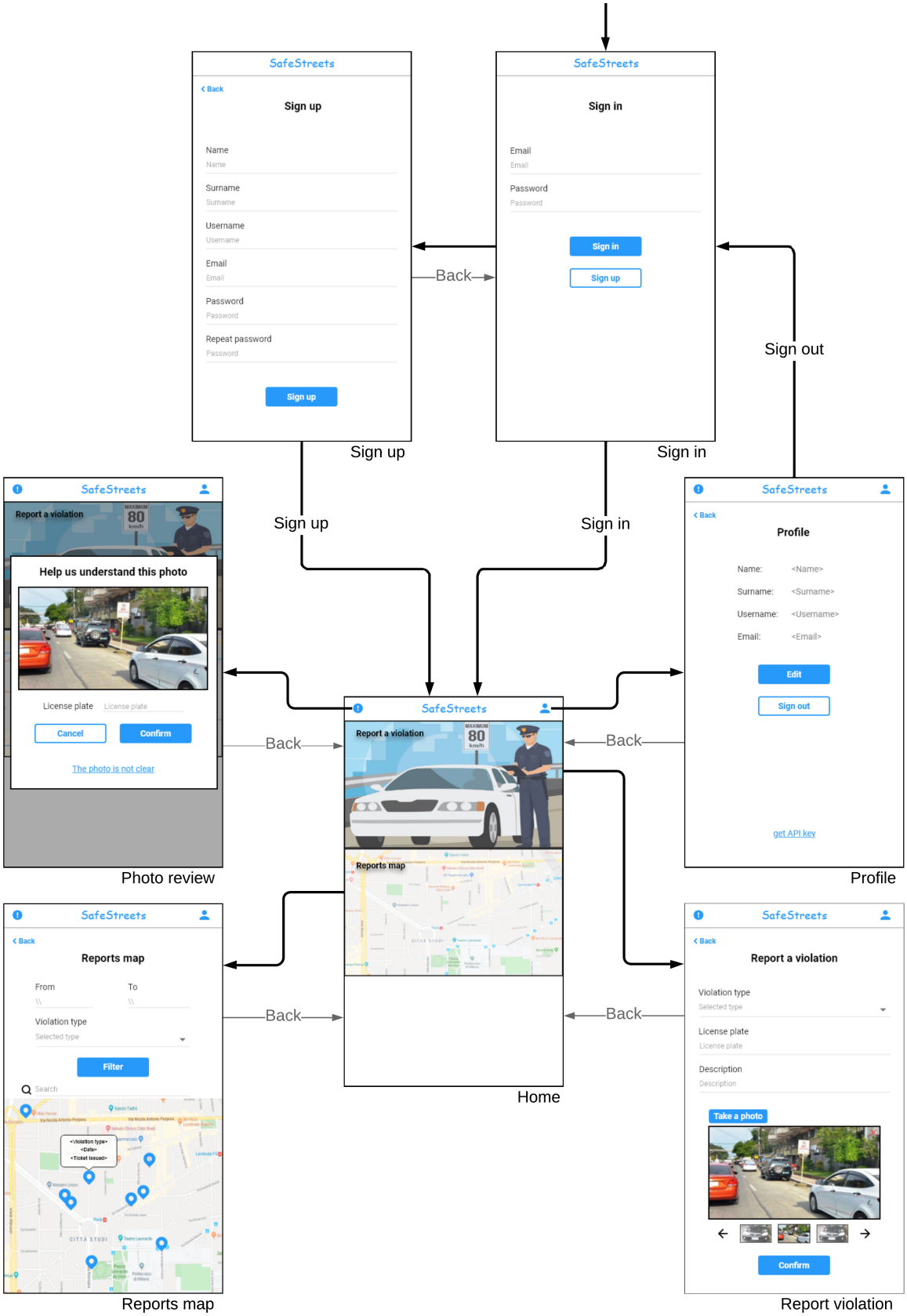

Figure 32: Mobile application flow.

4 Requirements traceability

Every design decision made and presented in this document was decided upon with the goals previously specified in de RASD in mind. From these goals, requirements were extracted; the following list provides a mapping between the goals and requirements, and the system components that help to accomplish them.

4.1 Functional requirements

[G1] - The user is able to report a traffic violation to authorities.
Requirements [FR1] - [FR2] - [FR3].

- <>

[G2] - The user is able to visualize reports in a specified area and time.
Requirements [FR4] - [FR5] - [FR6] - [FR7].

- <>

[G3] - It is possible to query report information in an easily parsable format to allow for data analysis.
Requirements [FR9] - [FR10] - [FR11] - [FR12] - [FR13].

- <>

[G4] - Only authorities are able to access report pictures and license plates.
Requirement [FR14] - [SR1].

- <>

[G5] - Compromised reports are detected and discarded.

- <>

[G6] - Authorities are able to access a curated list of reports which are considered to have a higher than average level of accuracy and reliability.
Requirements [FR15] - [FR16] - [FR17] - [FR18] - [FR19] - [SR2].

- <>

[G7] - It is possible to determine if a particular report contributed in issuing a traffic ticket.
Requirements [FR20] - [FR21].

- <>

5 Implementation, integration and test plan

5.1 Technologies

In order to talk about implementation, first we need to talk about technologies.

5.1.1 Mobile

To reach the highest possible amount of users, the application needs to be available in both major mobile operating systems, Android and iOS. Achieving this is possible without two different codebases by using a cross-platform development framework. A lot of alternatives are available, most of them consisting of a web view, providing the ability to use web technologies for development. For scalability and performance reasons, these options are discarded, in favor of a more native, approach possible with React Native or Flutter. These frameworks compile to native code, providing better performance and as a result more room for the app to grow in the future. Finally, between the two, Flutter is the preferred choice as it is a stable release (React Native is still in version 0.61 as of writing this document) and provides a big standard library of UI components, avoiding the need of third party packages and allowing for faster development.

5.1.2 Back-end server

The backend will be implemented in Kotlin utilizing the Spring boot framework. Spring is one of the most popular frameworks for the development of web applications. It allows for fast development and easy integration with documentation tools that help speed up the work. Because of its maturity and popularity, there is a great amount of documentation on the framework itself. There are a few alternatives, like Play and Grails, which offer similar functionalities, but Spring has been proven to be the most performant of the three. The web server will conform to the REST architectural design, providing a REST api, which allows for great flexibility.

5.1.3 Database

Due to the need of location searches on the report data, the database management system chosen needs to have support for geospatial queries. Also, there is no need for complex transactions in the system, as updates will be rare and not concurrent, reports are mostly appended to the existing data. For these reasons, the system chosen is MongoDB, a document-oriented database management system. It provides geospatial queries out of the box, good read and write performance and great scalability with distributed configurations.

5.1.4 Image analysis

A key point of the system is extracting important information from the submitted pictures in the reports. This then implies the necessity to have an image recognition algorithm to detect license plates and cars. Implementing this from scratch would consume too many resources and there is no need to reinvent the wheel, as there are tried and true libraries available. The chosen one is OpenALPR, which provides a public API, but also SDKs for multiple languages, including Java, which can be easily used from Kotlin in the application server.

5.2 Implementation and integration

Due to the small size of the development team, communication between team members and change making is not a problem. Because of this, the idea is to progressively integrate every part of the system as soon as they are ready. A component is considered ready once fully implemented and tested.

Both the web server and the mobile application will be developed in parallel, with the mobile app slightly behind in schedule so that any functionality required from the backend is already implemented and tested. This allows the mobile app to safely use the services provided by the backend without failures. In any case, external dependencies in the mobile application will be mocked for implementation and later integrated, which makes the development more predictable and easier for problem detection.

Integration with external services and libraries will take maximum priority, as these are the parts of the system where some modifications might have to be made, and where problems may arise. This is especially true with the photo analyser component, which covers of the core functionalities of the system and has a great amount of uncertainty.

In terms of the interaction between the backend and the DBMS, this integration will be made from the beginning. Spring boot makes this extremely easy and integration with MongoDB is fully supported.

Here the implementation and integration order for the application server is listed along with diagrams showing the dependencies between components:

The ReportAnalyzer is prioritized because of its core importance. The ReviewRecruiter is integrated later.

- LicensePlateRegistry
- PhotoAnalyzer
- ReportAnalyzer

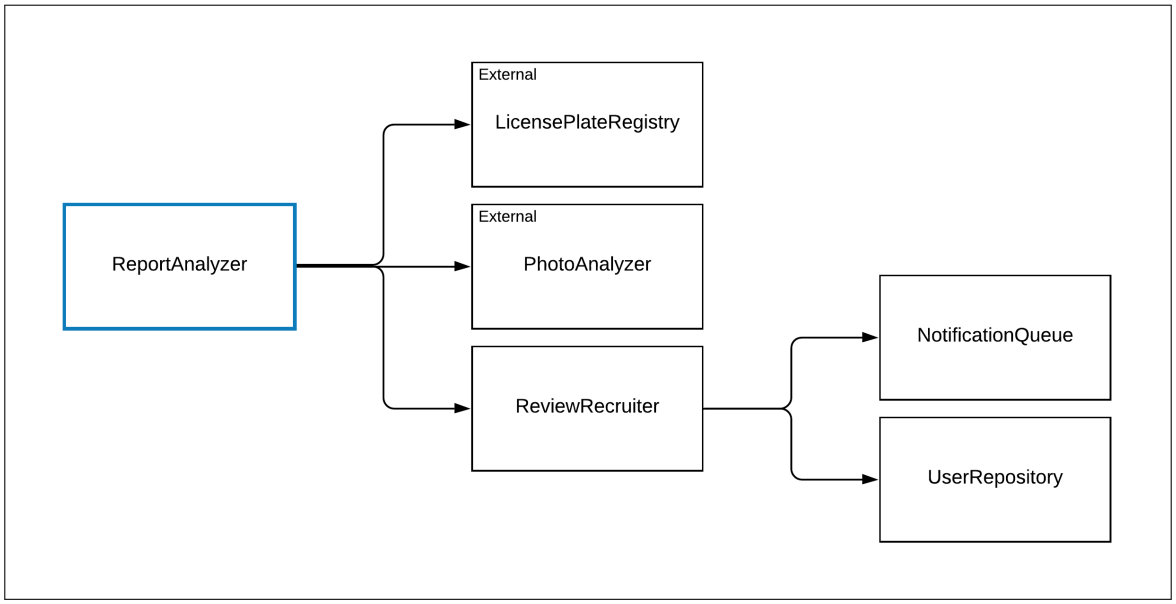

Figure 33: ReportAnalyzer dependencies.

- UserRepository
- IdentificationManager
- AuthorizationGuard
- UserManager
- UserService

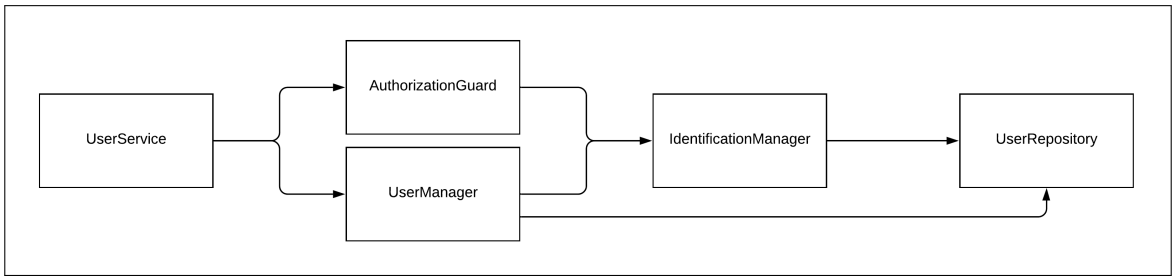

Figure 34: UserService dependencies.

- NotificationQueue
- ReviewRecruiter
- ReportRepository
- TicketingSystem
- ReportManager
- ReportService

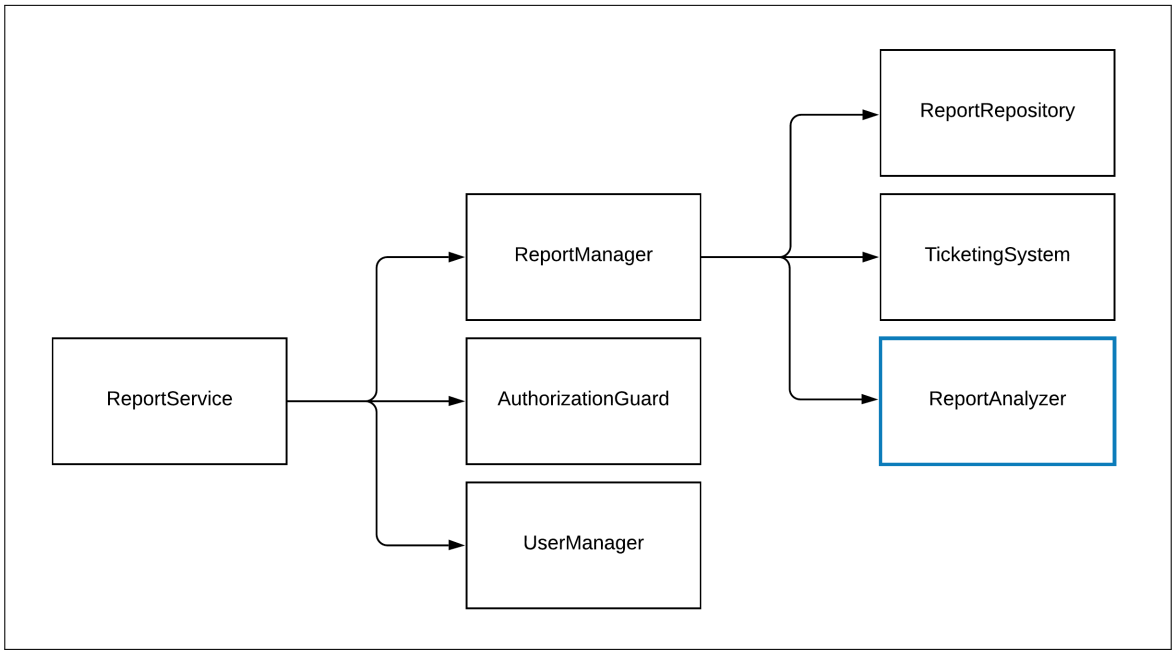

Figure 35: ReportService dependencies.

- ReviewRepository
- ReviewManager
- ReportReviewService

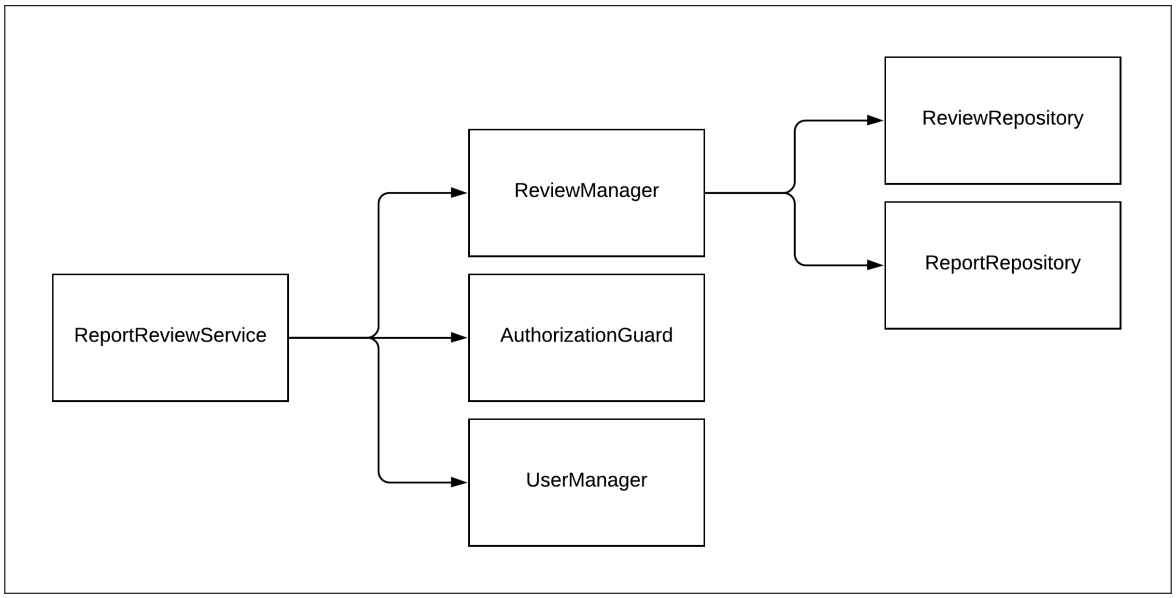

Figure 36: ReportReviewService dependencies.

- NotificationService

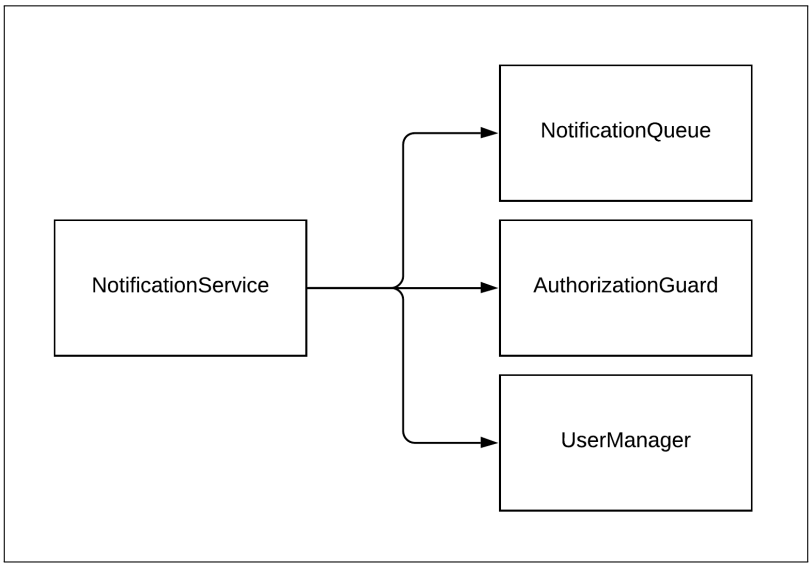

Figure 37: NotificationService dependencies.

As previously mentioned, the implementation and integration of the mobile app will be done in parallel, in the order listed below. Note that these include both the user interface and the underlying functionalities and integrations:

As in the application server, the core of the application is given priority. In this case, it consists of the Report submission and the Report map. Since these components depend on others not yet implemented, they will be mocked until necessary.

- Report submission
- Report map

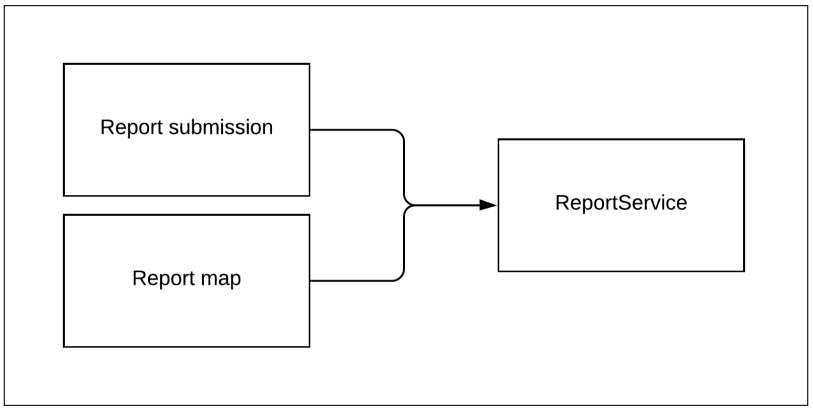

Figure 38: Report submission and Report map dependencies.

- Sign In
- Sign Up
- Profile

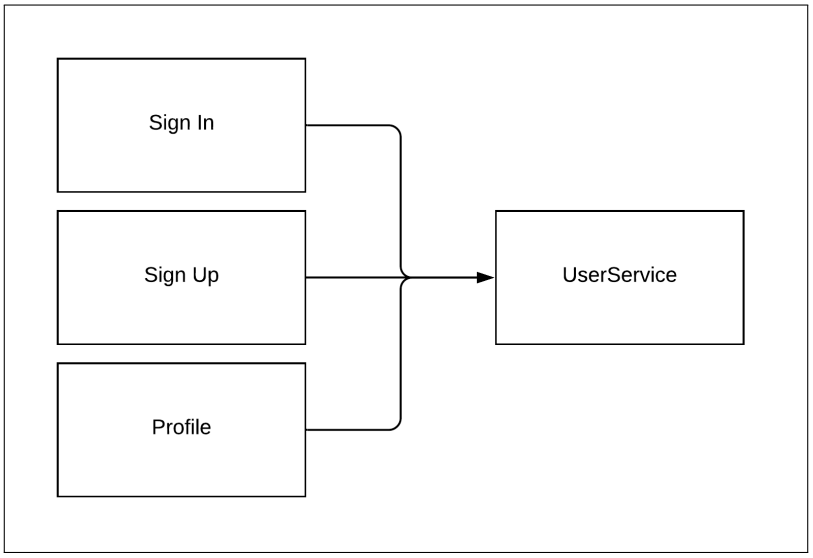

Figure 39: Sign In, Sign Up and Profile dependencies.

- Photo review

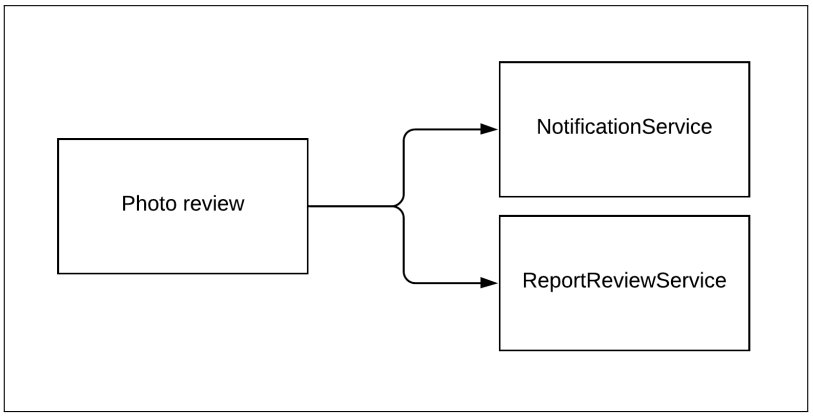

Figure 40: Photo review dependencies.

- Home

Here screen names are utilized for easy understanding, but note that the screens themselves are not considered components.

5.3 Testing

A component is only considered ready for production after its fully tested. There is no established order of testing, this is, the component is continually unit tested while being developed. Integration testing will mainly take place in the mobile application, since the external services used in the backend are already tested by their organizations. Once a feature is finished in both the mobile app and the backend, e2e tests will be performed to ensure all parts are in working order.

The following technologies will be used for testing:

- **JUnit5** integrated with spring boot for unit testing in the web server.
- For mobile, the libraries used will be **flutter_test** and **flutter_driver** which are designed for UI and E2E testing respectively.

6 Effort Spent

6.1 Manuel Pedrozo

Task	Hours
-	-

Table 1: Effort spent by Manuel Pedrozo

6.2 Tomás Perez Molina

Task	Hours
-	-

Table 2: Effort spent by Tomás Perez Molina

- “SafeStreets Mandatory Project Assignment”

