POLITECNICO DI MILANO

Academic Year 2019/2020

DD

Design Document

version 1.0 – 9/12/2019

Computer Science and Engineering

Software Engineering 2

Matteo Falconi 945222 - Davide Galli 944940

Contents

1 Introduction

1.1 Purpose

1.1.1 Project Description

SafeStreets is a cross-platform service available both on app and on web.

App service is focused to develop a software-based service that allows individual basic users to report traffic violations. Those reports consist in pictures of violation, type of violation, date, time and position. When a picture is uploaded, the system runs an algorithm in order to read the license plate. Finally, all those data are stored in *SafeStreets*' databases.

The system allows also authorities registration, who can receive notifications about new violations in a certain area. When a notification occurs, an authority can reserve it taking charge of that violation.

Both basic users and authorities can access to collected data in order to analyze the streets and the relative safeness. However, a basic user can only access to anonymized data clusters, that give an idea of how many violations occur in each area; whereas authorities can also access to specific anonymized data.

Web service, instead, let *SafeStreets* to develop a functionality, in partnership with the municipalities who provide accidents data, that can cross-reference data provided by the users with the accidents one, in order to identify unsafe areas and suggest possible interventions.

1.1.2 Goals

Basic users:

[G.BU1] Basic users can report traffic violations.

[G.BU2] Basic users can view a data clustering about violations that had occurred.

Authorities:

[G.A1] Authorities should choose to receive anonymous notifications in real time about new violations.

[G.A2] Authorities should view and reserve a violation.

[G.A3] Authorities can view both data clustering and specific data about violations that had occurred.

Municipalities:

[G.M1] Municipalities can identify potential unsafe zones.

[G.M2] Municipalities can receive a safety report with suggestions to reduce accidents.

1.2 Scope

1.2.1 World

There are three main types of actors in our world: citizen, authorities and municipalities. Citizen are interested in reporting traffic violations and receiving information about violations in certain areas, authorities and municipalities are interested in exploiting the data gathered from the citizen: the former want to get notified when new violations occur in order to generate traffic tickets, the latter want to identify unsafe zones and to receive possible solutions.
SafeStreets is the service that acts as a bridge between these actors' needs.

1.2.2 Phenomena

Phenomena that occur in the world and that are related to the system application domain are:

- Traffic violations occur in a city;
- Authority patrols an area and makes traffic tickets;
- People are interested in analyzing violation data;
- Municipality wants to reduce the number of accidents.

The system shares also some events with the world in order to communicate with it. Phenomena that occur in the world and are observed by the machine are:

- A user registers and logs in filling the various form;
- A basic user fills the violation data and sends a new report;
- An authority manages notifications, enabling or disabling them;
- An authority researches a violation among those of civilians in which he/she may be interested;
- An authority examines the details of a violation;
- An authority reserves a violation;
- App user views mined data on a map in his/her smartphone;
- Municipality analyzes unsafe zones;
- Municipality views safety report with suggestions for reducing accidents.

On the other hand, aspects generated by the machine and observed by the world are:

- The system tracks the position of users;
- The system uploads, receives and confirms data insert by users through an acknowledgement (login credentials, new violation reports, etc.);
- A connection error occurs, the system notifies the issue to users;
- The system generates notifications about new violations;
- The system loads safety reports for the municipalities, with suggestions to reduce accidents;
- The system loads and renders graphically data to the user (violations list, detail of a violation, etc.)
- The system loads into a map the mined data as highlight zone and shows them to users.

Finally, phenomena inside the machine and not visible directly from the world are:

- The system stores and reads data (users' data, report data, etc.);
- The system deletes data account;
- The system checks authority's personal ID;
- The system retrieves data from municipality database;
- The machine mines data, clustering both violation data and accidents data ;
- The machine compiles safety report;
- The algorithm analyzes photos reading the license plate of a vehicle;

1.3 Definitions, acronyms, abbreviations

1.3.1 Definitions

User	Any kind of person who uses the system (basic user, authority and municipality).
App User	Any kind of person who uses the system through the App (basic user and authority).
Basic user	Citizen who can report a traffic violation and view a data clustering about violations that had occurred.
Authority	Recognized entity which can empower the law (ex. local police).

Municipality	Authority recognized by the State who holds the government in an area.
Unsafe zone	Area of the city where accidents happen frequently.
Safety report	Document with all information about the safety of a municipality. It consists in three parts: raw data, accidents' analysis and possible solution. *(RASD, section 2.2.4)*
Report / Violation report	Organized set of information collected by basic users in order to denounce a traffic violation.
Matricula / Personal ID	A code able to identify uniquely an authority, stored in state DB.
Area / zone	Region on a map of at least 5 Km^2.
Highlight zone	Area on a map where are shown the data cluster. The information shown consists in location, type of violation and occurrences.

1.3.2 Acronyms

S2B	Software to Be
IEEE	Institute of Electrical and Electronics Engineers
DB	Database
DW	Datawarehouse
DBMS	Database Management System
API	Application Programming Interface
DFM	Dimensional Fact Model
ER	Entity-Relationship
VM	Virtual Machine

1.3.3 Abbreviations

…

1.4 Revision history

Date	Version	Log

9/12/2019	v. 1	First DD release

1.5 Reference Documents

...

1.6 Document Structure

This document is composed into 6 sections, organised as follow:

- *Section 1*
- *Section 2*
- *Section 3*
- *Section 4*
- *Section 5*
- *Section 6*

2 Architectural Design

2.1 Overview

The application architecture is a four-tier architecture; it enables the distribution of application functionality across four independent systems: a presentation layer, a web layer, a business logic layer and a data layer. The modular approach allows maintenance and scalability.

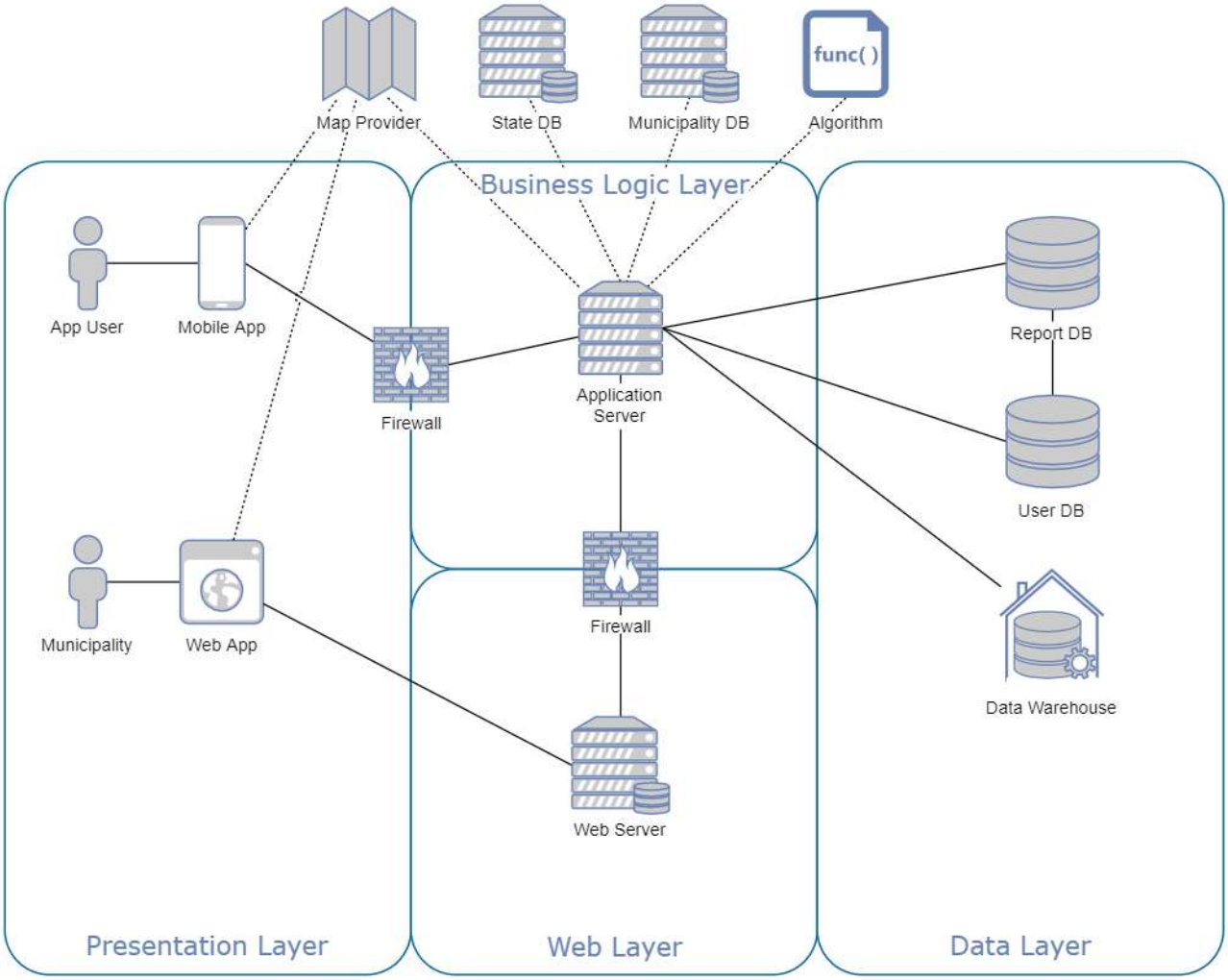


Figure 1: *Overall architecture of the system*

The main components of the systems are the following:

App Application installed on App Users' device that communicate with the system. Its purpose is to show data to the App User and to forward

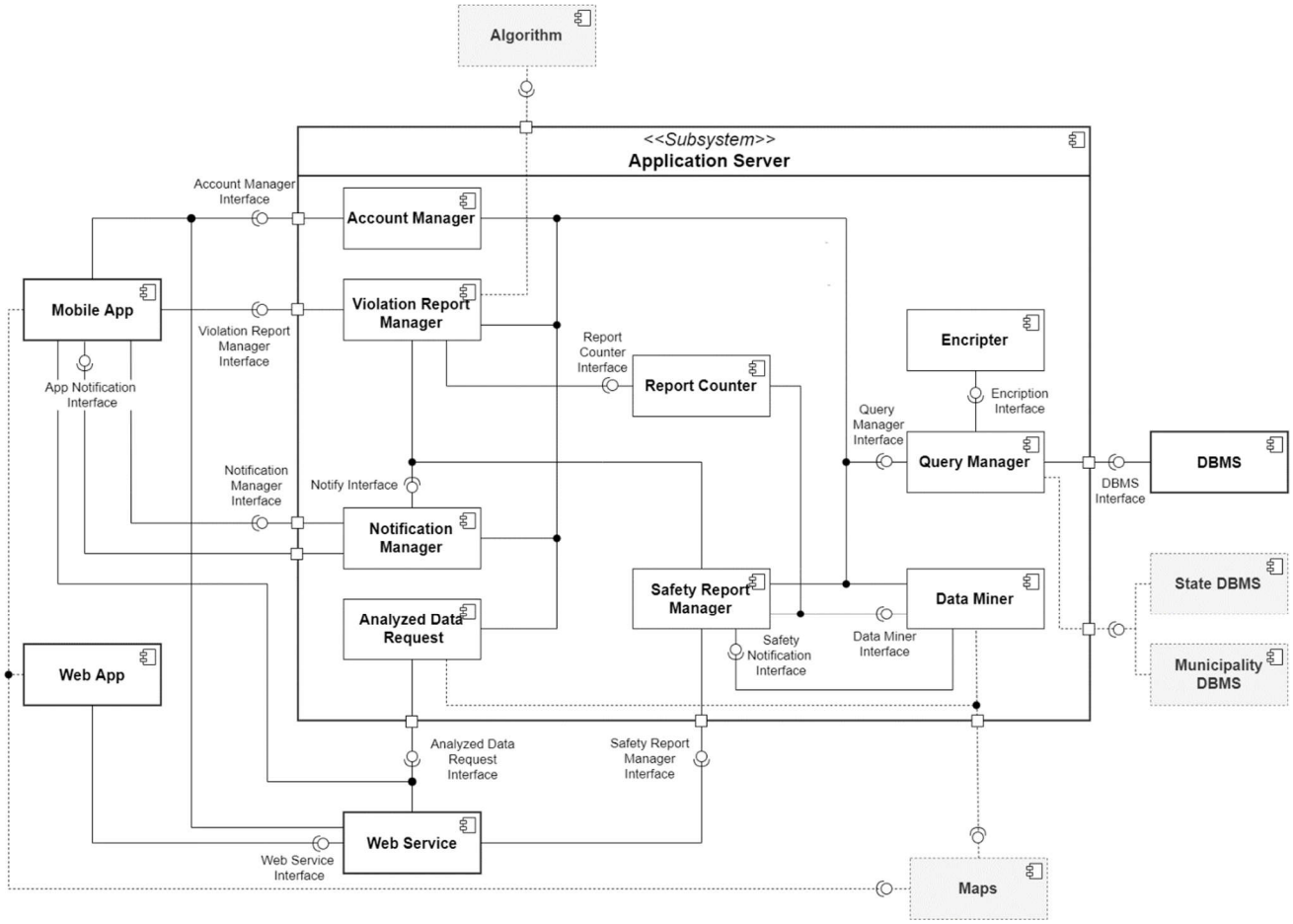
	his/her requests to the Application Server. The application will be available for both Android and iOS systems.
Web App	Web page accessible from *SafeStreets* main site, that communicate with the system. Its purpose it to show safety report to the municipality and to forward its requests to the Application Server through the Web Server. The web app will work with the most modern internet browsers.
Application Server	Main back-end component on which the logic of the application takes place: it elaborates the requests coming both from app and web app, it interacts with the data layer and web layer and it communicates with the external systems.
Web Server	Back-end component of the web app, it forwards the requests from the municipalities to the application server.
Databases	Components responsible for data storage; it is divided into two main DBs (Report DB and User DB) and into one main Data Warehouse.
External Systems	Systems that interact with *SafeStreets*; they provide functionalities not internally developed (such as the algorithm that reads the plate from the vehicles) or they are needed in order to provide *SafeStreets* main functionality (such as State DB for authorities' registration).

2.2 Component View

In this section we will analyze every high-level component in terms of its subcomponents and provide the main interface interaction between different components.

External services, such as *Maps* and *Algorithm*, are presented as black-box and expose only the interfaces used by our system.

For details on component interfaces see Section 2.5.



2.2.1 App Components

The application component is the front-end of the system, through with user interact with the system. It shall only render interfaces in order to exploit all system functionality and all computational tasks shall oversee the Application Server, so the front-end shall be able to communicate frequently with it. The only logic incorporated in the application component is the checking of basilar incorrect data type, e.g. incomplete forms, invalid date or email without @ character.

Map pictures are provided thanks to maps provider's API which render maps directly on the client side.

The application component consists in two subcomponents as shown in FIGUR XXX:

Mobile app is provided to basic user and authorities in order to manage user's account, report a violation, view violation details, view statistic map and receive notifications.

Web app, instead, is provide to partner municipality for managing account, requesting a safety report and viewing unsafe areas.

User Interfaces for both mobile app and web app are shown in section 2.3 (and 3.1 of RASD).

2.2.2 Web Service

A ***web server*** is required in order to provide a web site for the municipality.

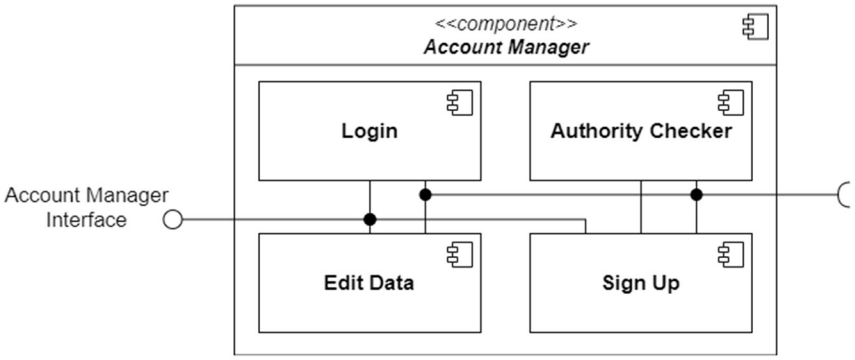
The server receives HTTPs requests from the user and forward them to the application server in order to collect all data required for rendering web pages.

2.2.3 Application Server Components

Application Server exploit the business logic of the S2B, its role is to compute all data needed by the system and coordinate the flow of information between application layer and data layer. Indeed, it is the only component directly connected with the DBMS and no one else can access the data, neither external system.

Components of the Application Server are:

Account Manager handles all the operations link with users' account. A connection with the DBMS is required to read, store and delate account data. In order to exploit *SafeStreets* functionality users must create an account and be authenticated by this module.



This component relies on *Sign Up* subcomponent to allow user registration, it checks if the credentials are unique or if an account already exists; through the *Authority Checker* verifies Personal ID and department of the authorities connecting to the state DBMS trough the *Query Manager*, in order to let only real authorities to register. *Login* subcomponents handle account authentication by checking credentials, while *Edit Data* modify in the DB user info, such as password, or deleting the account on user request.

Violation Report Manager receives new report submissions by basis users and requests from authorities to load violation report data. When a new violation is reported, this component asks to the image scanning algorithm to read the license

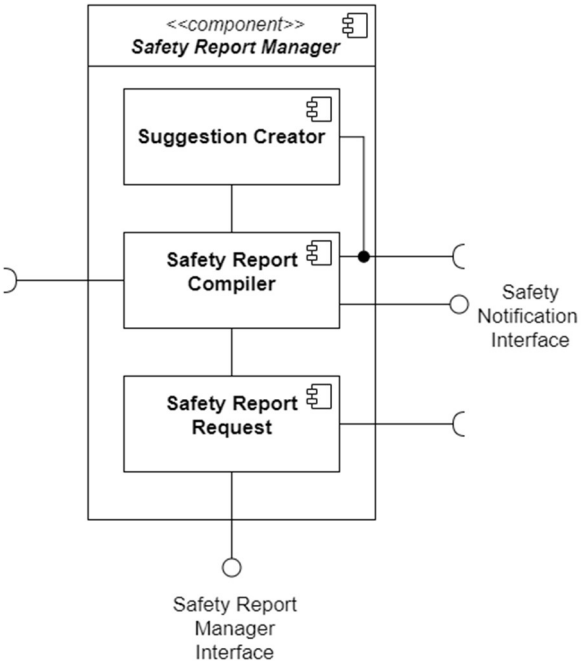
plate of the vehicle from the photos taken by the user. Lastly, it manages the reservation of a report.

Report counter keeps the account of how many reports have been recorded in order to activate the *data miner* component once XXX is reached. Every time data miner is activated, the counter is reset.

Notification Manager receive authorities' notifications preferences and stores them. It sends notifications to mobile app every time a new report is submitted in a zone interested by the authority. It also sends an email to municipalities, notifying them a new safety report has been generated.

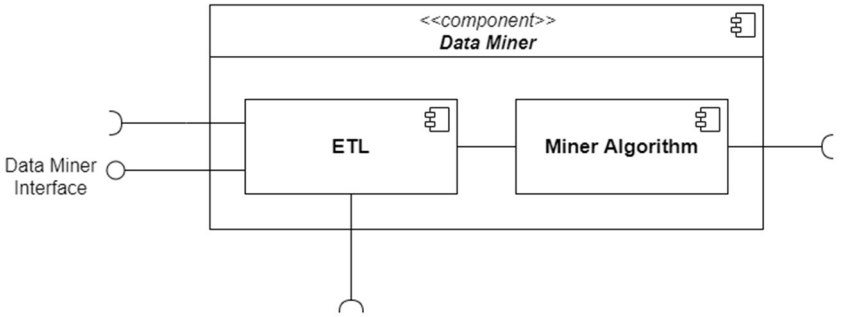
Analyzed Data Request sends to mobile app and web server highlighted and unsafe areas data of the zones requested by the users render on a map, retrieving them from the Datawarehouse.

Safety Report Manager handles safety report request, creating a new one or loading the last generated.



Safety Report Request subcomponent activates the *data miner* if new report is requested, otherwise asks directly to *Safety Report Compiler* to compile the last one, loading data from the database. *Safety Report Compiler* is also notified when data miner ends. *Suggestion Creator* find suggestion to reduce accidents crossing accidents mined data with violation reports one. It works only when new report is request, otherwise suggestions are loaded from the database. For details on safety report compiling see paragraph 2.4.

Data Miner classifies both violation reports data and municipality's accidents data separately. It's activated asynchronously by *Report Counter* or *Safety Report Manager* and can notify other components, i.d. *Notification Manager*, when it ends.



ETL subcomponents retrieves data needed to mine correctly violation and accidents, extraction dynamically only data not already mined. For location mining, is used information provided by *maps* external system. It also transforms data, deleting duplication (e.g. two report from two different users for the same vehicle in the same violation) and standardizing data format. Particular attention is paid to the data of the municipalities as they come from an external system and may exist inconsistencies. *Miner Algorithm* runs the mining algorithm on data provided by the *ETL*. All data are lastly store in the Datawarehouse through the DBMS. For details on mined data see paragraph 2.2.4.

Query Manager is the only which interfaces with the data layer, it receives request from other components about what read, store or delate from the database. It computes queries both for *SafeStreets*' DBMS and for external DBMS, i.e. municipalities and state ones. Queries can be refused by DBMS according to visibility privileges of user (i.e. basic user can't see report details).

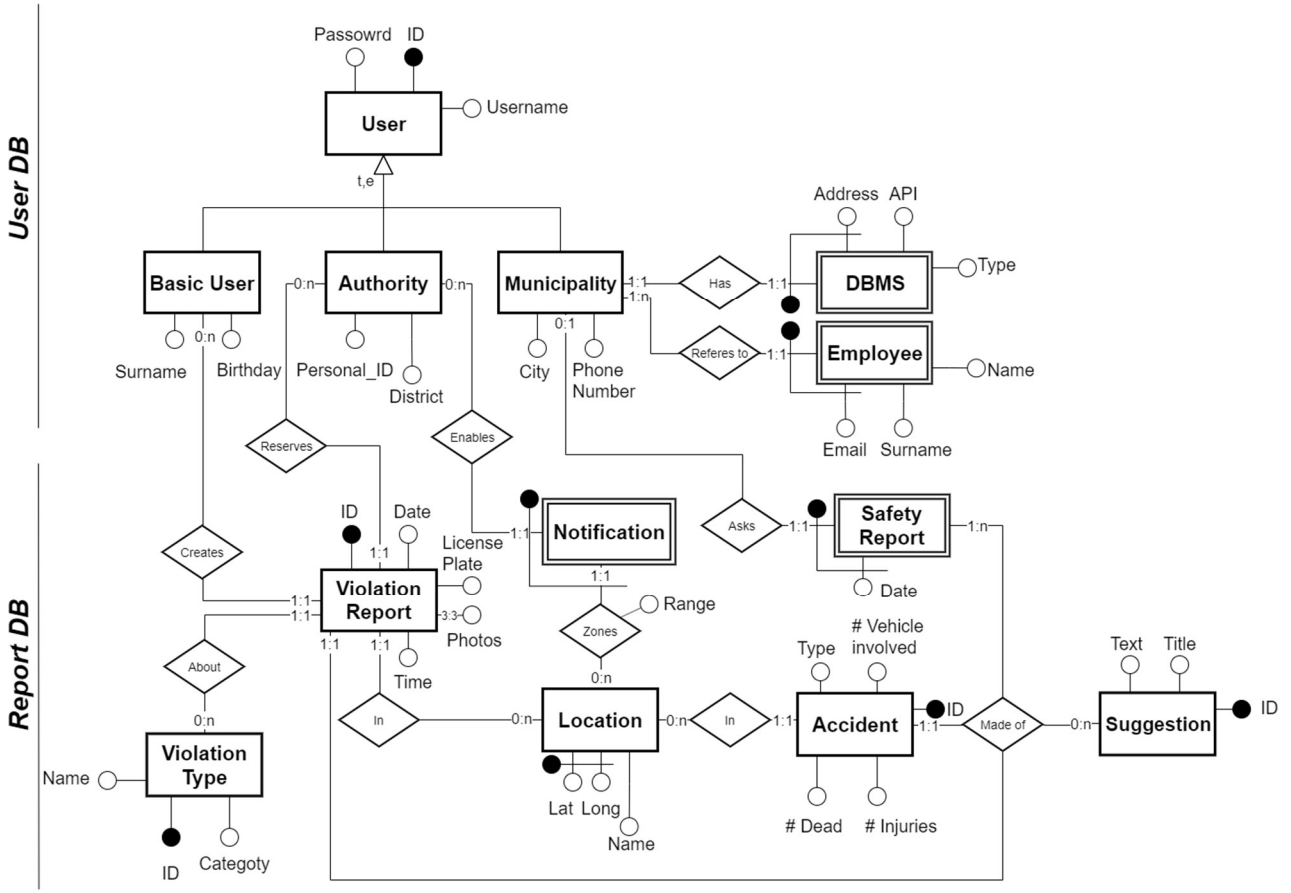
Encrypter runs encryption/decryption algorithm on some data, such as account password, by request of *Query Manager*.

2.2.4 Data Components

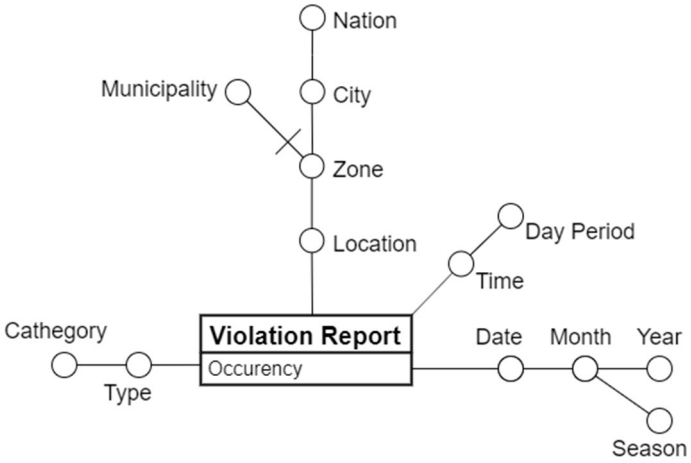
Data layer is composed by two relational databases (the former for user data and the latter the reports data) and a Datawarehouse where are stored the mined data. Query are managed by a unique DBMS that optimizes and elaborates them in parallel.

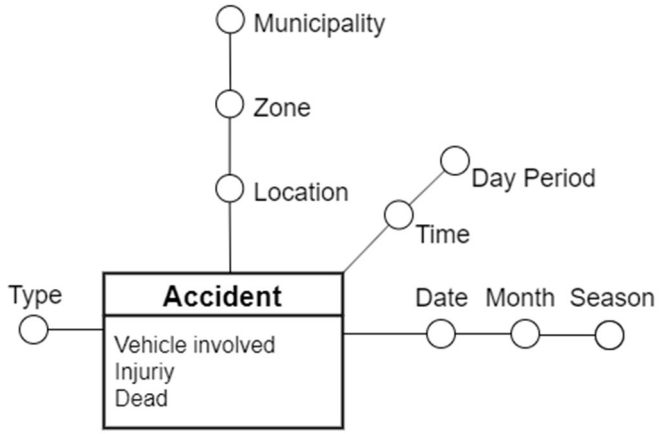
An entity-relationship diagram is provided for relational databases, users and reports DBs are linked for a better overall view of the system. Some observations about the ER:

- Cardinality between *municipality* and *safety report* is 0 if a safety report has never been generated, otherwise will be always 1. Indeed, when a new report is compiled, the old one will be overwritten.
- *Suggestion* and *violation type* entities contain all the possible value, so suggestions are generic and pre-loaded in the database and they will be linked to a specific safety report by the *Safety Report Manager* component.
- *Accidents* doesn't contain all accident data retrieved by municipality but only the ones shown in the safety report. This choice has been made to keep the size of the database contained.
- *Username* attribute in *user* entity could be either an email or an identification code base on the type of user.



The dimensional fact models below are provided in order to better illustrate the dimensions through which data are mined. Accidents and violation reports have similar DFM in order to facilitate the integration between each other in the safety report compiling phase.


Nation
Municipality
City
Zone
Location
Day Period
Time
Cathegory
Violation Report
Occurency
Date
Month
Year
Type
Season


Municipality
Zone
Location
Day Period
Time
Type
Accident
Vehicle involved
Injuriy
Dead
Date
Month
Season

2.3 Deployment View

We decided to deploy our system exploiting the services provided by Microsoft Azure (Figure 2). Microsoft Azure handles server calls in a distributive way, and offers reliability, security and scalability. The pros may be summarized as follow:

- **Load balancing:**

Microsoft Azure offers load balancing of the traffic that arrives at the front-end to back-end pool instances, in order to distribute equally the incoming traffic.

- **Virtual Machines:**

Microsoft Azure provides VMs that will host out apps and services, without having to buy and maintain physical hardware. Azure VMs' guarantee flexibility of virtualization.

- **SQL Databases:**

Azure offers managed SQL relational Databases.

- **Scalability and Flexibility:**

The flexible structure of Azure allows to scale up the data storage capabilities dynamically.

- **Cost-Efficiency:**

Microsoft Azure offers a *pay-as-you-go* payment plan, that allows to manage budget efficiently, paying only for the services that are used in each billing period.

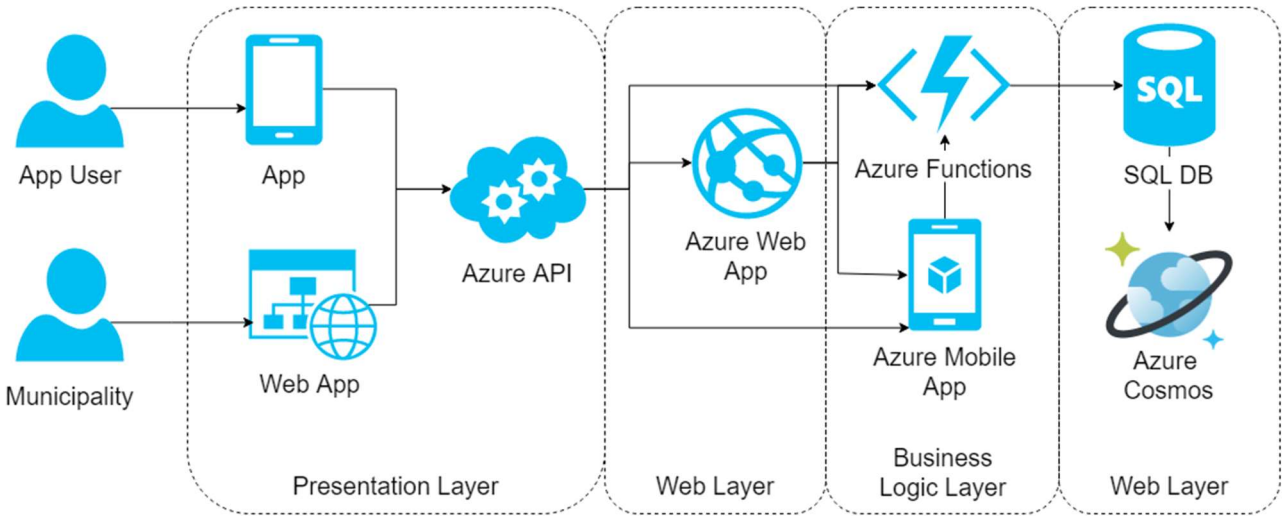


Figure 2: *Architectural layers on Azure services*

In the deployment diagram we have the following components:

Mobile device Runs the mobile application of the presentation layer; the application is compatible with both Android and iOS. The mobile device shall have a built-in camera and GPS functionalities.

PC	Needs to reach *SafeStreets* web page; the PC shall use modern browsers, such as Chrome.
Azure Web App	Platform where is possible to create and deploy web applications
Azure Mobile App	Service able to provide authentication capabilities for mobile applications.
Azure Functions	Platform that allows to accelerate and simplify application development; connection to other services without hard-coding integrations.
Microsoft Azure SQL	Microsoft Azure relational database service where resiliency, scale, and maintenance are primarily handled by the platform.
Microsoft Azure Cosmos	zzz
Azure API	

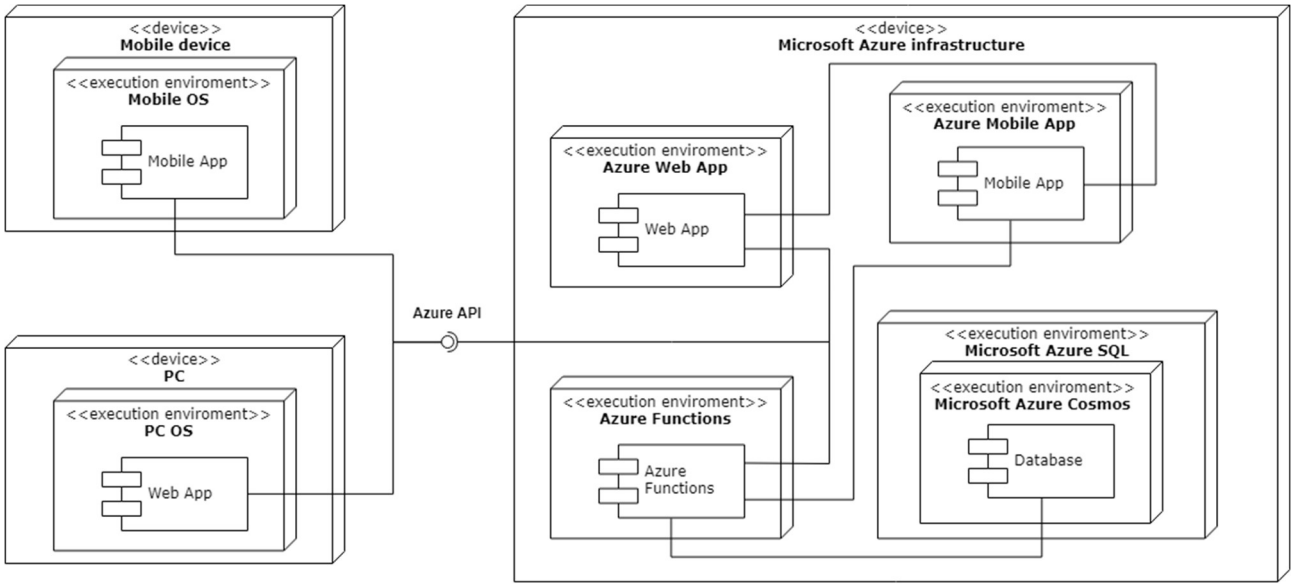


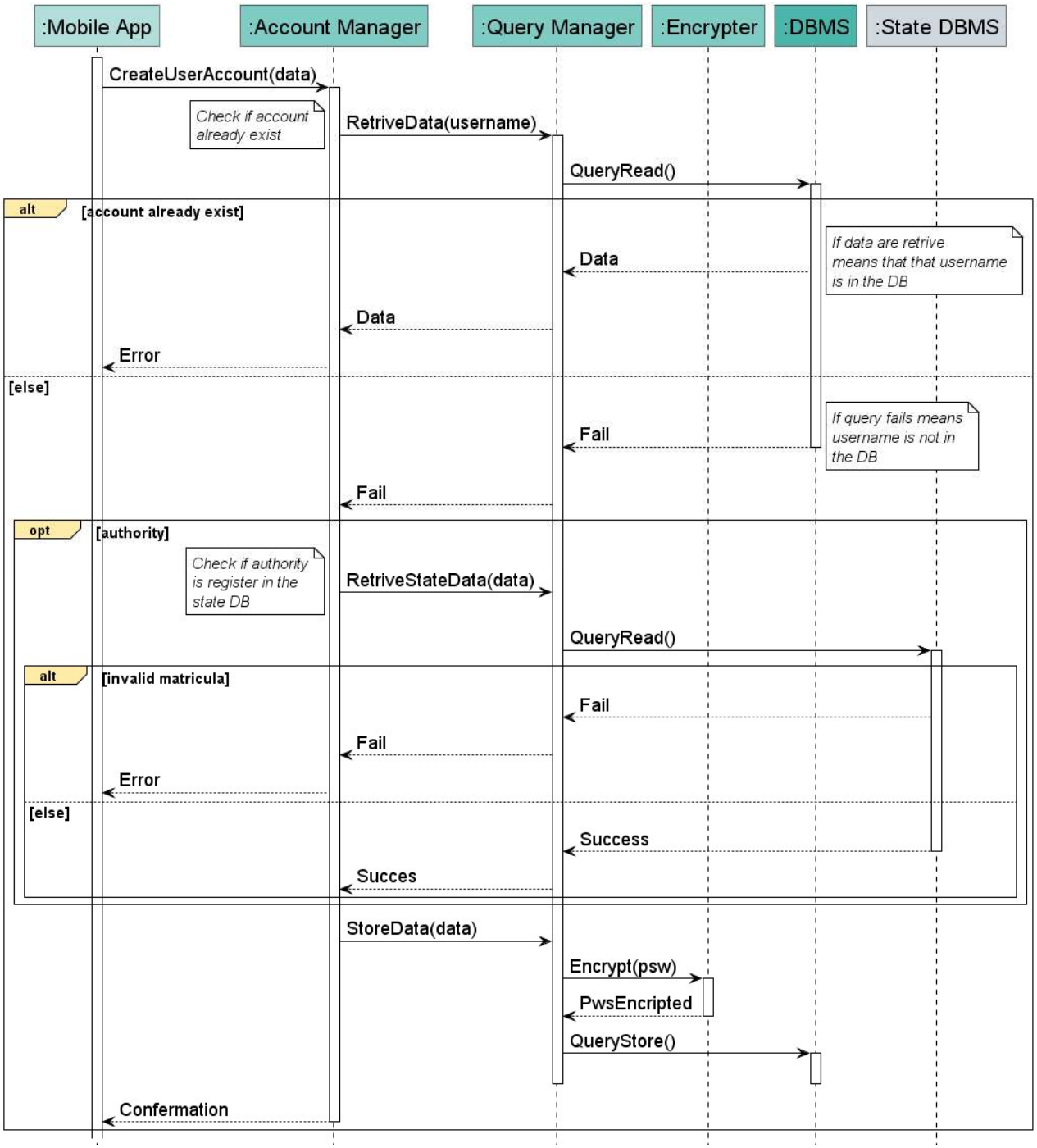
Figure 3: *Deployment Diagram*

2.4 Runtime View

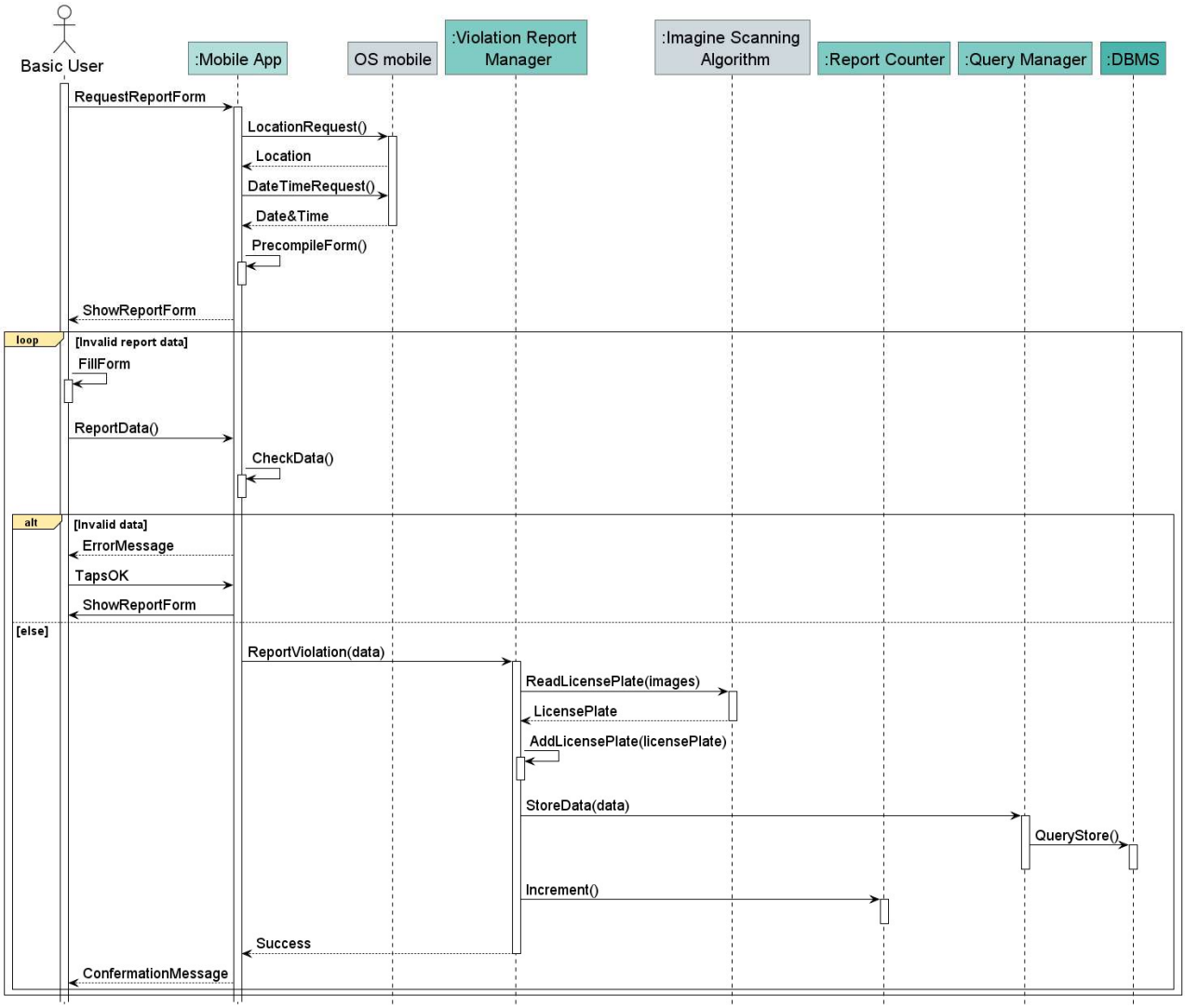
In this section we will present some sequence diagrams that show the major interaction processes between the system components. All the methods performed between components are described in Section 2.5 and 2.6.

In the following sequence diagram all operation, unless otherwise specified, have been successful.

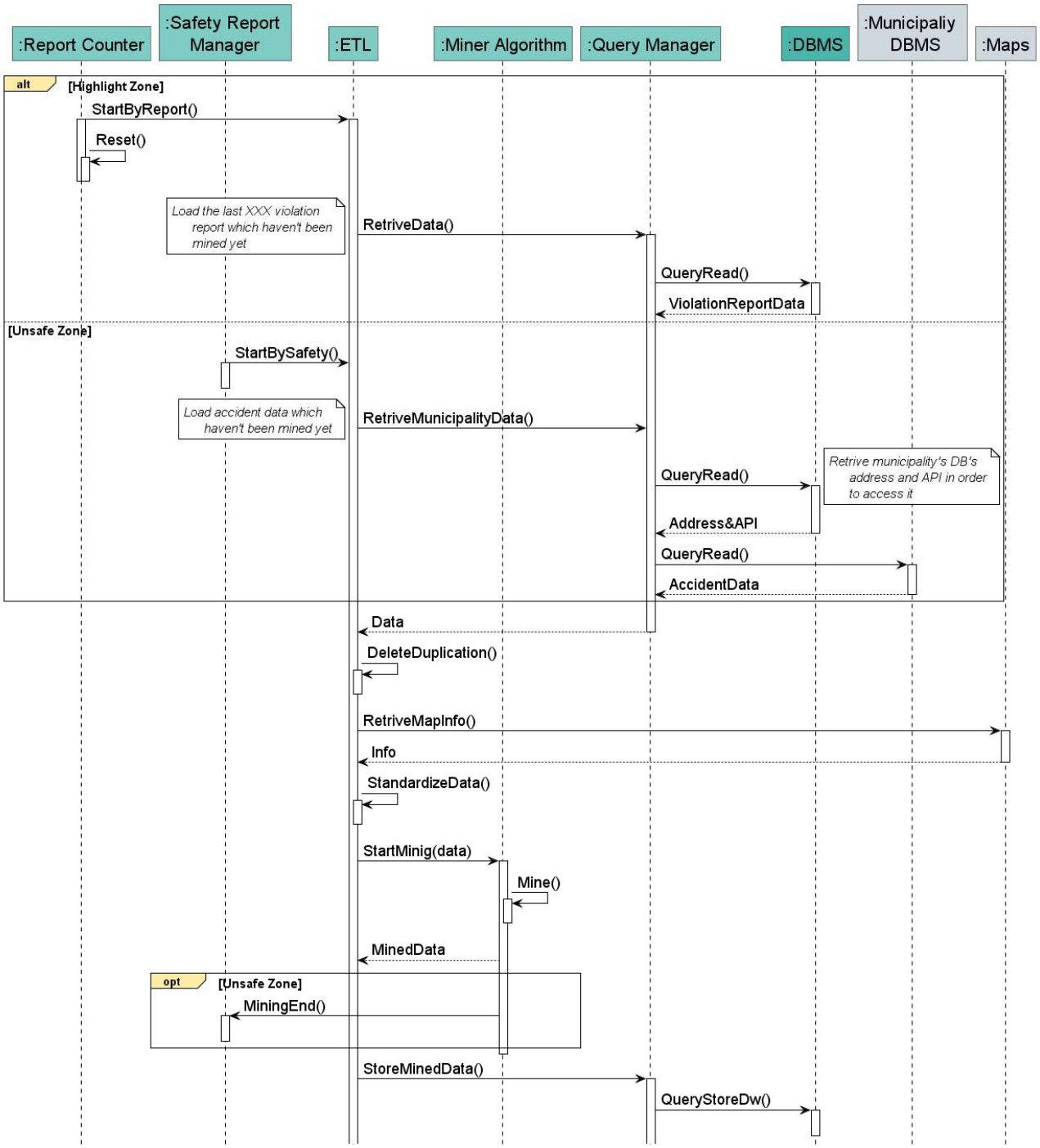
2.4.1 App user registration



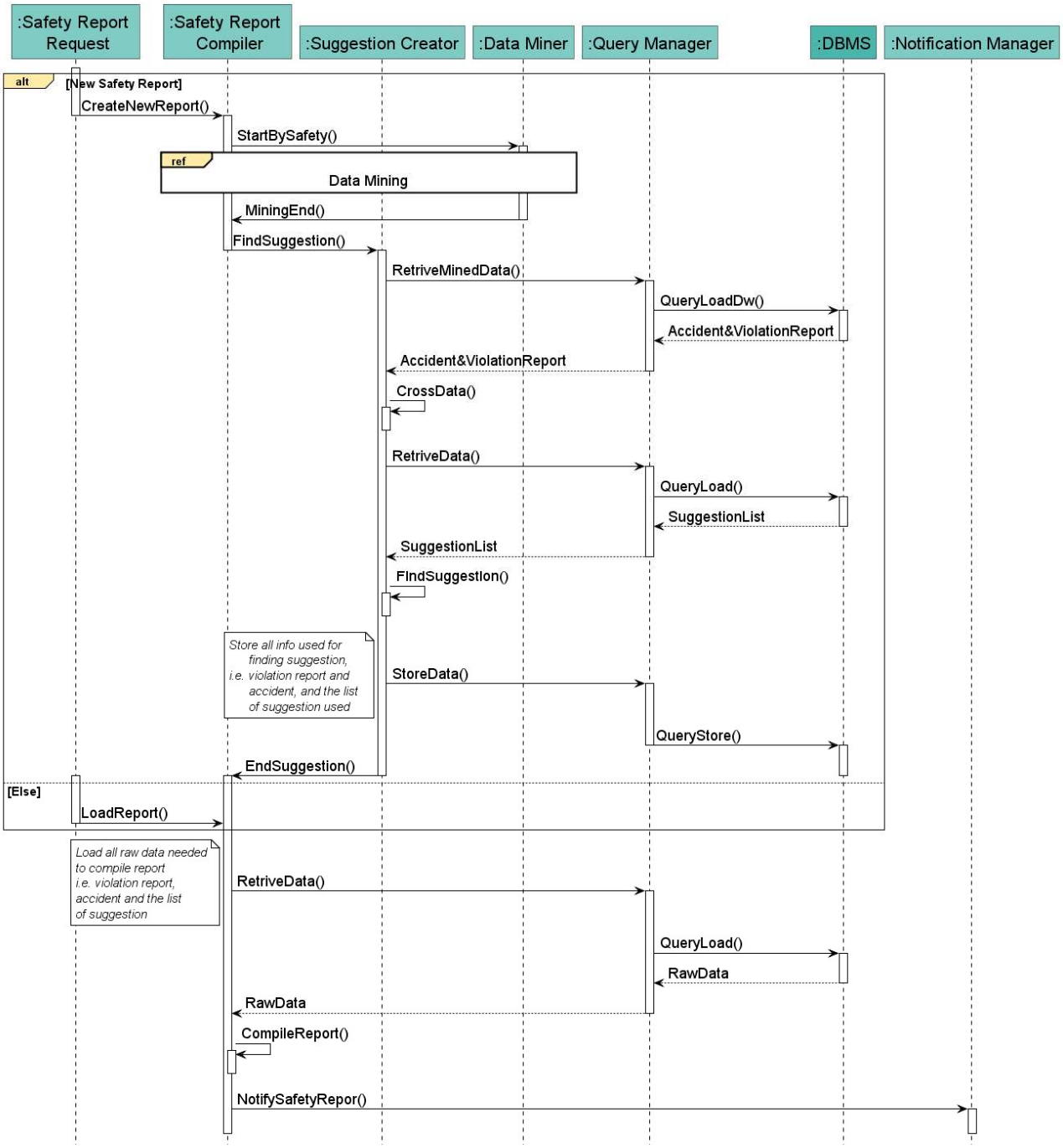
2.4.2 Violation reporting



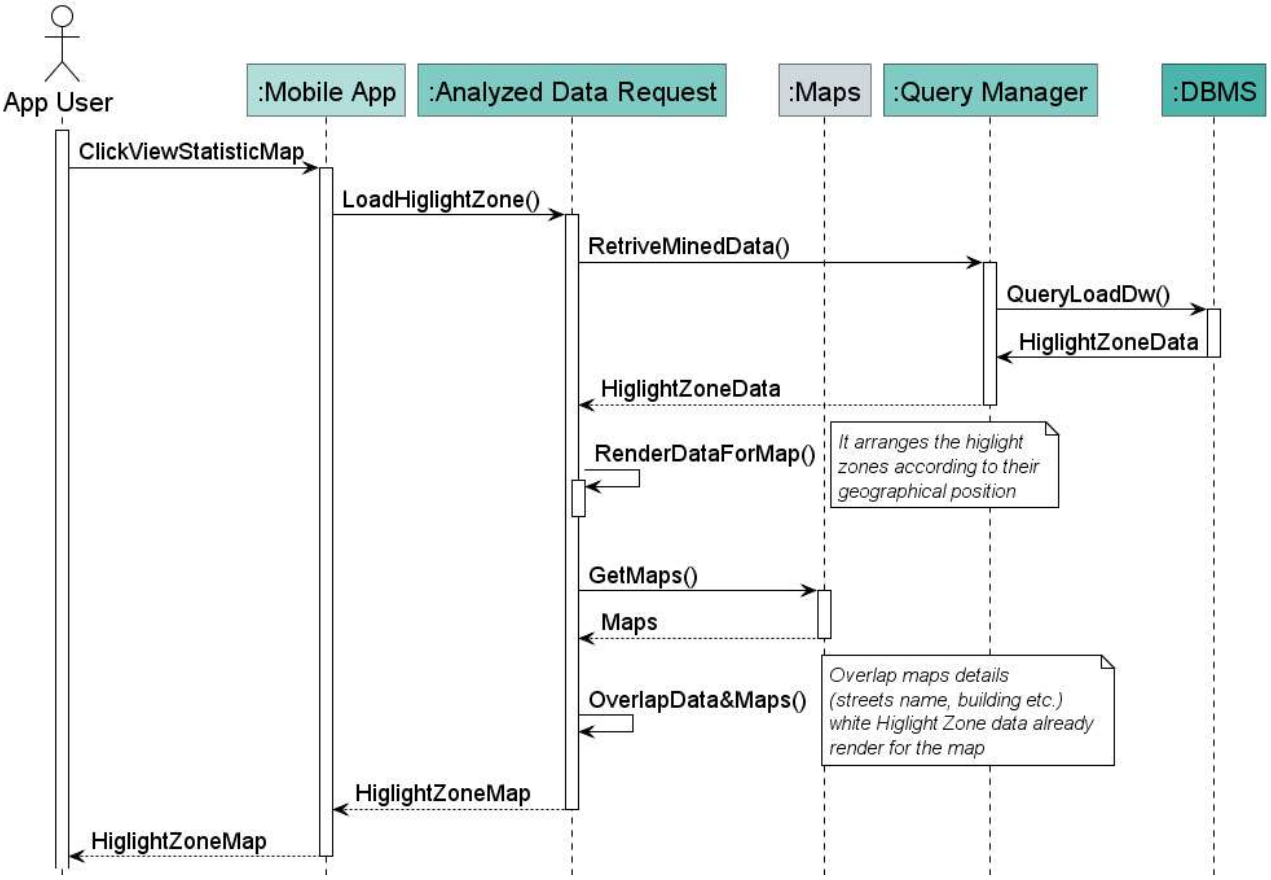
2.4.3 Mining data

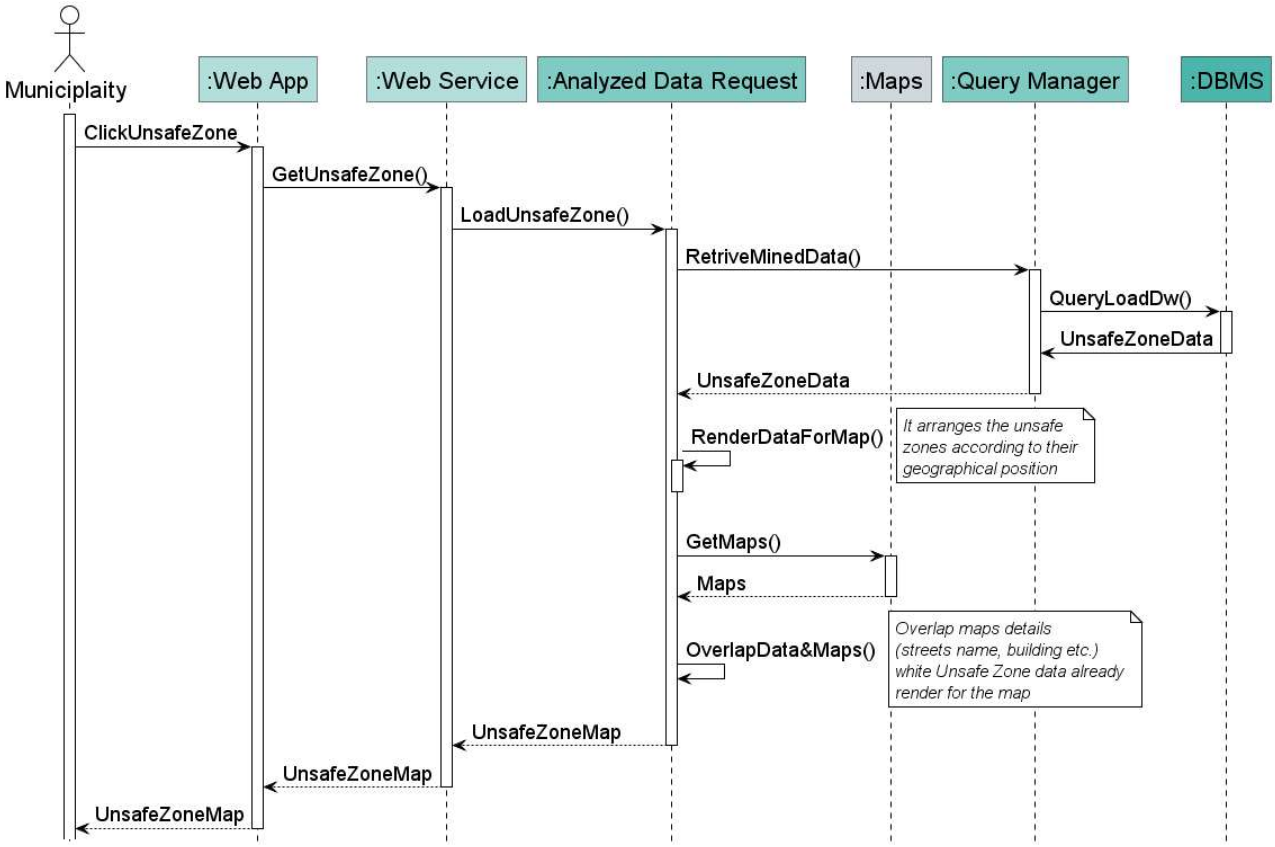


2.4.4 Safety report compiling



2.4.5 Analyzed data rendering





2.5 Application Server Interfaces

In this section we describe the methods offered by the interfaces of each Application Server component as show in section 2.2. Parameters and return type are to be considered as mere indications and have not been provided for all the methods listed because strictly connected to implementation and may change during the development.

2.5.1 Account Manager Interface

CreateUserAccount	Create a new account for a basic user providing name, surname, email address, password, date of birth as specified in the RASD section 2.2.1. The return value specified if the procedure was successful or if an account with some credential already exist.
CreateAuhorityAccount	Create a new account for an authority providing name, district's name, personal ID, email, password as specified in the RASD section 2.2.1. The return value specified if the procedure was successful, if an account with some credential already exist or if the checking of personal ID failed.
CreateMunicipalityAccount	Create a new account for a municipality providing name of the municipality, referencing email, referencing name, referencing surname, referencing phone number as specified in the RASD section 2.2.1. The return value specified if the procedure was successful or if some incorrect information made it abort.
Login	By providing email/identification code and password, a user can login into his/her account and exploit system functionalities. The return value is positive if information provided are correct and negative if there's no account that matches given credentials.
DeleteAccount	Delete permanently a user account providing email/identification code and password. The return value is positive if the deletion was successful and negative if something made the deletion abort.

EditInfo	Updates user's profile information with the new set of information passed as parameter. The return value confirms if the procedure ended correctly.

2.5.2 Violation Report Manager Interface

ReportData	Upload a new violation report by a basic user, providing as parameters all the info required in a report (RASD section 2.2.3). The return value confirms if the procedure ended correctly.
LoadNotifiedReport	Return to an authority a list of all violation reports for the last 24h located in zones where he/she have enabled the notification. Error code shall return, or an exception shall be thrown in case of unsuccess.
LoadSingleReport	Return all the details of the report passed as parameter (basic user who report the violation is omitted). Error code shall return, or an exception shall be thrown in case of unsuccess.
ReserveReport	Reserve a report by an authority, when return the report shall not be reservable by anyone else. The return value confirms if the procedure ended correctly.
CancelReservationReport	Cancel a reservation of a report by an authority, when return the report shall be reservable again. The return value confirms if the procedure ended correctly.

2.5.3 Analyzed Data Request Interface

LoadHighledZone	Return all mined violation report data of the zone passed by parameter. Error code shall return in case of unsuccess.
LoadHighledZoneByType	Return mined violation report data of the zone and type passed by parameters. Error code shall return, or an exception shall be thrown in case of unsuccess.

LoadUnsafeZone	Return all mined accidents data of the zone passed by parameter. If municipality tried to load data outside its area an error code shall return, or an exception shall be thrown.

2.5.4 Notification Manager Interface

AddNotification	Add a new zone to the areas where an authority wants to be notified when a violation report is uploaded. Error code shall return, or an exception shall be thrown in case of unsuccess.
DeleteNotification	Delete a zone from the area where an authority wants to be notified when a violation report is uploaded. Error code shall return, or an exception shall be thrown in case of unsuccess

2.5.5 Notify Interface

NotifySafetyReport	Called when a new safety report has been compiled and it is ready for the municipality. It shall send an email to municipality employee with a link to the report. Error code shall return, or an exception shall be thrown if the email can't be sent.
NotifyViolationReport	When a new violation report is correctly uploaded, this method is called and notify all the authority interested in the zone where the violation occurred. Error code shall return, or an exception shall be thrown if the notification can't be sent to all authorities.

2.5.6 Safety Report Manager Interface

CreateNewReport	Start the process to create a new safety report. Error code shall return, or an exception shall be thrown if process can't start correctly.
LoadReport	Load from database the last safety report compiled for the municipality who called this method. Error code shall

return, or an exception shall be thrown if report can't be load correctly.

2.5.7 Safety Notification Interface

MiningEnd	Called from *Data Miner* component when the process of mining accident data is end. This method starts the compilation of the safety report retrieving the data just mined from the Datawarehouse.

2.5.8 Report Counter Interface

Increment	Increment by one the report counter, called whenever a new violation report is stored. The return value confirms if the procedure ended correctly.

2.5.9 Query Manager Interface

RetriveData	Make a query in order to read some data from the relational database. Data to be queried are passed as parameter. Error code shall return, or an exception shall be thrown if query can't be done.
RetriveMinedData	Make a query in order to read some data from the Datawarehouse. Data to be queried are passed as parameter. Error code shall return, or an exception shall be thrown if query can't be done.
RetriveStateData	Make a query in order to read some data from state database. Data to be queried are passed as parameter and only read query are allowed. Error code shall return, or an exception shall be thrown if query can't be done.
RetriveMunicipalityData	Make a query in order to read some data from municipality relational database. Databased and data to be queried are passed as parameter and only read query are allowed. Error code shall return, or an exception shall be thrown if query can't be done.

StoreData	Make a query in order to store some data in the relational database. Data to be stored are passed as parameter. Error code shall return, or an exception shall be thrown if query can't be done.
StoreMinedData	Make a query in order to store some mined data in the Datawarehouse. Data to be stored are passed as parameter. Error code shall return, or an exception shall be thrown if query can't be done.
DeleteUserData	Make a query in order to delete user data from the relational database. Data to be deleted are passed as parameter and only user information can be deleted. Error code shall return, or an exception shall be thrown if query can't be done.

2.5.10 Encrypter Interface

Encrypt	Return the value passed by parameter encrypted. Error code shall return in case of unsuccess.
Decrypt	Return the value by parameter decrypted. Error code shall return in case of unsuccess.

2.5.11 Data Miner Interface

StartByReport	Start the process to mine violation report data retrieved from the DB. Error code shall return, or an exception shall be thrown if process can't start correctly.
StartBySafety	Start the process to mine accident data retrieved from municipality DB. The municipality where data are retrieved shall be passed as parameter. Error code shall return, or an exception shall be thrown if process can't start correctly.

2.6 External Interfaces

For external interfaces are generally providing only the required method, names are omitted because could change according to the partner who provides the interface.

Maps Interface shall exploit a method for rendering map graphically on client side and another method for retrieving geographical information during the mining phase and for pre-rendering analyzed data.

Image Scanning Algorithm Interface shall provide a method where given an image with a vehicle as input, the license plate of that vehicle shall be the output.

Municipality DBMS Interfaces is required for each partner municipality and shall provide methods for only retrieving accidents data.

State DBMS Interfaces is required for each partner state and shall provide methods for only checking if an authority matricula is real or fake.

2.7 Selected architectural style and patterns

Mvc, observer – observsable

Suggestions:

adapter per potersi interfacciarsi alle diverse API di state DBMS and municipality DBMS
Observable per le notifiche alle autorità: lista di tutte le autorità divise per zona o una roba simile

3 User Interface Design

In the RASD document we have shown a series of mock-ups that show the application screens. Here we will extend the UI by providing the navigation flow between the screens.

The graphs should be read as follow: nodes for screens, referenced by figure number in the RASD document; arcs for buttons and square brackets for conditional branches.

3.1 Flow Graph in mobile App

Although there is only one mobile application, authorities and users have two partly different flow charts; as clearly shown on *Figure 1-2.* All mobile users must log in in order to exploit the functionalities; from the main page in possible to access to the sidebar, that allows users to log out or to delete their account; in addition, authorities can manage their notification settings.

In order to receive a notification, the authority must be logged into the app.

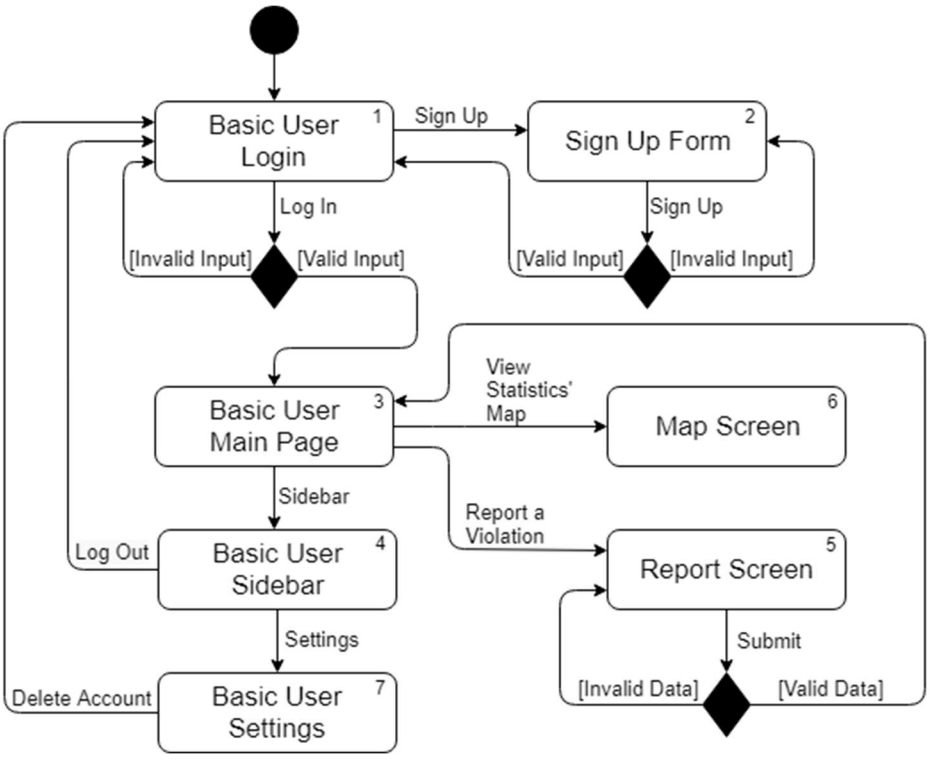


Figure 4: *Basic User's flow of screens*

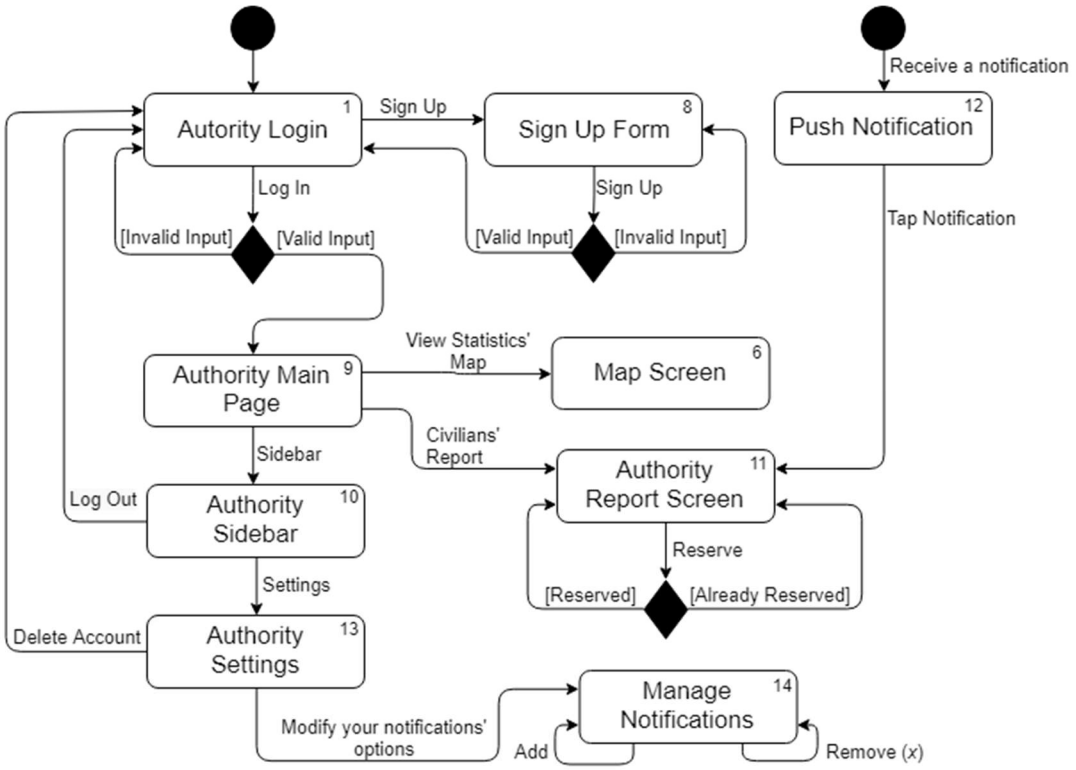


Figure 5: *Authority's flow of screens*

3.2 Flow Graph in Website

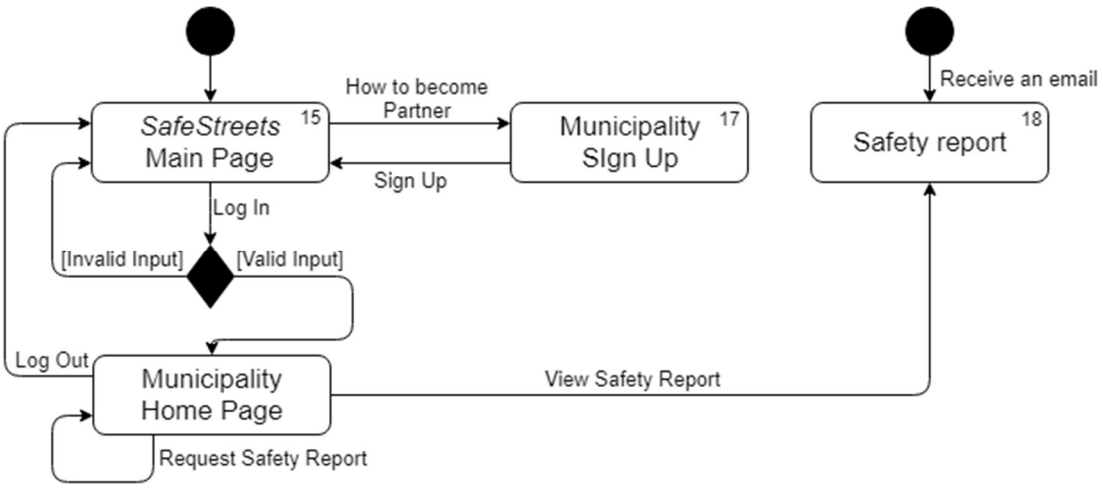


Figure 6: *Municipality's flow of screens*

4 Requirements Traceability

In the following table the requirements described in the RASD document (reported along with a short description) are matched with the components described in the Section 2.2.

Requirement	Description	Components
R.1	*User registration*	Account Manager
R.2	*App users' distinguishability between basic users and authorities*	Mobile Application, Account Manager
R.3	*Authority validation upon registration*	Account Manager
R.4	*Account uniqueness*	Account Manager
R.5	*Users login*	Account Manager
R.6	*Users can exploit functionalities only after the log in*	Mobile Application, Web Application, Account Manager
R.7	*Account deletion*	Account Manager
R.8	*Basic user violation report's upload*	Violation Report Manager
R.9	*Data is saved internally*	Query manager, DBMS
R.10	*Data can't be erased*	Query manager, DBMS
R.11	*Violation reports are anonymous for Authorities*	Query manager, DBMS
R.12	*It is allowed to authorities to receive notifications regarding new violations*	Mobile Application
R.13	*The system shall generate notifications to authorities when a new report is registered*	Notification Manager
R.14	*Authority access to report details anonymously*	Violation Report Manager, Query Manager
R.15	*Basic user does not access to single report*	Mobile Application, Query Manager
R.16	*Authority reservation of violations*	Violation Report Manager
R.17	*New violation classification*	Report Counter, Data Miner
R.18	*Render violations graphically*	Mobile App
R.19	*List of all violations for authorities*	Mobile App

R.20	*Management of connection errors*	Web Server, Web App, Mobile App
R.M1	*Municipality's sign up*	Account Manager
R.M2	*Availability of accidents data from municipalities*	Data Miner
R.M3	*Data integration between SafeStreets and Municipality's DB*	Data Miner
R.M4	*Data mining for generate unsafe zones*	Data Miner
R.M5	*???*	Web App
R.M6	*???*	Safety Report Manager
R.M7	*Safety Report generation*	Safety Report Manager
R.M8	*Municipality's access to its own Safety Reports*	Safety Report Manager, Query Manager

5 Implementation, integration and test plan

//todo

Date	Falconi	Galli	Theme
20/11	2.5	2.5	Section 1, Overview Diagram, Component Diagram
21/11		2	ER, DFM
23/11	3.5		Flow graphs
24/11	3		Section 3, Section 4
26/11		3	Component Diagram
27/11		3	Section 2.2.2
28/11	2		Revision
29/11		4	Section 2.2.4, Sequence Diagram
30/11	4	3	Revision, Section 2.5, 2.1, diagrams
1/12	5	3	Section 2.3, 2.4, 2.5, 2.6, revision sections 2.5, 2.2
TOTAL	20	20.5	