POLITECNICO
MILANO 1863

Davide Perugini, Antonio Pipita, Stefano Panzeri

SafeStreets

Design Document
Software Engineering 2 Project

Deliverable:	DD
Title:	Design Document
Authors:	Davide Perugini, Antonio Pipita, Stefano Panzeri
Version:	1.0
Date:	9-December-2019
Download page:	https://github.com/fafrullo2/PanzeriPeruginiPipita
Copyright:	Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

Contents

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

List of Figures

List of Tables

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

1 Introduction

1.1 Purpose

The purpose of this document is to provide a more detailed description than the one already given in the RASD document about the SafeStreets system. This document is mainly addressed to developers, as it focuses on the illustration of specific components and design choices. The aim is to guide developers during implementation, integration and testing.

The aspects covered by the design document are:

- High level architecture of SafeStreets.
- Structure of the main components and their respective interfaces through which they communicate.
- Expected runtime behaviour.
- How requirements defined in the RASD map to the design elements of this document.
- Implementation, Integration and Testing plan.

1.2 Scope

SafeStreets is a software that aims to collect data coming from registered users in the form of reports and allow either basic users, policemen and municipalities to mine this data with different levels of detail. The stakeholders of the system will be:

- Regular Users: Simple users that want to report traffic violations or have insight of the most unsafe areas of a city.
- Policemen: Members of the local police that need to access the data about users' reports and possibly take actions based on these.
- Municipal Authorities: Members of the municipality that need to analyse data in order to strengthen city safety.

Data coming from the users is collected by SafeStreets servers that organise reports and build statistics out of them. These data allows the system to offer the basic application feature: providing violations information to users, visualized mainly in map format.
The system can also collect data coming from municipalities subscribed to SS advanced function and cross them with informations collected from the users. In this way we can enable advanced analysis features, such as intervention suggestion on a city metropolitan aerea.
Finally SS can offer to municipalities which request it a traffic ticket emissions tracking service, providing statistics and acces also on this kind of data.
SS sytems manage requests from different users and save and protect a relevant ammount of data.
A registration is needed to exploit the various functionalities, even for Regular Users, reducing the possibility of false reports. Personal and sensitive informations required during registration process, are quite limited and only collected in order to reduce service abuse. Thus the system ensures an adeguate level of user privacy, reducing at the same time the costs of sensitive data management and protection.
In addiction, every data treatment will be carried on respecting every applicable privacy regulation, such as GDPR, and users will be informed about data collection and its porpuses in a dedicated privacy policy.
To ensure an even higher level of transparency, SafeStreets will ask users permission to access device data or modules, such as GPS and camera, according to last mobile OSes privacy policies.

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

1.3 Definitions, Acronyms, Abbreviations, Conventions

1.3.1 Definitions

- Smartphone: Any mobile phone that has software like the one running on a small computer, and that connects to the internet.
- Computer: Any machine or device that performs processes, calculations and operations based on instructions provided by a software or hardware program.

1.3.2 Acronyms

- RASD: Requirements Analysis and Specification Document
- DD: Design Document
- API: Application Programming Interface
- GPS: Global Positioning System
- MVC: Model View Controller
- GUI: Graphical User Interface
- DB: DataBase
- DBMS: DataBase Management System
- SS: SafeStreets
- HTTP: Hyper Text Transfer Protocol
- HTTPS: HTTP over Secure Socket Layer
- GDPR: General Data Protection Regulation
- OS: Operative System

1.3.3 Abbreviations

- (Gn) : n-Goal
- (G1.n): n-Goal for advanced function 1
- (G2.n): n-Goal for advanced function 2
- (Rn): n-Requirement

1.4 Revision History

- Version 1.0 - 9/12/2019

1.5 Reference documents

- Slides about DD from the course "Software Engineering 2"
- Davide Perugini, Antonio Pipita, Stefano Panzeri, *SafeStreets Requirement Analisys and Specification Document*

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

1.6 Document Structure

- Introduction: this chapter contains the purpose and the scope of the design document. In order to make the document more comprehensible, this chapter contains also an explanation of terms, acronyms and conventions used.
- Architectural Design: this section describes in general the architecture of the system including the three most important views: component, deployment and runtime. In this chapter is also described how the various components of the system interact and the reason behind particular design choices.
- User Interface Design: this chapter presents a reference to the mock-ups previously presented in the RASD document.
- Requirements traceability: this section shows how the components introduced in this document map the requirements given in the RASD.
- Implementation, integration and test plan: describes how to plan the implementation and integration of the various components and how to validate the system following the requirements.
- Effort spent: shows the number of hours each member of the group spent for every chapter of the document.

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

2 Architectural design

2.1 Overview

During the definition of this architecture both the top-down and bottom-up approach were used: at first we defined the main somponents of the system (top-down) considering them as black boxes, then we studied how to connect and use the aformentioned modules in order to make the system satisfy its requirements and reach its goals (bottom-up).
Given the nature of the application, a three layer approach has been chosen.

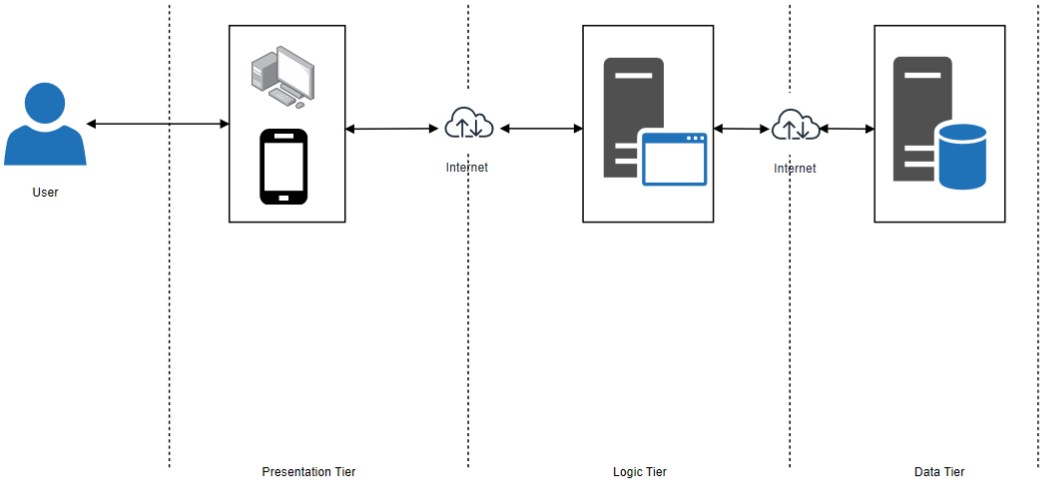

Figure 1: Three layer structure

As shown by the image above, the three layers consist in the presentation tier, the logic tier and the data tier

- Presentation Tier: consists of the user interfaces for all of the three clients (RegularUser, Policeman and MunicipalAuthority) and is used by the user to interact with both the application logic and Google Maps APIs.

- Logic Tier: consists of the servers used to control the functionalities of the application, interacts with both the presentation and data tiers.

- Data Tier: consists of the database server (where both user data and data generated by the application are stored), interacts only with the logic tier.

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

2.2 Component view

2.2.1 Introduction

The following diagram represents the interface structure of the system, focusing on the application server structure.

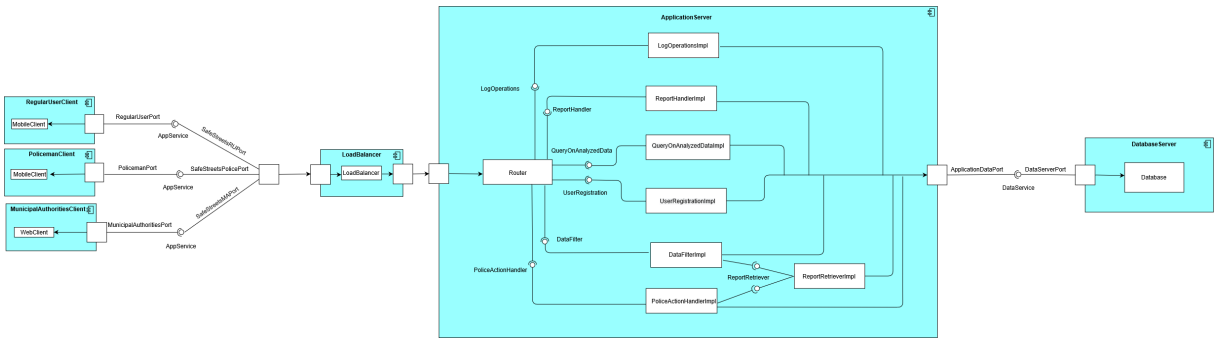

Figure 2: Component diagram

As shown above, the application logic is stored on the application server, while the clients just contain what is mandatory in order to contact the application server and to carry on basic operations (thin client).

2.2.2 Server

We will now proceed to give a brief summary of the role of the components listed above:

- QueryOnAnalizedData: allows the RegularUser profiles to access data about the reports in a certain area or other information already analized by the MunicipalAuthorities.
- PoliceActionHandler: allows the policemen to take action on user made reports or to compile reports themself. The actions performed by the police consist in compiling reports (of which they will take care right away), marking themselves as dispatched towards a report, mark a report as wrong or signal that a traffic ticket has been written as a consequence of a reported traffic violation.
- ReportRetriever: this component is used by PoliceActionHandler and DataFilter in order to retrieve all the reports regarding the same violation (not just violation type, but also time, plate and location) as a given one.
- ReportHandler: this component manages users' submissions of reports about traffic violations. The reports are analyzed, authenticated and then strored inside the application DB.
- UserRegistration: this component allows the registration of all types of users. Note that, while RegularUser profiles registration is done by the user themselves, Authority profiles (Policeman and MunicipalAuthority) can only be registered by a system admin.
- LogOperations: this component takes care of the login and logout operations for all types of User profiles
- DataFilter: allows MunicipalAuthority profiles to perform data-mining activities, as well as cross-analyze external information (such as data about crashes in an area provided by a municipality) with user-submitted reports.

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

- Router: this component manages the flow of messages from the clients towards the various components of the application server. Any kind of message for any server component will be handled and enrouted by this component

All of the aformentioned components are stateless.
The logic of the application works based on the aformentioned components: they grant all the functionality needed to satisfy the system's goals (for a more in-depth analysis see chapter 4). In the next two pictures more information about the system will be provided: the first represents a more accurate version of the class diagram presented in the RASD; the second includes a schema of the relationship between the classes interfaces and implementations of said interfaces.
Note that, when compared to the class diagram present in the RASD, the class Photo is missing: this is so because of the Base64 encoding for the photos that has been chosen to represent them inside reports.
Please note also that, in the second diagram, the Router component isn't shown. This is so since, given that the task of this component is just to redirect messages, ther's no need for further specification.

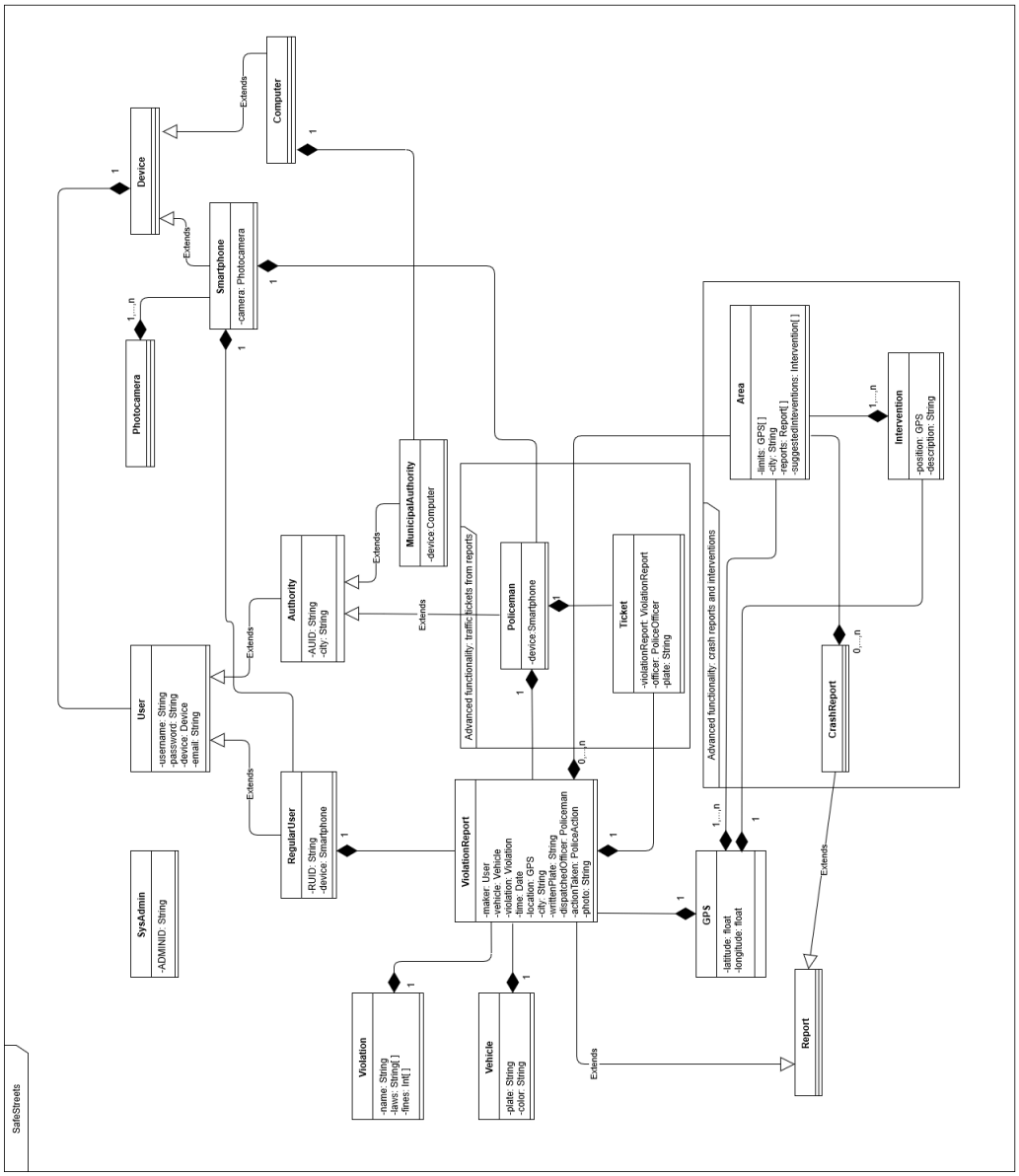

Figure 3: Class diagram

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

Figure 4: Relation between components and already specified classes

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

2.2.3 LoadBalancer

The proprietary server approach chosen for the system may limit its scalability, since the available resources cannot be increased on the demand as in a cloud-based environment. Even though an high level of scalability is not a strict requirement for the application, we decided to insert a load balancer in order to achive better flexibility and scalability. This component should allow the system availability and funcionality even in case of heavy load, granting enough room to plan server expantions and simplifying these operations at the same time.
A CISCO off-the-shelf solution (or an equivalent one from other vendors) is to be considered.

2.2.4 Client

In this section, we will give a description of the three clients functions showed in the introduction:

- RegularUserClient: allows regular users to register and login to the service, to send reports about traffic violations and visualize on a map the most unsafe areas around him.
- PolicemenClient: allows policemen login to the service, to retrieve reports sent by the regular users (visualized in a map format) and to communicate that they are working to resolve a specific report.
- MunicipalAuthoritiesClient: allows municipal authorities to login to the service, to mine data about their municipality from the DataBase and to visualize violation maps, suggested interventions, advanced informations based on crossing SS data and accidents informations (if provided by the municipality).

In the following pictures we will provided more detailed informations about the three clients' components, focusing on one at a time.
Please also note that, like the server-side components, also all the client-sides ones are stateless.

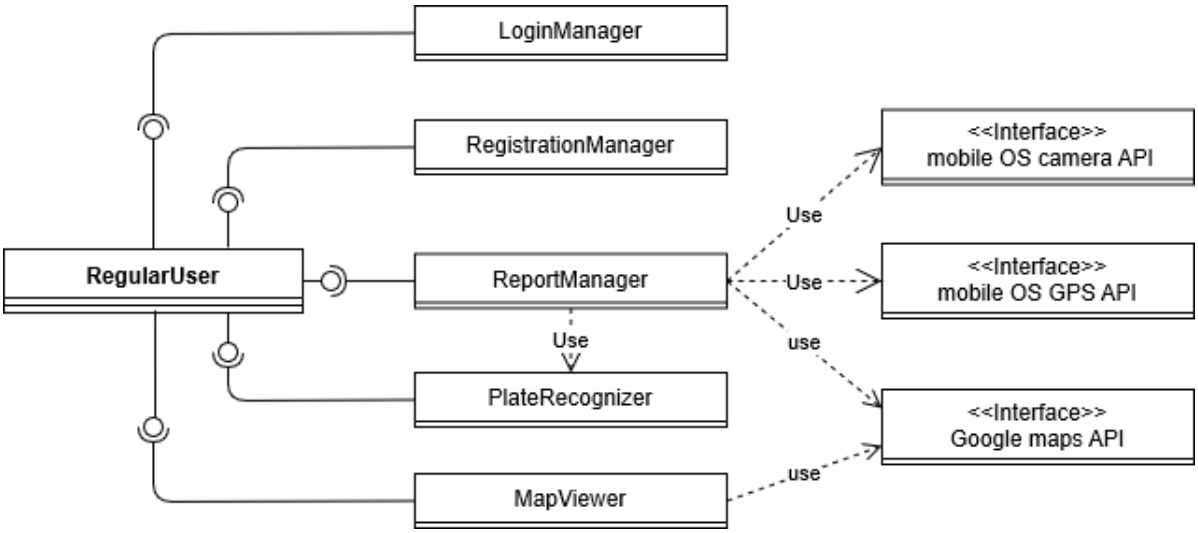

Figure 5: Relation between components inside regular user client

The components represented in this diagram are:

- LoginManager: Allows users to send their log data to the server in order to login or logout from the system.
- RegistrationManager: Allows users to send their registration data to the server in order to create a new account.

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

- ReportManager: Allows users to send a report about traffic violations, packing all the required data in a suitable format to be sent to the server. This component also uses Google maps API in order to recover informations about the area around user's geographical position.
- PlateRecognizer: Allows plate number recognition exploiting a classification Machine Learning algorithm. This component will be used by ReportManager during the process of building a report.
- MapViewer: Allows the user to visualize a geographical map from Google maps API enriched by data about unsafe areas, statistics and traffic violations coming from the server.

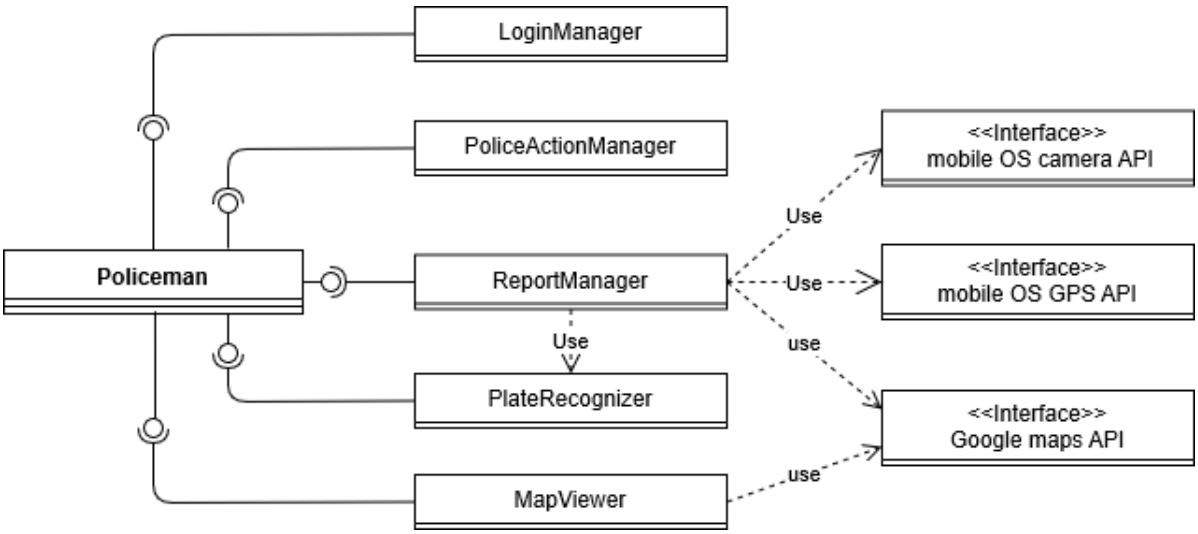

Figure 6: Relation between components inside policeman client

The components represented in this diagram are:

- LoginManager: It has the same function as in the regular user client.
- ReportManager: Allows Policeman to send a traffic violation report with all the required data, packed in a correct format, to the server. This component also uses Google maps API in order to recover informations about the area around user's geographical position.
  It's worth noting that if a policeman sends a report about a violation, he is automatically assigned to solving it.
- PlateRecognizer: Allows plate number recognition exploiting a classification Machine Learning algorithm. This component will be used by ReportManager during the process of building a report.
- MapViewer: Allows a policeman to visualize the list of the various pending reportsfor his municipality. The locations of theeports will be shown on a map provided through Google maps API.
- PoliceActionManager: Allows a policeman to signal that his intention to take care of a report. This component will send a suitable message to server which will update its data accordingly, marking the considered report as under analisys.

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

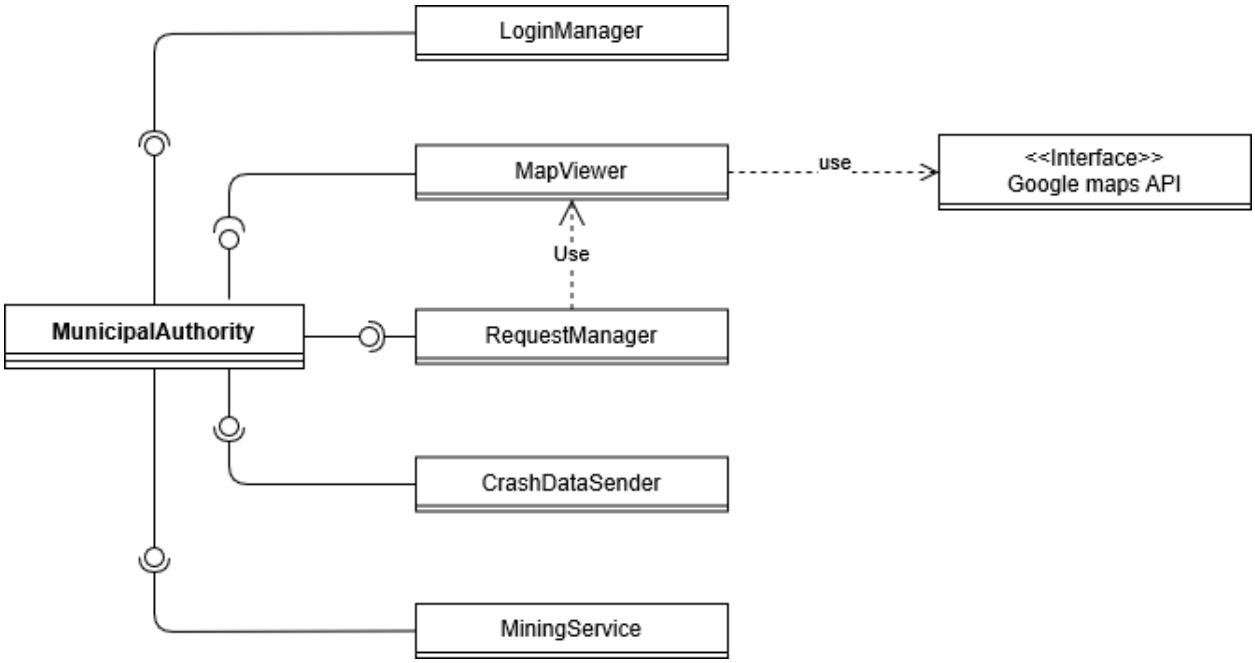

Figure 7: Relation between components inside municipal authority client

The components represented in this diagram are:

- LoginManager: It has the same function as in the regular user client.
- RequestManager: Allows the municipality to request data about Reports (solved or not) from the server. This componet uses MapViewer in order to provide violation data in a map format.
- MapViewer: Allows the visualization of the mined data in a map format. This component exploit Google Maps API to provide a representation of the local map.
- CrashDataSender: Allows the municipal authority to send Data about car crashes around the municipality to the server in order to be crossed with data already collected.
- MiningService: Provides an interface to simplify the acess of collected data to the municipality.

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

2.3 Deployment view

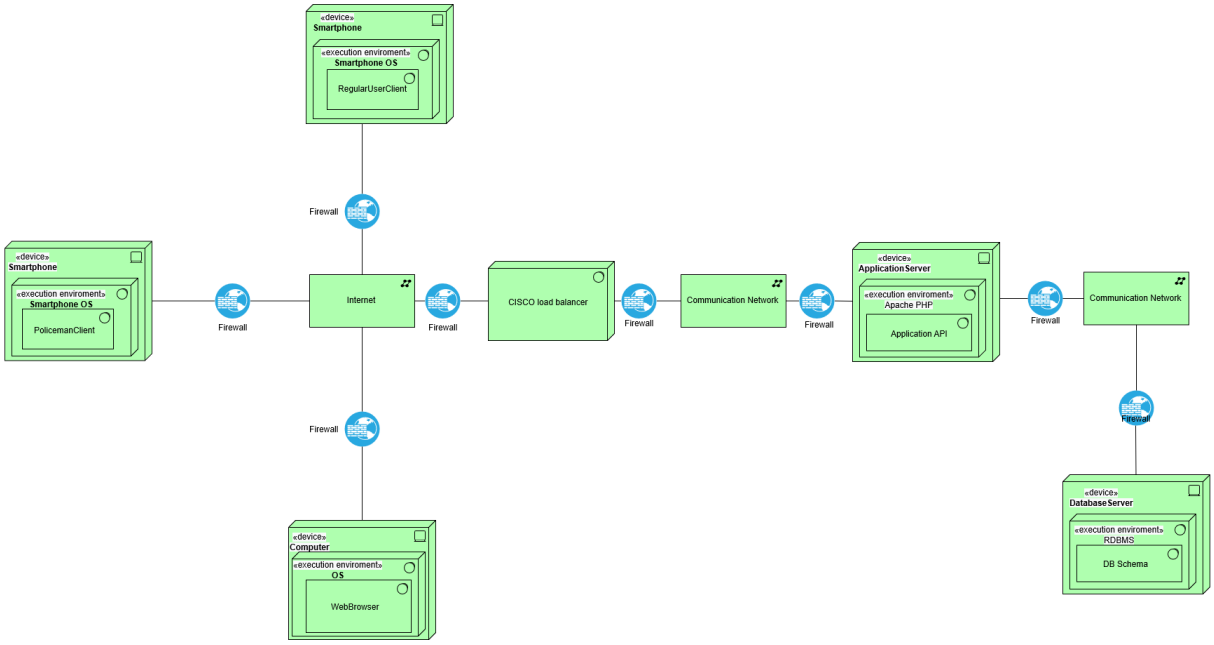

Figure 8: Physical view

The image above shows the system's architecture from a physical standpoint. Components:

- Smartphone: Device used by both RegularUser and Policeman users to interact with the application. Two different clients will be developed: one for regular users, that will allow them to look at already analyzed data and submit reports; an adavanced one for policemen, that will allow the officers make reports (upon which they will take actions right away) and take actions on reports.
  The RegularUser mobile client should be distributed as an apk/ipa, available through mobile OSes applications repositories/markets. The Policeman client will be directly delivered to Municipal Authorities subscribed to our ticket generation service.

- Computer: device used by MunicipalAuthorities users. They will interact with the application via a web page, accessible through a web browser. SS webpage will allow them to interact in various ways with the Reports submitted by the other users and will offer advanced analysis on data collected.
  Subscription to advanced feature, such as suggested intervention and traffic ticket generation, is accesible through this service. Moreover, a section with useful information is present: it will include an in-dept description of SS APIs for authorities that desire to access our data.

- Load Balancer: CISCO off-the-shelf load balancer. Note that we provided a possible load balancer solution, but an equivalent one from a different vendor (providing the same features) can be used.

- Application server: this server contains the application logic and will interact with the different clients in a client-server way.

- DatabaseServer: this server contains the actual data, both about the registered users and related to the user submitted reports, the actions taken by the police and the results of the data mining operations.

Please note that Google servers have not been included, since those servers are not part of the systems and will not be involved in the system deployment.

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

2.4 Runtime view

In this section the main functionalities of the system are analyzed through the use of sequence diagrams. The participants involved in the diagrams are either system componets (from client and server side) or external services APIs (e.g. Google Maps API) previuosly specified in this document.
A colour schema has been used in order to increase diagram clarity: Client components are represented with light blue labels; Server components with yellow labels; third party API with green labels.

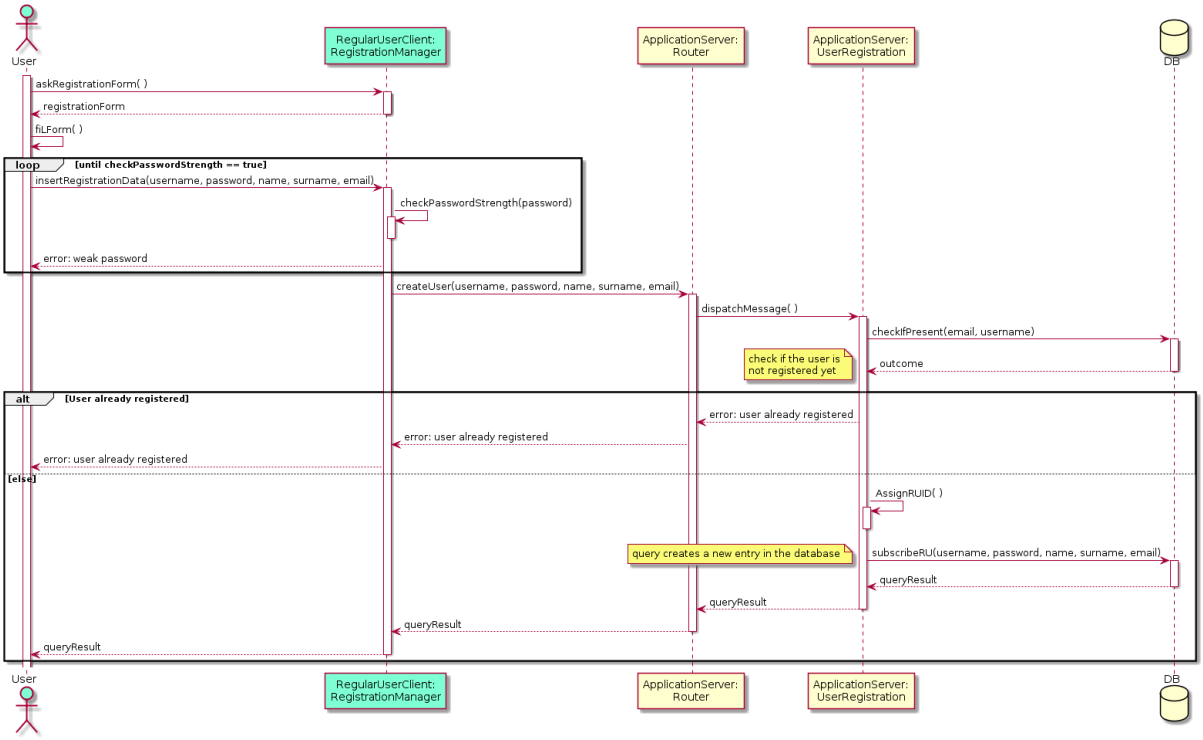

Figure 9: User registration process

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

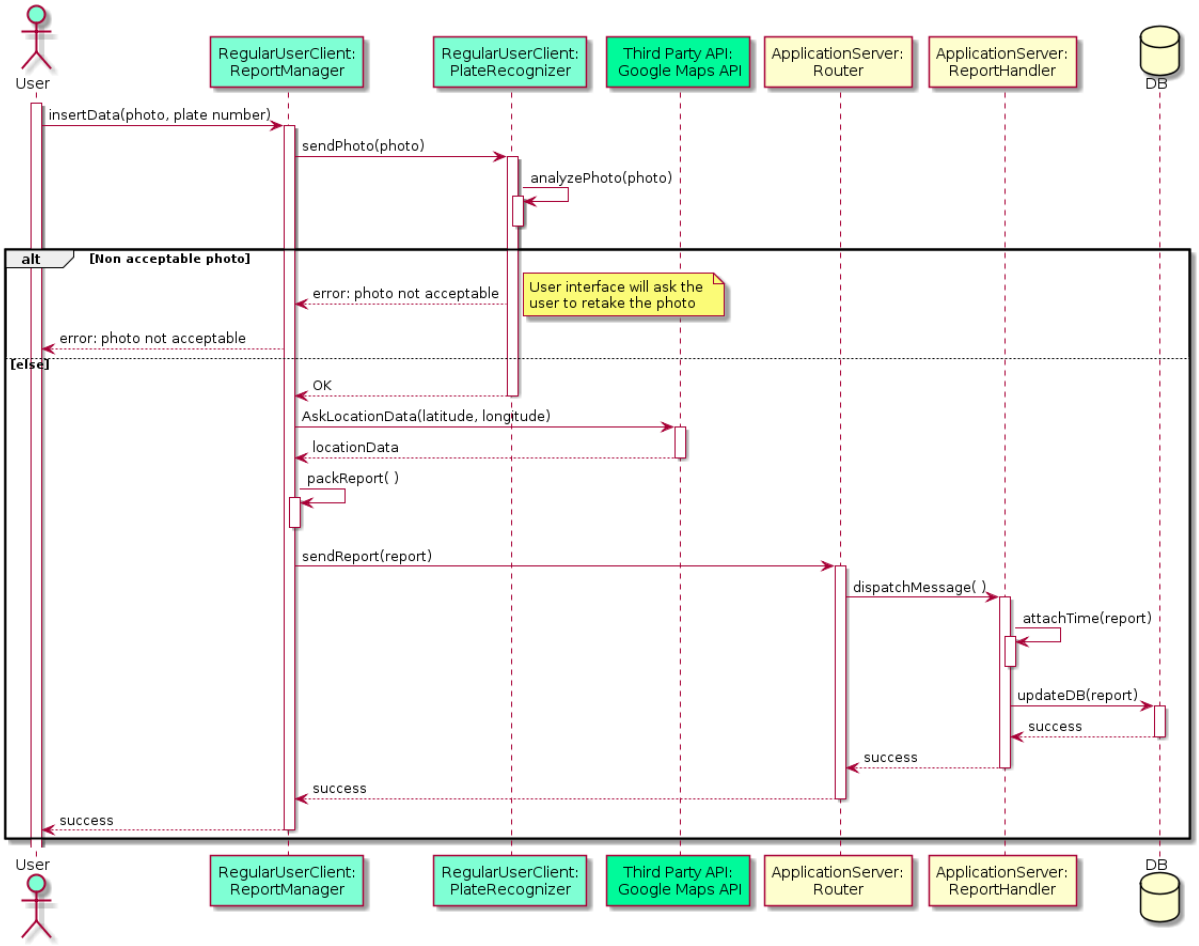

Figure 10: Report signaling procedure

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

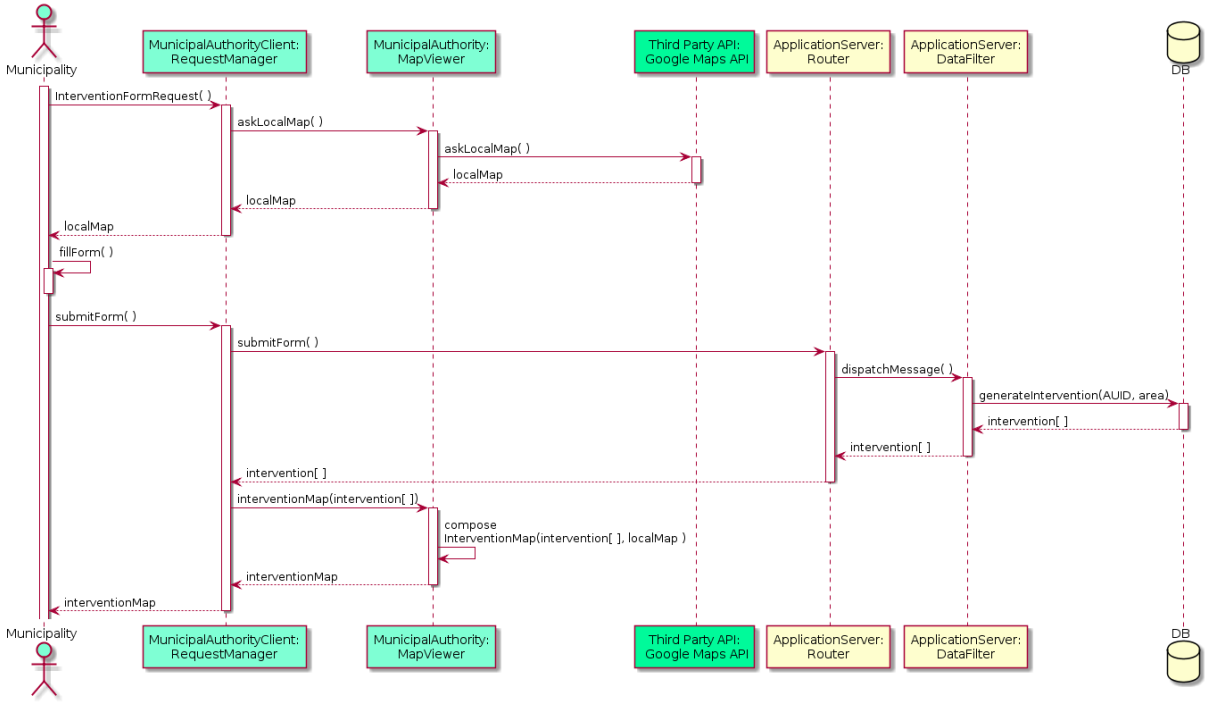

Figure 11: Municipality query for possible intervention in its metropolitan area

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

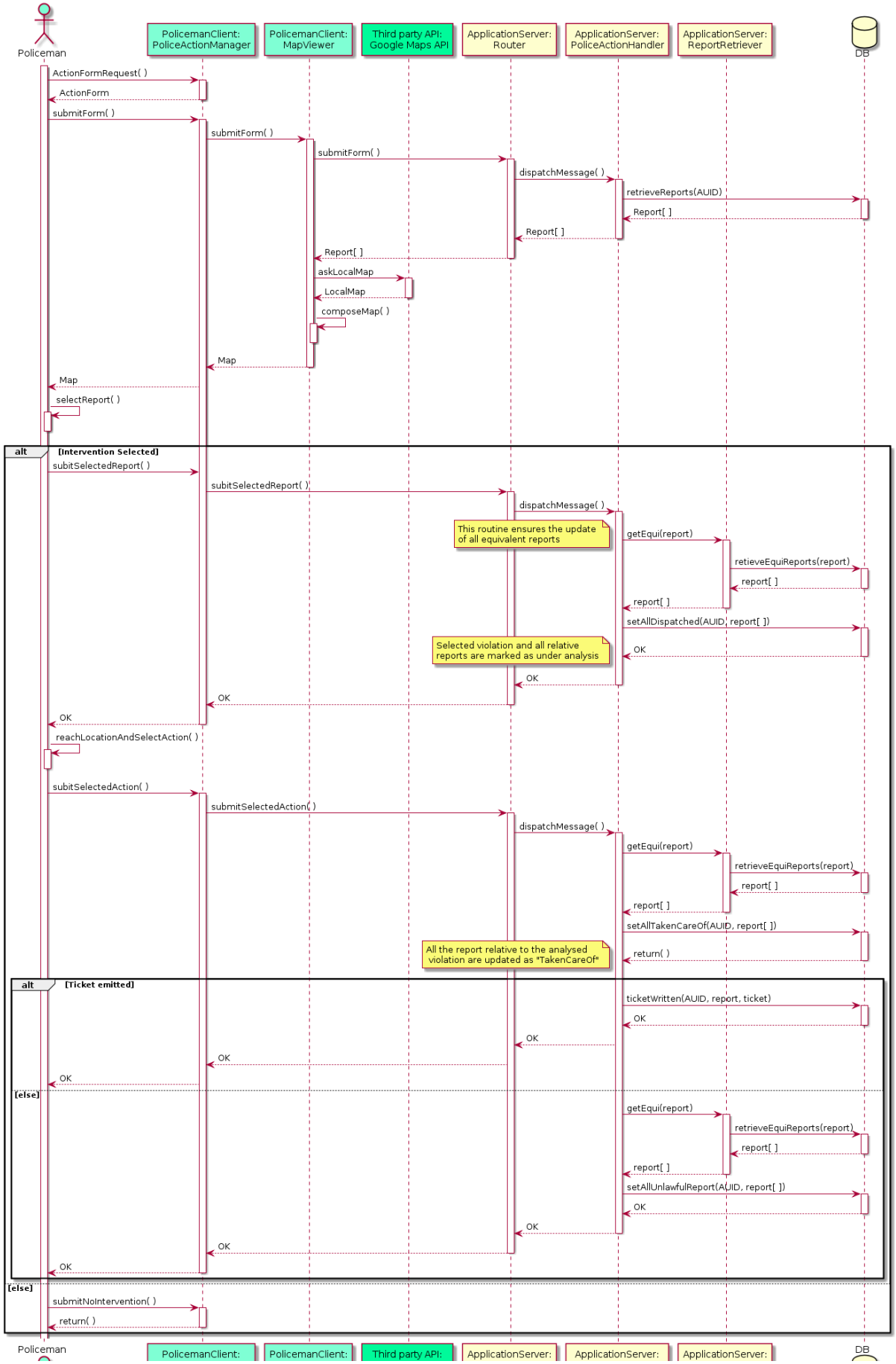

Figure 12: Policeman takes care of a signaled report

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

2.5 Interfaces

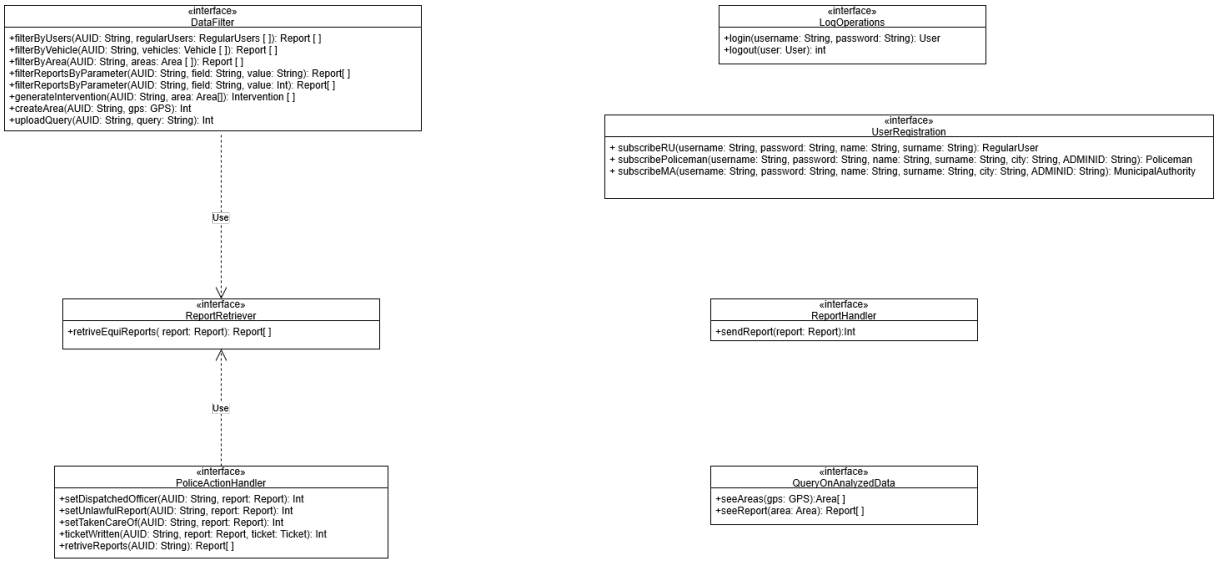

Figure 13: Components' interfaces

Other than the relation between PoliceActionHandler, DataFilter and ReportRetriever, already explained in section 2.2, it is worth noting that the interfaces don't actually expose all the methods of the objects that implement them. This choice was made in order to hide the methods that will be called only from inside the object in which they are defined.

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

2.6 Architectural and implementative styles and patterns

2.6.1 Architectural styles

- Client-server: given the nature of the application, a distibuted system in which data submitted by some users has to be accessible by other users, the figure of a mediator becomes necessary. Therefore a client-server architecture seems to be a good solution. Another alternative could have been a cloud computing architecture (SaaS, flex tenacity approach), but we opted for a client-server architecture since the workload and scalability issues would not be big enough to justify the increased costs.
- Thin client: in the aformentioned client-server architecture, the role of the server is more than just a message manager. As a matter of fact, the server also takes care of the application logic, allowing a light client with limited functionalities.
- Three tiers architecture: given the client-server with thin client architecture described, the most natural solution is a three tier architecture in which the presentation layer is on the client, the application layer on the application server and the data layer on the database server.

2.6.2 Design patterns

- Model-view-controller: given the three-tiers and thin client architecture, using a MVC design pattern is quite easy: the model will be represented by the database server, the control by the application server and the view will be managed by the client.
  The development of a different view for each type of user will be necessary, considering the different privileges and functions offered. Thus, in this section of the system modularity of the application will be central: sharing of common modules will simplify the development.
- Facade: used to hide components complexity; useful to isolate functionality, improve portability and maintainability. In particular this will ease clients development since the 3 client proposed share many modules.
- Oberserver-Oberservable: considering the message-driven architecture of the system, this pattern can be used to model components comunication.
- Factory: used in clients to create reports and in general messages and requests.

2.7 Other design choices

This sections explores many meaningfull design decisions that deserve further specification.

- Plate Recognition Procedure: Here we present a brief description of the steps of the photo analysis routine, contextualizing it in the report signaling flow.
  - Entry conditions: during a standard report procedure (the whole procedure is handled by ReportManager component), the user is propted to take a photo of the violation.
  - Procedure steps:
    1. Image acquisition: the ReportManager client component, interfacing with OS camera API, presents the user a in-app camera interface. The user takes a photo of the violation.
    2. Image cropping: the user is asked to crop the image, in order to obtain a picture representing only the plate involved in the violation. This step is necessary to avoid possibile problems with pictures including more than a plate. In addiction, the image cropping reduces a lot the computation resources needed for a plate recognition algorithm.
       The GUI has to offer a basic tool for image cropping, such as a floating selection rectangle.

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

3. Plate number recognition: the client runs locally an image recognition algorithm.
   A classification machine learning algorithm has been chosen for this task. Given the small set of possible simbols involved and the standardization of this simbols, the neural net required will be easly developed and trained. Moreover, virtually every smartphone can easly run a net of this kind. Thus, we opted to integrate the net into a client component, the PlateRecognizer.
   Note that the performance consideration presented apply exclusively to an image representing only a plate: so the image cropping step is vital.

– Output conditions: the recognition algorithm return a plate number that will be compared to the number manually inserted by the user.

- Garbage collector implementation: Considering that the traffic ticket generation functionality requires to store a relevant ammount of images in the DB, the implementation of an automated mechanism to get rid of no longer needed data is essential for resource management. Thus, it is strongly reccomended to implement a garbage collector component that periodically checks each reports timestamps and elminates images related to reports older than a threshold (e.g. 72h).
  Note that the others reports metadata have to be kept.

- Security concerns: as expressed in the RASD, there are various measures needed in order to obtain a good level of security:

  – Filtering and escaping registartion requests of all kinds of user profiles and RegularUser submitted reports (Policeman submitted ones and MunicipalAuthority queries are considered trustwothy) in order to avoid code injections
  – Escaping and filtering login fields in order to avoid code injections
  – All communication must happen over a secure channel (HTTPS)
  – An anti cross site request forgery token is needed, in particular for Municipal Authority web interface
  – Firewall use for components expoliting an internet connection in order to guarantee (at least) a basic malitious data filter.

- Database: for this application a relational database has been chosen, since involving any other kind of solution would be a waste of resources for no sensible inprovement on the system

- All exchanged data are in xml format (all the photos are hashed via base 64 and embedded in the document)

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

3 User Interface

Interfaces mock-ups have already been presented in the RASD documentation. They can be found in the section 3.1.1 of the RASD.

It's remarkable that, since regular users and policemen clients interfaces share a lot of functions, it is recommended a modular approach in order to simplify the development.

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

4 Requirements traceability

In this section the reations between Requirements, Goals and Components will be highlighted.

Row ID	Goals	Requirements	Client Component	Server Component
r1	G1	R1	RegistrationManager	UserRegistration
r2		R2	/	UserRegistration
r3		R3	ReportManager	ReportHandler
r4		R4	LoginManager	/
r5	G2	R5	ReportManager	/
r6		R6	ReportManager	/
r7	G3	R7	ReportManager	/
r8	G4	R8	PlateRecognizer	/
r9		R9	ReportManager	/
r10	G5	R10	/	DataBase
r11		R11	/	DataBase
r12		R12	https and security measures	https and security measures
r13	G6	R13	MunicipalAuthorityClient	/
r14		R14	RegistrationManager	UserRegistration
r15		R15	MiningService	DataFilter, QueryOnAnlyzedData
r16		R16	MapViewer	DataFilter, QueryOnAnalyzedData
r17	GA1.1	R17	CarshDataSender	DataFilter
r18		R18	MiningService	DataFilter
r19		R19	ReportManager, MiningService	/
r20	GA1.2	R16	MapViewer	/
r21	GA2.1	R20	PolicemanClient	DataFilter, QueryOnAnalyzedData
r22	GA2.2	R2	/	UserRegistration
r23		R3	/	UserRegistration
r24		R21	ReportManager	/
r25		R6	ReportManager	/
r26		R22	https and security measures	https and security measures
r27	GA2.3	R23	PoliceActionManager	PoliceActionHandler
r28		R24	/	PoliceActionHandler
r29		R25	/	PoliceActionHandler
r30	GA2.4	R26	PoliceActionManager	PoliceActionHandler
r31		R27	PoliceActionManager	PoliceActionHandler
r32	GA2.5	R16	MapViewer	DataFilter, QueryOnAnalyzedData

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

5 Implementation, integration and test plan

In this Section SafeStreets implementation, integration and testing will be discussed.

5.1 Implementation Plan

SafeStreets implementation wil start from the data tier moving towards presentation tier. The decision of this development order aims to semplify the unit-test of every component and to guarantee testing for all component implemented at a given time. This implementation plan follows the flow of information that derives from the proposed client-server structure.
Thus, SafeStreets implementation will follow this order:

- Database server: the database server will be the first part to be implemented, since all of its functions (ranging from submitting reports to even allow users to log in and out of the app) depend on it. A relational database management system has been chosen
- Database server - application server connection
- Application server: the application server components will be implemented in order of relevance, while still paying attention to any relation between the components. Thus, the implementation order will be the following:
  - UserRegistration
  - LogOperation
  - ReportHandler
  - ReportRetriever
  - PoliceActionHandler
  - DataFilter
  - QueryOnAnalyzedData

  This sequence is necessary because, in order to be able to perform any kind of action, there must be registered users (UserRegistration) and said users need to be able to log in their accounts(LogOperations) and submit reports(ReportHandler). Also, police has to be able to take action on said reports (ReportRetriever, then PoliceActionHandler) and municipal authorities must be able to mine data and cross database informations (ReportRetriver, then DataFilter). Last but not least, users need to query data mined by the authorities (QueryOnAnalizedData).
- clients - application server connection
- clients: in order for the application to work in its basic functions RegularUser client will be implemented first, Policemen and MunicipalAuthority clients will follow.
  Order of implementation:
  - RegularUser client:
    - RegistrationManager
    - LoginManager
    - ReportManager
    - PlateRecognizer
    - MapViewer

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

  - Policeman client:
    * PoliceActionManager
    * ReportManager (almost equal to the RegularUser client one, but it packs ViolationReport with "dispactchedOfficer" field already compiled)

    Please note that LoginManager, PlateRecogniser and MapViewer have already been implemented
  - MunicipalAuthoritiesClient:
    * RequestManager
    * CrashDataSender
    * MiningService

    Please note that LoginManager and MapViewer have already been implemented

- Load balancer: off-the-shelf CISCO solution (for reference, not a mandatory solution)

5.2 Integration Plan

5.2.1 Basic conditions

Integration is the process through which the various implemented components are connected and start interacting with each other. In order to achieve a meaningful integration, there are some basic condition that must be fulfilled.

First of all, the most important condition is that every single component must be working correctly and their functionalities must be tested. In this case, if during integration a problem arises, this must be related only to the integration phase.

Another important requirement is that is not necessary for every single component's functionality to be implemented, but at least the main ones and those needed to interact with other components must be tested and correctly operating.

5.2.2 Integration: Database - Application server

Since Database and Application Server will be the first parts to be implemented, integration between them is of primary importance to ensure the proper conduct of the following operations. The specific integrations are:

- UserRegistration - Database
- LogOperation - Database
- ReportHandler - Database
- ReportRertiever - Database
- PoliceActionHandler - Database
- DataFilter - Database
- QueryOnAnalyzedData - Database

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

5.2.3 Integration within the Application Server

The integration within the application server will be as follows:

- Router - LogOperations
- Router - ReportHandler
- Router - QueryOnAnalyzedData
- Router - UserRegistration
- Router - DataFilter
- Router - PoliceActionHandler
- PoliceActionHandler - ReportRetriever
- DataFilter - ReportRetriever

5.2.4 Integration: Application server - RegularUser Client

Regular user client will be the first one to be integrated with the application server in order for the application to work in its basic functions. Some client Components will also need an integration with Google maps API to operate correctly. The specific integrations are:

- RegistrationManager - Router
- LoginManager - Router
- ReportManager - Router
- ReportManager - PlateRecognizer
- ReportManager - mobile OS camera API
- ReportManager - mobile OS GPS API
- ReportManager - GoogleMapsAPI
- MapViewer - Router
- MapViewer - GoogleMapsAPI

5.2.5 Integration: Application server - Policeman Client

As already said for the regular user client, also policemen client will have components that need an integration with Google maps API and mobile OS APIs.

For policemen client, the specific integrations are:

- LoginManager - Router
- PoliceActionManager - Router
- ReportManager - Router
- ReportManager - PlateRecognizer

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

- ReportManager - mobile OS camera API
- ReportManager - mobile OS GPS API
- ReportManager - GoogleMapsAPI
- MapViewer - Google Maps API

5.2.6 Integration: Application Server - MunicipalAuthority Client

For municipal authorities, the specific integrations are:

- LoginManager - Router
- RequestManager - Router
- RequestManager - Router
- RequestManager - MapViewer
- MapViewer - GoogleMapsAPI
- CrashDataSender - Router
- MiningService - Router

5.2.7 Integration: load balancer

The load balancer component will be integrated when the application server development will be complete. The integration of this component may require a remarkable ammount of effort, considering the choice of a third party component. However, we opted for this option considering that the costs of the development of an in-house solution would not be sustainable.

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

6 Effort Spent

- **Antonio Pipita**

Section	Hours
Introduction	1
Architectural: Overview	1.5
Architectural: Component view	16
Architectural: Deployment view	6
Architectural: Runtime view	0.5
Architectural: Interfaces	1
Architectural: Styles and patterns	3
Architectural: Other design choices	3
Requirements traceability	3
Implementation plan	2
Integration plan	1
Total	38

- **Davide Perugini**

Section	Hours
Introduction	2
Architectural: Overview	1
Architectural: Component view	15
Architectural: Deployment view	3
Architectural: Runtime view	0.5
Architectural: Interfaces	0.5
Architectural: Styles and patterns	2
Architectural: Other design choices	1
Requirements traceability	2
Implementation plan	1
Integration plan	4
Total	32

- **Stefano Panzeri**

Section	Hours
Introduction	1
Architectural: Overview	3
Architectural: Component view	3.5
Architectural: Deployment view	1.5
Architectural: Runtime view	12
Architectural: Interfaces	2
Architectural: Styles and patterns	2
Architectural: Other design choices	2
Requirements traceability	3
Implementation plan	3
Integration plan	2
Total	35

Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved