DD

Carlo Dell'Acqua, Adriana Ferrari, Angelica Sofia Valeriani

November 15, 2019

Contents

Chapter 1

Introduction

1.1 Purpose

1.1.1 Purpose of the Platform

SafeStreets is a service that aims to improve the safety of the general traffic. This is achieved by creating a community of users who are able to report any violation they see while the system manages all the aspects of data validation and statistical analysis. Different services contribute to this purpose:

- The first service offered by the front end application is the report service. Any registered user can submit violation reports and SafeStreets will validate them as described in the following sections with the help of the community.
- The second service offered by the front end application is the the Unsafe Areas Map. SafeStreets will provide statistics about areas that have a higher risk of violations based on the reports it receives and the danger of the infractions. Data can also be collected from public services if available to increase accuracy.
- The third service is the ticket generation. Traffic policemen will have access to a dedicated section of the application where SafeStreets will collect validated reports. This will enable any registered policeman to take actions against those violations.

1.1.2 Purpose of this document

In this document a more detailed approach with respect to the RASD will be provided to explain how we intend to build the platform. The following paragraphs will describe the architecture of the physical system and its abstract software components, how they interact with each other and how they create the services we described in the RASD.

1.2 Scope

1.3 Definitions, acronyms and abbreviations

1.3.1 Definitions

- **Safe (or Unsafe) Area**: A geographical region, usually a set of streets, where less (or more) accidents occur than the average based on neighboring regions
- **Report**: A set of data containing all the information about a traffic violation
- **System/Platform**: The SafeStreets platform
- **Web Application**: A Rich Internet Application that enables users to access the functionalities of the system through a modern web browser without having to manually install any other software
- **Reverse Proxy**: A server that acts as an intermediary between the client and the application server.

1.3.2 Acronyms

- **DD**: Design Document
- **API**: Application Programming Interface
- **GPS**: Global Positioning System
- **HTTP**: HyperText Transfer Protocol
- **HTTPS**: HTTP over TLS
- **JSON**: JavaScript Object Notation
- **SQL**: Standard Query Language
- **HTML**: HyperText Markup Language
- **CSS**: Cascading Style Sheets
- **JS**: JavaScript
- **DBMS**: DataBase Management System
- **RDBMS**: Relational DBMS

1.3.3 Abbreviations

- **Web App**: Web Application

Version	Major changes
1.0.0	First release

1.4 Revision History

1.5 Reference documents

- **Assignment:** SafeStreets Mandatory Project Assignment
- **Previous project example:**
  - **Assignment:** Mandatory Project Assignment AY 2018-2019
  - **Example document:** DD to be analysed AY 2019 2020

1.6 Document Structure

1. **Introduction**: The first section is a general description of the system's scope and purpose. It also includes references of the document and definitions, abbreviations and acronyms used along the paper
2. **Architectural Design**: The second section describe the architecture of the platform from different views.
3. **User Interface Design**: The third section includes an overall description of the user interface, explaining how the user will interact with the system and how the user interface will help them through the different interaction scenarios
4. **Requirements Traceability**: The fourth section provides a traceability matrix that allow keeping track of the requirements
5. **Implementation, Integration and Testing**: The fifth section describes how the system will be implemented, how the integration with existing system will be made and how the test will ensure the stability of
6. **Effort Spent**: The sixth section includes the detailed information about the time spent for each part of the document and how the work has been divided between group members

Chapter 2

Architectural design

2.1 Overview

2.2 Component View

In this section, the individual components will be presented in terms of their functionalities, their role and the needed sub-parts, as well as how the sub-elements interface one with another within the overlaying component. Moreover, in this section it will be specified which of the considered sub-elements are in charge of interfacing with the other components of the system.

2.2.1 Database

The database layer must include a DBMS component, in order to manage operations like insertion, update or deletion as well as logging of transactions on data inside the storage memory. The DBMS must guarantee the correct functioning of concurrent transactions and the ACID properties. As the application doesn't require a more complex structure than the one provided by the relational data structure, the database has to be a relational DBMS. The data layer will be exclusively accessible through the Application Server via a dedicated interface. Besides, the Application Server must provide a persistent unit to handle dynamic behavior of all the contained application data.
Sensible data, such as passwords and personal information, have to be encrypted properly and salted before being stored. The users will be granted access only after verification of correct and valid credentials.
Pertinence and integrity are granted, as well as isolation in concurrent transactions and atomicity, all ACID properties are satisfied.

2.2.2 Application Server

This layer must handle a great part of the application logic, together with the connections with the data layer and the multiple ways of accessing the application from different clients. The main feature of the Application Server is the set of specific modules of logic, that describe rules regarding violations, their identification, and the following procedures. Work-flows for each of the functionalities

provided by the application itself are also provided in this component.
The interface with the data layer must be handled by a dedicated persistent unit, that will be dedicated also to the dynamic data access and management, besides the object-relation mapping. In this way the only Application Server is granted to have access to the database.
The Application Server must also provide a way to communicate with external systems, by adapting the application to already existing external infrastructures. Furthermore, the Application Server must provide a way to interface to Web Server and the mobile via specific APIs, in order to decouple the different layers with respect to their individual implementation.
The main logic must include:

- **UserManager**: This module will manage all the logic involved with the user account, management, and operations like registration, login, and also profile customization and update, as allowed to authority, that can register with special permission after they are authorized. Furthermore, the module will deal with the generation and provision of user credentials, connected to the optional two-factors authentication, that can be requested by the client.

- **MapManager**: This module contains the logic used to locate violations and users; besides it deals with the definition of the Unsafe Area boundaries.This module must provide useful data to the logic that is at service of the unit that deals with Authorities, since it can need localization information, in order to perform its task.

- **AuthorityManager**: This module contains all the logic that grant access to an authority to its specific functionality: generation of traffic tickets and access to personal information and sensible data regarding offenders. Besides, the module is also devoted to the fundamental task of keeping the chain of custody of a certain notification with license plate, in order to preserve the integrity of information.

- **ReportManager**: This module provides the logic behind the report of violations, with particular focus on timing restrictions and on the confirmation of the report by other users. This module is also in charge to detect multiple reports of the same violation.

- **NotificationManager**: This module is used as a gateway from all the modules that need to interact by sending an email to the clients. Its task is managing the logic behind the email notification services.

2.3 Deployment View

The SafeStreets platform can be divided into the following Tiers that separate different aspects of the application.

- Client: it represents the client application. It will be the Web Application running in the browser or the native mobile application running on the target OS

- Reverse Proxy: it represents the intermediary between the client and the server application. It will provide basic functionalities such as data compression and caching.
- Server Application: it represents both the static file serving service and the application that provides the REST APIs for accessing and creating dynamic information
- Database: it represents the data storage solution. This tier identifies both the RDBMS and the media storage

This is more of a logical separation due to the fact that we'll be using a Cloud Infrastructure Provider, thus the real architecture may vary based on the specific company offering the service.

In the following page a simplified graphical representation of the architecture previously described is provided.

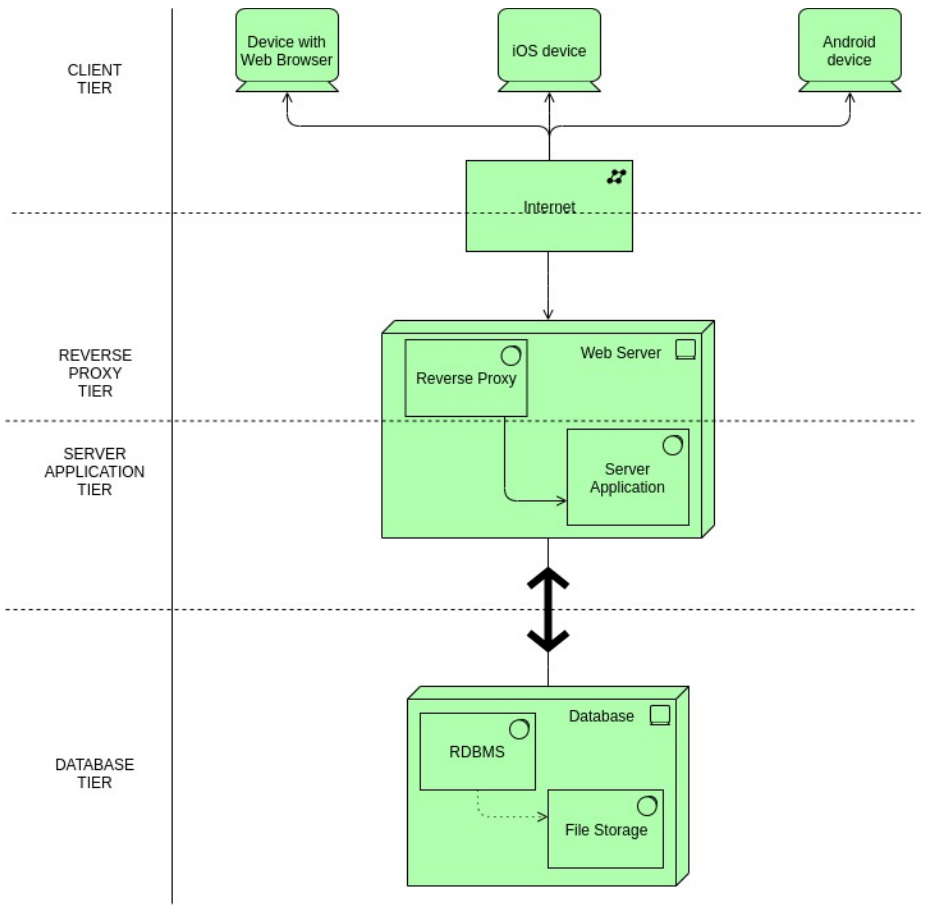

Figure 2.1: *Simplified Architecture diagram*

2.4 Runtime View

2.5 Component Interfaces

SafeStreets will be built using a RESTful architecture. The web application and the mobile application will interact with the server using HTTP endpoints communicating using the JSON format. The main advantages of these choices are:

- separation of the presentation layer from the application logic at a physical level: delegating all the presentation logic to the client device will reduce the load of the server (no Server Side Rendering)
- the JSON format will ensure the compatibility with all the platforms: being a standard communication format guarantees the existence of a variety of libraries for all client platforms

The server application will be structured using the concept of middlewares. Middlewares are chainable components in the HTTP Request-Response flow and guarantee a good level of isolation among the different functionalities they offer, thus providing a simple pattern that encourages the employment of the Single Responsibility Principle.

The middlewares we'll be using are:

- CompressionMiddleware: it reduces the overall bandwidth needed by the server using a compression algorithm that decreases the size of the data sent (and received) by (and to) the server. This middleware will be provided by the Reverse Proxy

- AuthenticationMiddleware: it has the purpose of filtering requests, rejecting those which lack the permissions to access a specific resource

- LoggingMiddleware: it logs the server traffic. This can be useful for debugging purposes and statistical analysis on further improvement of the infrastructure

The SafeStreets APIs will be developed using **express**, a lightweight library for Node.js that provides an easy programming interface for building middlewares and REST applications. The main language we'll be using is TypeScript due to its statically-typed nature. Moreover, being it a superset of JavaScript, it guarantees the compatibility with a large amount of web-oriented libraries and first-class Functions. Functional programming fits well with the stateless concept of a RESTful architecture, we'll be following a Functional style for our code, trying to avoid state whenever possible to reduce the possibility of error caused by mutability.

The database system we'll be using is PostgreSQL, a common RDBMS, on which we'll store all kind of data except for multimedia content. This content, being non-relational and prevalent on our platform will be stored on a Cloud Storage solution (such as Google Cloud Storage) and linked to the RDBMS using its own identifiers.

2.6 Other Design Decisions

Chapter 3

User interface design

Chapter 4

Requirements traceability

Chapter 5

Implementation, integration and test plan

Chapter 6

Effort spent

Carlo Dell'Acqua

Task	Time spent (hours)
Project setup	0.5
Introduction	1
Architectural Styles	1
Deployment View	1.5

Adriana Ferrari

Task	Time spent (hours)
Architectural Styles	1
Deployment View	1.5

Angelica Sofia Valeriani

Task	Time spent (hours)
Component View	2

References