POLITECNICO
MILANO 1863

Software Engineering 2

Design Document

Amorosini Francesco
Casali Alice
Fioravanti Tommaso

8 December 2019

Introduction

1.1 Purpose

While the RASD presented a general view of the SafeStreets appliction and its features, this document aims to further analyze the system's design and architecture, by describing for each of its components their runtime behaviour, integration, interfaces, implementation and testing plans. This document is mainly intended to be used by the test and development teams as a guidance in the development process, but also to prevent structural degradation during maintainance and extension phases. Nonetheless, the document is also addressed to all the stakeholders who are interested in supervising the development process.

1.2 Scope

SafeStreets: an application that aims to improve the safety of urban areas by giving its users the possibility to report traffic violations to authorities. users are logged in either as citizen, those who report the violations, and authorities, those who are notified about newly reported violations and are supposed to take action on them.
The system is in charge of collecting all the Reports, storing them, and notifying the authorities about them. The stored Reports are then used to build statistics, find unsafe areas, and compute suggestions on how to improve the safety of such areas. The system can also communicate local Municipalities' Systems in order to retrieve information about accidents and iussued traffic ticket: in this case some of the above functions are enhanced, and some new functions are enabled (g.e computing statistics on traffic ticked).

Further information about the scope of the application can be found in the Chapter 1 of the RASD.

1.3 Definitions, Acronyms and Abbreviations

1.3.1 Definitions

- *Client*: a piece of computer hardware or software that accesses a resource or a service made available by a Server.
- *Server*: a device or a computer program that provides resources or functionalities to other programs or devices.
- *Reverse Proxy*: particular proxys that are responsible of forwarding requests to one or more Servers which will handle it.
- *Firewall*: a network security systems that monitors incoming and outcoming network traffic, applying predefined predefined security rules.
- *Port*:

1.3.2 Acronyms

- **RASD**: *Requirement Analysis and Specification Document*, the document in which all the requirements and goals of the application are throughly described.
- **API**: *Application Programming Interface*, interface, or communication protocol between Client and Server intendend to simplify the building of the Client-side software.
- **OCR**: *Optical Character Recognition*, software dedicated to the detection of characters contained in a document and to their transfer to digital text that can be read by a machine. In this context, OCR will be used to read license plates.
- **UML**: *Unified Modeling Language*, is a standard visual modeling language intended to be used for analysis, design, and implementation of software-based systems.
- **GPS**: *Global Positioning system*, technology widely used to get the user's position.
- **DBMS**: *Data Base Management system*, software that provides organized space memory to store information.

1.3.3 Abbreviations

- [R_i]: i-th requirement.

1.4 Reference Documents

- Specification document: *SafeStreets Mandatory Project Assignment.pdf.*
- Requirement Analysis and Specification Document: *RASD.pdf.*

1.5 Document Structure

This document is presented as it follows:

1. **Introduction** presents a general overview, the scope and the purpose of the document.
2. **Architectural Design** shows the main components of the system and their relationships. This section will also discuss the arichitectural choices of the design process.
3. **Algorithm Design** presents and discusses the algorithms that will enable the system's functionalities.
4. **user Interface Design** provides some further details on the user interface defined in the RASD.
5. **Requirement Traceability** maps all the functional requirements defined in the RASD over the components that will accomplish them.
6. **Implementation, Integration and Testing Plans** shows the order in which the implementation and the integration of the components will occur, and how the testing phase will be carryed out.
7. **Effort spent** displays the time spent writing this document by each member of the team.

Architectural Design

2.1 Overview

In this chapter the architectural structure of the system will be discussed at multiple levels of abstraction. A high-level view of the components and their interactions is represented in Figure **??**. The details will be explained in the next sections.

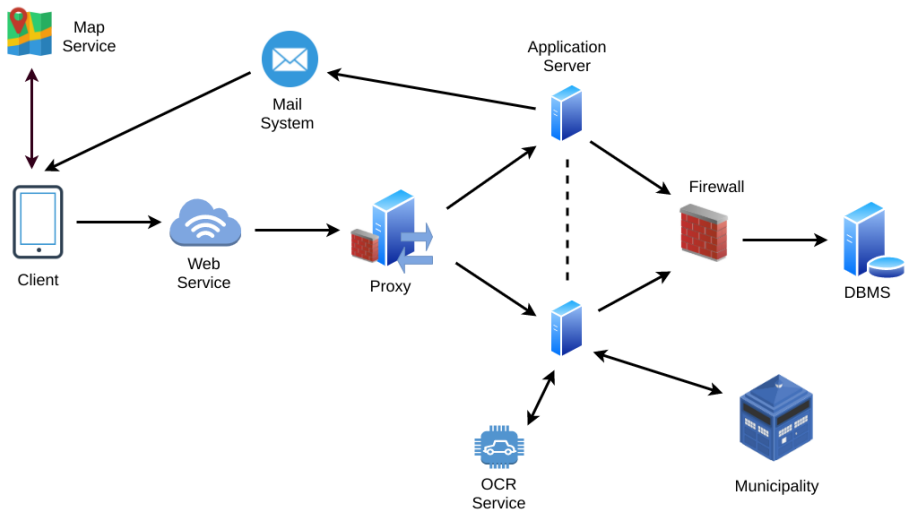

Figure 2.1: Overview of the system.

2.2 Component view

2.3 Deployment view

In this section it is described the deployment view of the components inside the system. The deployment diagram describes the distribution of components of the hardware.

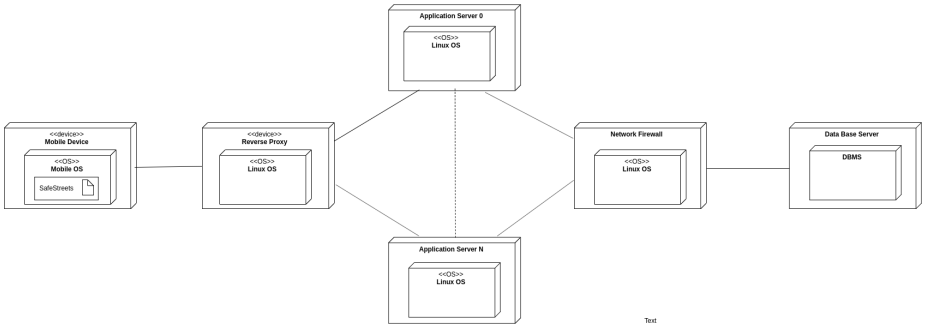

Figure 2.2: Deployment diagram of the system.

The system is composed of a multitier architecture, each specific role is clarified below.

- **Client**
  The first tier contains the client mobile machines which have installed SafeStreets application to access all its functionalities.

- **Reverse Proxy**
  The second tier contains a reverse proxy to implement load balancing on the several requests to access the application servers. Further, it is a cacheble component who can speed-up the most frequent requests. We decide to use a Linux machine for safety and semplicity reasons.

- **Application servers**
  This is the middleware level of the application where all the computations happen. The servers are distributed to increase the scalability of the network, they are also part of the second tier.

- **Firewall**
  The access to the Database is protected from a firewall to avoid unauthorized accesses to sensible data.

- **Database Server**
  This is the last layer of the architecture. All the data are stored here structured in a relational DBMS.

2.4 Runtime view

2.5 Component interfaces

2.6 Selected architectural styles and patterns

The following architectural patterns are used to build the structure of the system in order to provide all the services of SafeStreets application.

Client-Server Architecture

Client-Server architecture is a computing model that features two roles: a Server that hosts, delivers and manages most of the resources and services, and a client which exploits them.

Motivations

This structure provides several advantages.The Server can be replicated to have more scalability. Accessibility: users can use the services without the need for a specialized device or Client. The security is granted because the server allows only registered user to consult data. Furthermore, it is easy to update, replace, repair, move a server without affecting Clients.

Three-tiered Architecture

This type of architecture is a kind of Client-Server paradigm where three tiers are phisically separated:

- The *presentation tier* provides an interface that allows the users to consult data and result of services executed in a different layer. It is the top-most tier and the only one accessible from the Client.
- The *application tier* executes the logic of the application throw functions that elaborate data. A reverse proxy is needed to handle the Client requests and to balance the workload, the requests are forwarded to the application serves in order to provide the right data.
- The *database tier* includes the data persistence mechanisms and the data access layer that encapsulates the persistence mechanisms and exposes the data.

Motivations

A multi-tier application architecture provides a model with several advantages: developers can create flexible and reusable applications and acquire the option of modifying or adding a specific layer, instead of reworking the entire application.

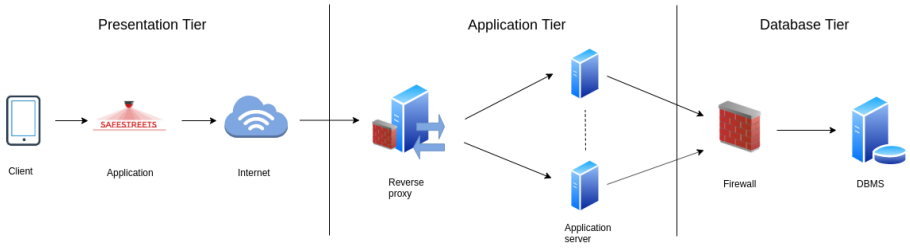

Figure 2.3: Overview of the system.

Thin Client

A thin client is a lightweight computer which does most of the computation in a remote server. The servers takes care of several duties as storage of data and performing calculations.

Motivations

In this case the application is thought for mobile phone which could not have great power of computation. Whit a thin client is sufficient to have internet connection to use the application and not an ultimate generation phone. Furthermore, this architecture semplify the client-side shifting most of the execution to the server.

MVC Design Pattern

Model-View-Controller is a software design pattern particulary usefull to develope user interfaces. Indeed, this pattern is used for the frontend implementation of the application. It divides the program logic in three interconnected elements: *model* that directly manages data and rules of the application, *view* which handles any representation of data, *controller* that accepts input and converts it to commands for the model or view. This is done to separate internal representations of information from the ways information is presented to and accepted from the user.

Motivations

MVC allows full encapsulation of object. This means that each component can be changed without creating issues to other components. Furthermore, with MVC developers are able to work in parallel on different components without blocking another.

Reverse Proxy Design Pattern

A simple proxy act as an interface to refere an object in another machine. The reverse proxy offers a single point of access (with HTTP) to multiple client which want access to several application servers. In practice, it is a wrapper of an object behind the scenes. An extra functionality is also caching of data and security throw a firewall on client-side.

Motivations

It is useful to organize the requests of multiple clients and to save frequent requested informations. It protects also the application servers from external attack.