SafeStreets DD

Matteo Secco, Mohammad Rahbari

release date: December 9, 2019
version 1

Contents

1 Introduction

1.1 Purpose

The required system, called SafeStreets, is a distributed system to allow the citizens to signal parking violations to the competent autorities.

The system must allow the citizen to submit pictures of the violation, attaching data such as date, time and position. The user will have to specify the type of the violation when sending these data.

When reciving such data, the system must store them, toghether with the plate of the car that performed the violation (elictated from the picture the citizen sent), the informations about the violation itself (in particular the type of violation), and the name of the street where the violation occourred (which can be retrieved by the positioning information the user sent).

In addition, the application must allow both autorities and citizens to analyze the stored data, for example highlighting streets or plates with most violations registered. Different levels of security can be offered.

Finally, the application must be enable authorities to automatically generate traffic tickets using its data for people commiting violations, if and only if it can be verified that the data concerning it has not been altered. In this case, SafeStreets could use this informations to build additional statistics.

1.1.1 Goal list

`G.1` Allow citizens to notify parking violations and be acknowledged of the result as soon as possible

`G.2` Force citizens to provide all the needed data about violation, in particular infraction type, picture, date, time and position

`G.3` Prevent the autorities to have to manually address parking tickets

`G.4` Ensure no tickets can be emitted if the notification's data has been modified somehow

`G.5` Ensure no tickets can be emitted if the plate of the car that committed the infringment owns a permission for that infringiment

`G.6` Every notification not covered by `G.4` or `G.5` will be eligible for ticket generation

`G.7` Allow both citizens and authorities retrive informations about previous violations and released tickets, possibly in an aggregated form. Every violation reported to the system and stored will appear in some statistics

1.2 Scope

The world where the system must fit is an everday city, with focus on the traffic of moterized veichles.
The events the system aims to influence are the parking of motorized veichles, in particular the ones considered infractions.
In the context of the system, when any user notices an illegal parking, he/she may notify the system and provide any needed informations to the competent authorities. In particular, the notification is composed by a picture of the infraction, a timestamp (date and time), the geographical location of the infraction and the type of infraction wich is to be notified. Some of these informations can be gathered automatically from the user's device. Notice that, for legal reasons, SafeStreets will just make the data available to the auth, and will not generate tickets itself.
In addiction, the user may interrogate the system to gather aggregated informations about the locations with more violation incidence, and the cars which committed more violations.

Table 1: Phenomena

Phenomenon	Shared	Who controls it
A citizen parks its car	N	W
A citizen spots a car in an illegal parking	N	W
A citizen wants to notify a violation	Y	W
The user fills the data needed to notify a violation	Y	W
The machine revices a notification, analyzes it and stored stores it if it's trusted	N	M
A user requests map statistics	Y	W
A citizen requests top violators statistics	Y	W
The machine analyzes data and builds statistics	N	M
Statistics are organized and presented to the user	Y	M
An authority wants to generate tickets	N	W
An authority requests some violation notified and stored in the machine	Y	W
The machine provides some notification to an authority requesting them	Y	M
The authority decides whether generate a ticket for a violation or not	Y	W
The authority generates a ticket for a violation	N	W
The authority informs the machine she has generated a ticket for a given violation	Y	W
An authority requests informative statistics about its competence area	Y	W

1.3 Definitions, Acronyms,Abbreviations

Person: A person in the real world. Every Citizen is a person, generally an Authorithy is not

User: A person, an organization or a system which somehow uses SafeStreets

Citizen (cit): This term will be used to denote every user not owning particular privileges or permissions. A citizen is only allowed to notify violations and see some aggregated data

Authority (auth): This term will denote every user (phisical or digital) having privileged access to the stored data. An example of Authority is the Local Police.

Notification: Represents a set of data submitted by any user composed by:

- A picture of a parking infraction
- A timestamp of when the notification occourred, containing date and time (may be gathered automatically by the citizen's device)
- A geographical position of where the infraction occurred (may be gathered automatically by the citizen's device)
- The type of infraction notified

Car: The word car will be used to issue every mothorized vehicle

Plate: Identifies a car

Permisison: A document released by a verified authority, granting to a car the permission to park in a set of reserved parkings (ex: permission to park on parking reserved for disabled people).

1.4 Revision history

1.5 Reference documents

1.6 Document Structure

2 Architectural Design

2.1 Overview

High-level components and their interaction In this section the architectural design of SafeStreets is presented. The presentation is divided into the following parts:

- **High level architecture:** This section will provide an high level view of the system architecture.
- **Data model:** This part will describe the Database structure.
- **Component view:** This part will introduce all the components the system will be made from, their functionalities and their interactions.
- **Deployment view:** This part describes the hardware that will compose the system as well as the software that will be runned on it.
- **Runtime view:** This parts describes in detail the executon behaviour of each component.
- **Component interfaces:** This pat describes the interface that the components will use to communicate one with the other, as well as the interfaces with other systems.
- **Selected architectural styles and patterns:** This parts describes the design choices about the architecture of the application.

2.2 High level architecture

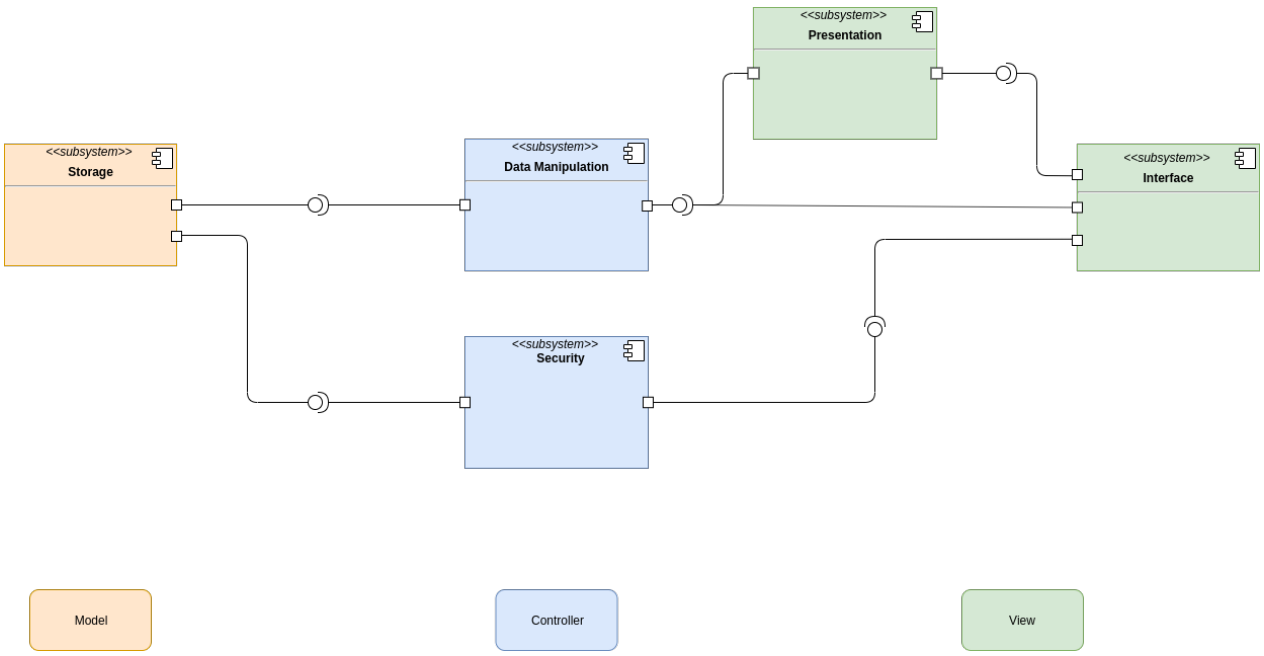


Figure 1: High level architecture

As displayed in fig. 1, the system embraces the MVC architectural pattern (see section 2.8) and can be divided into 5 subsystems:

- **Storage:** The Storage subsystem is responsible of encoding and persist the data needed by the system. As the system even embraces the REST pattern, this is the only subsystem that stores data. This subsystem is only composed by the Database Management System. This subsystem coincides with the Model subsystem.

- **Data manipulation:** This is the main Controller subsystem. Its components are responsible to handle the User's requests, recive data (and forward them to the Storage), and in general to cover the functional sides of the system.
- **Security:** The Security subsystem is an auxiliary controller. It will grant Authentication functionalities to the system, but plays no role in fullfilling funcitonal requirements.
- **Presentation:** This subsystem is responsible of displaying the outputs of the Data Manipulation subsystem in a way appreciable by an average user. It takes care of the presentation for both authorities (exploiting HTML pages) and citizens (exploiting the GUI of the mobile application).
- **Interface:** This subsystem manages the communication between the server(s) and the clients. It is responsible for handling HTTP requests for both webpages and raw resources (used later by the mobile application), and of sending emails as well. In a typical working scenario, the previous subsystems are activated by this one, which forwards user requests.

2.3 Data model

The system's data will be stored in a cloud-based relational database. The Entity-Relationship diagram in fig. 2 explains which data the system will store and manage. In detail:Citizens can have multiple Cars, Tickets and sent Notifications but all of these must be identified by only one citizen. All of Officers can access to all of the Tickets that issued by themselves to specific Citizen also each Officer belongs to one Authority organization releated to their job and their location for having different access(for instance officers in Milan have distinct location and radius than officer in Como). Moreover Officers are linked to Notifications and its reachable for them to observe all of them.

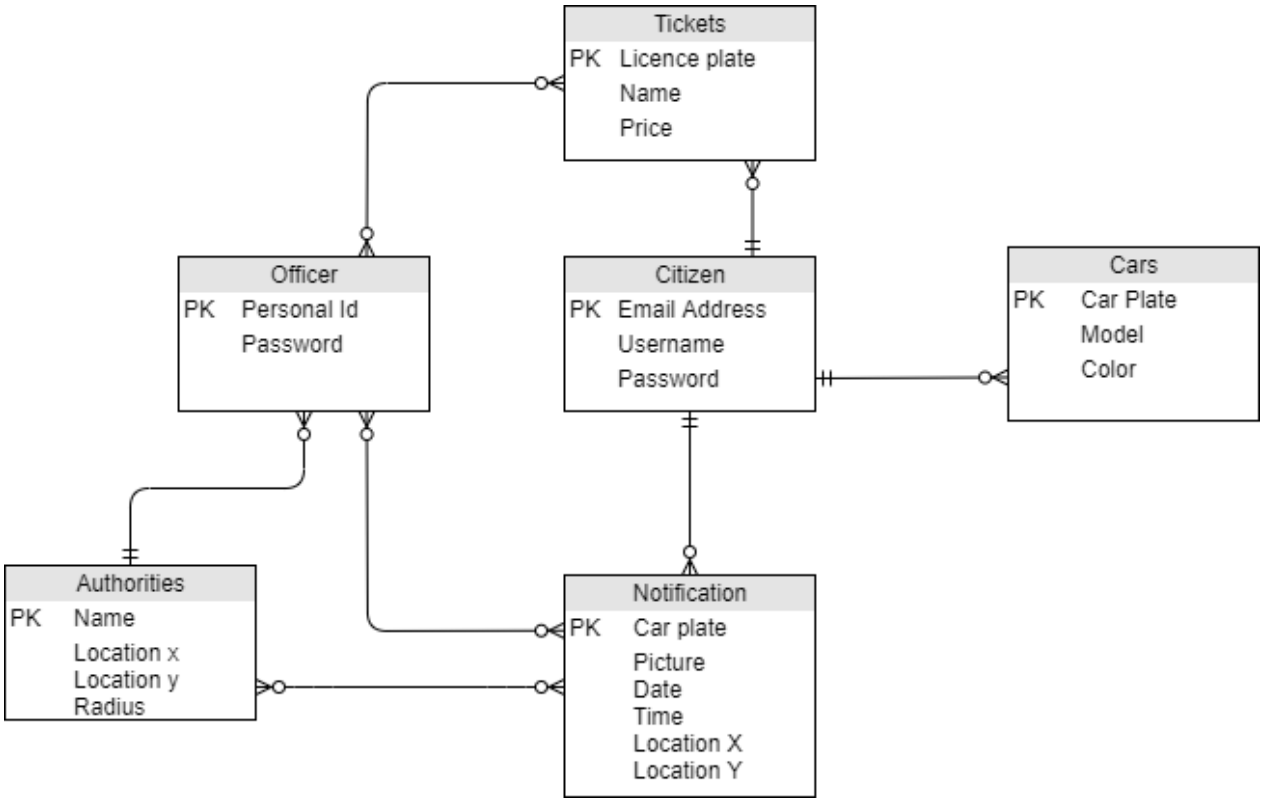


Figure 2: Data model diagram

2.4 Component view

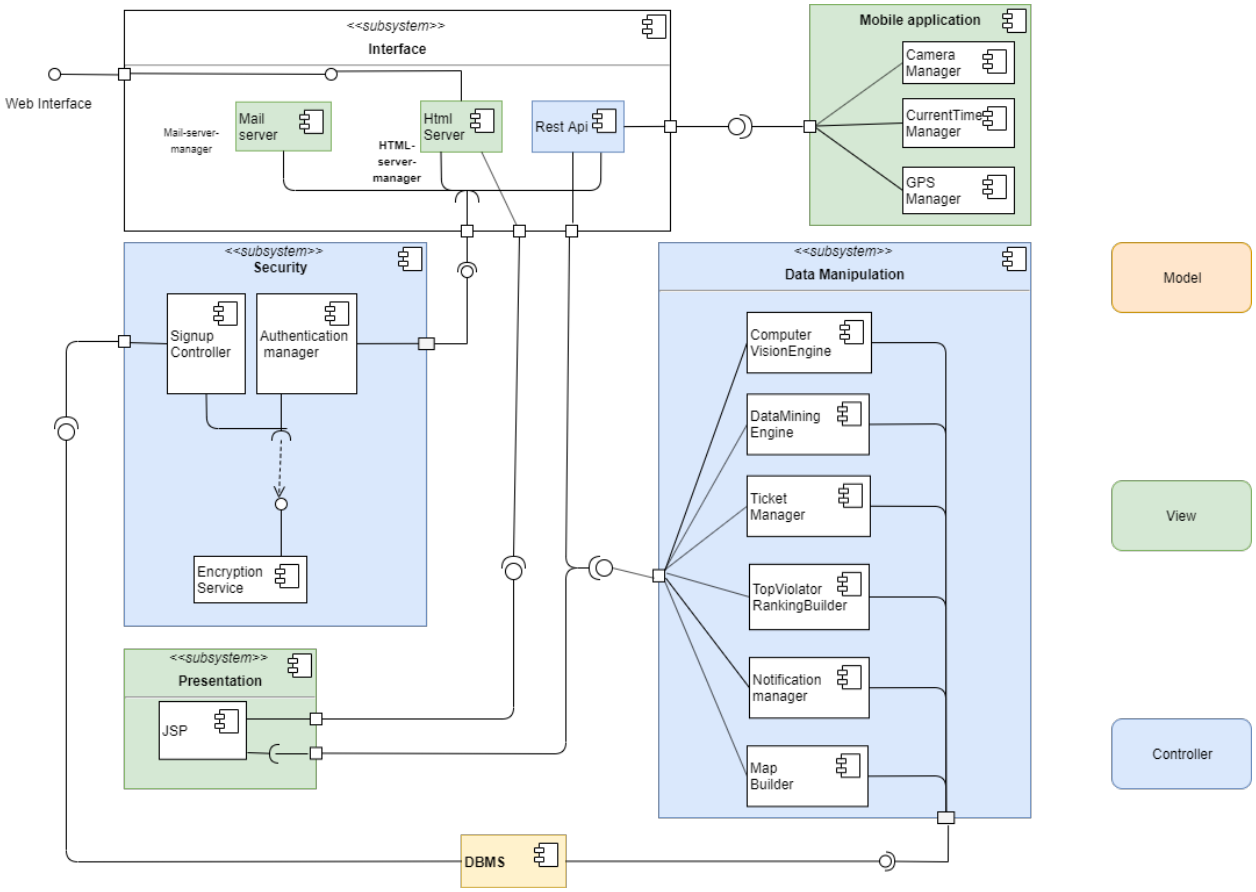


Figure 3: Component Diagram (Some client components are omitted for brevity)

In fig. 3 the components of the system are presented with their connections, and inserted in the propter subsystem. In table 2 summarizes the belonging of each component to its subsystem.

Table 2: Components

	Security	Interfaces	Presentation	Data manipulation	Storage
Server	AuthenticationManager SignupController EncryptionService	HtmlServer MailServer RestApi	NotificationManager MapBuilder TopViolatorsRankingBuilder StatisticsPresenter TicketPresenter JSP	ComputerVisionEngine DataMiningEngine TicketManager	DBMS
Citizen	EncryptionService	RestApi Mobile Application	MapDisplayer TopViolatorsDisplayer NotificationService	CameraManager GPSManager CurrentTimeManager	
Authority		WebBrowser	WebBrowser		

The following list provides a detailed description of each module's functionalities (calls to components are omitted here as they will be datailed in sequence diagrams):

COMP.1 **AuthenticationManager:** This module will authenticate the user comparing the provided credentials with the ones stored to the database, and if a match is found will generate a session token for authentication. The provided password will be hashed before comparison (as passwords won't be stored in plain text). The usage of a token (nounce) can prevent many malicious attacks, such as Csrf attacks or Replay attacks.

COMP.2 **SignupController:** This component will manage the signup procedure, in particular it's main functionalities will be: reciving a signup request, check the constraints of the provided data (in example, unicity of username and email, strength of the password), hash the password, temporary store these data, generate and send a token to the given email to verify it, persistently store the data. The email validation can provide some protection against DoS attacks, but full protection is currently unachievable (many researches are running to find a solution)

COMP.3 **EncryptionService:** BCrypt

COMP.4 **HtmlServer:** This component is an interface to handle http requests for the HTML pages to be displayed on a web browser.

COMP.5 **MailServer:** This component is an interface for mail sending emails to the users, in particular notifications (i.e: a ticket has been issued to them) and the first email used to confirm the email address.

COMP.6 **RestApi:** This component will handle the custom messaging protocol with Citizen's devices. The communication will rely on HTTPS, in order to protect the system from Man in the Middle attacks.

COMP.7 **JSP:** This component, relying on a well-known technology, will dynamically create HTML data starting from given templates and data.

COMP.8 **NotificationManager:** This component will decide when is needed to send a notification to an User, by sending an email in case of Authorities, or by displaying a notification on the application and/or sending an email in case of Citizens, according to the preferences they setted.

COMP.9 **MapBuilder:** This component is responsible to aggregate data from the database according to a geographical location, and to create the data needed to display them using the Google Maps API.

COMP.10 **TopViolatorsRankingBuilder:** This component will query the database for Violations based on a geographical location, hide the plate from pictures and generate the list for the Top Violators page.

COMP.11 **ComponentVisionEngine:** The main functionality of this component is to recive a picture and extract the car plate out of it. It will also be used to hide the plates from the TopViolatorsRanking.

COMP.12 **DataMiningEngine:** This component will run data mining algorithms on the database to discover relevant informations with regard to specific geographical locations.

COMP.13 **Mobile Application:** The application running on the Citizen's mobile device.

COMP.14 **TicketManager:** This component will filter incoming notifications in order to avoid to store non-ticketable violations, and provide stored violations that are issuable on demand.

COMP.15 **CameraManager:** This component will grant secure access to the camera of a Citizen's device, preventing it from notifying violations if the camera is not available (for hardware fault or denied permission).

COMP.16 **GPSManager:** This component will regulate the access to the GPS of the citizen device, as well as the gathering of the position.

COMP.17 **CurrentTimeManager:** This component will check if the date and time settings on the user device are set to auto (the device synchronizes it's clock using internet or the GPS). If they are, the current time is collected, otherwise the notification is blocked until the settings are changed

COMP.18 **DBMS:** This component will be respopnsible of managing the database.

2.5 Deployment view

Designing the system, our main focuses were costs (both initial and maintenance) and complexity (of the system architecture and of the development). For this reason, we decided to put as many components on the cloud, relying on well-known and trusted providers (Amazon and Google). The final hardware system will so be composed by:

- Android devices: used by Citizens to interface with the system.
- iOs devices: used by Citizens to interface with the system. We needed to distinguish between the two kind of devices because of the possible differences in the development of the Mobile Application for different operative systems.
- Personal Computers: used by Authorities to interface with the system.
- Web Server: a replicated device used to hande HTTP requests.
- HTML Server: a replicated device used to generate dynamic HTML pages. We are planning to merge this first two servers in a following version of this document.
- Data Analysis Server: this replicated device will handle the heavy load of the computation, covering a big portion of the Data Manipulation subsystem.

In fig. 4 summarizes the datails of the deployment, highlighting the hardware, the operative systems, the running software and the resources available in the different hardwares.

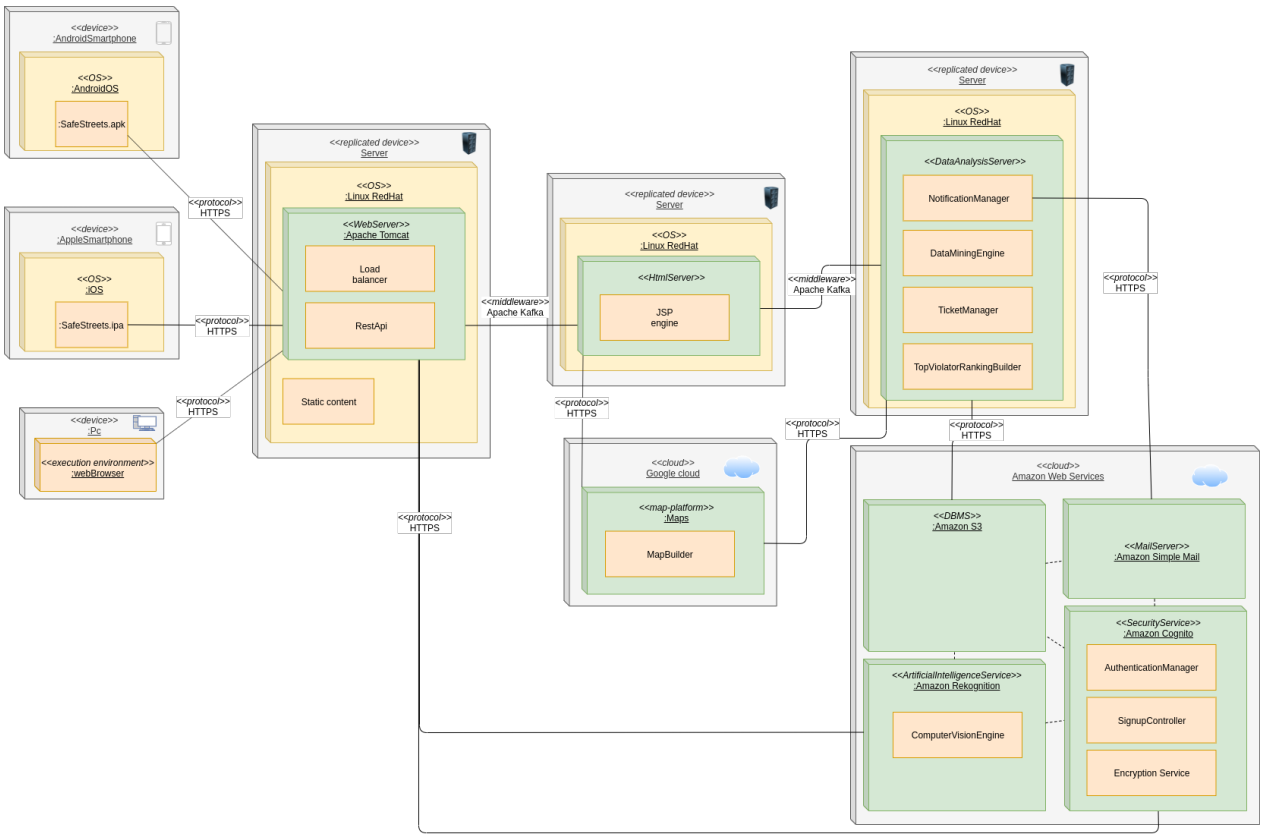


Figure 4: Deployment diagram

2.6 Runtime view

The following diagrams show the interactions between components in order to achieve the required functions

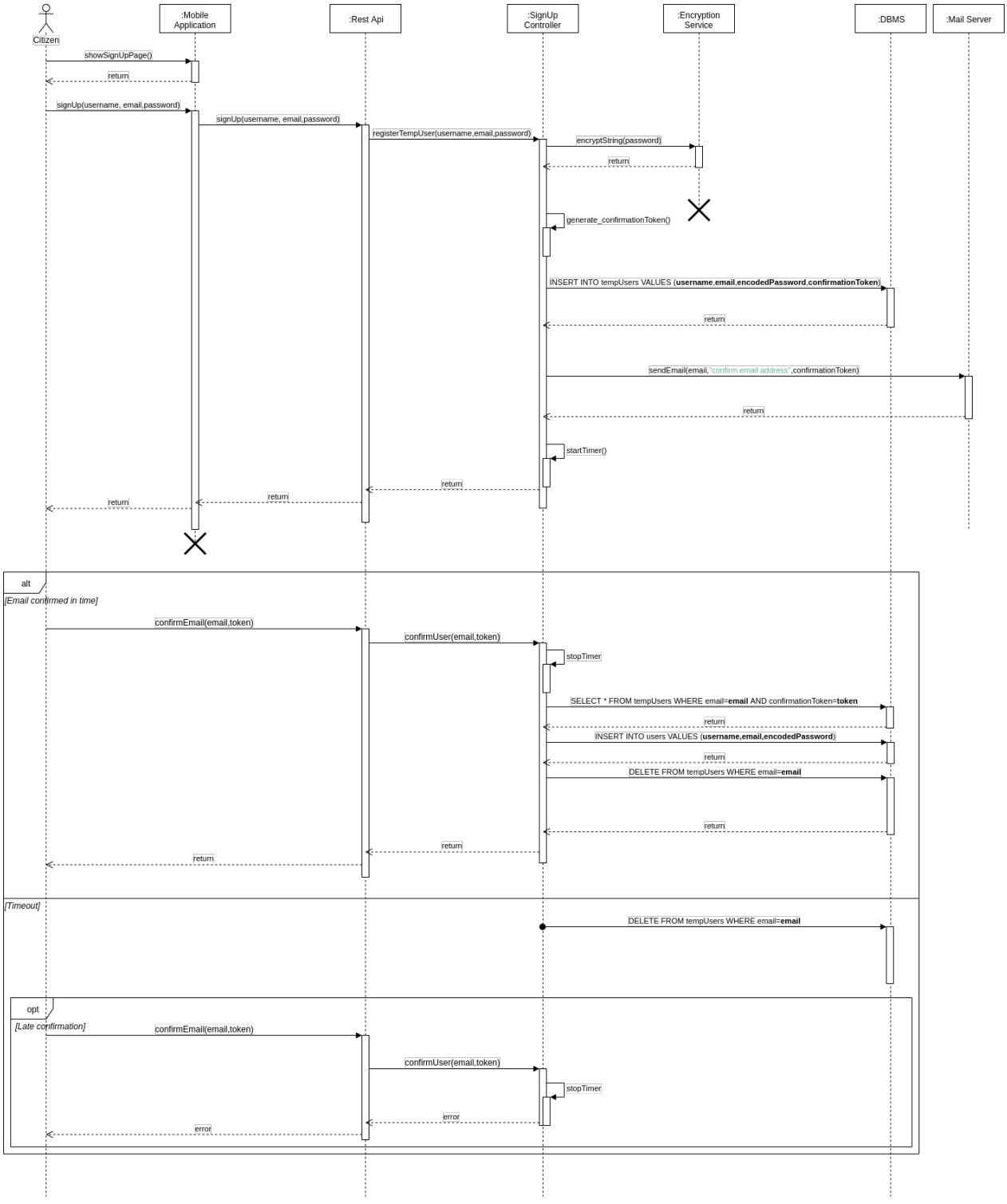


Figure 5: Register citizen sequence diagram

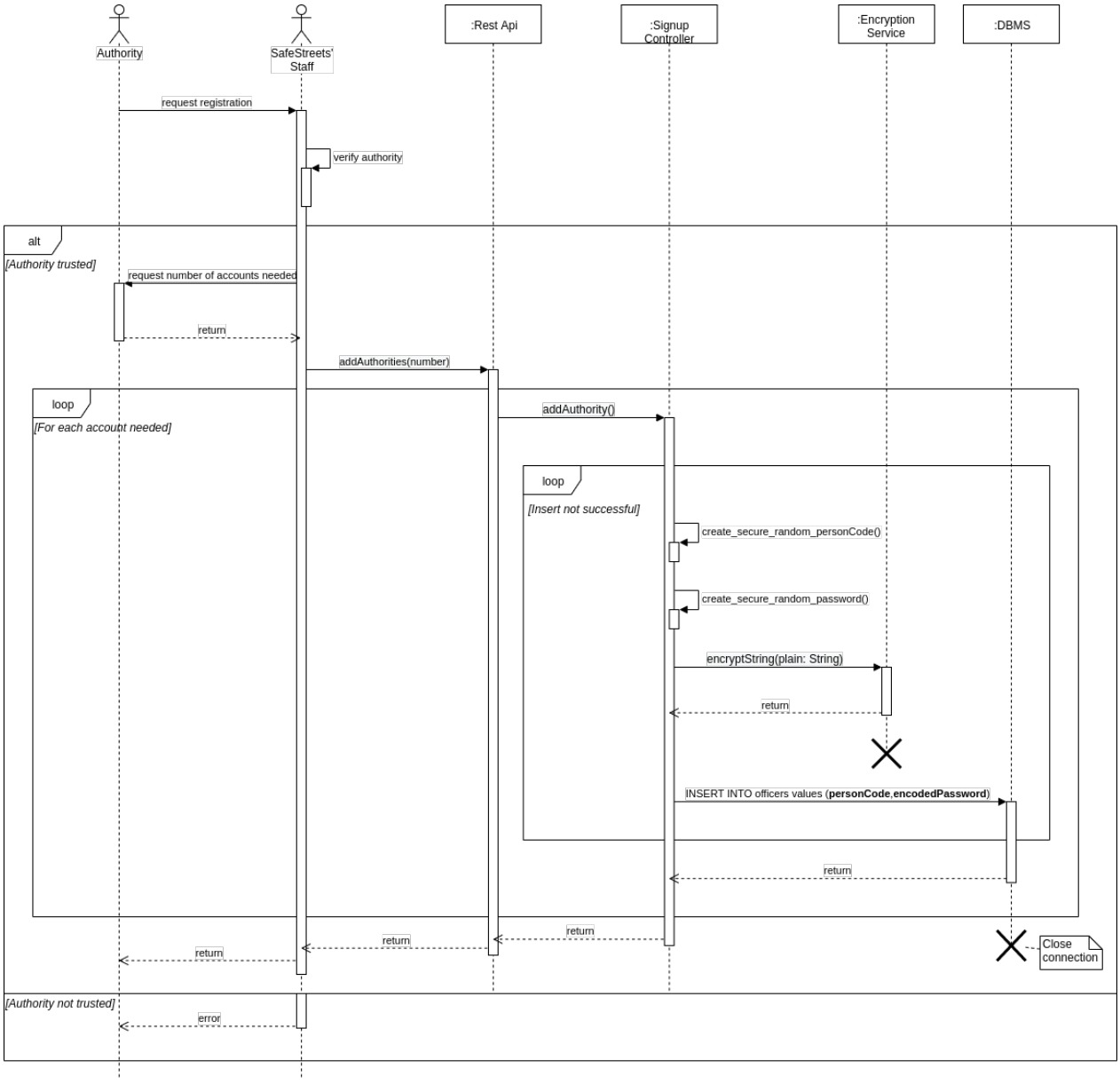


Figure 6: Register authority sequence diagram

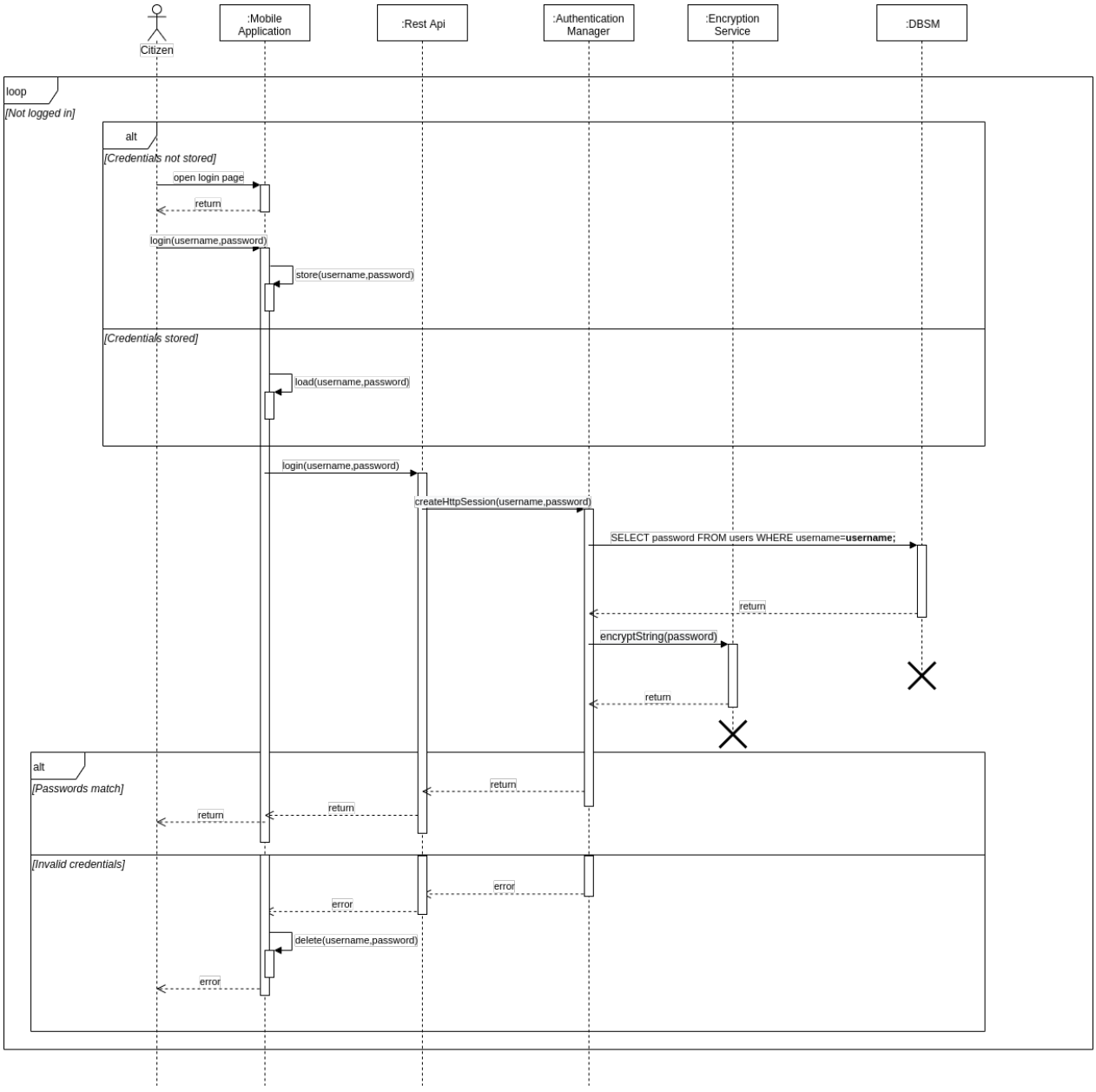


Figure 7: Login citizen sequence diagram

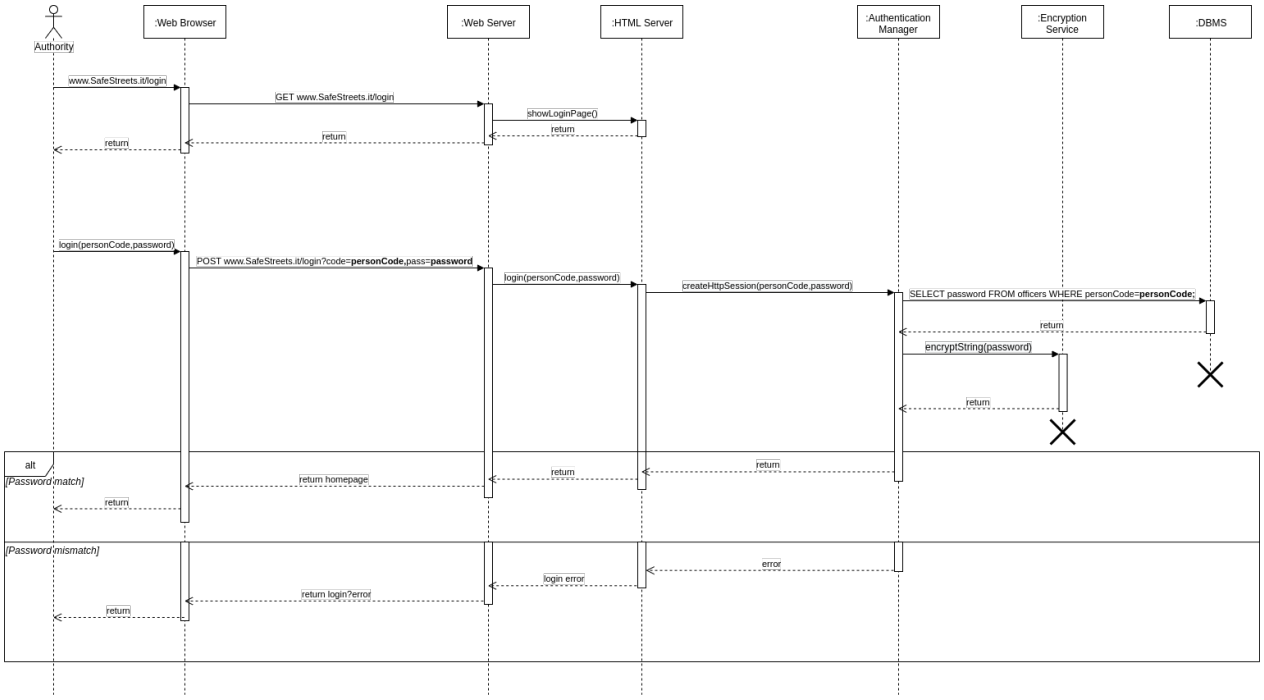


Figure 8: Login authority sequence diagram

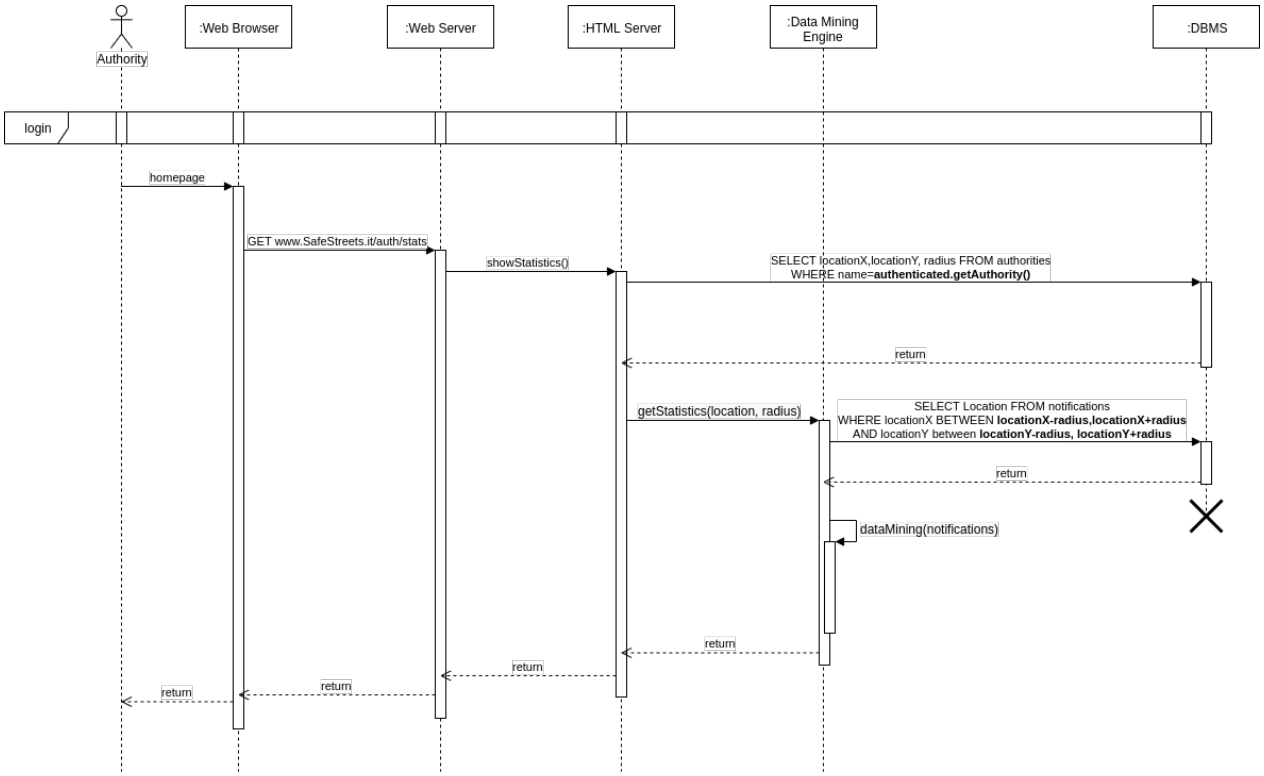


Figure 9: See statistics sequence diagram

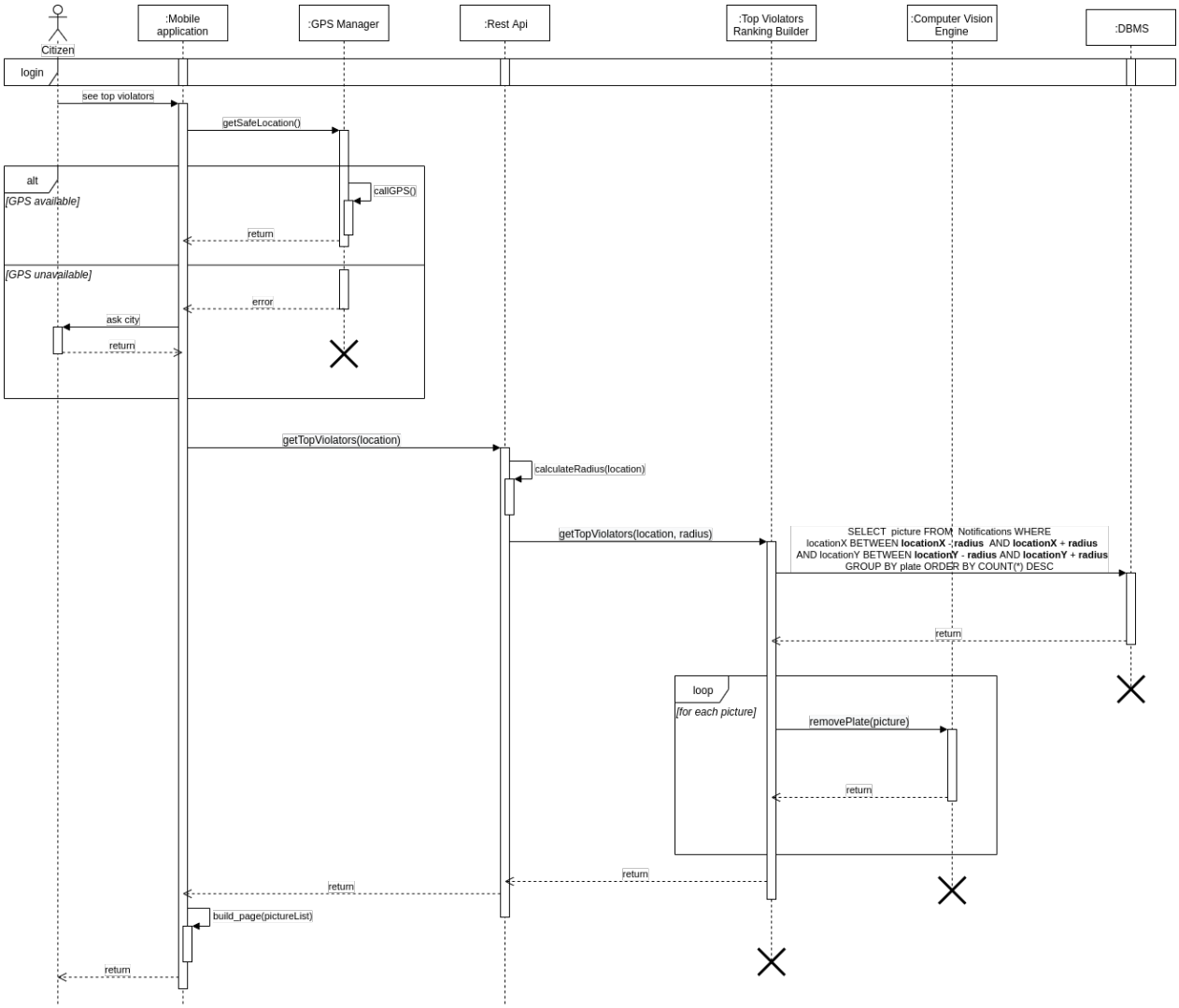


Figure 10: See top violators ranking sequence diagram

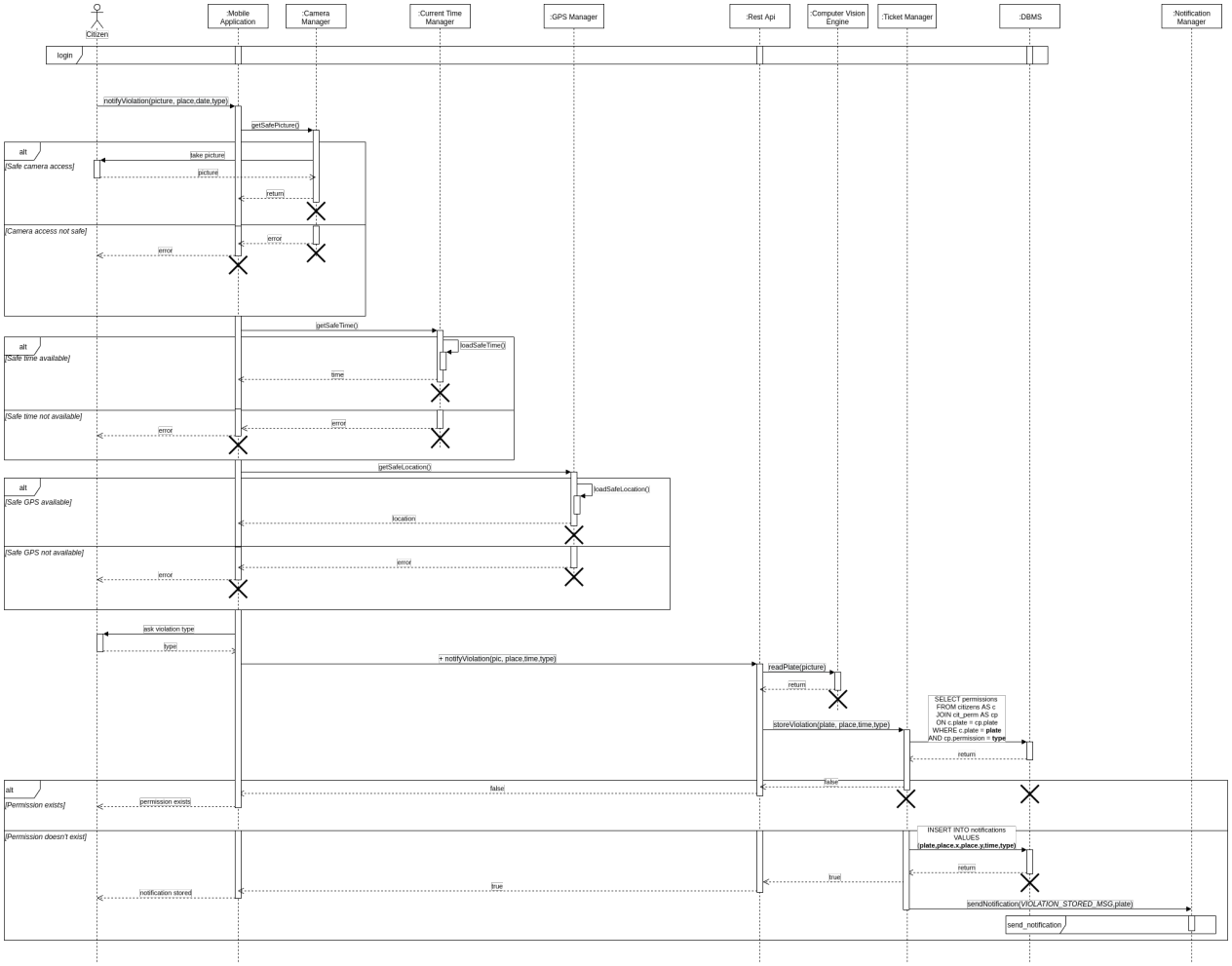


Figure 11: Notify violation sequence diagram

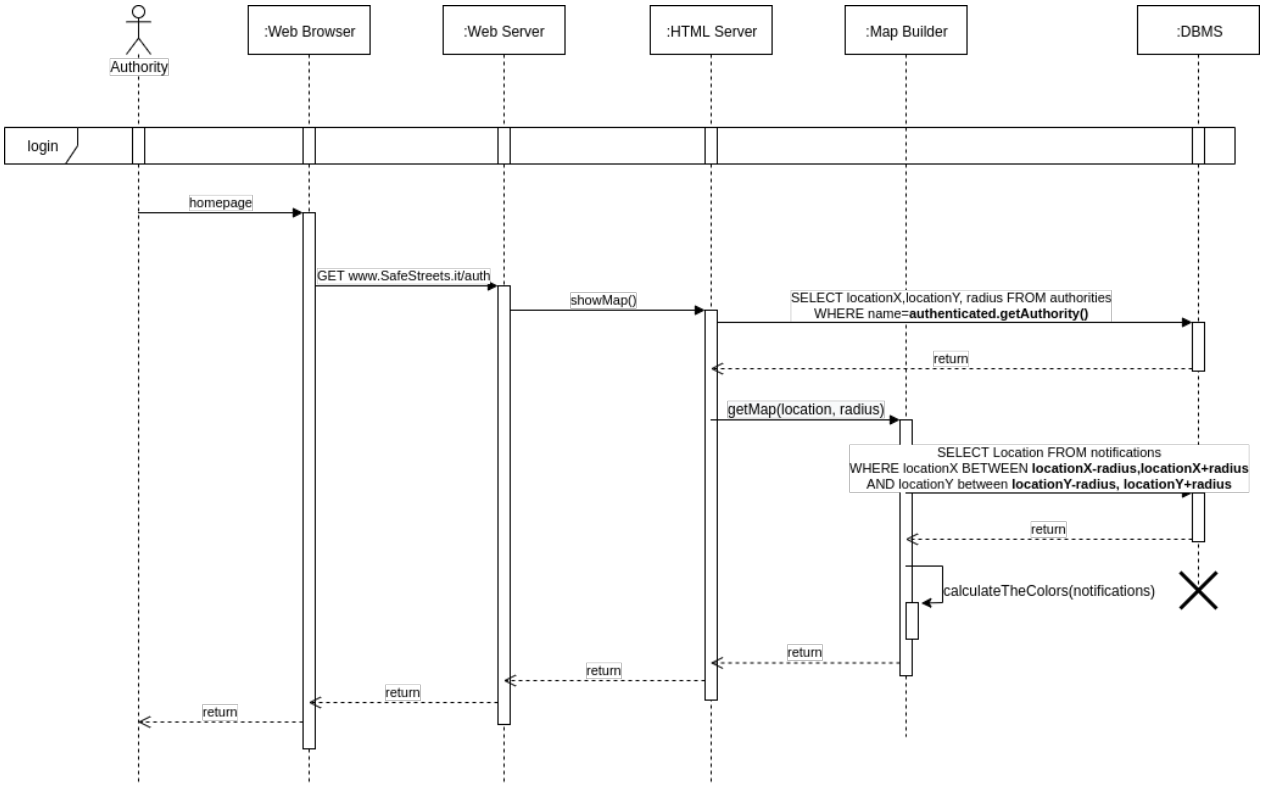


Figure 12: See map (authorities) sequence diagram

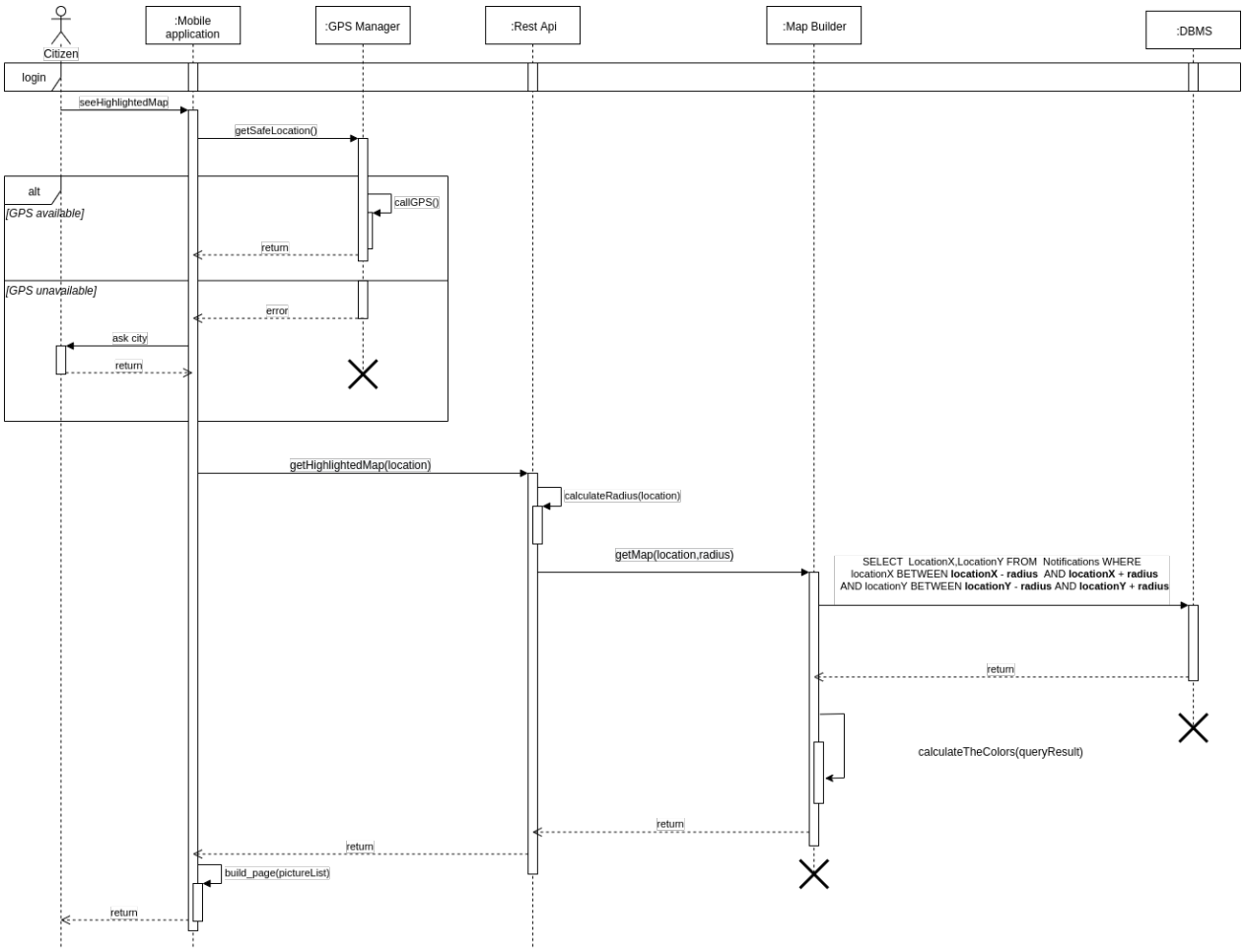


Figure 13: See map (citizens) sequence diagram

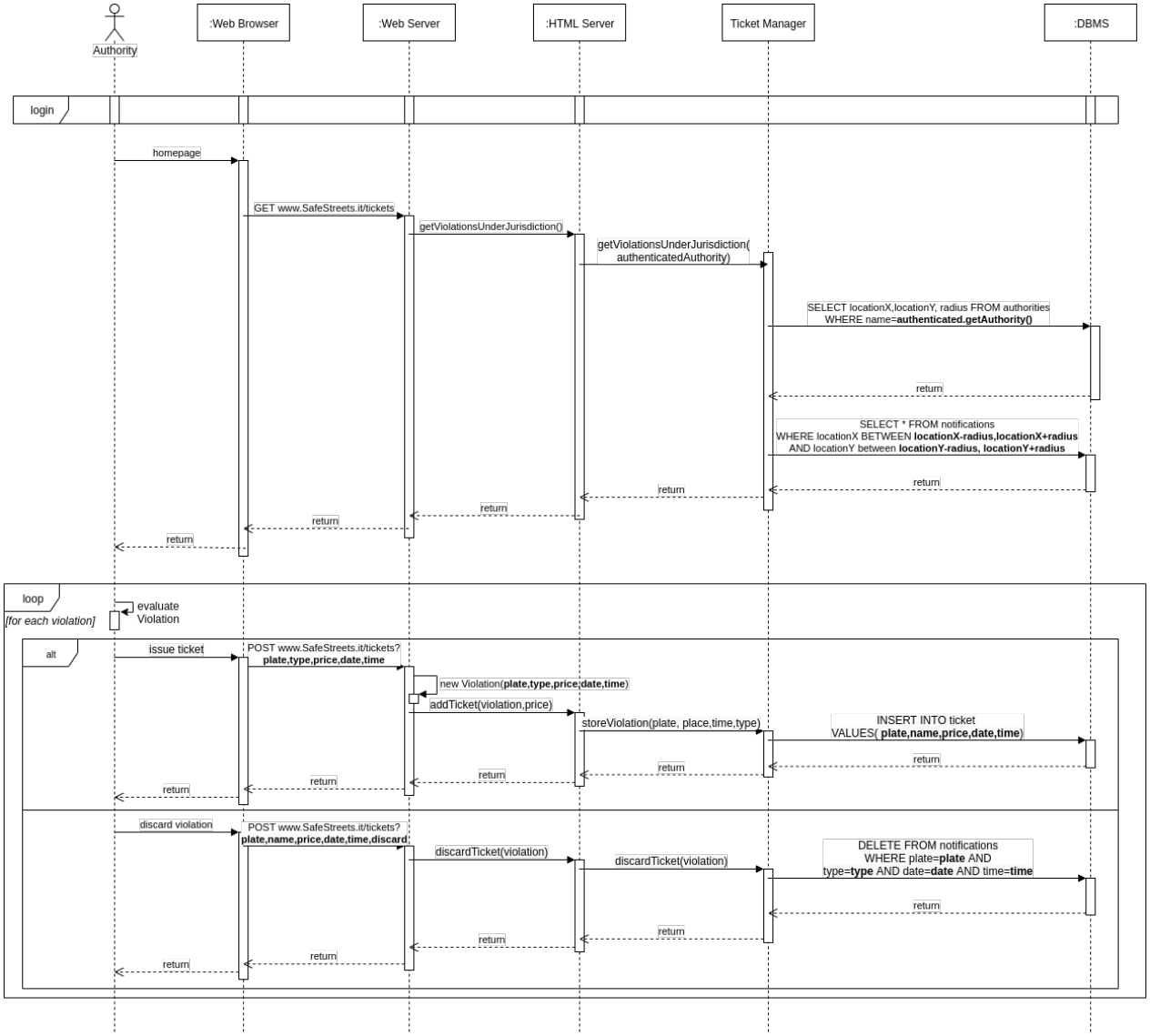


Figure 14: Issue ticket sequence diagram

2.7 Component Interfaces

The following diagram illustrates the interfaces and the dependencies between different components, highlighting user interfaces and programming interfaces.

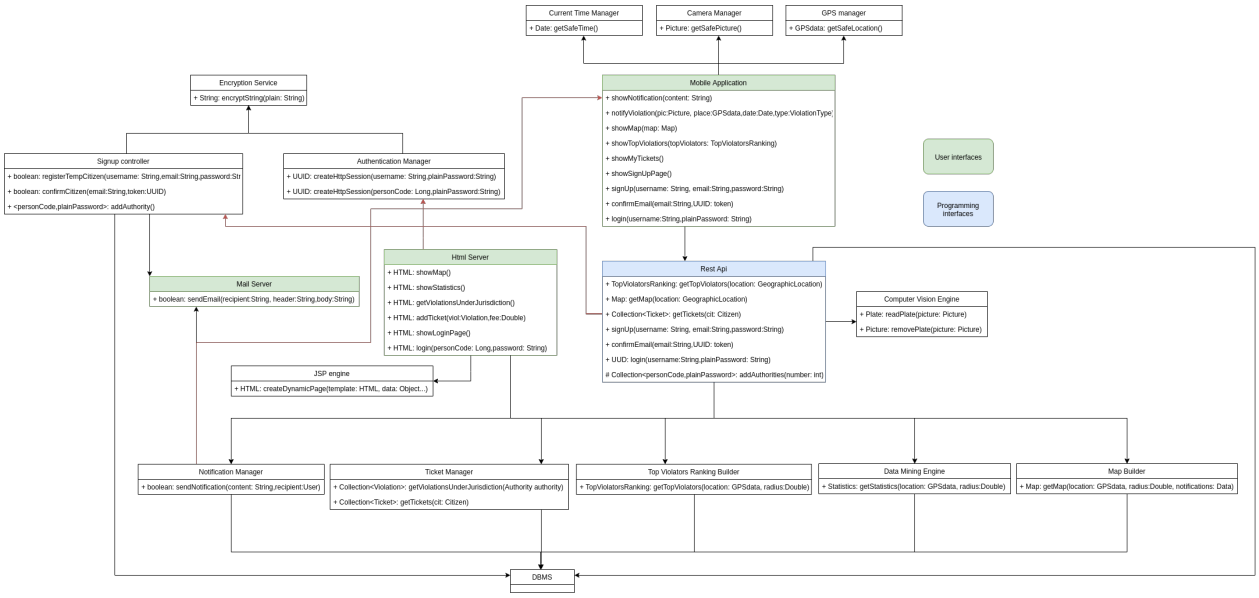


Figure 15: Component interfaces diagram

2.8 Selected architectural styles and patterns

2.8.1 REST paradigm

We decided to design the system following the REpresentational State Transfer paradigm, in order to reduce the design complexity and improve reusability and maintainability. Following are stated the key concepts of REST and how they're

- **Client-server architecture**
- **Stateless:** The components does not store information of what they are doing. Every request comes with all the needed data to provide a response, or any missing data can be retrieved from the database or by transforming the already available data.
- **Cacheability:** All the responses are marked as cacheable or not (if a response can be cached, overall latency can be significantly reduced)
- **Layered:** As pointed in 2.4, the system is organized in layers and provide interfaces to external processes (controlled by users or automated=
- **Uniform interface:** Uniformity of the interface is granded by exposing HTTP endpoints, which transmit resources using the JSON format.

  Resource identification: Evevry data exposed to the outside and not representational (Webpages) is encoded in the JSON format. Web pages are obviously provided in HTML format, in order to be displayable by common web browsers.

  Resource manipulation through representations: The format of the resources can be easly modified, as are represented in a structured text format

  Self-descriptive messages: The JSON format names every data exchanged. The names we use are expressive enought to be self-descriptive

Hypermedia as the engine of application state: Concerning the API, we provide a service broker to expose all of the interface mathods. With regard to the web application, it's navigable using web links (see fig. 19)

2.8.2 MVC pattern

The Model, View, Controller pattern is a widely used pattern. Its main advantage is a significant reduction of the complexity, granted by a layered structure that subdivides responsibilities. In detail:

- **Model:** the portion of the system responsible for data definition and storage
- **Controller:** the portion of the system responsible for business and functional logic
- **View:** the portion of the system responsible of data presentation and displaying

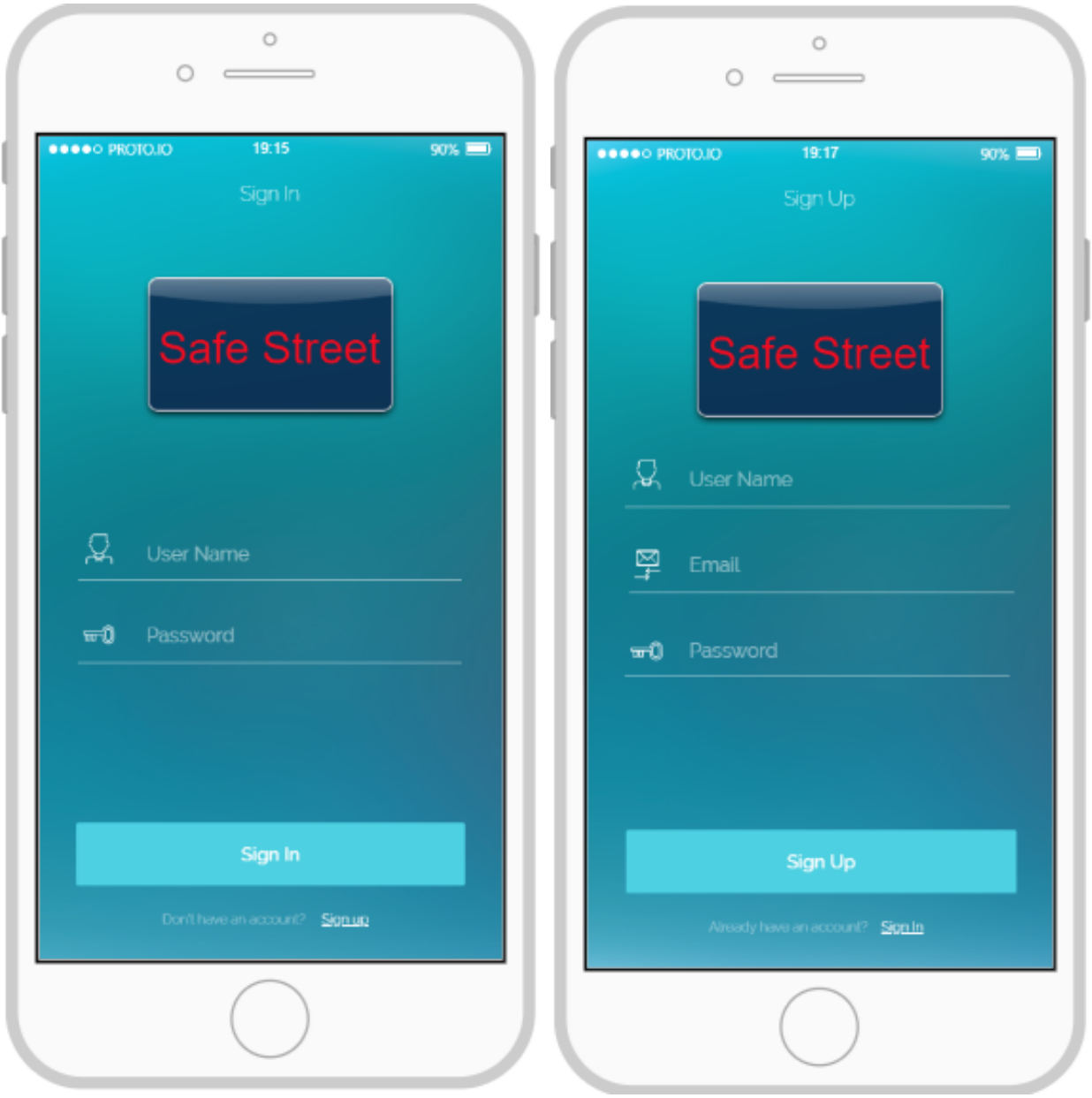
3 User Interface Design

Provide an overview on how the user interface(s) of your system will look like; if you have included this part in the RASD, you can simply refer to what you have already done, possibly, providing here some extensions if applicable.

3.1 Mobile application interface

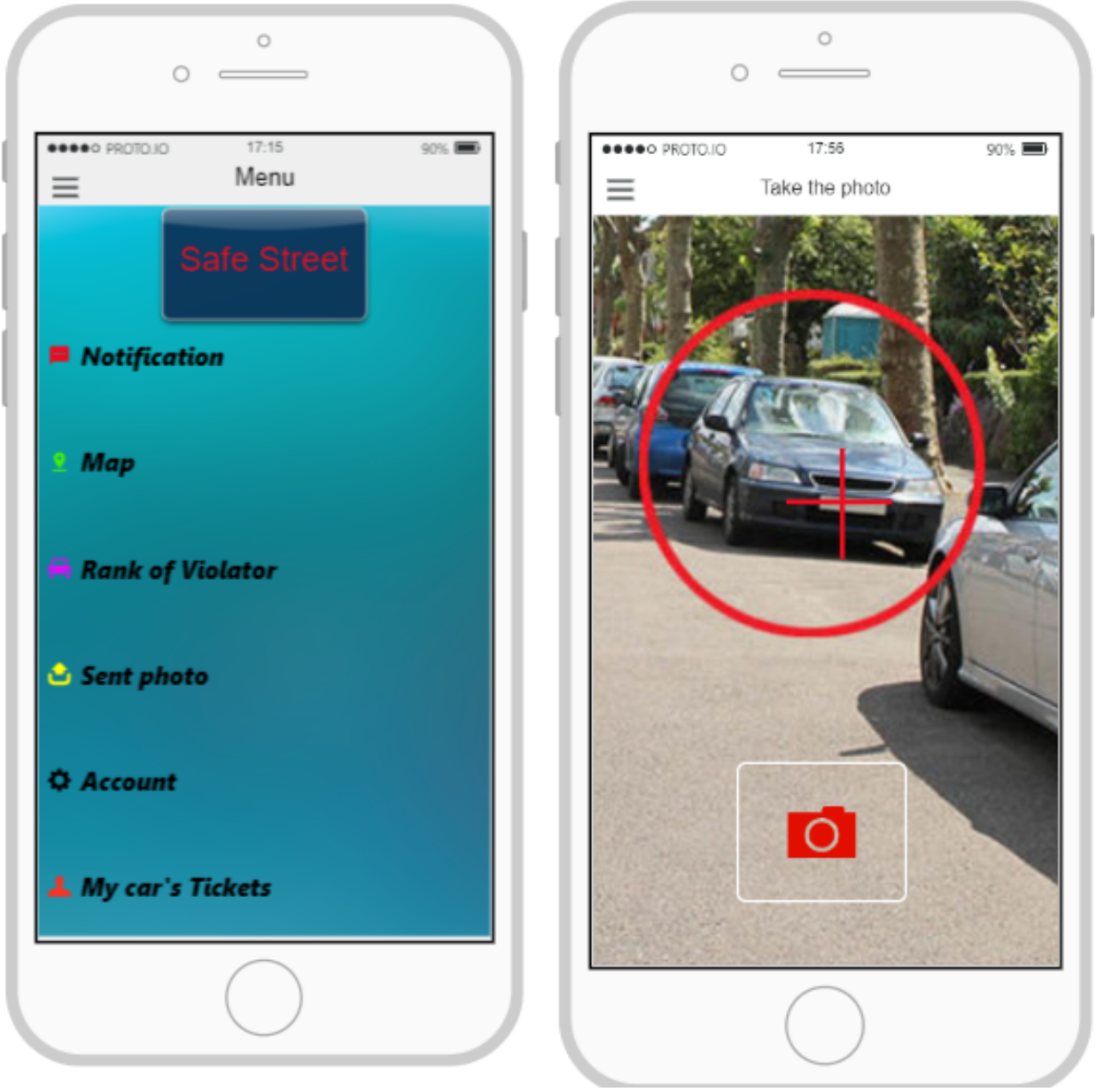
Here we have mockups about some part of Mobile App and below you can see a short description for each one of them:

- Figure 16a: Users need to insert their username and password to login.
- Figure 16b: If users do not have any account they must choose username and password and verify their email to make an account.
- Figure 17a: By clicking on the three-line icon users can access to Menu and choose any item that they need.
- Figure 17b: The first page after they open the application is Camera for having a rapid photo taking at the right moment.
- Figure 18a: Through the Menu users by clicking on the map option they will see a city map with highlighted streets that gives some useful information about more risky area.
- Figure 18b: After choosing the Top violator option from the Menu the list of top ranked violators will be represented.



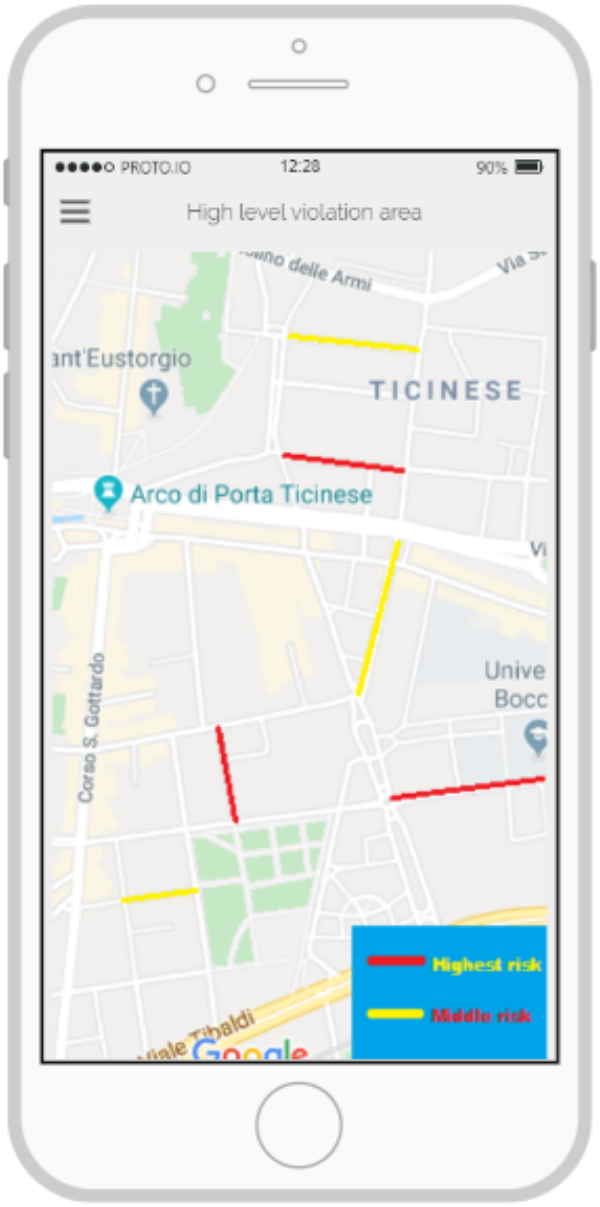
(a) Sign in

(b) Sign up



(a) Menu

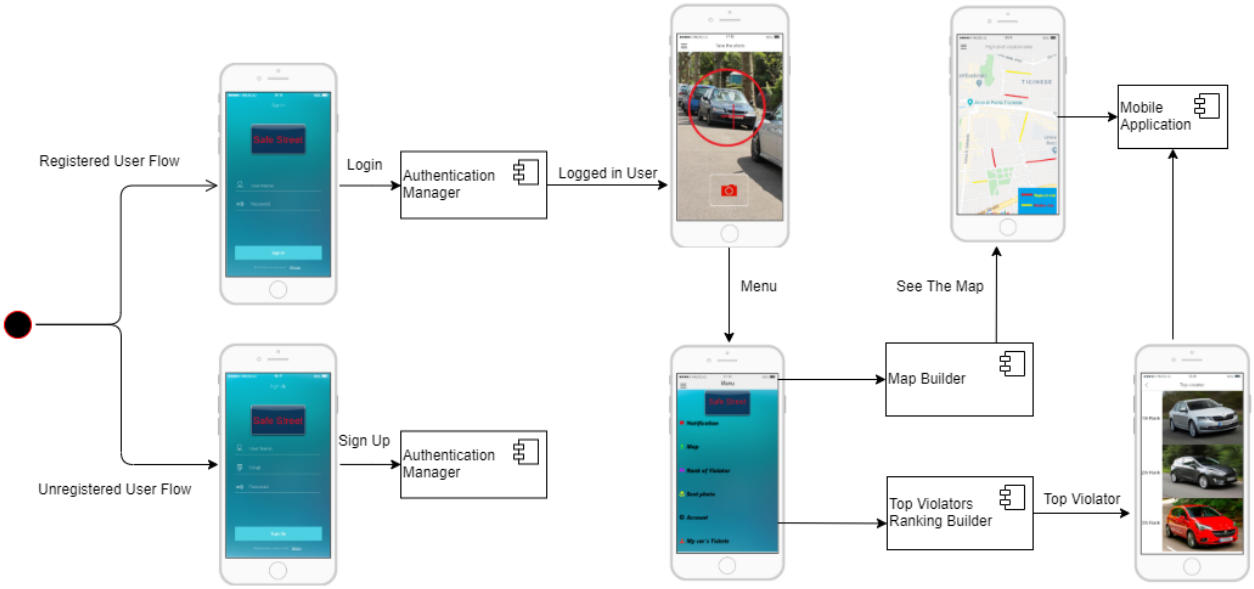
(b) Camera



(a) Map

(b) Top Violators

3.2 Mobile application UX diagrams



3.3 Web application interface

As you can see we provide some mockups for authorities interface through the web application.

- Figure 19 : After signing in, officers have access to the home page with all the features that they need.
- Figure 20 : By scrolling down through the home page or clicking on the map option officers can represent the map that they are already working in with highlighted streets in different sort.
- Figure 21 : Officers can access to Tickets page through the up bar and by filling obligatory data they can issue tickets.
- Figure 22 : With this approach that officers are already registered into the system they just need to insert their personal code and their Pre-prepared password for login to the system.

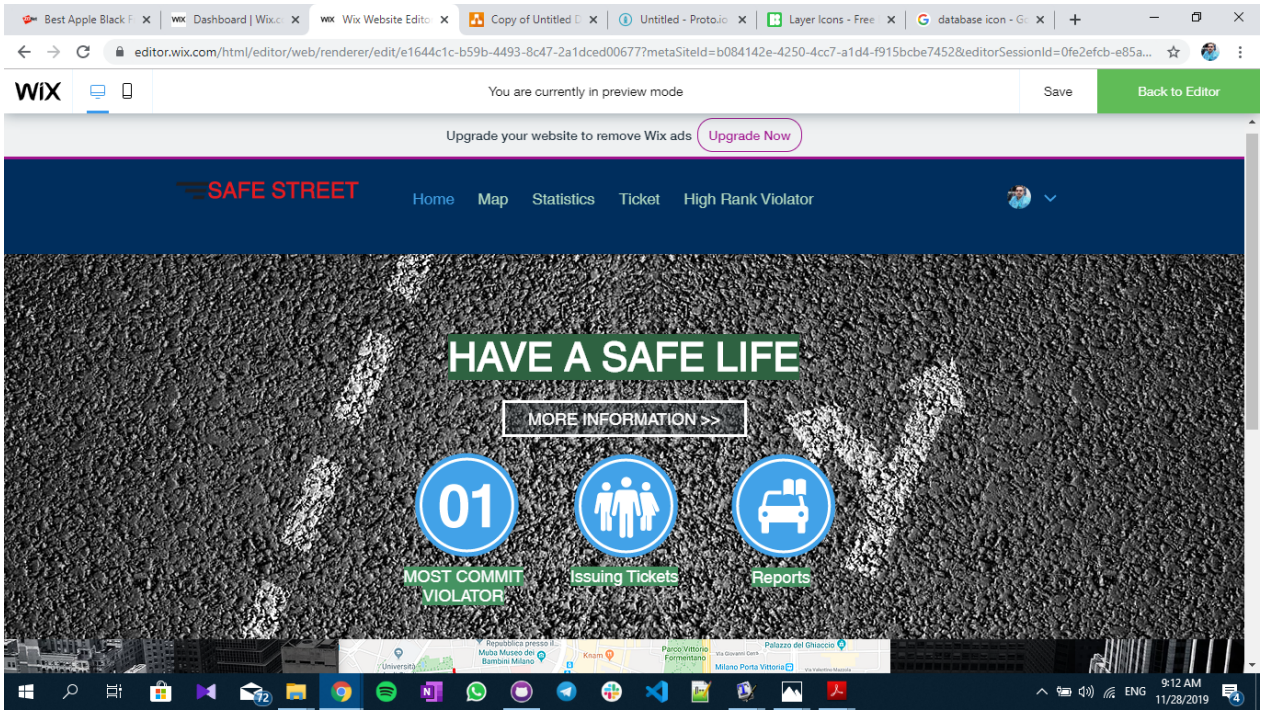


Figure 19: Home page

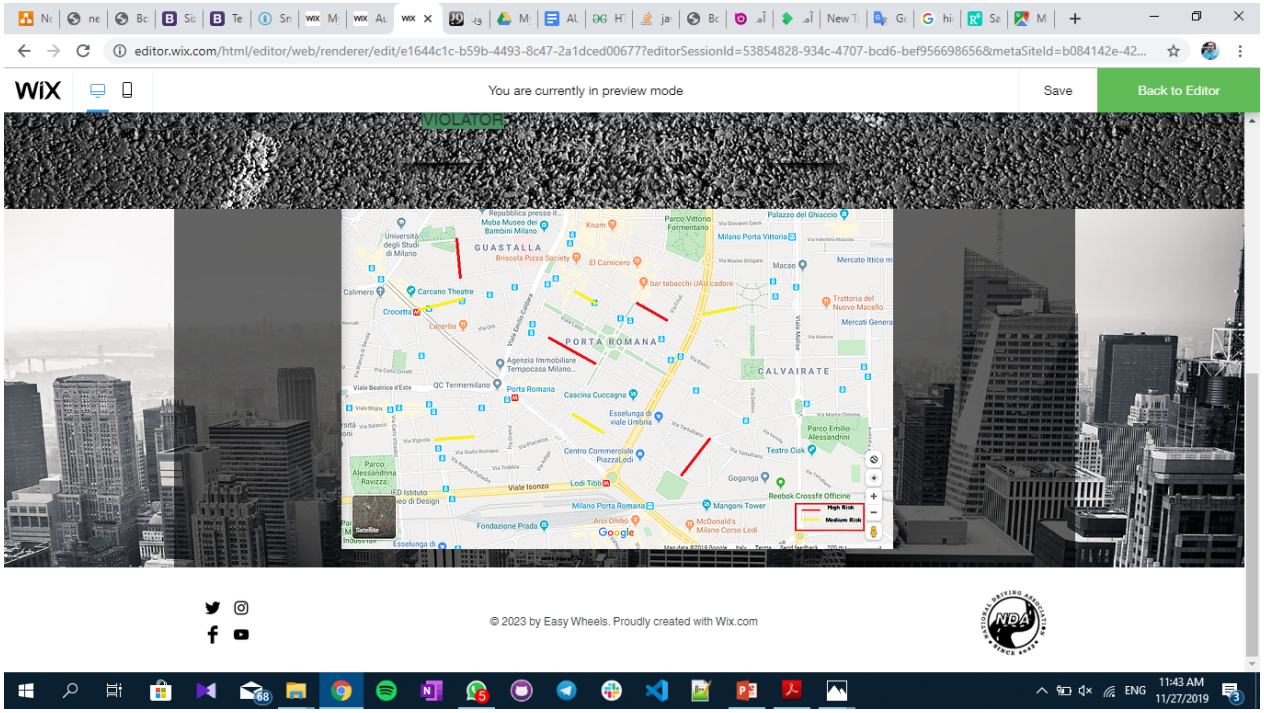


Figure 20: City map with risky streets marked

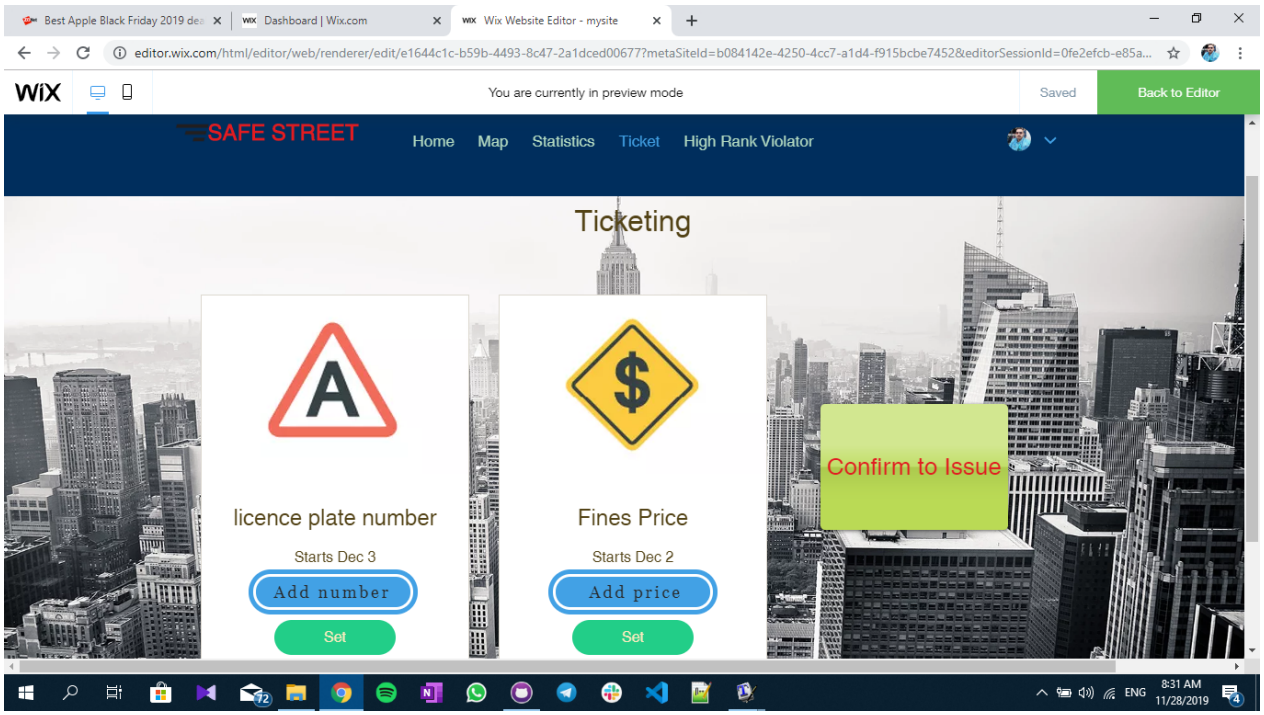


Figure 21: Insert ticket informations

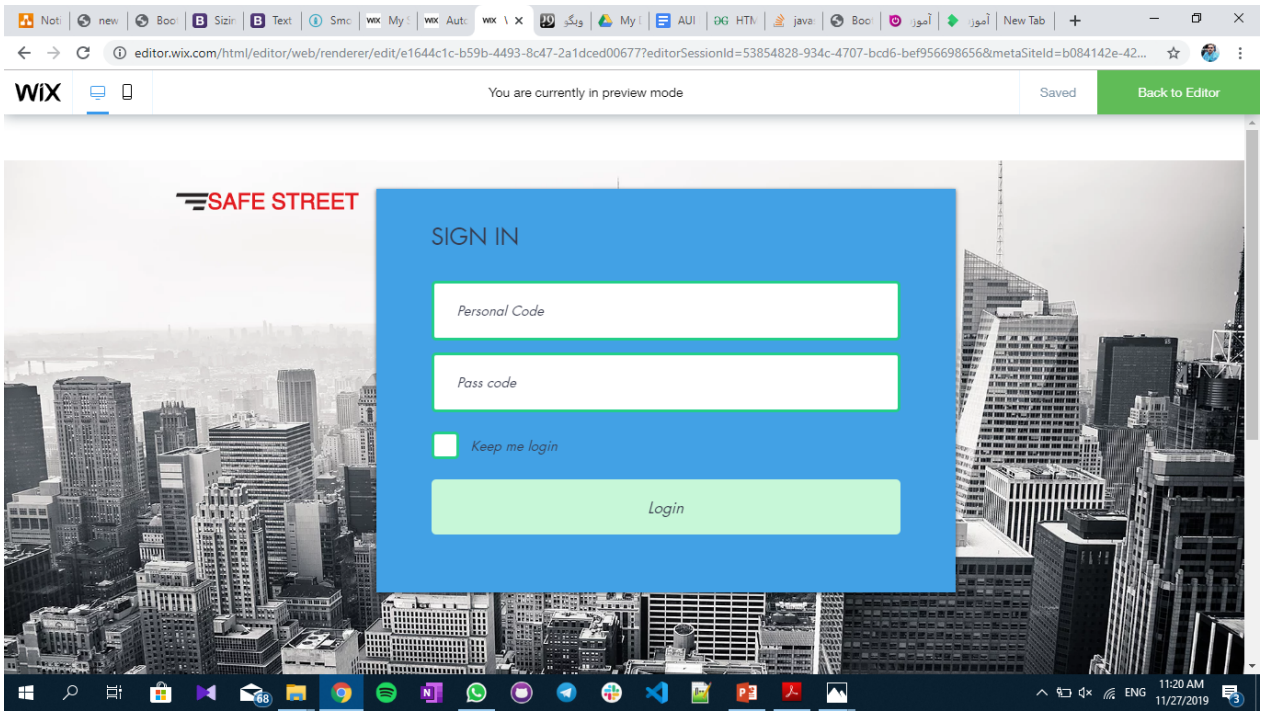


Figure 22: Login screen

4 Requirement traceability

4.1 Traceability matrix

Table 3: Traceability matrix

Goal ID	Req ID	Scenario ID	Responsible component
G.1	R.2	S.1	COMP.1
	R.4	S.3	COMP.15, COMP.16, COMP.17
	R.5	S.4	COMP.8
	UI.2		
	P.1		
	HW.2		
G.2	R.4	S.1	COMP.15, COMP.16, COMP.17
	R.6		COMP.13
	UI.2		COMP.13, figs. 16a to 18b
	P.2		COMP.15
	HW.2		
	OT.1		
G.3	A.1	S.3	
	A.2	S.7	
	A.3		
	A.4		
	A.5		
	A.6		
	R.2		COMP.11, COMP.15
	R.3		COMP.13, COMP.3
	R.6		COMP.13
	UI.1		COMP.4, figs. 19 to 22
	SW.1		COMP.6
	HW.1		
	OT.1		
G.4	A.5	S.4	
	A.6		
	R.3		COMP.13, COMP.3
	R.4		COMP.15, COMP.16, COMP.17
	R.6		COMP.13
	HW.2		
	OT.1		
G.5	A.1	S.3	
	A.2		
	A.4		
	R.2		COMP.1
	SW.1		COMP.6
	HW.1		
G.6	A.1	S.1	
	A.2		
	A.4		
	P.2		COMP.15
	HW.2		
G.7	R.1	S.6	COMP.4, COMP.14
	R.4		COMP.15, COMP.16, COMP.17
	R.6		COMP.13
	UI.1		COMP.4, figs. 19 to 22
	UI.2		COMP.13, figs. 16a to 18b
	SW.1		COMP.6
	SW.2		COMP.9
	HW.1		
	HW.2		

5 Implementation, Integration and Test plan

Identify here the order in which you plan to implement the subcomponents of your system and the order in which you plan to integrate such subcomponents and test the integration.

First, each component has been prioritized for both value generated for each user-type and implementation difficulty. The results of this prioritization are displayed in the table 4. For cloud-placed and bought components, the expected price and the configuration effort for the chosen implementation are both written in parenthesis. Empty cells indicate components not significant for that type of user. Finally, components fullfilled by the same service (cloud-based or bought) are not separated by horizontal lines. Analyzing the

Table 4: Component priorities

Component	Value (for citizens)	Value(for authorities)	Difficulty of implementation
Authentication Manager	Low to Medium	High	Cloud (cheap,low)
Signup Controller	Low to Medium	High	Cloud (cheap,low)
Encryption Service	Low	High	Cloud (cheap,low)
Mobile Application	High		Medium to High
HTML Server		High	Buy (free,low)
Rest Api	High		Buy (free,low)
Mail Server	Low	Low	Cloud (cheap,low)
JSP		High	Buy (free,low)
Notification Manager	Medium	Medium to Low	Low
Map Builder	High	High	Cloud (cheap,medium)
Top Violators Ranking Builder	Low to Medium		Low
Computer Vision Engine	Medium to High	Medium to High	Cloud (cheap,medium to high)
Data Mining Engine		Medium to High	High
Ticket Manager		High	Low
Camera Manager	Medium	High	Medium to Low
GPS Manager	Medium	Medium	Medium to Low
Current Time Manager	Medium	Medium	Low
DBMS	High	High	Buy (cheap,low)

table, we indentified 5 implementation phases: Prototype, Basic Logic, Authentication, Mobile Security, Extended features. In table 5 describes, for each phase, which component are to develop in that phase, how it's planned to develop them, their estimated effort and the estimate effort of the phase. Finally, fig. 23 shows which phases must happen before and which later (note that some may be performed in parallel).

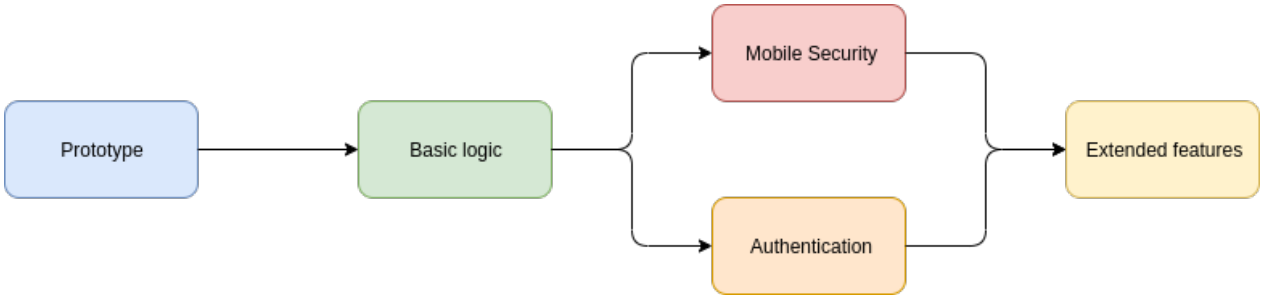


Figure 23: Phase priority

Unit testing For testing, we have to assume that every outsourced solution has been tested and is perfectly working (as we can't perform unit testing on them). With regard to the Mobile Application, the Ticket Manager, the Data Mining Engine, the Notification Manager and the Top Violators Ranking Builder, a set of tests will be generated before the beginning of the coding phase in order to guide it, and to define some

Table 5: Development phases

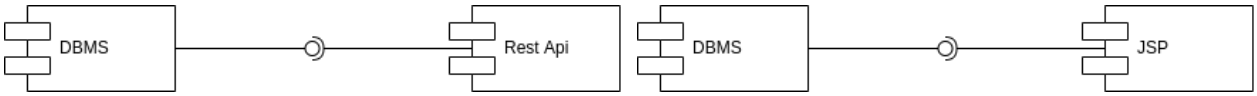
Implementation Phase	Adopted solution	Covered Components	Solution effort	Phase Efforts
Prototype	Amazon S3	DBMS	Low	Low
	Tomcat Web Server	Rest Api HTML Server	Low	
	Custom made Application	Mobile Application	Medium to High	
	JSP	JSP	None	
Basic logic	Amazon Rekognition	Computer Vision Engine	Medium to High	Medium
	Google Cloud: Maps	Map Builder	Medium	
	Custom controller	Ticket Manager	Low	
Authentication	Amazon cognito	Authentication Manager Signup Controller Encryption Service	Low	Low
	Amazon Simple Mail	Mail Server	Low	
Mobile Security	Mobile Application Update	Camera Manager GPS Manager Current Time Manager	Medium to Low	Medium to Low
Extended features	Custom controllers	Data Mining Engine Notification Manager Top Violators Ranking Builder	High Low Low	Medium

initial acceptance criteria. More precise tests will be developed during the coding phase, in order to test the coponent behaviour more in detail.

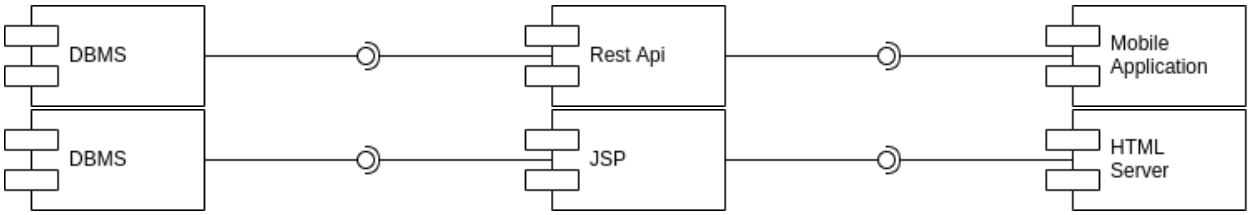
Integration testing Into each phase, integration testing will be performed to integrate components of the same phase one with the other and with the pre-existing system (assumed to be working perfectly and already tested exhaustively). Figures figs. 24 to 27 shows for each implementation phase the corresponding integration phases.

Acceptance test A set of acceptance tests will be generated at the beginning of each phase. This of acceptance tests will cover a great number of test cases, and will be usable as Regression tests as well

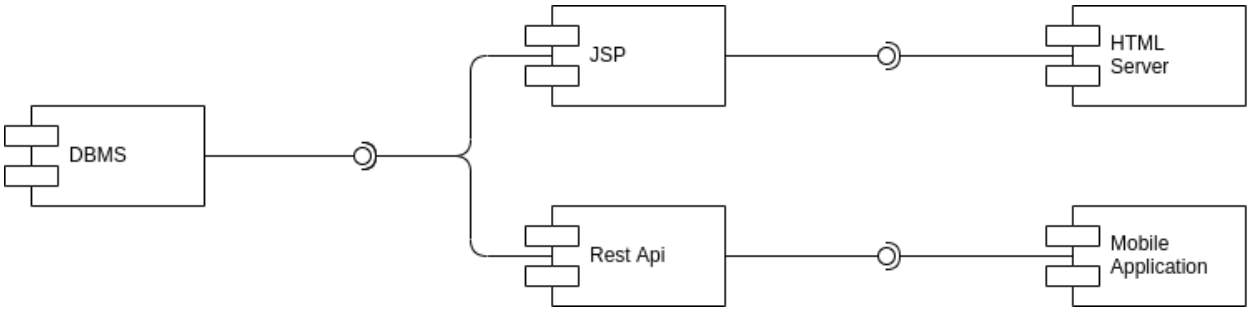
Figure 24: Prototype integration



(a) Model-controller integration

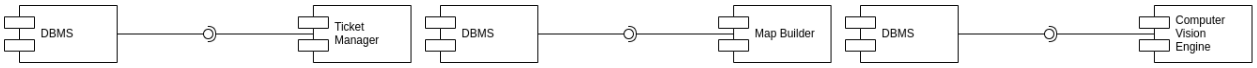


(b) Model-controller-view integration

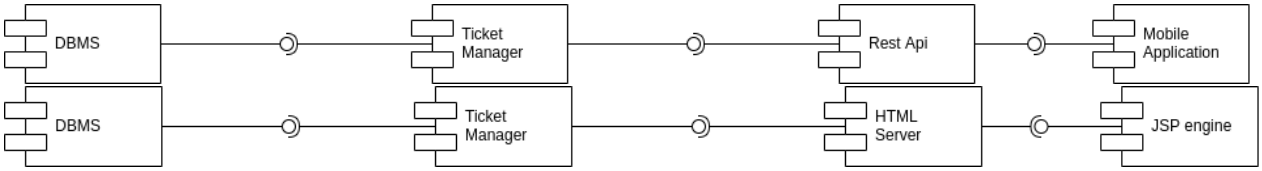


(c) Final integration

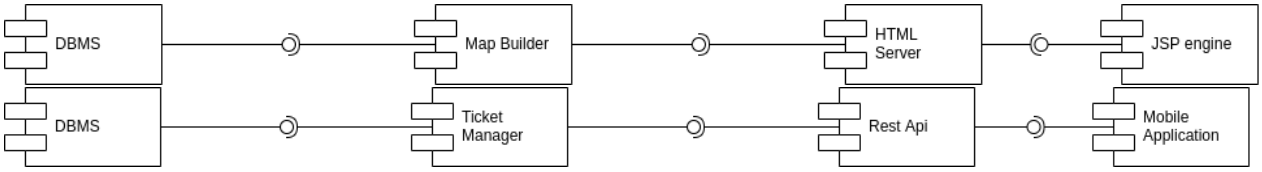
Figure 25: Basic Logic Integration



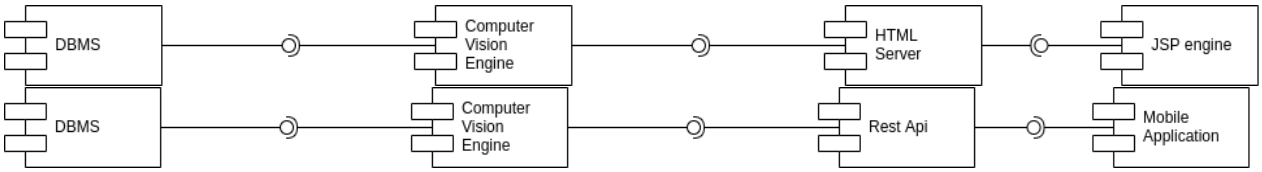
(a) Model-controller integrations



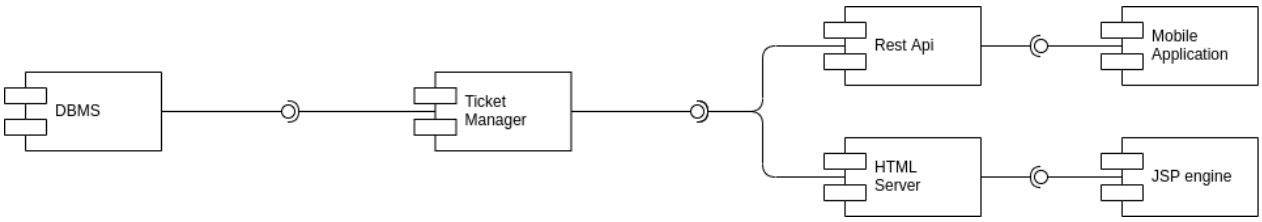
(b) Model-controller-view integration for Ticket Manager



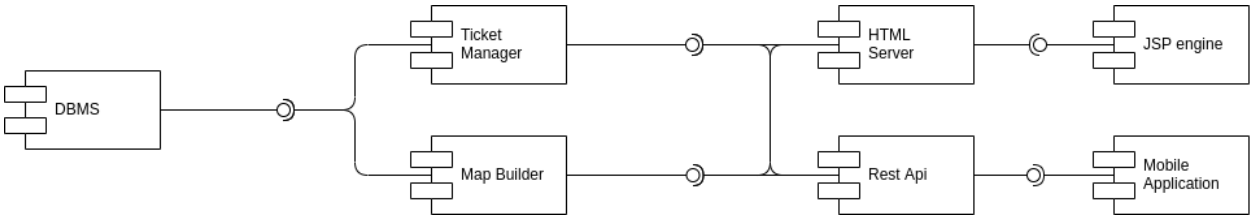
(c) Model-controller-view integration for Map Builder



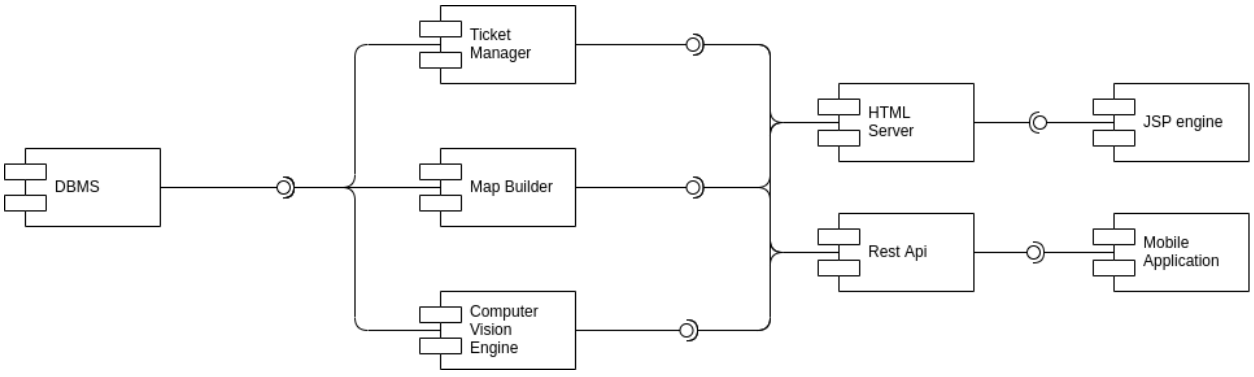
(d) Model-controller-view integration for Computer Vision Engine



(e) Final integration: Ticket Manager

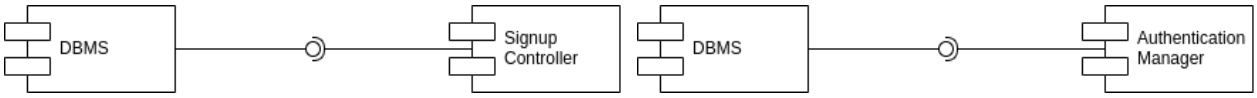


(f) Final integration: Map Builder

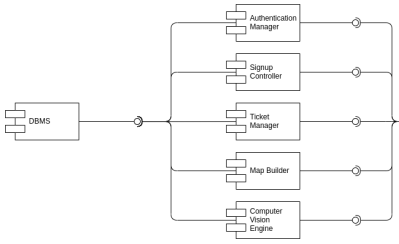


(g) Final integration: Computer Vision Engine

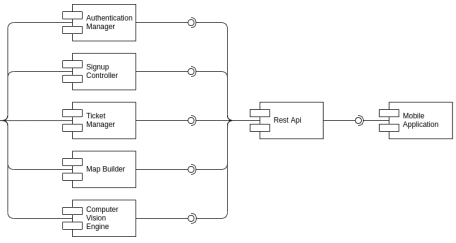
Figure 26: Authentication Integration



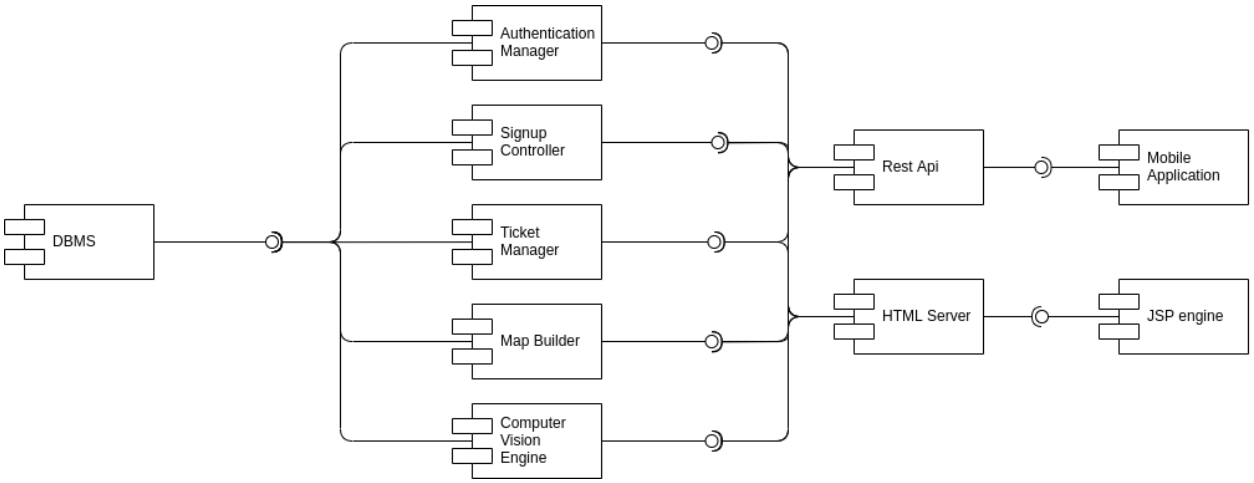
(a) Model-controller integrations



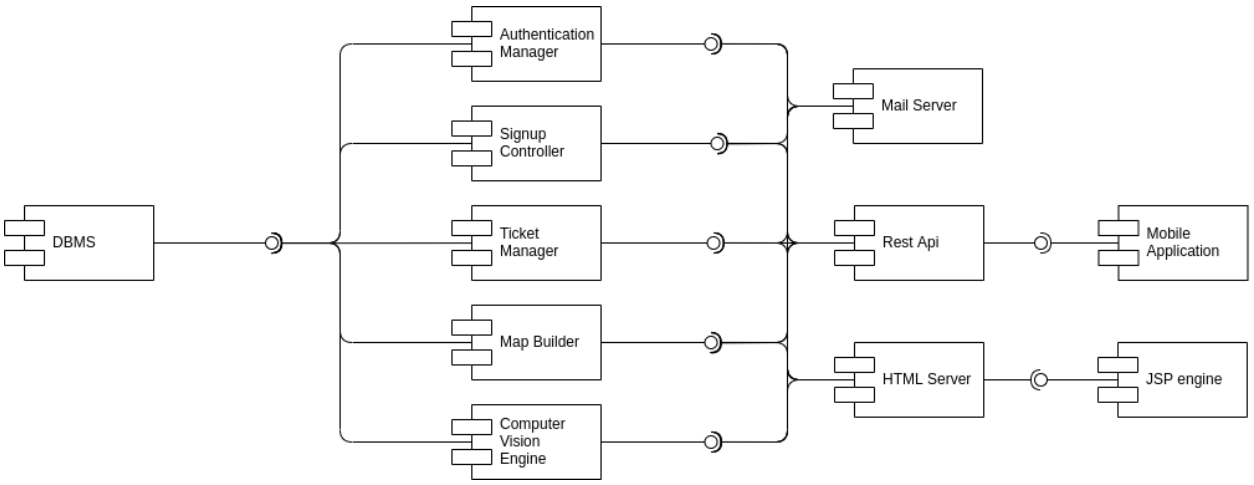
(b) HTML-authentication integration



(c) Rest-authentication integration

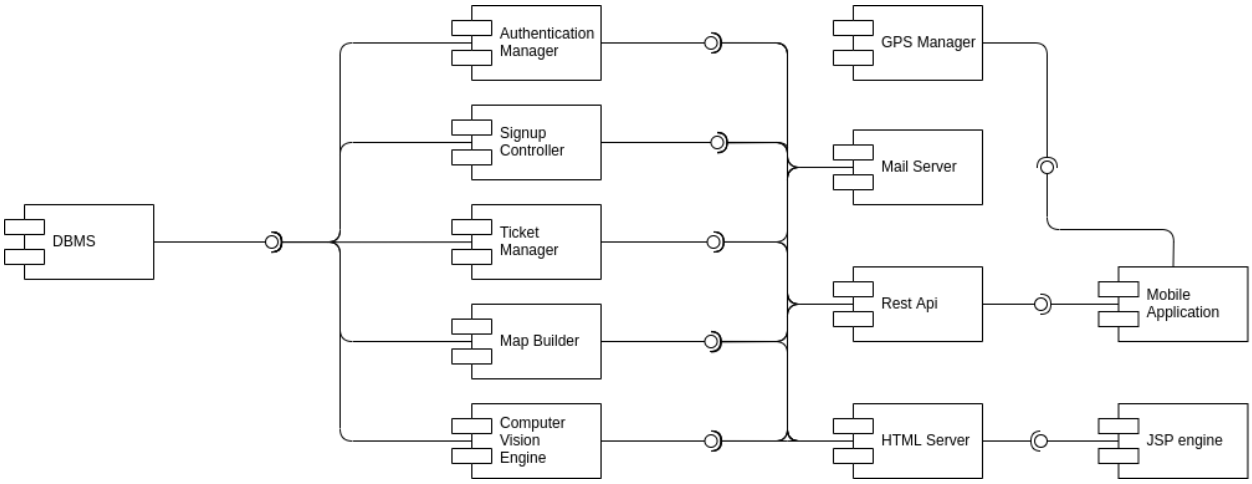


(d) Controllers integration: authentication

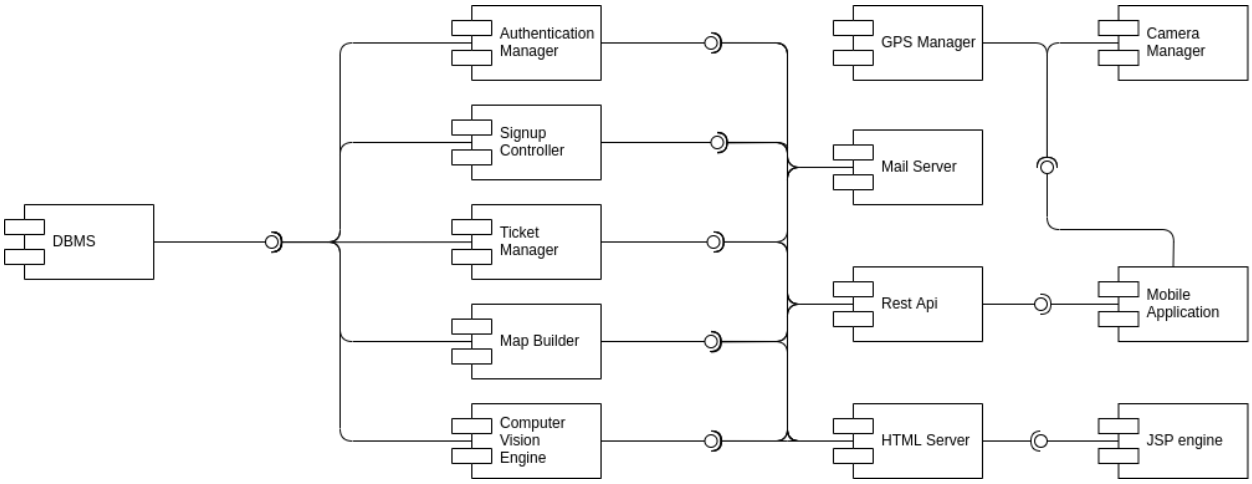


(e) Final integration: authentication

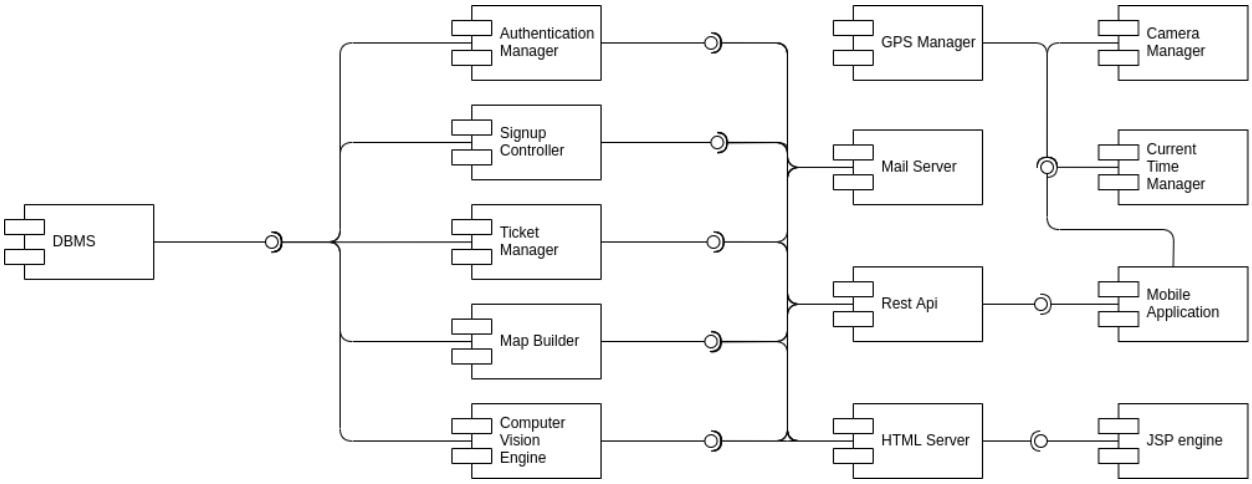
Figure 27: Mobile Security Integration



(a) GPS Manager integration

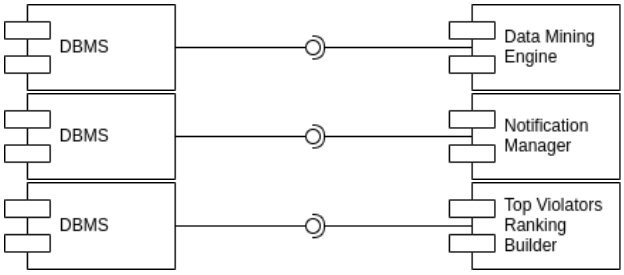


(b) Camera Manager Integration

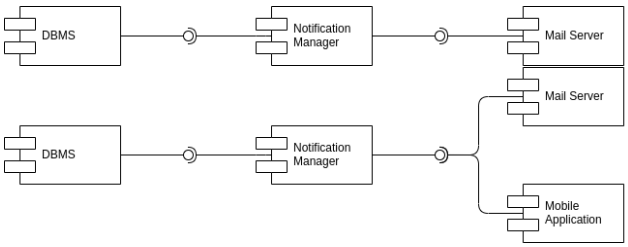


(c) Current time manager integration

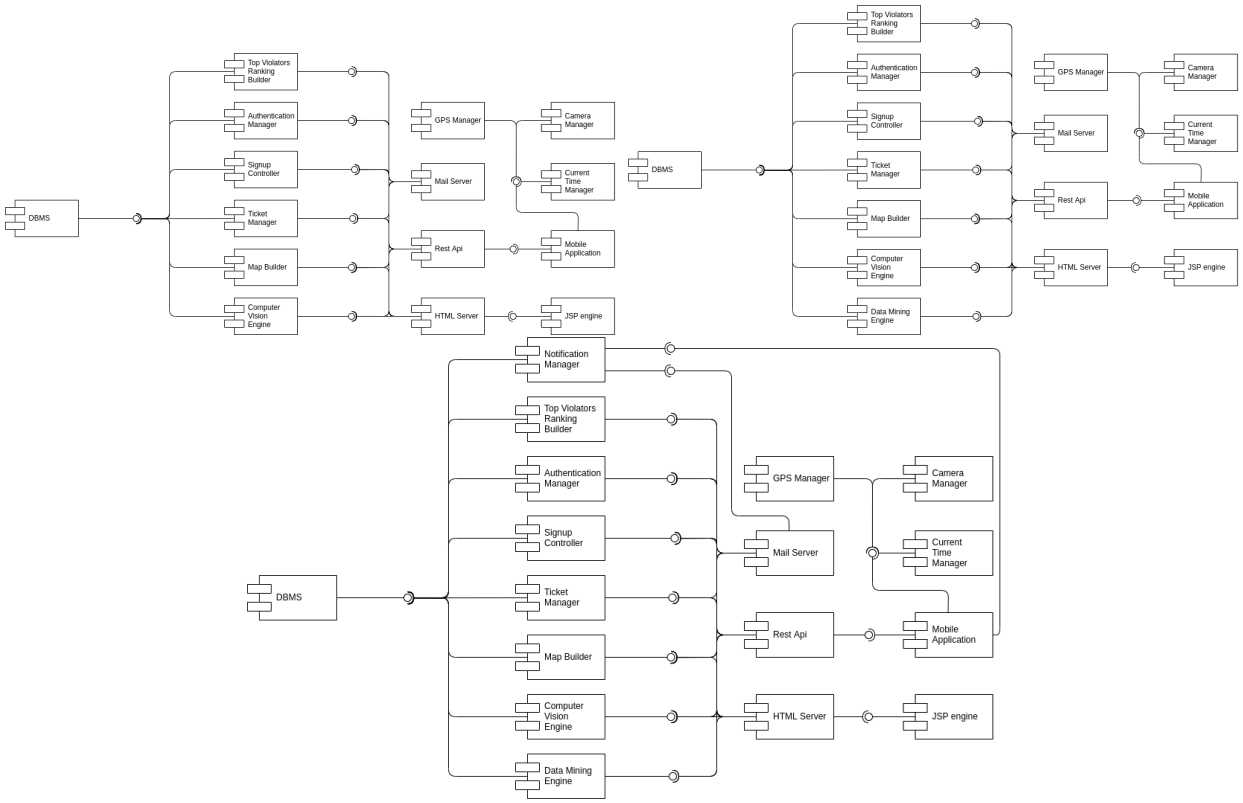
Figure 28: Extended Features Integration



(a) Model-controller integrations



(b) Notification Manager Integration



(c) Progressive full-system integrations

6 Effort spent

In this section you will include information about the number of hours each group member has worked for this document.

Matteo Secco

- 25 November: Components, toghether. 3h
- 27 November: Components. 3h
- 29 November: Deployment. 4h
- 5 December: Inserted images, refactor structure. 1h
- 7 December: Sequence diagrams. 5h
- 7 December: Implementation, integration and testing. 0.45h

Rahbari

- 25 November: Components, toghether. 3h
- 27.28 November: Interface. 3h
- 4 December : Component Diagram 5h
- 8 December: Data Model Diagram 3h

7 References