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.
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	Data Warehouse
DBMS	Database Management System
API	Application Programming Interface

DFM	Dimensional Fact Model
ER	Entity-Relationship
VM	Virtual Machine
ETL	Extract Transform Load

1.3.3 Abbreviations

[G.BU$_n$] Basic users' nth goal;

[G.A$_n$] Authorities' nth goal;

[G.M$_n$] Municipalities' nth goal;

[R.$_n$] Nth requirement;

[R.M$_n$] Municipalities' nth requirement;

[R.P$_n$] Performance nth requirement.

1.4 Revision history

Date	Version	Log
9/12/2019	v. 1	First DD release

1.5 Reference Documents

Specification document: "SafeStreets Mandatory Project Assignment"

1.6 Document Structure

This document is composed into 7 sections, organised as follow:

- *Section 1* gives a short introduction to the project, giving a clear idea of who are the actors and what are the goals of the S2B;
- *Section 2* describes the high-level architecture, highlighting the main components, their interaction with runtime views and other design decisions;
- *Section 3* provide an overview about the UX flow;
- *Section 4* contains mapping between software requirements, described in the RASD, and design elements;
- *Section 5* identifies the order in which the component will be implemented, integrated and tested;
- *Section 6* shown the effort spent in developing the DD;
- *Section 7* contains the tool used and references.

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.

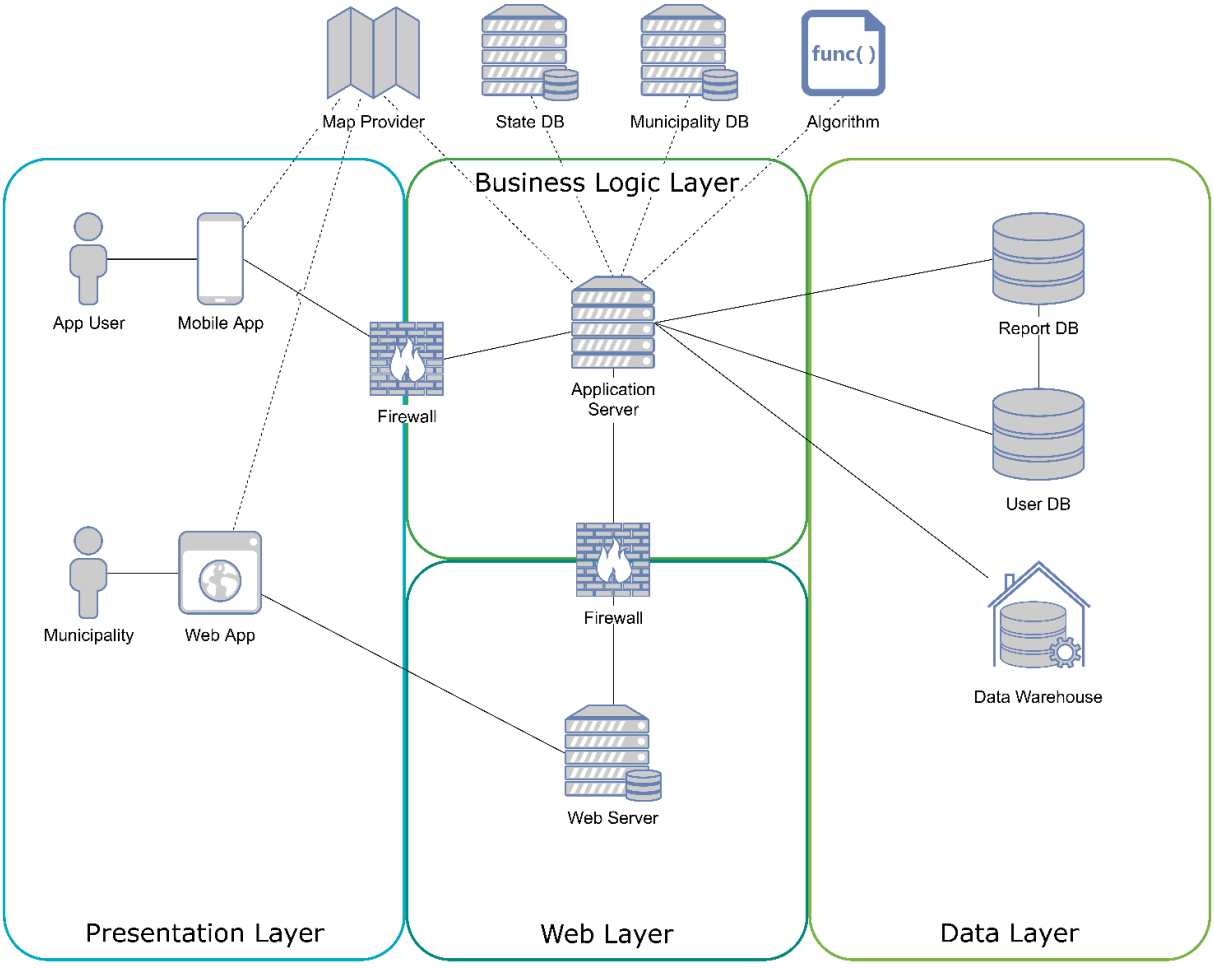


Figure 1: *Overall architecture of the system*

The main components of the systems are the following:

Mobile 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 unsafe zones and 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 web client 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.

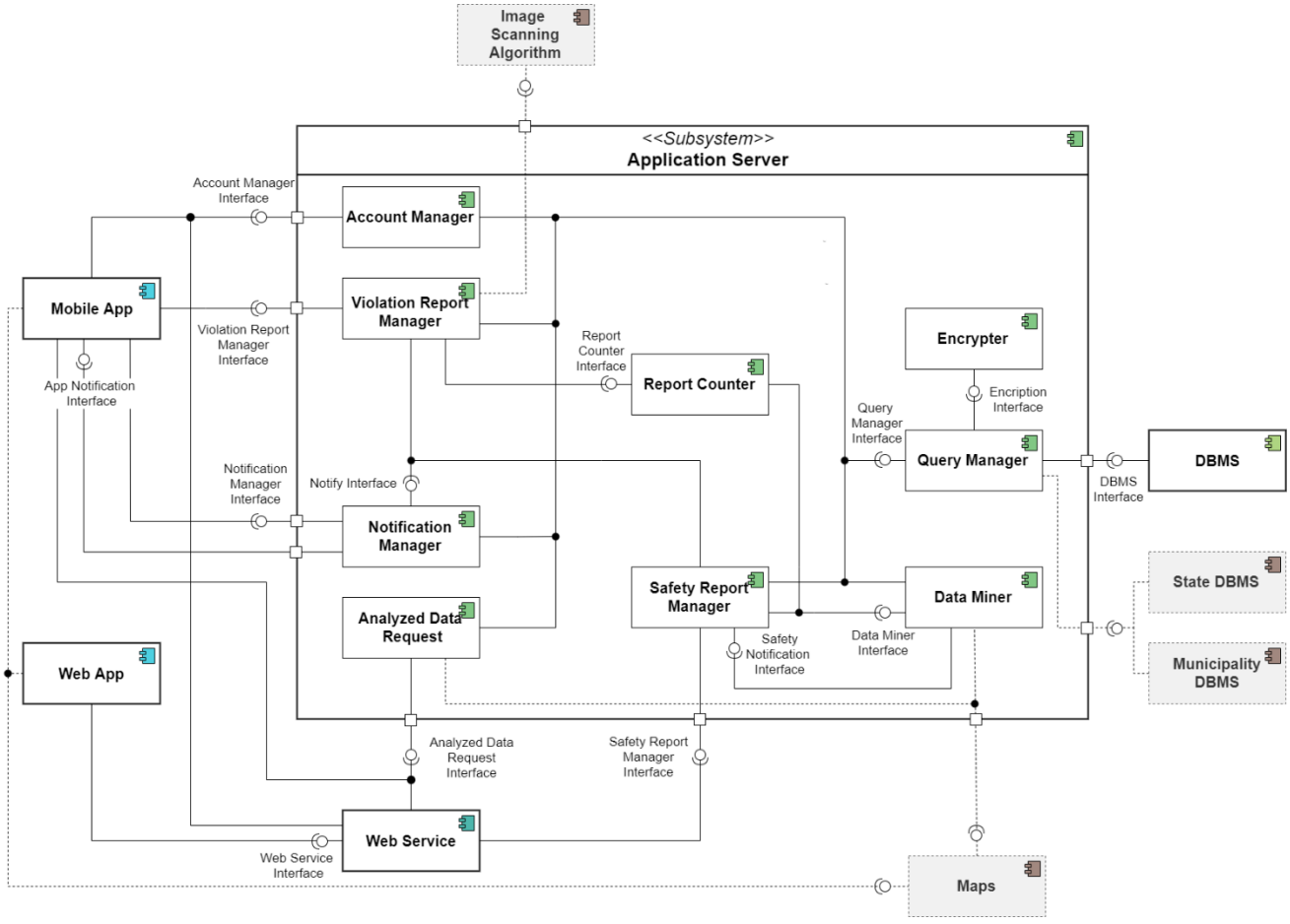


Figure 2: *Component Diagram*

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 ability to check basic 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 *Figure 2*:

Mobile app is provided to basic user and authorities in order to manage user's account, report a violation (exclusive to basic users), view violation details (exclusive to authorities), 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 delete account data. In order to exploit *SafeStreets* functionality users must create an account and be authenticated by this module.

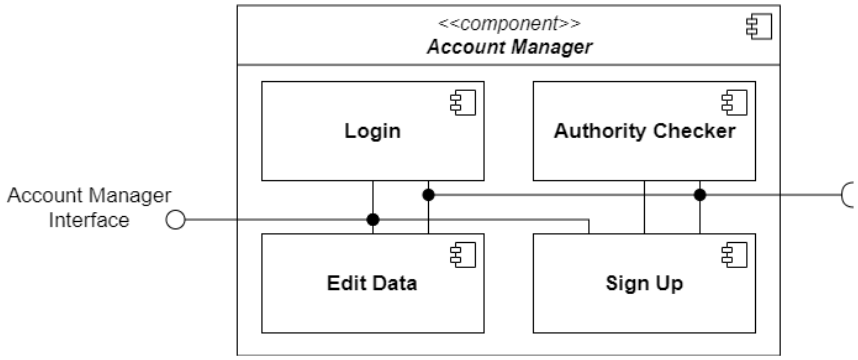


Figure 3: *Account Manager Sub-Component Diagram*

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 basic 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 5000 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 when a new safety report is 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 data warehouse.

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

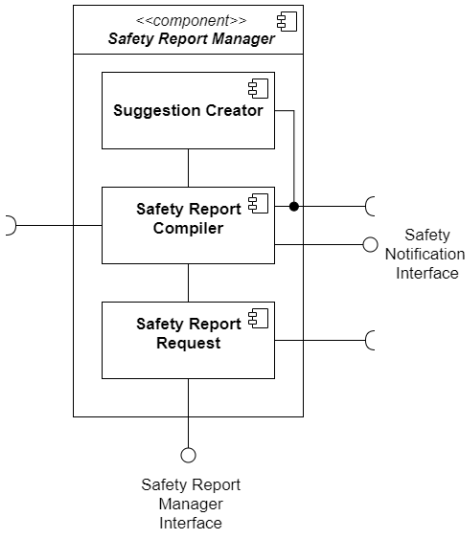


Figure 4: *Safety Report Manager Sub-Component Diagram*

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.

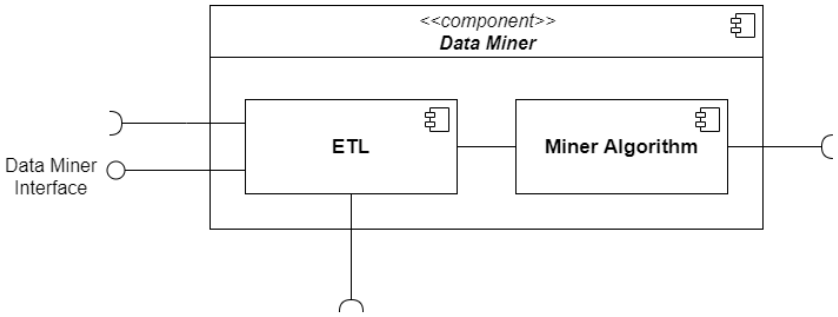


Figure 5: *Data Miner Sub-Component Diagram*

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 data warehouse 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 for reading, storing and deleting data 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 data warehouse 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.

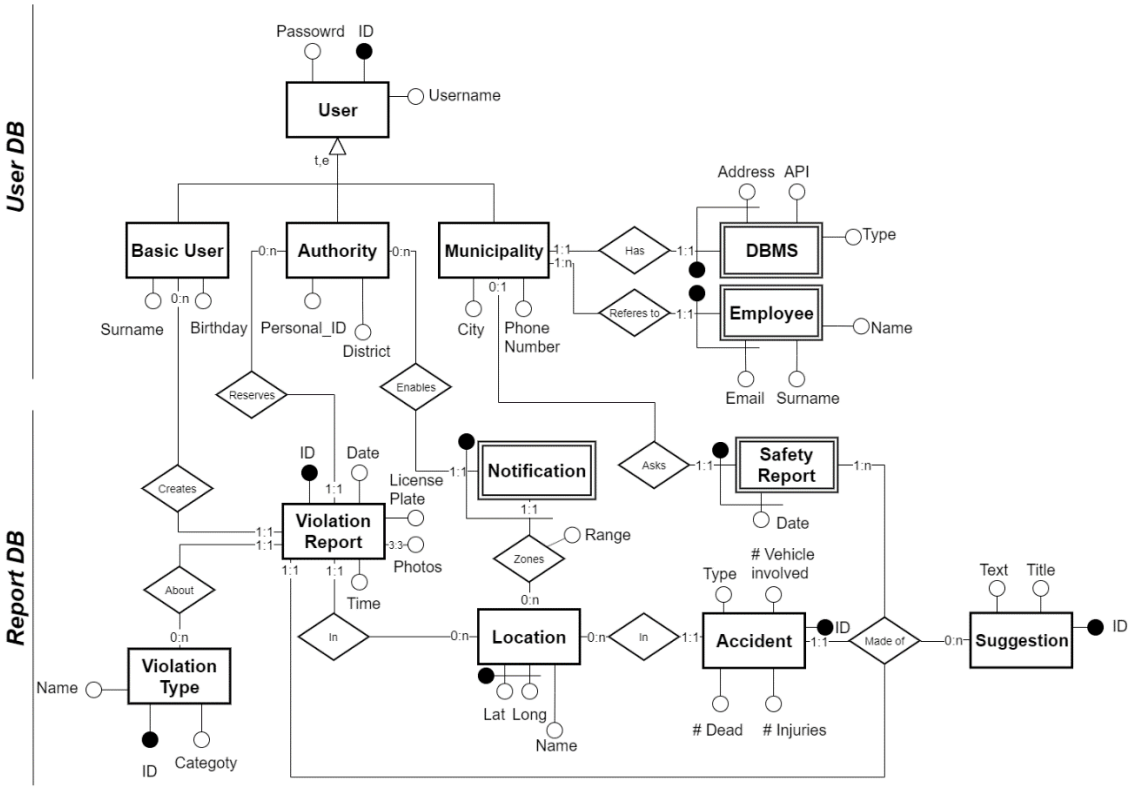


Figure 6*: Entity-Relationship Diagram*

The dimensional fact models (DFM) 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.

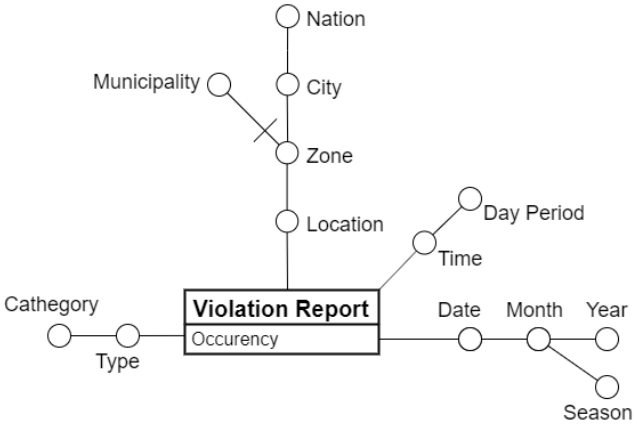


Figure 7: *Violation Reports DFM*

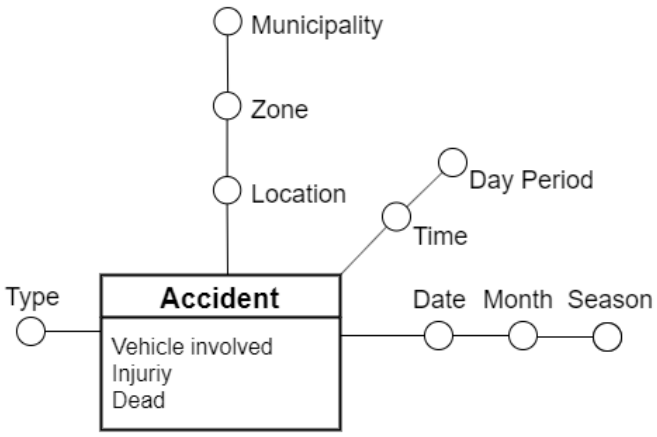


Figure 8: *Municipality Accidents DFM*

2.3 Deployment View

We decided to deploy our system exploiting the services provided by Microsoft Azure (*Figure 9*). 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.

Scalability and Flexibility: The flexible structure of Azure allows to scale up dynamically according to the computational power needs and let *SafeStreets* to scale out its business proportionally in the future.

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.

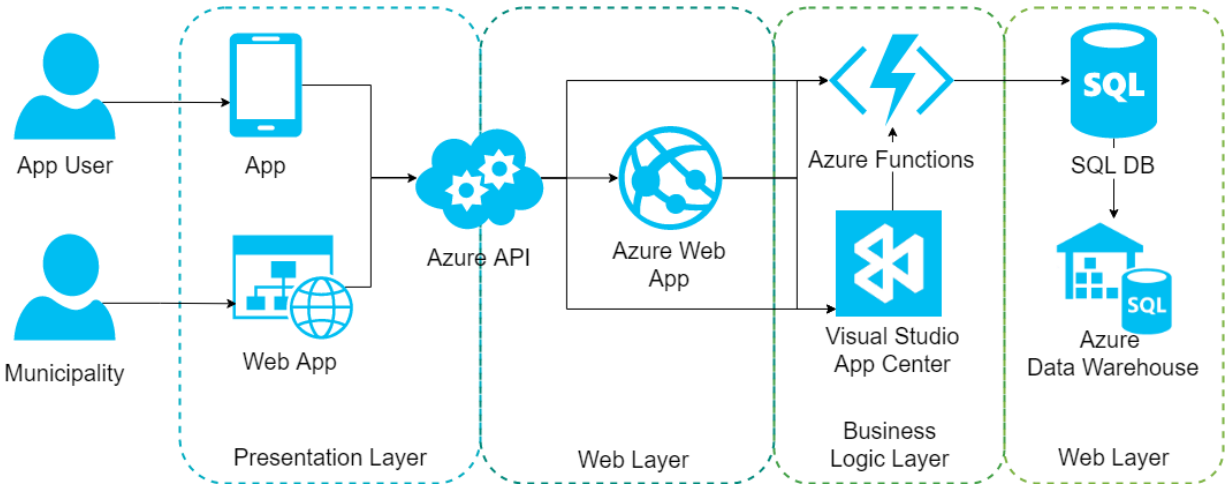


Figure 9: *Architectural layers on Azure services*

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 in .NET, Java, Node.js and many other coding languages. It's where Web Service component will be hosted.
Visual Studio App Center	Service able to provide both front-end and back-end authentication capabilities for application. Exploit all the functionality of Account Manager component.
Azure Functions	Platform that allows to accelerate and simplify application development; connection to other services without hard-coding integrations. Application Server component will be deployed using Azure Functions services.
Microsoft Azure SQL	Microsoft Azure relational database service where resiliency, scale, and maintenance are primarily handled by the platform.
Azure SQL Data Warehouse	Microsoft Azure Data Warehouse service, built on-premise in the Azure SQL service.
Azure API Management	Improves the development and management of *SafeStreets* API.

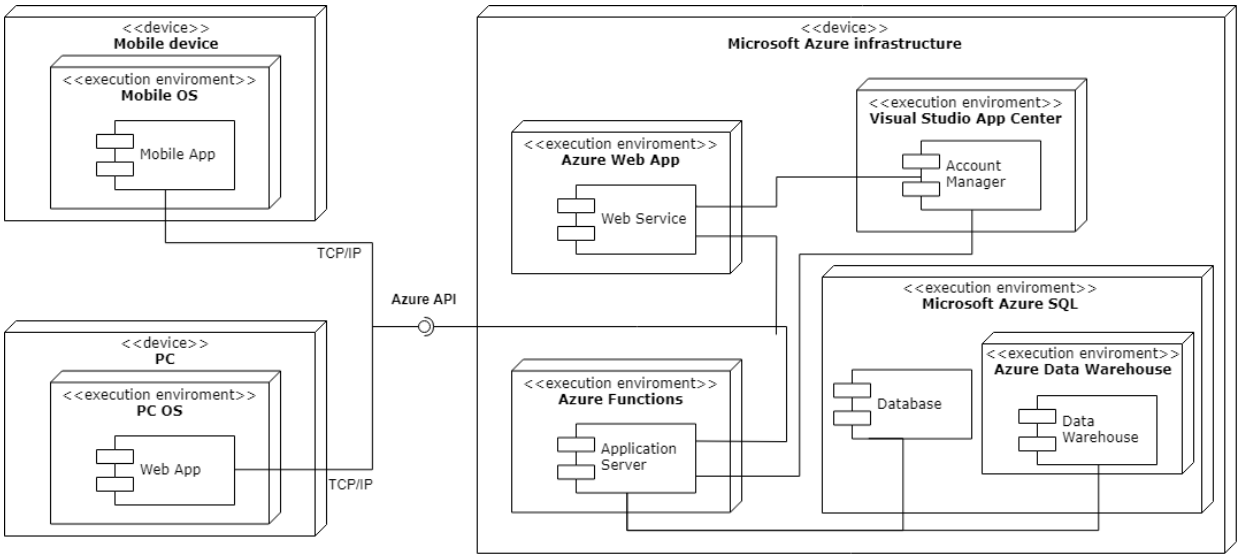


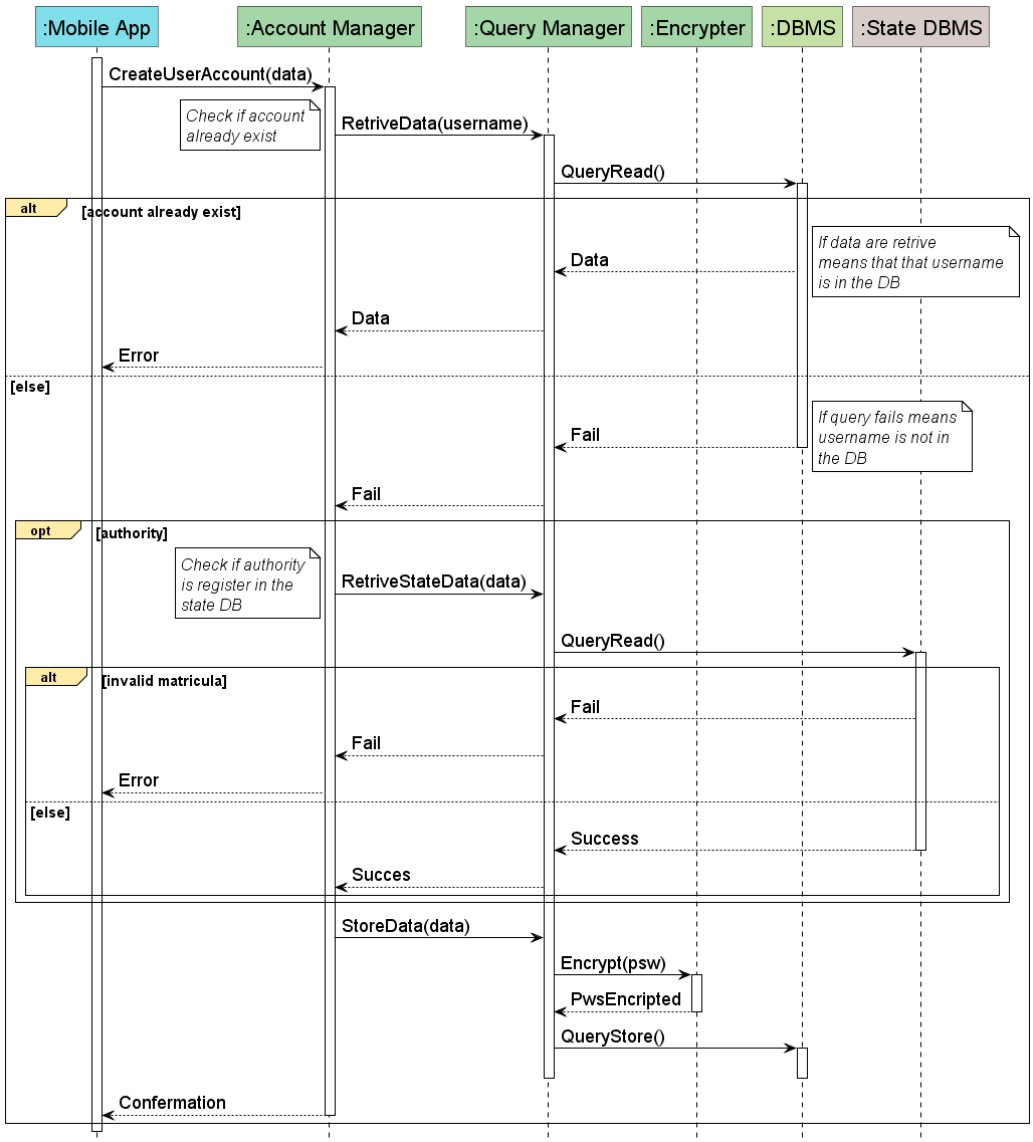
Figure 10: *Deployment Diagram*

2.4 Runtime View

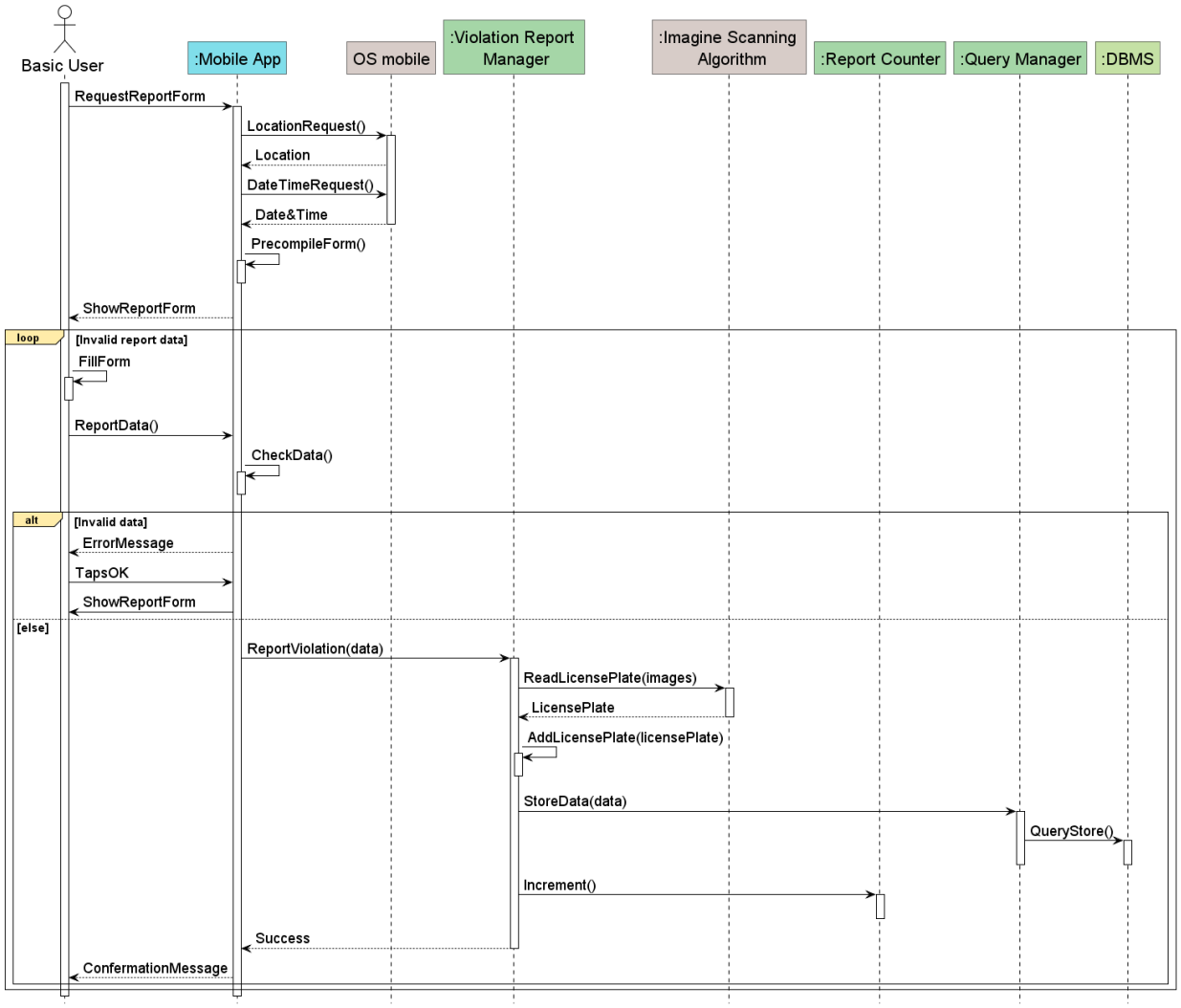
In this section we will present some sequence diagrams that show the major interaction processes between the system components. All the methods performed between components are described in Section 2.5 and 2.6.

We decided to use a color coding to better highlight each tier role: cyan for application layer, teal for web layer, green for application layer, light green for data layer and brown for external service. In the following sequence diagram all operation, unless otherwise specified, have been considered successful.

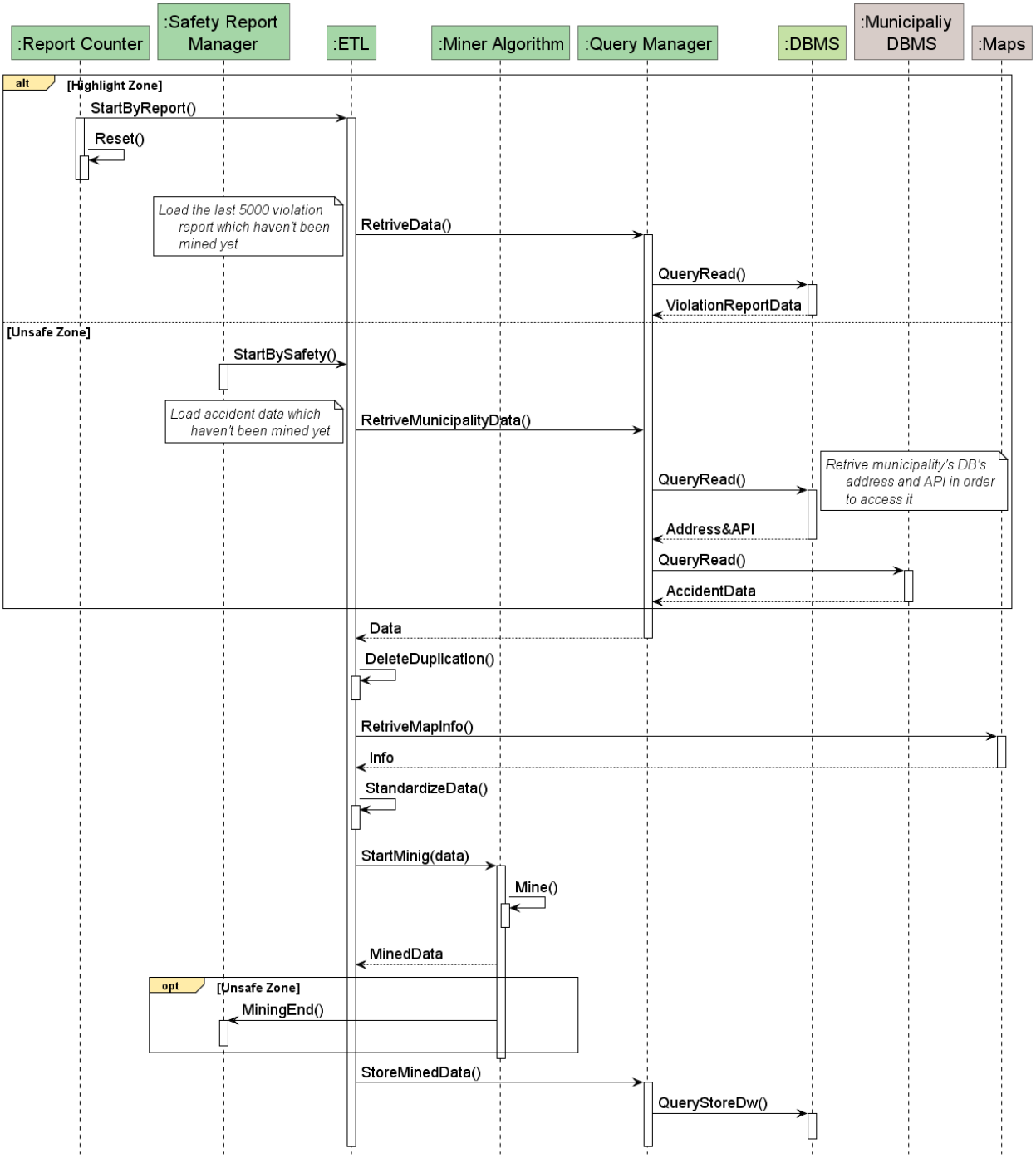
2.4.1 App user registration



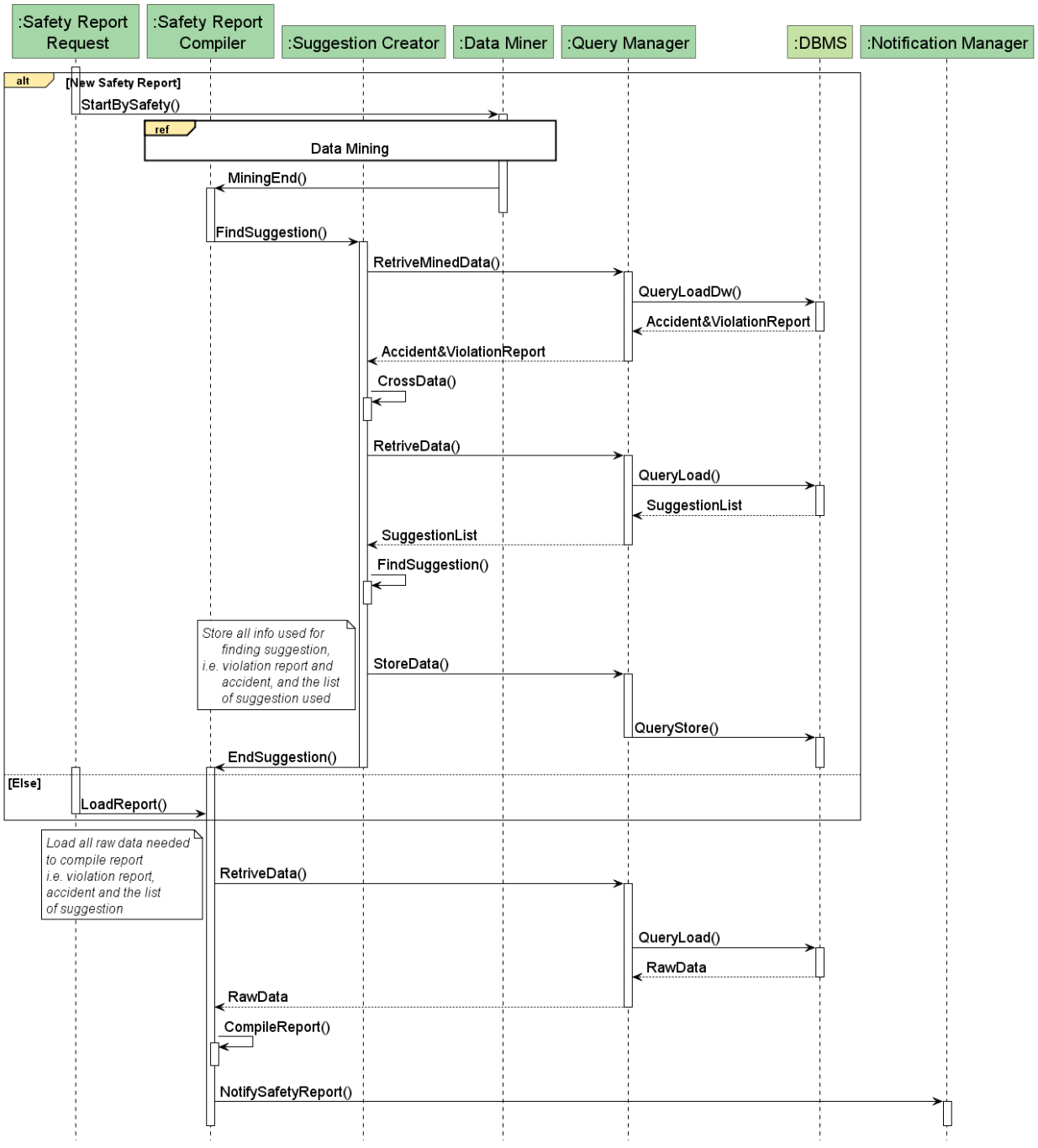
2.4.2 Violation reporting



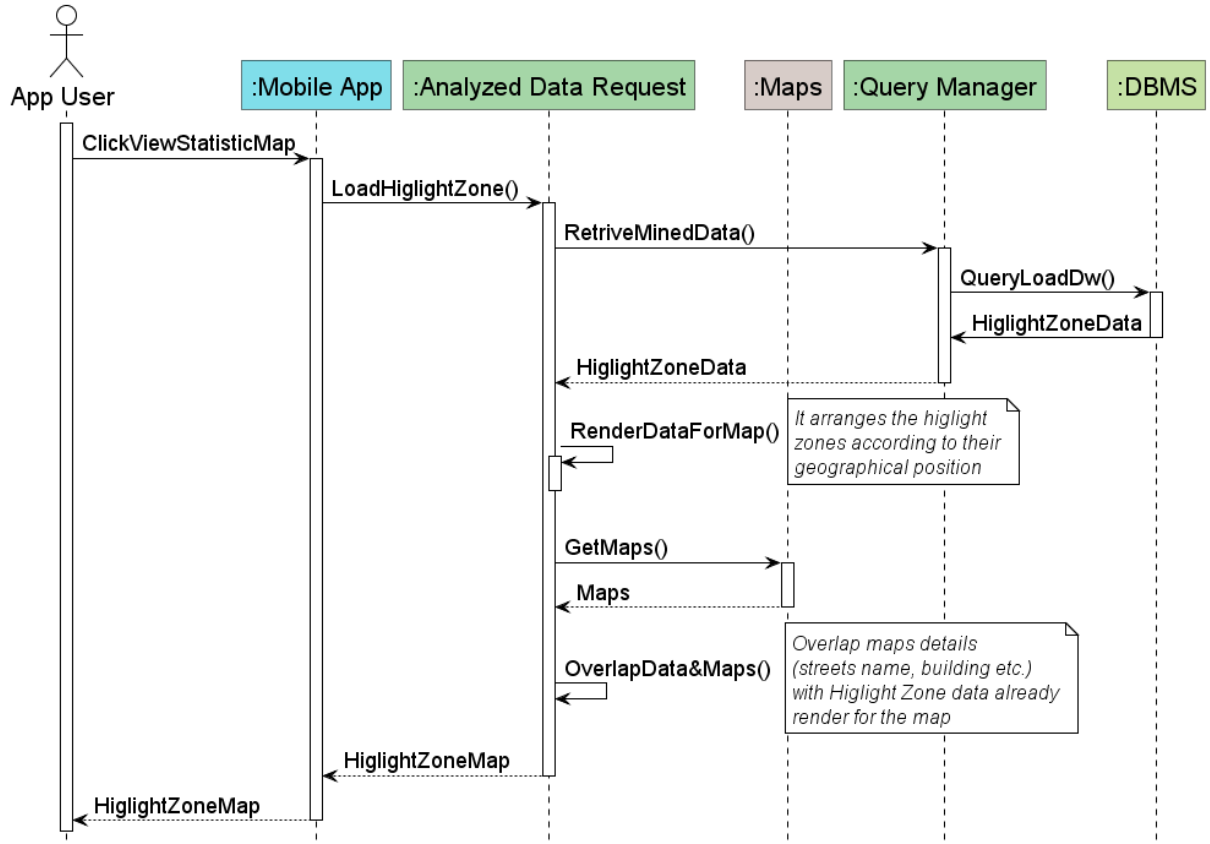
2.4.3 Mining data

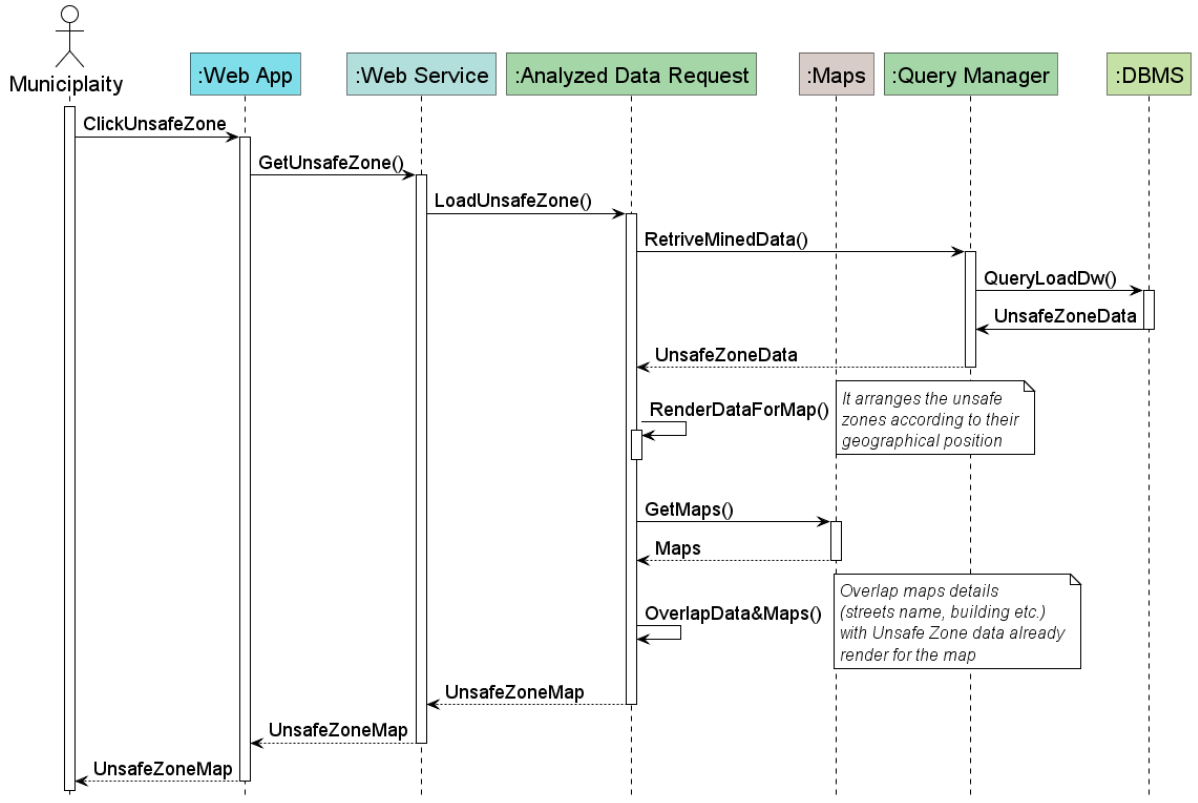


2.4.4 Safety report compiling



2.4.5 Analyzed data rendering





2.5 Application Server Interfaces

The methods offered by the interfaces of each *Application Server* component (Section 2.2) will be descried in the following section.

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.

For each method, an error code shall return, or an exception shall be thrown in case of unsuccess.

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 method will be unsuccess if an account with the same email addressalready 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. If an account with same email address already exist or if the personal ID provided is not valid (not found in *State DB*) or if the Personal ID is already linked to another account, the method will be unsuccess.
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.
Login	By providing email/identification code and password, a user can login into his/her account and exploit the system functionalities.
DeleteAccount	Delete permanently a user account providing email/identification code and password.

EditInfo	Updates user's profile information with the new set of information passed as parameter.

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).
LoadNotifiedReport	Return to an authority a list of all violation reports for the last 24h located in zones where he/she has enabled the notification.
LoadSingleReport	Return all the details of the report passed as parameter (basic user who report the violation is omitted).
ReserveReport	Reserve a report by an authority, when return the report shall not be reservable by anyone else.
CancelReservationReport	Cancel a reservation of a report by an authority, when return the report shall be reservable again.

2.5.3 Analyzed Data Request Interface

LoadHighledZone	Return all mined violation report data of the zone passed by parameter.
LoadHighledZoneByType	Return mined violation report data of the zone and type passed by parameters.
LoadUnsafeZone	Return all mined accidents data of the zone passed by parameter.

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.
DeleteNotification	Delete a zone from the area where an authority wants to be notified when a violation report is uploaded.

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.
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.

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.

2.5.7 Safety Notification Interface

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

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.
RetriveMinedData	Make a query in order to read some data from the data warehouse. Data to be queried are passed as parameter.
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.
RetriveMunicipalityData	Make a query in order to read some data from municipality relational database. Database and data to be queried are passed as parameter and only read query are allowed.
StoreData	Make a query in order to store some data in the relational database. Data to be stored are passed as parameter.
StoreMinedData	Make a query in order to store some mined data in the data warehouse. Data to be stored are passed as parameter.
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.

2.5.10 Encrypter Interface

Encrypt	Return the value passed by parameter encrypted.
Decrypt	Return the value by parameter decrypted.

2.5.11 Data Miner Interface

StartByReport	Start the process to mine violation report data retrieved from the DB.
StartBySafety	Start the process to mine accident data retrieved from municipality DB. The municipality where data are retrieved shall be passed as parameter.

2.6 External Interfaces

For external interfaces only the required methods are generally provided, names are omitted because they are strictly related to the partner providing the API and may change during the system lifecycle.

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

2.7.1 Architecture Style

RESTful API

Both web client and mobile application shall use RESTful API in order to communicate with the server. REST is designed to build systems that are lightweight, maintainable, and scalable: every HTTPS request encapsulated all the info needed for its execution and the server doesn't have to maintain any session information allowing different requests to be handled from different server nodes.

Moreover, Azure's architecture works on the REST principle, so developing the S2B using this architecture will exploit at its best Azure's platform.

Four tiers client-server

This modular structure increase maintainability, flexibility and scalability allowing *SafeStreets* to improve and upgrade each tier independently. The developing speed also increase, letting different team code different tier parallelly. As explained in Section 2.1 the four tiers are: application, web, business logic and data.

Thin client

The presentation layer has the only role of showing data to the user and performing requests; those requests, along with data and logic are managed by data layer, business layer and web layer. As already said, the only logic incorporated in client side is checking of basilar incorrect data (incomplete forms or invalid date).

2.7.2 Design Patterns

Adapter

In order to be able to interface with different API provided by multiple DBMS (i.e. *State DBMS, Municipality DBMS*), we shall adopt adapter pattern, mainly in the *Query Manager* component.

Façade

In our system there are some complex components made of different subcomponents; in order to improve the readability and usability of interfaces provided by those components we shall use the façade pattern hiding all the complexity of internal deployment.

Observer

In order to develop correctly the notifications manager, we shall use the observer pattern. The S2B should keep and update a list of all authorities that are willing to be notified in a certain area and notify them when a new report that interest that area is submitted.

2.8 Other design choices

Maps

SafeStreets relies on an external component for providing maps information, both on client and on server side. We decided to use Google Maps Platform since it provides an SDK for all major mobile platform and since it allows us to customize maps, simplifying the creation of statistic maps. Moreover, it offers an API for exploiting information about geographic conformation, used in the back end of our S2B.

Image Scanning Algorithm

Image Scanning Algorithm for license plate reading will be provided by Google Cloud Vision through optical character recognition (*OCR*) API. It detects and extracts text from any image and it's one of the best on the market.

Node.js

Node.js shall be use in developing web server component and query manager component. The former, indeed, must let many clients as possible to connect to the web site, while the latter is a bottle neck of Application Server since the majority of component interact frequently with it. A high throughput is required and the asynchronous event-driven paradigm of Node.js is a natural choice for this aim.

HTTPS

For better security, all communication and RESTful API shall be based on HTTPS protocol.

3 User Interface Design

In the RASD 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; 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.

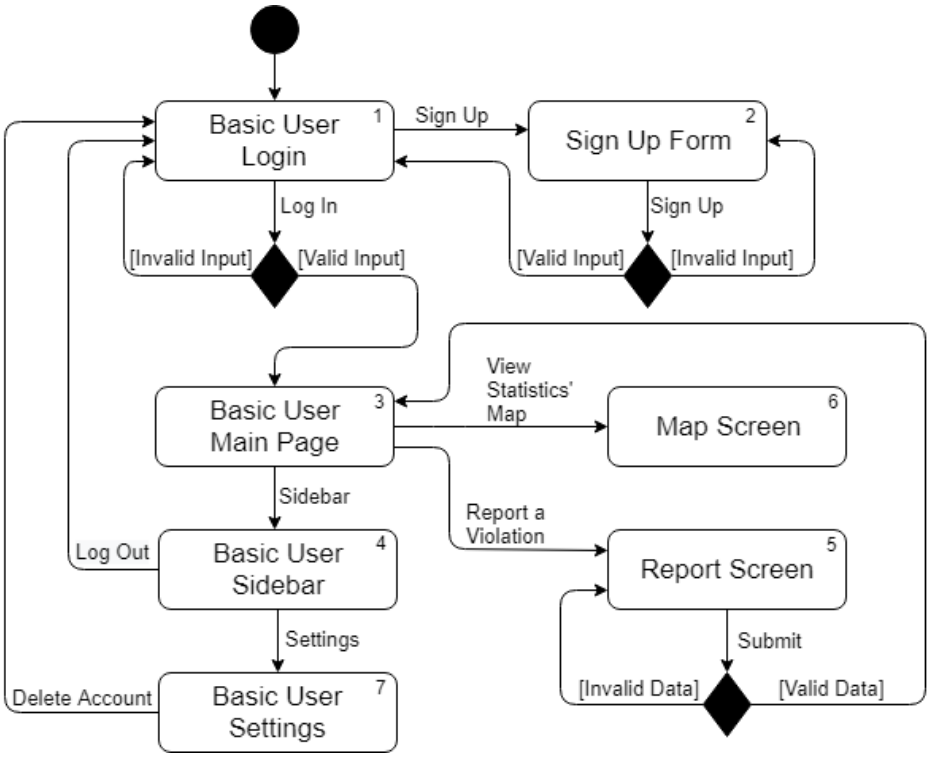


Figure 11: *Basic User's flow of screens*

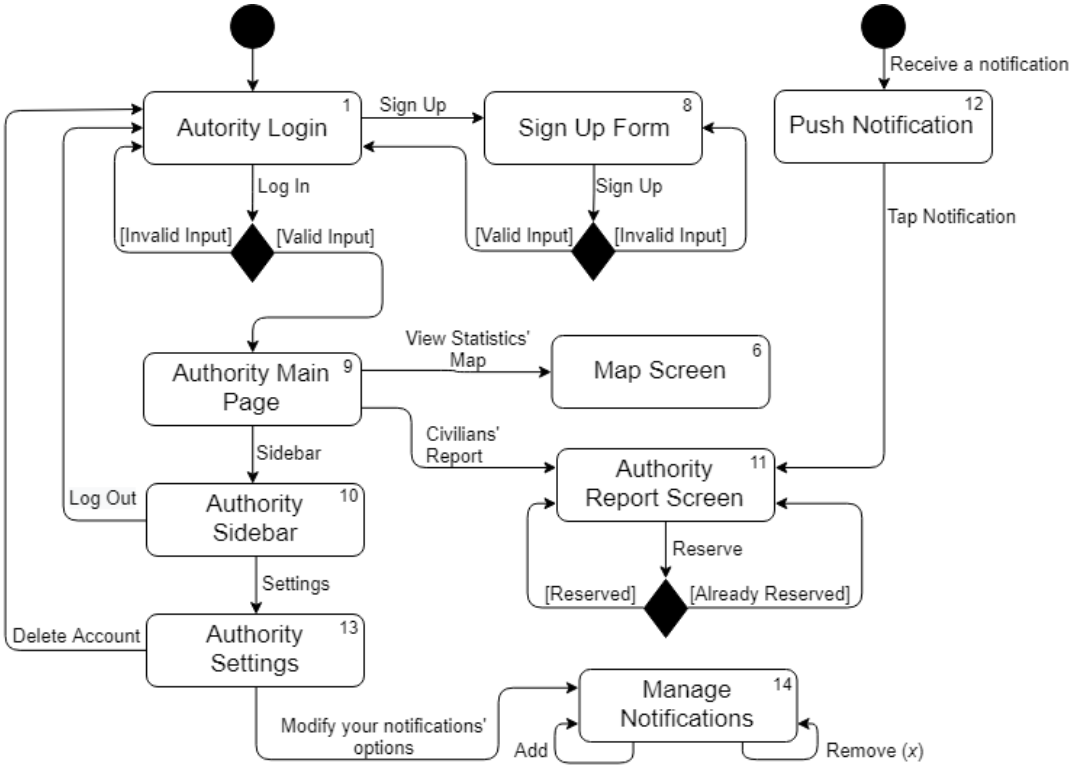


Figure 12: *Authority's flow of screens*

3.2 Flow Graph in Website

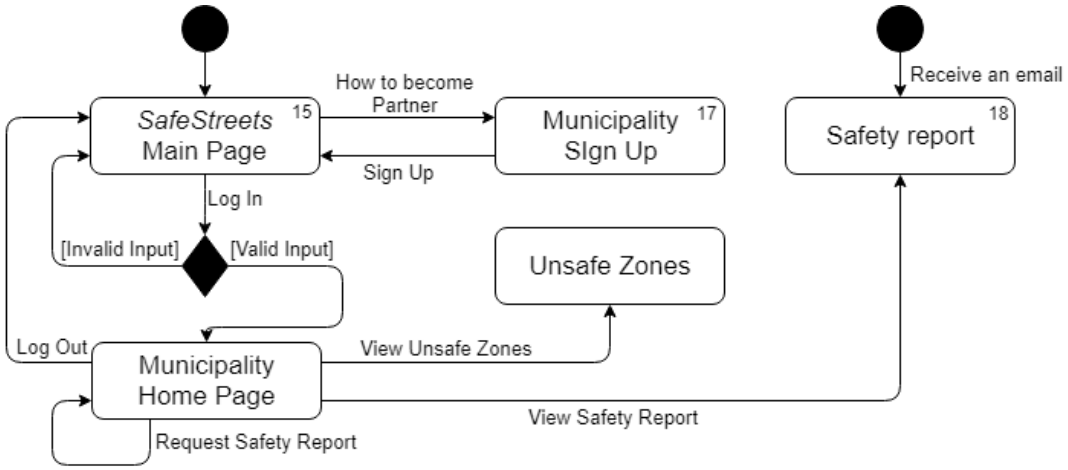


Figure 13: *Municipality's flow of screens*

4 Requirements Traceability

In the following table the requirements described in the RASD (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, Mobile Application
R.2	*App users' distinguishability between basic users and authorities*	Mobile Application, Account Manager
R.3	*Authority validation upon registration*	Account Manager, Query Manager
R.4	*Account uniqueness*	Account Manager
R.5	*Users login*	Account Manager, Mobile Application
R.6	*Users can exploit functionalities only after the log in*	Mobile Application, Web Application, Account Manager
R.7	*Account deletion*	Account Manager, Mobile Application
R.8	*Basic user violation report's upload*	Violation Report Manager, Mobile Application
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	*Authorities shall choose to receive notifications*	Mobile Application
R.13	*The system shall send notifications to authorities when a new report is registered*	Notification Manager

R.14	*Authority access to report details anonymously*	Violation Report Manager, Query Manager, Mobile Application
R.15	*Basic user does not access to single report*	Mobile Application, Query Manager
R.16	*Authority reservation of violations*	Violation Report Manager, Mobile Application
R.17	*New violation classification*	Report Counter, Data Miner
R.18	*Render highlighted zone graphically*	Mobile App, Analyzed Data Request
R.19	*List of all violations for authorities*	Mobile App, Violation Report Manager
R.20	*Management of connection errors*	Web Server, Web App, Mobile App
R.M1	*Municipality's sign up*	Account Manager, Web App
R.M2	*Availability of accidents data from municipalities*	Query Manager
R.M3	*Data integration between SafeStreets and Municipality's DB*	Data Miner
R.M4	*Data mining for generate unsafe zones*	Data Miner
R.M5	*Render unsafe zones graphically*	Web App
R.M6	*Find correlations between accidents and data violations*	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, Web App

Non-functional requirements, instead, are verify by:

Requirement	**Description**	**Design choice**
R.P1	*High throughput*	Azure Infrastructure, Node.js
R.P2	*Fast safety report compiling*	Optimized mine algorithm, Azure SQL
R.P3	*Fast report notifications*	Azure Functions, observer pattern
Maintainability		Azure Infrastructure, façade and adapter pattern
Availability		Azure Infrastructure
Security		Encrypter component, DBMS privileges, HTTPS protocol
Reliability		Azure Infrastructure, Node.js

5 Implementation, integration and test plan

The four tiers of the system can be implemented in parallel and integrated at the end of the development. However, business logic tier is the most critical and difficult to implement and this section is focused on its realization.

For the implementation and integration procedure, we will adopt a bottom-up approach, starting from the components that have no use relation entering in them, as shown in the dependency diagram (*Figure 14*).

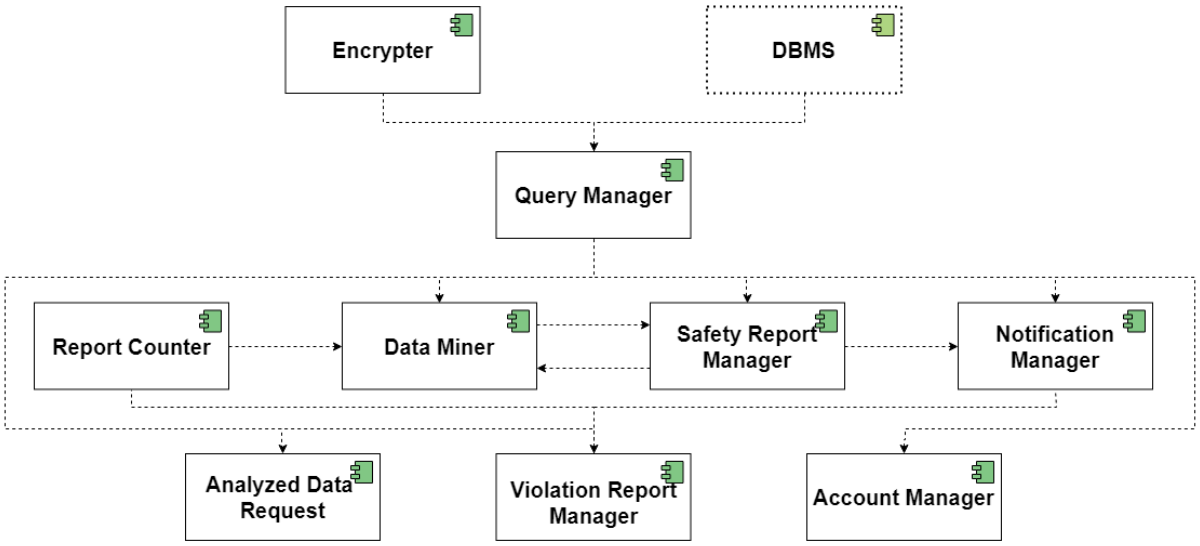


Figure 14: *Dependency Diagram*

From the diagram, it is clear that the systems' components are divided into four levels of dependency. For each level of dependency, we will develop the components using a critical approach, in order to test the most critical components first. Query Manager emerges as a bottle neck of the system, so stressful tests and benchmarks are required for this component in order to test the whole system's reliability.

In the interests of understanding which components should have priority among those who are on the same dependency level, we provide below a table that shows the relationship between each component and their difficulty of implementation:

Component	Difficulty of implementation
DBMS (*data layer*)	*Easy*
Encrypter	*Easy*

Query Manager	*Hard*
Report Counter	*Easy*
Notification Manager	*Easy*
Safety Report Manager	*Hard*
Data Miner	*Hard*
Account Manager	*Easy*
Analyzed Data Request	*Medium*
Violation Report Manager	*Medium*

Each component on the same level and with the same priority can be developed simultaneously. In order to move to the next step, every component must be unit tested and since we used a bottom-up approach, skeletons are required.

Integration shall start as soon as possible in order to discover and fix malfunctions. Every time a dependency level is completed, it shall be integrated and tested with the ones below.

It is important, before release, to test that all the requirements have been implemented correctly, that the system can handle heavy loads from its communication interfaces and that it behaves correctly even when receiving unexpected inputs.

6 Effort Spent

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
4/12	5	4.5	General revision, sections 2.7, 2.8, dependency graph
5/12	1		Section 5 beginning
7/12	1	2	Section 5, Revision
8/12	1.5	1	Final Revision, page layout
TOTAL	28.5	28	

7 References

7.1 Tool used

In this section we will list the tools used to produce this document:

- *Office Microsoft Word* for document writing and building;
- *draw.io* for flows diagram; ER, DFM, components diagrams, dependency diagram, architectural diagram and deployment diagram;
- *PlantUML* (with Visual Studio Code editor) for UML sequence diagrams;

7.2 Document References

- Microsoft Azure's services documentation
  https://azure.microsoft.com/en-US/
- Google Cloud Vison OCR API
  https://cloud.google.com/vision/docs/ocr
- Google Maps Platform API
  https://cloud.google.com/maps-platform/
- PlantUML documentation
  https://plantuml.com/sequence-diagram/
- Node.js documentation
  https://nodejs.org/it/docs/
- RESTful API
  https://restfulapi.net/