POLITECNICO
MILANO 1863

SAFESTREETS

DESIGN DOCUMENT

Sergio Cuzzucoli
Daniele De Dominicis

9 DECEMBER,2019

Deliverable:	DD
Title:	Design Document
Authors:	Cuzzucoli Sergio , De Dominicis Daniele
Version:	1.0
Date:	9-December-2019
Download page:	https://github.com/Danieledede/CuzzucoliDeDominicis.git
Copyright:	Copyright © 2019, Cuzzucoli Sergio, De Dominicis Daniele – All rights reserved

Contents

List of Figures

1 Introduction

1.1 Purpose

This document outlines the SafeStreets service, both basic and advanced funcionalities, introduced in the corresponding RASD

1.2 Scope

SafeStreets is a service that basically allows users (k̇.a. normal citizens) to upload reports about violations. This reports are seen by authorities and investigated, if deemed appropriate, or dropped if not. Every user is also allowed to inspect data regarding violations w.r.t. the area interested by them, with limitations based on the type of user exploiting the service. In addition to the basic function, Safestreets can be used to acknowledge statistics about accidents in a fashion similar to that of the violations.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **Client:** Piece of software or hardware that can access services offered by a server in different forms.
- **Server:** Piece of software or hardware that offers different services (that can constitute a part or the entirety of an application) to one or more clients.

1.3.2 Acronyms

Acronym	Meaning
DB	Data Base
DBMS	Data Base Management System
DD	Design Document
API	Application Program Interface
UI	User Interface
UX	User Experience
OS	Operating System
RASD	Requirement Analysis and Specification Document
GPS	Global Positioning System
OTP	One Time Password

1.3.3 Abbreviations

G.th	: n-th Goal
D.th	: n-th Domain Assumption
R.th	: n-th Functional Requirement

1.4 Revision history

1.5 Reference Documents

- Project assignment specifications:[1]
- UML: [2]
- DD to be analyzed: [3]

1.6 Document Structure

- **Introduction:** summary of the concepts already expressed in the RASD document.
- **Architectural Design:** detailed description of the architectural design w.r.t components and design patterns.
- **User Interface Design:** addition of details on the UI previously sketched in the RASD document by means of UX modeling.
- **Requirements Traceability:** analysis on the requirements of the RASD and how they are satisfied by the design choices of the DD.
- **Implentation, Integration and Test plan:** showing implementation and integration of subcomponents in the defined order and giving details on the subsequential testing for the integration.

2 Architectural Design

2.1 Overview

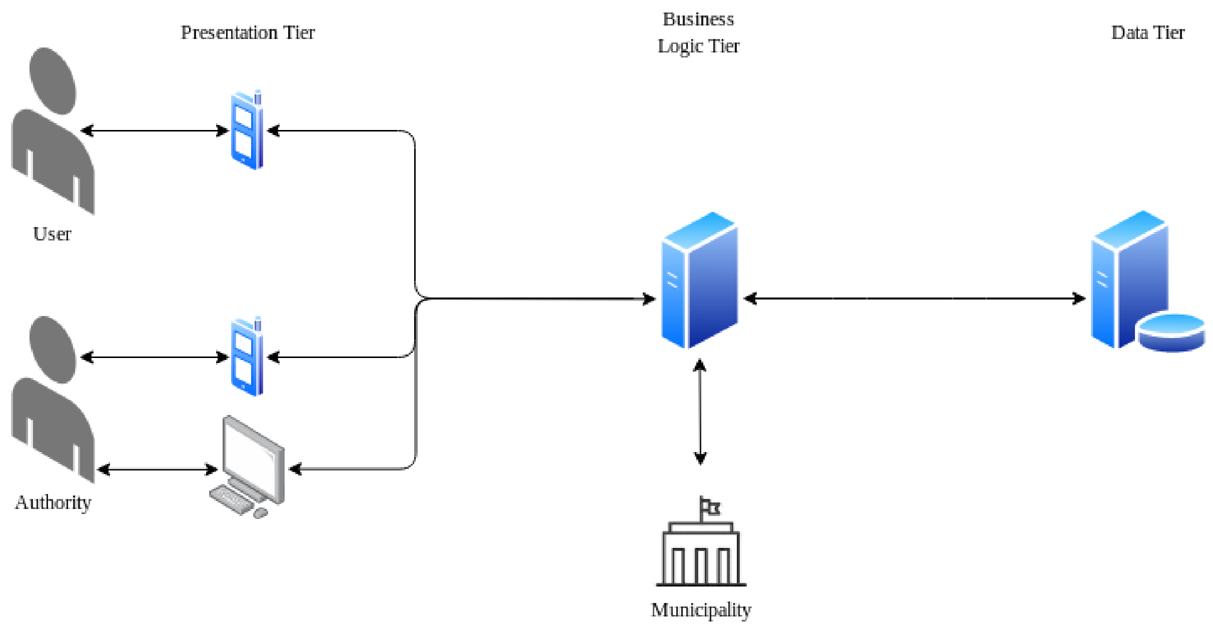
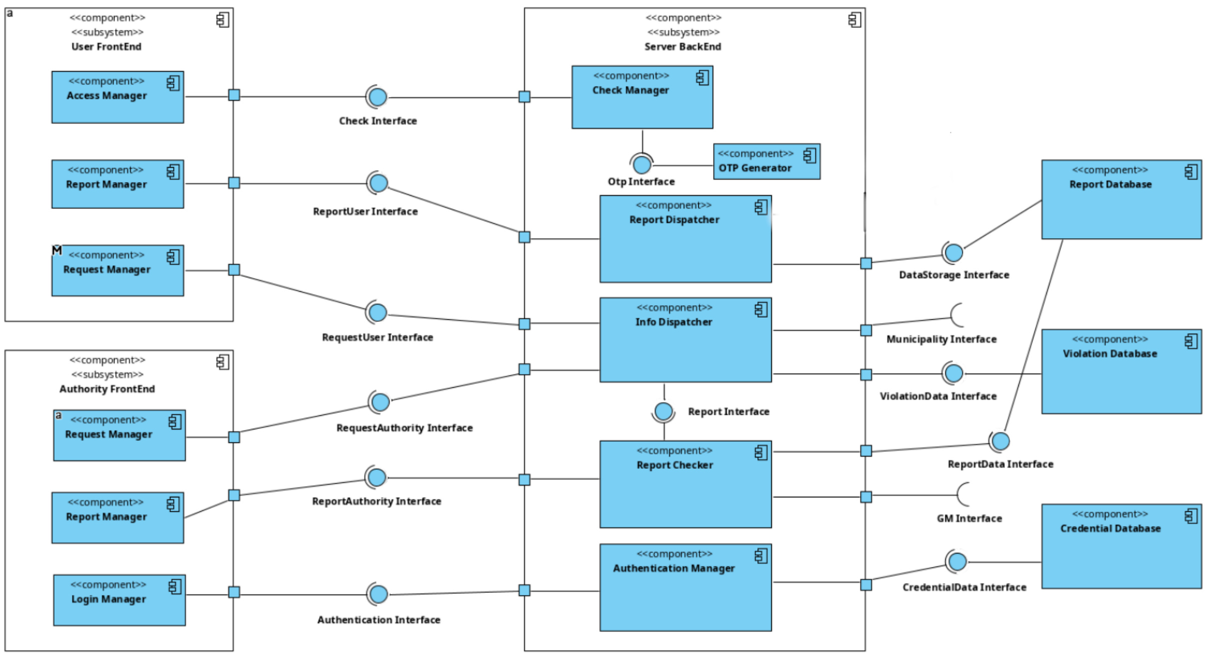


Figure 1: High level overwiew of the system

2.2 Component View



<<component>> <<subsystem>> User FrontEnd
<<component>> Access Manager
<<component>> Report Manager
<<component>> Request Manager
<<component>> <<subsystem>> Authority FrontEnd
<<component>> Request Manager
<<component>> Report Manager
<<component>> Login Manager
Check Interface
ReportUser Interface
RequestUser Interface
RequestAuthority Interface
ReportAuthority Interface
Authentication Interface
<<component>> <<subsystem>> Server BackEnd
<<component>> Check Manager
<<component>> OTP Generator
Otp Interface
<<component>> Report Dispatcher
<<component>> Info Dispatcher
Report Interface
<<component>> Report Checker
<<component>> Authentication Manager
DataStorage Interface
Municipality Interface
ViolationData Interface
ReportData Interface
GM Interface
CredentialData Interface
<<component>> Report Database
<<component>> Violation Database
<<component>> Credential Database

Figure 2: Component diagram

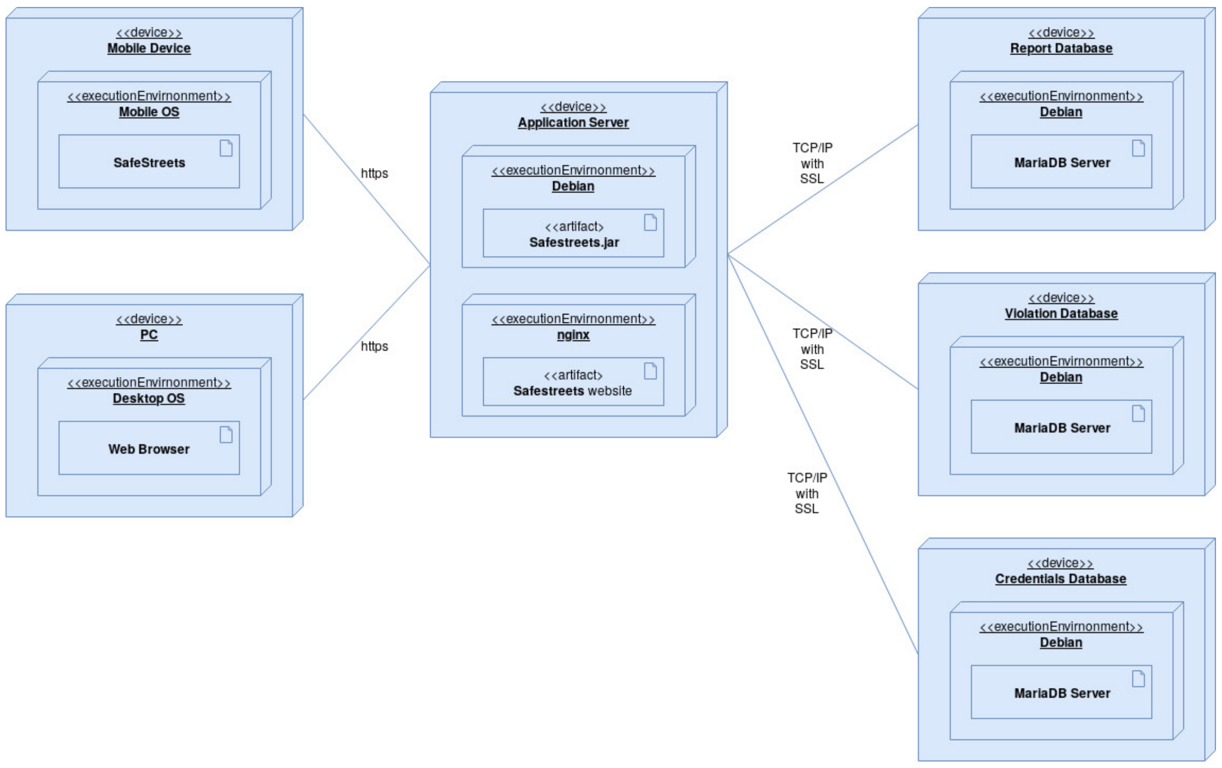
The UML Component diagram shows the internal modular structure of the components and their connections. To communicate, they can require a variable number of interfaces and attributes. A brief description of each component, w.r.t the most importants interfaces, is presented:

- Clients: Users' and authorities' machines, they can be either mobile devices or computers.
  - User Frontend
    - Report Manager: Establishes an exchange of messages with the Server and sends data regarding the alleged violation, to be later examined.
    - Request Manager: Retrieves the list of violations or accidents using the Server.
  - Authority Frontend
    - Report Manager: Retrieves a report and orders to approve it or to discard it, based on the information stored.
    - Request Manager: Retrieves the list of violations or accidents using the Server.
- Server Backend: Interacts with the clients and the Databases and contains the logic of the Application.
  - Check Manager: Requests and receives the OTP by OTPGenerator to withstand the login of the user.
  - Report Dispatcher: Receives the information about the reports filed by users and pushes them to the Report Server if the report is correctly fulfilled, aborting the incorrect ones.
  - Authentication Manager: Exchanges information with the Database containing the credentials of the authority that wants to login and ultimates the login process.

  - Info Dispatcher: Communicates with both the types of users to retrieve information about violations or accidents (these are retrieved by the Municipality API). *Note: different kinds of users receive different quantities of information, for example, users do no obtain the license plates of the offenders.*
  - Report Checker: Sends information about the open reports to AuthorityFrontend/ReportManager, locates the position using GM API and receives the approval of dismissal of said reports to store them as Violations on the Violation Database or not, by sending them to InfoDispatcher.

- Databases: They allow the retrieval and addition of data.

2.3 Deployment View



<<device>> Mobile Device
<<executionEnvirnonment>> Mobile OS
SafeStreets
<<device>> PC
<<executionEnvirnonment>> Desktop OS
Web Browser
https
https
<<device>> Application Server
<<executionEnvirnonment>> Debian
<<artifact> Safestreets.jar
<<executionEnvirnonment>> nginx
<<artifact> Safestreets website
TCP/IP with SSL
TCP/IP with SSL
TCP/IP with SSL
<<device>> Report Database
<<executionEnvirnonment>> Debian
MariaDB Server
<<device>> Violation Database
<<executionEnvirnonment>> Debian
MariaDB Server
<<device>> Credentials Database
<<executionEnvirnonment>> Debian
MariaDB Server

Figure 3: Deployment diagram

The purpose of the Deployment diagram is to specify the distribution of the components of the system, showing the configuration of run time processing nodes and components that live on them.

2.4 Runtime View

2.4.1 User Access

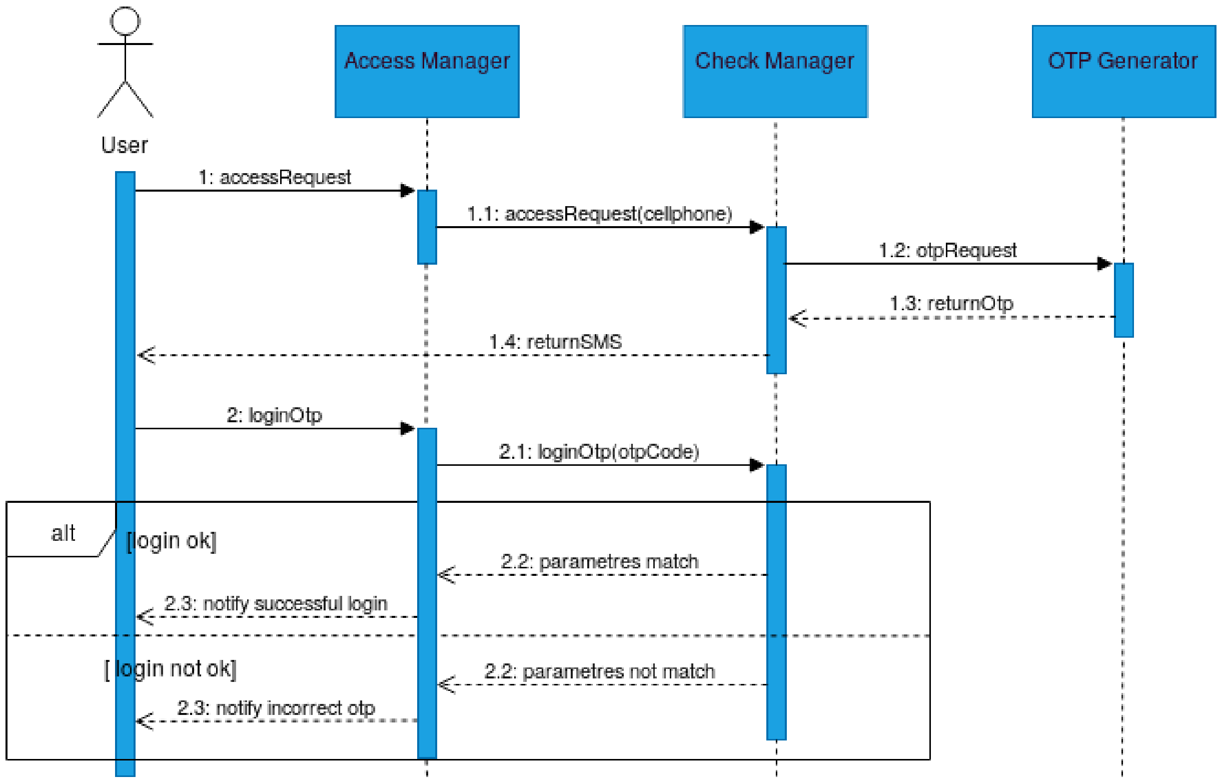


Figure 4: User access sequence diagram

2.4.2 Authority Access

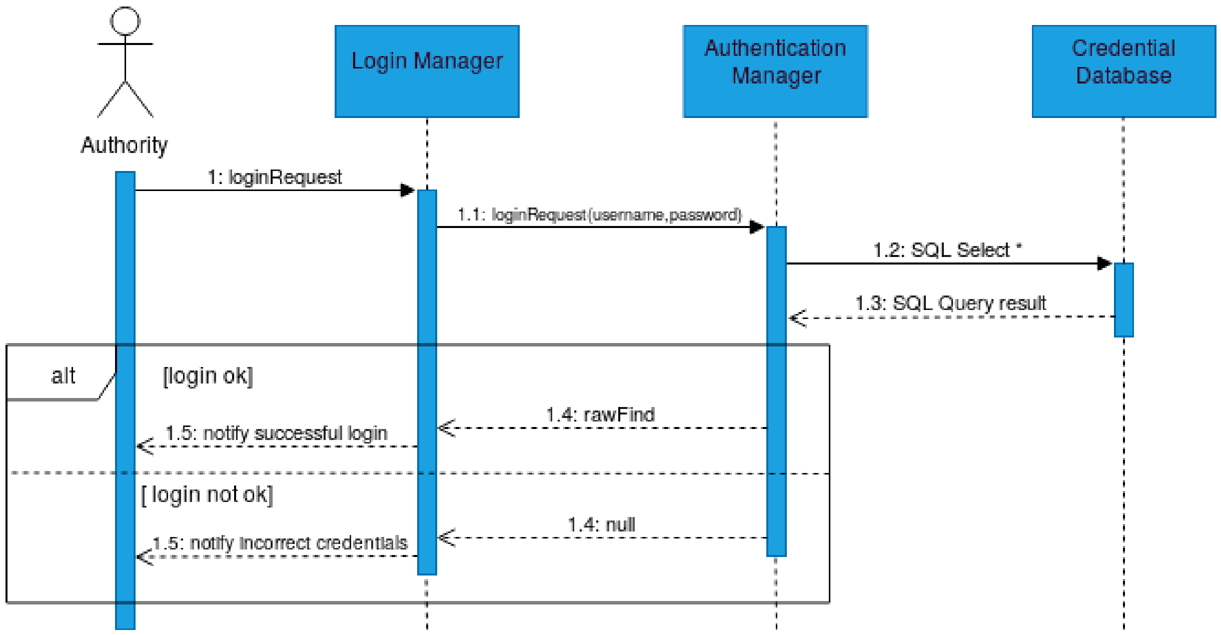


Figure 5: Authority access sequence diagram

2.4.3 User Files a report

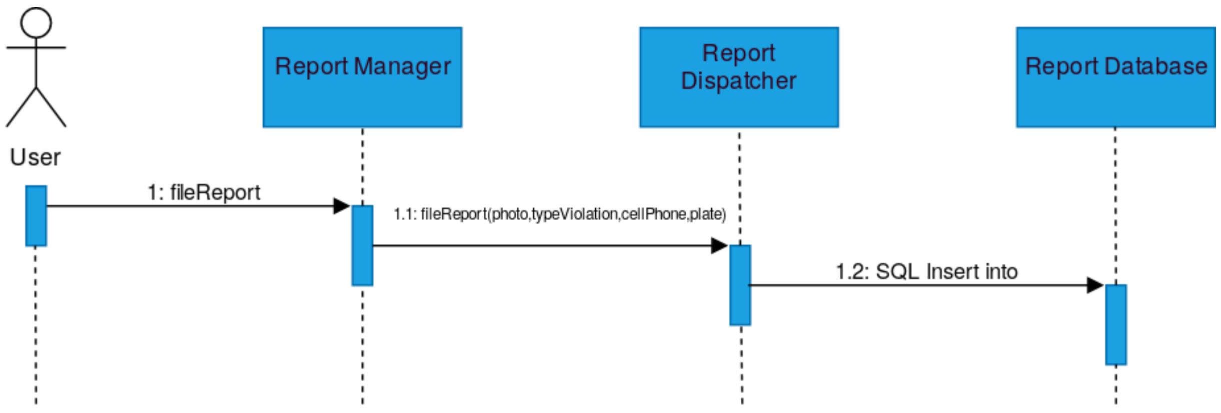


Figure 6: User files a report sequence diagram

2.4.4 Authority Checks a report

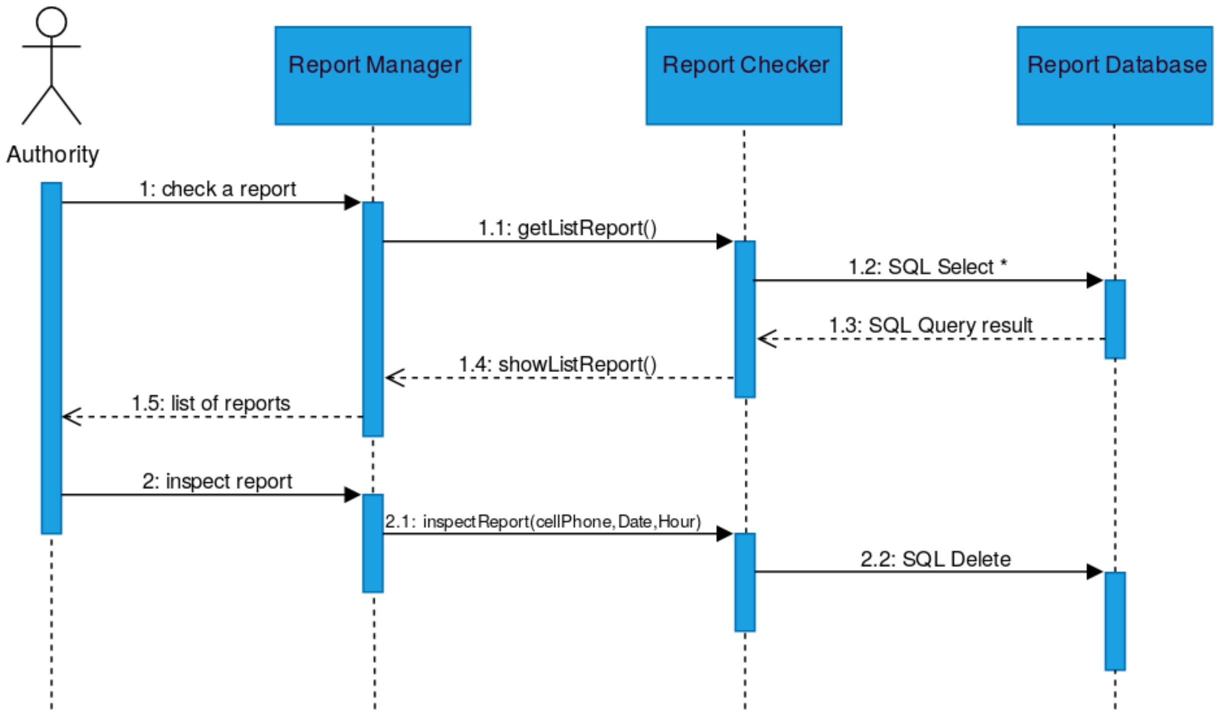


Figure 7: Authority checks a report sequence diagram

2.4.5 Authority/User reads list of violations

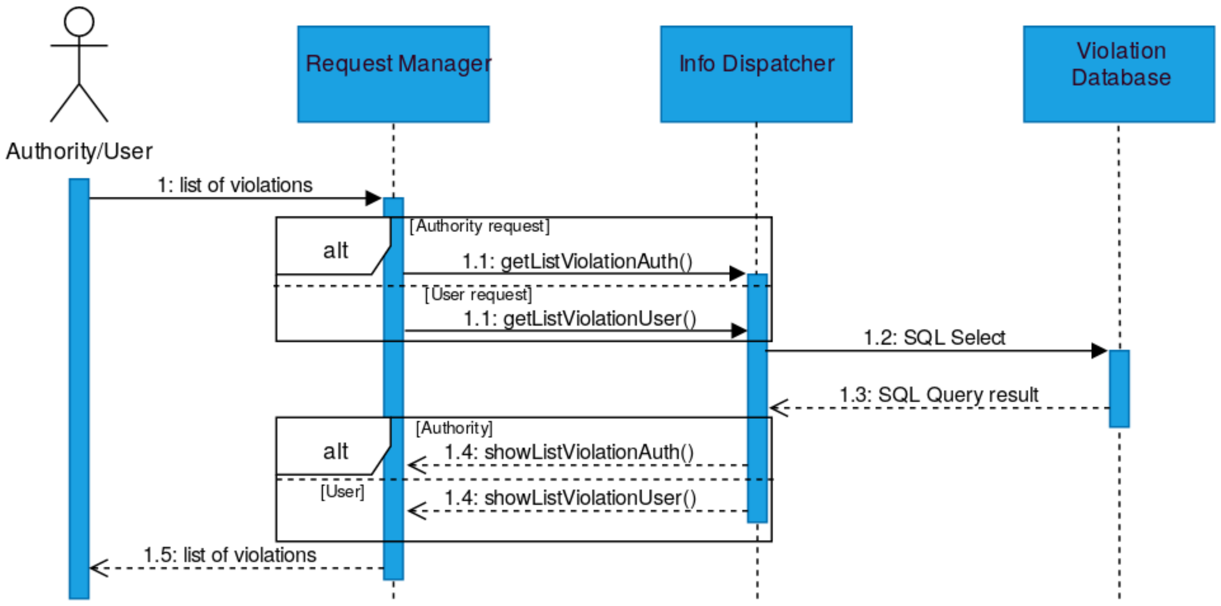
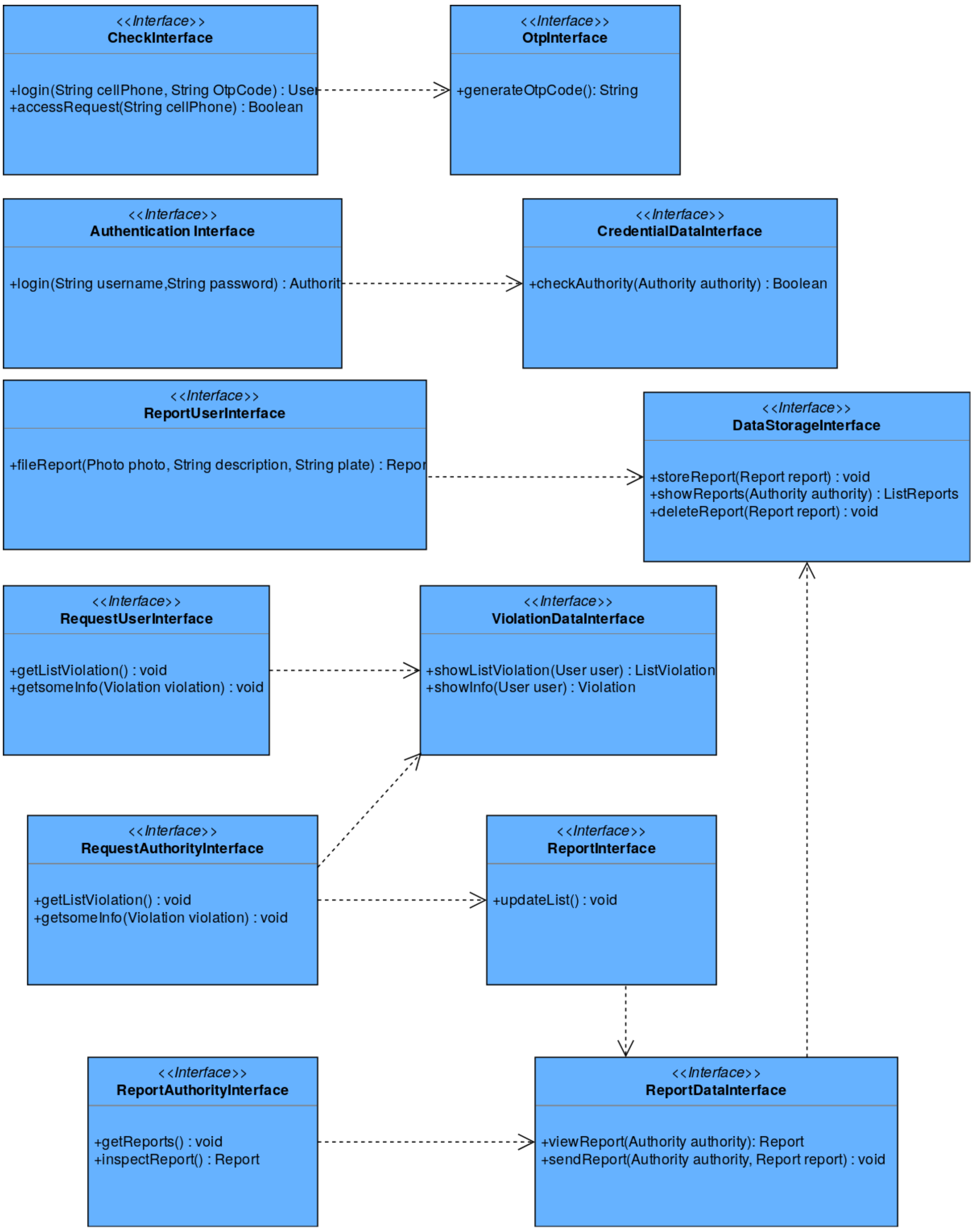


Figure 8: Authority/User reads list of violations sequence diagram

2.5 Component Interfaces



<<Interface>>
CheckInterface
+login(String cellPhone, String OtpCode) : User
+accessRequest(String cellPhone) : Boolean
<<Interface>>
OtpInterface
+generateOtpCode(): String
<<Interface>>
Authentication Interface
+login(String username,String password) : Authorit
<<Interface>>
CredentialDataInterface
+checkAuthority(Authority authority) : Boolean
<<Interface>>
ReportUserInterface
+fileReport(Photo photo, String description, String plate) : Repor
<<Interface>>
DataStorageInterface
+storeReport(Report report) : void
+showReports(Authority authority) : ListReports
+deleteReport(Report report) : void
<<Interface>>
RequestUserInterface
+getListViolation() : void
+getsomeInfo(Violation violation) : void
<<Interface>>
ViolationDataInterface
+showListViolation(User user) : ListViolation
+showInfo(User user) : Violation
<<Interface>>
RequestAuthorityInterface
+getListViolation() : void
+getsomeInfo(Violation violation) : void
<<Interface>>
ReportInterface
+updateList() : void
<<Interface>>
ReportAuthorityInterface
+getReports() : void
+inspectReport() : Report
<<Interface>>
ReportDataInterface
+viewReport(Authority authority): Report
+sendReport(Authority authority, Report report) : void

Figure 9: Component Interfaces

2.6 Selected architectural styles and patterns

2.6.1 Three Tier client-server

The architectural style chosen for Safestreets is the **client-server** in its **three-tier** variant. It allows the separation of the access to data, presentation and logic of the application. The main advantage of a multilayered architecture is the decoupling of Logic and Data. This leads to a high level of Mantainability and Scalability, qualities that are extremely useful when working with cities that tend to expand their territory and consequently the number of streets to be controlled. At the same time the system is safe as the access to data is transparent to the end user.

2.6.2 MVC

MVC is a desing pattern that breaks an application into different layers of functionalities: Model, View and Controller. This allows for the internal representation of information to differ from the information presented to the user. This produces easily reusable and flexible code and allows parallel development.

`The division:`

- **View:** Clients.
- **Controller:** Application.
- **Model:** DBs.

2.7 Other Design Decisions

2.7.1 Thin Client

In SafeStreets the client contains no logic of the Application at all, but it is just able to format and send reports and to request lists. This guarantees the App to be "light" on resources and power consumption (users do not need to have a high end smartphone) while keeping it easily upgradable.

2.7.2 Data Division

Data is spread among three different DBs as different types of data requires different levels of security and replication:

- Report Databse: The data saved on this DB is not important as reports are very dependent on time. The images and information stored are not legally binding (only authorities can administer fines), thus it must be able to handle large quantities of data and be operative in a short span of time but data loss can be sustained.
- Violation Database: The data stored here is composed of light elements (only text) which do not require particular encryption thus the server is able to sustain a great deal of ACID operations but requires data duplication.
- Credentials Database: This DB contains information for the Authentication of the authorities so, not only does it require data duplication to be quickly operative also in case of failure, but encryption is also crucial to this device. Data is stored following good pratices of security (e.g. only the salted hashes of the password are saved).

3 User Interface Design

Please examine the UI of the Application on the corresponding section of the RASD document. Here the UX for the for said UIs are shown.

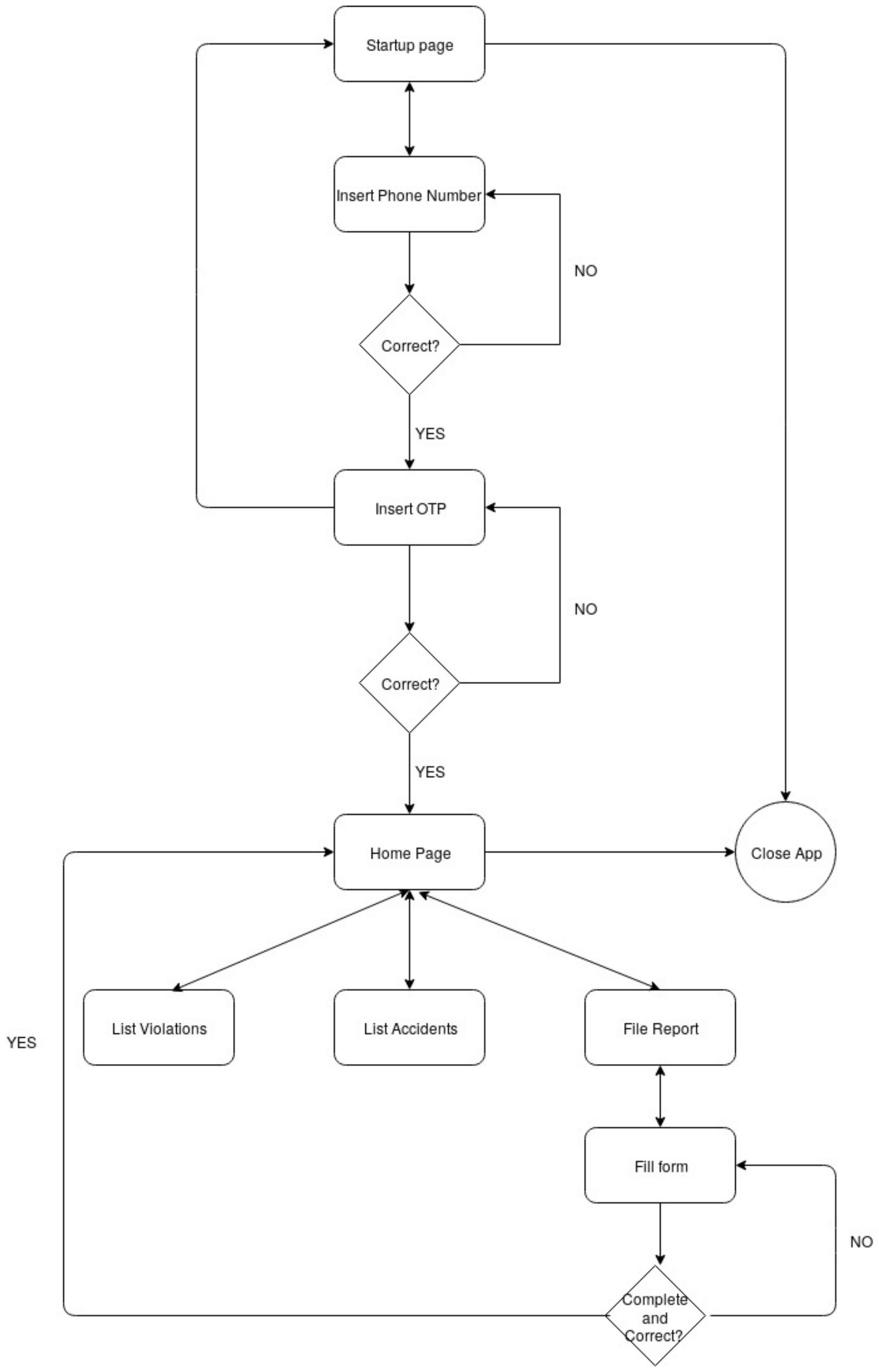


Figure 10: User UX

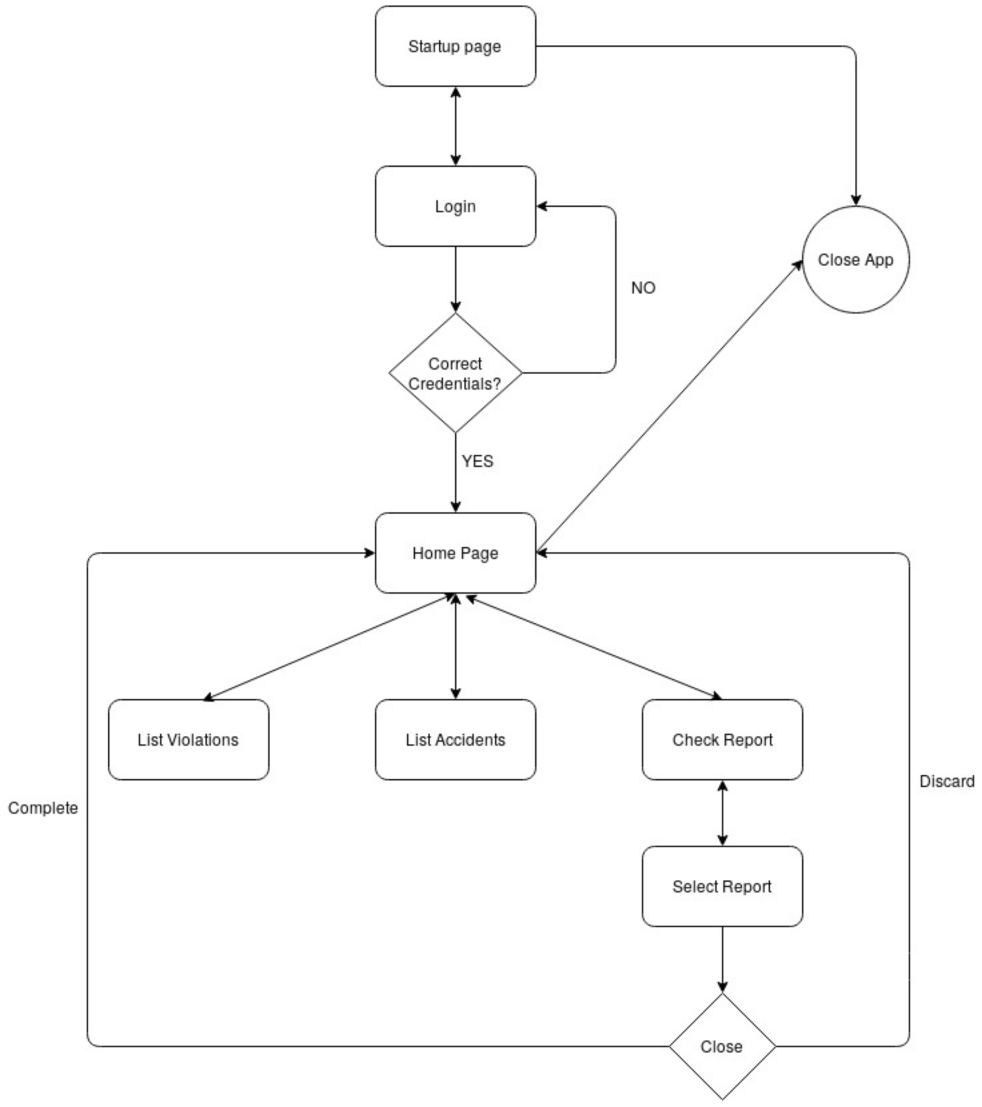


Figure 11: Authority UX

4 Requirements Traceability

Component	**Requirements**
Check Manager	R1: Users must login to SafeStreets
Report Dispatcher	R2: Users must be connected to the internet with their GPS enabled to use the service R3: Users must fill the log to send a report
Info Dispatcher	R7: Users are able to retrieve info about violations for every street R8: Users are able to retrieve info about accidents for every street R9: Authorities are able to retrieve info about violations for every street R10: Authorities are able to retrieve info about accidents for every street
Report Checker	R6: Authorities can close open reports
Authentication Manager	R4: Authorities must be connected to the internet with their GPS enabled to use the service R5: Authorities must login to use Safestreets
Databases	R11: The system can store data permanently

5 Implementation, Integration and Test Plan

As shown on the Diagram, the system is composed of UserFrontEnd, AuthorityFrontEnd, Server-Backend, ReportDatabase, ViolationDatabase and CredentialDatabase. Coherently with the pictures below, the development begins with the definition of the schemas (one for each DB), this is known as Phase 0. Phase 1 is composed of the development of the modules regarding the exchange and management of data (Report Dispatcher, Info Dispatcher and Report Checker), as they provide the services central to Safestreets. Phase 2 begins with the development of the modules and interfaces used for login and authentication (CheckManager and AuthenticationManager on the ServerBackend, AccessManager on UserFrontend and LoginManager on AuthorityFrontEnd) while the integration of the external APIs is carried in parallel. The last phase, labeled as 3, is dedicated to the completion of the FrontEnds, with the addition of the modules ReportManager and RequestManager. The process is bottom-up as first the components to be developed first are the core ones, followed by those one who rely on their services, and the testing is costant: every single unit is tested upon completion and then aggregate testing is carried out at the end of each phase to verify the ability of said components to cooperate. Altough the focus is cast principally on functional testing , performance testing is used to measure and improve the throughput, still with the idea that the system does not have to be extremely fast to offer an optimal experience to users and authorities. Load and stress testing are also carried, to ensure availabilty even in critical conditions, such as the ones that can be found during rush hours in big cities.

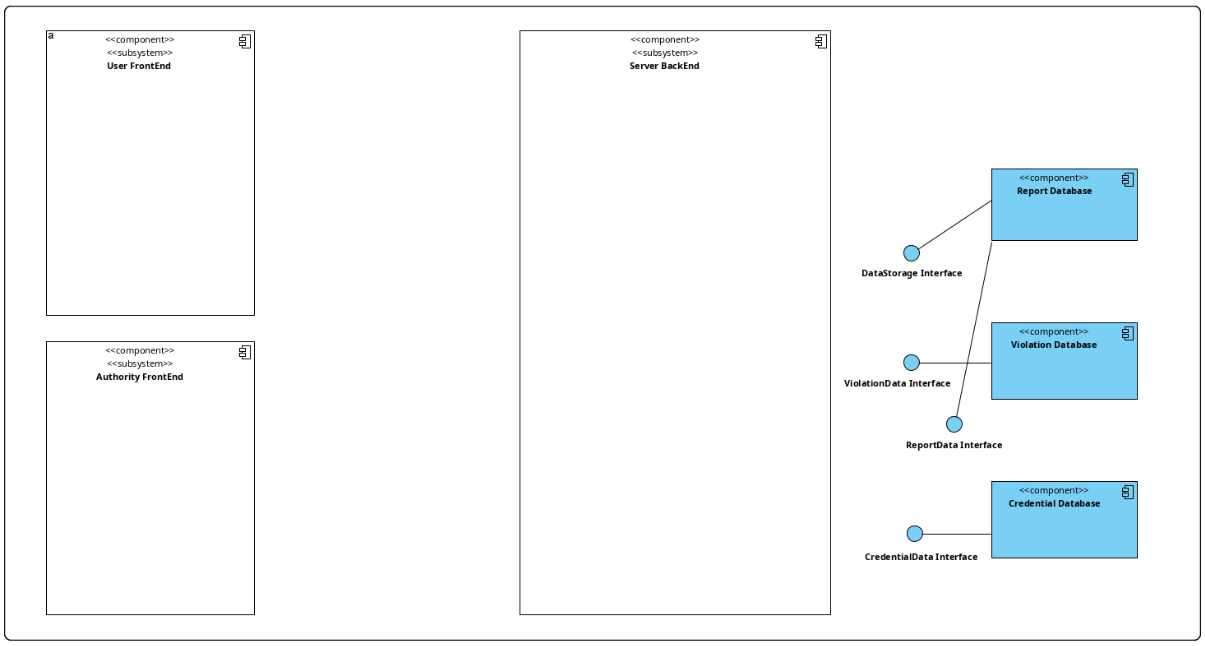


Figure 12: Phase 0

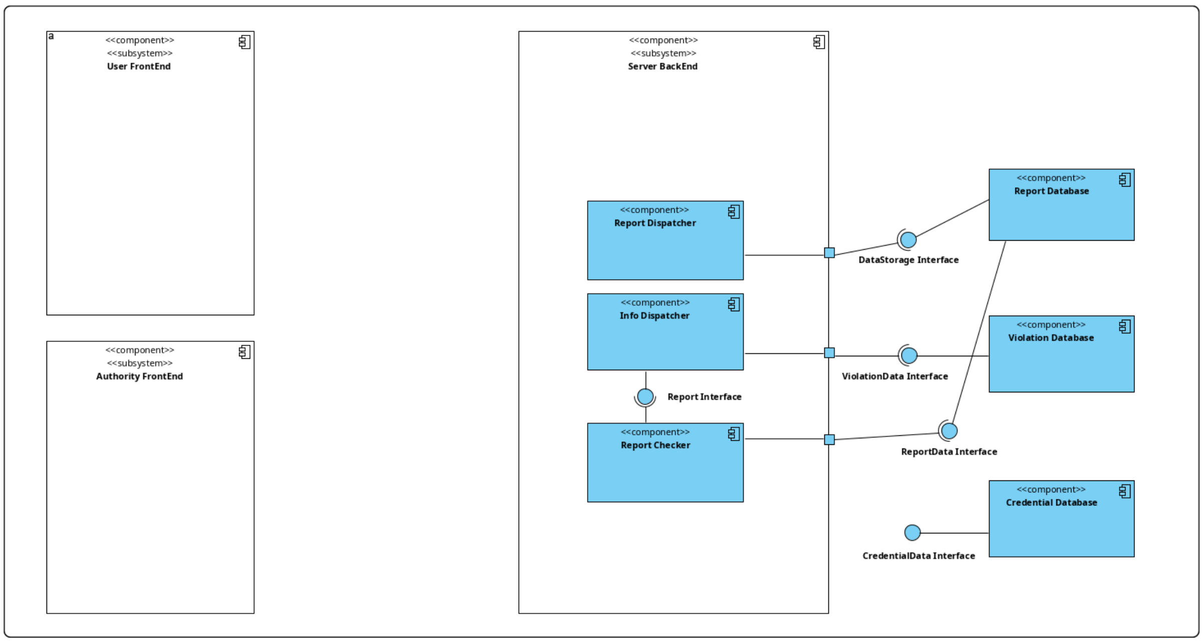


Figure 13: Phase 1

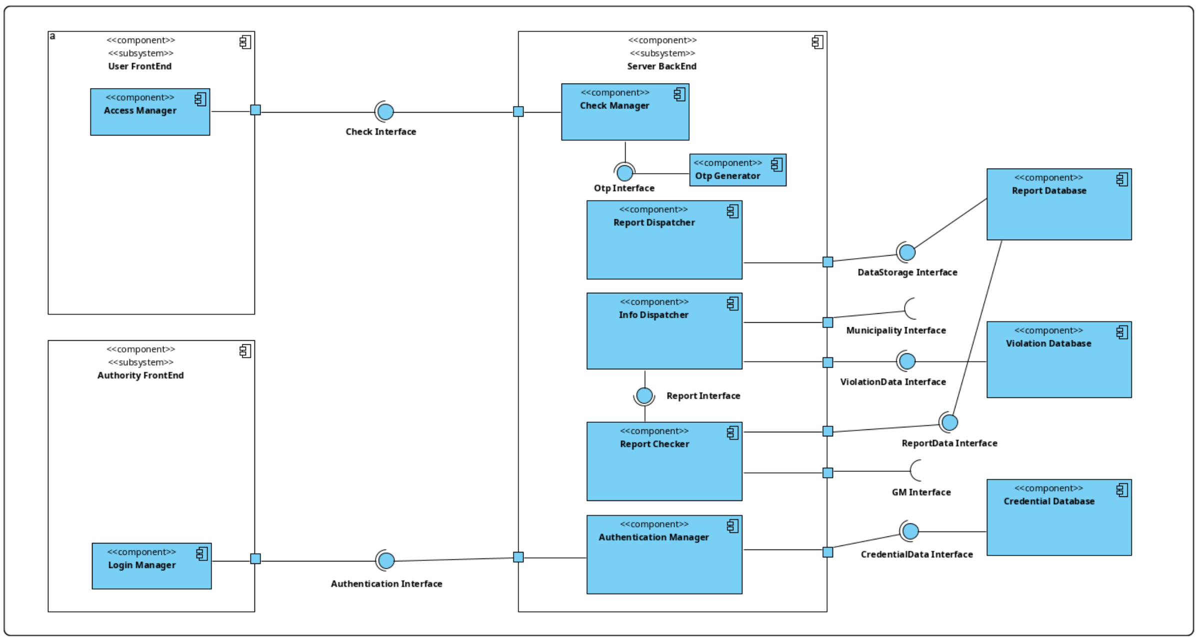
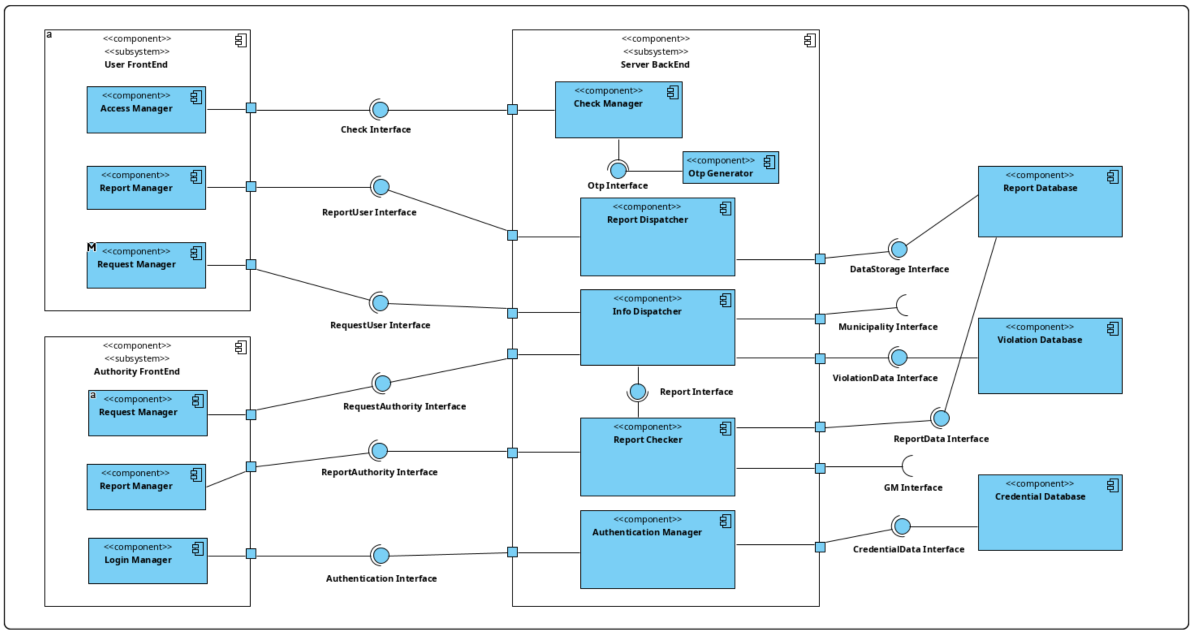


Figure 14: Phase 2



<<component>> <<subsystem>> User FrontEnd
<<component>> Access Manager
<<component>> Report Manager
<<component>> Request Manager
<<component>> <<subsystem>> Authority FrontEnd
<<component>> Request Manager
<<component>> Report Manager
<<component>> Login Manager
Check Interface
ReportUser Interface
RequestUser Interface
RequestAuthority Interface
ReportAuthority Interface
Authentication Interface
<<component>> <<subsystem>> Server BackEnd
<<component>> Check Manager
<<component>> Otp Generator
Otp Interface
<<component>> Report Dispatcher
<<component>> Info Dispatcher
Report Interface
<<component>> Report Checker
<<component>> Authentication Manager
DataStorage Interface
Municipality Interface
ViolationData Interface
ReportData Interface
GM Interface
CredentialData Interface
<<component>> Report Database
<<component>> Violation Database
<<component>> Credential Database

Figure 15: Phase 3

6 Effort Spent

Summary of the work dedicated to the project by the individuals composing the group

Cuzzucoli Sergio

Date	**Task**	**Hours**
25/11/2019	Introduction	2
26/11/2019	First steps and discussion on Architectural Design	3
27/11/2019	Additions to Architectural Design	5
28/11/2019	Explication of design choices	4
29/11/2019	User Interface Design	4
30/11/2019	Layout	2
1/12/2019	Refinements	2
2/12/2019	Minor Refinements	1
3/12/2019	Implementation,Integration and Test Plan	3
4/12/2019	Refinements	1

De Dominicis Daniele

Date	**Task**	**Hours**
25/11/2019	Introduction	1
26/11/2019	First steps and discussion on Architectural Design	4
27/11/2019	Additions to Architectural Design	3
28/11/2019	Addition of runtime views	4
29/11/2019	Finish runtime views	5
30/11/2019	Component interfaces	3
1/12/2019	Finish component interfaces	1
3/12/2019	Implementation,Integration and Test Plan	3
4/12/2019	Refinements	1

References

[1] Di Nitto Elisabetta. Mandatory project: goal, schedule, and rules, 2019.

[2] The Object Management Group (OMG). Unified modelling language: Infrastructure, 2011. version 2.4.1. omg document: formal/2011-08-05. Technical report, , 2011.

[3] Unknown Students. Dd–design document, 2018.