1. Introduction

1.1 Purpose

This document will go into more detail on architecture and design of the software to be developed. With referencing the requirements present in RASD file, design and the architecture details will be explained in detail. General architecture, components and their interactions, classes, runtime sequences, implementation, integration and testing will be explained in high detail. The order of the sections is as follows:

* Overall Architecture
* Component Design
* Component Interfaces Services
* Runtime Views with Sequence Diagrams
* Deployment View
* Implementation and Testing Plan
* Integration and Testing Plan

1.2 Scope

As described in the RASD document, SafeStreets provides three services but for the purpose of this project we delve into two services. SafeStreets is an application that intends to bring public participation into reducing traffic offences, in particular parking violations. The application allows users to send pictures of violations, including their date, time, and position to system. There are 2 possible services basic service and advanced service.

* for the basic service SafeStreets stores the information provided by users. In particular, when it saves the report of violation the user recognize the plate number of vehicle and stores the retrieved information with the type of violation, which select by users and the name of the street where the violation occurred and also can be retrieved from the geographical position of the violation. In addition, the application allows both end users and authorities to mine the information that has been received, for example by highlighting the streets (or the areas) with the highest frequency of violations, or the vehicles that commit the most violations. Of course, different levels of visibility could be offered to different roles.
* In addition, the municipality (and, in particular, the municipal agent) could offer a service that takes the information about the violations coming from SafeStreets and generates traffic tickets from it. In this case, mechanism should be put in place to ensure that the chain of custody of the information coming from the users is never change, and the information is never altered (e.g., if a manipulation occurs at any point of the image showing the violation, for example to alter the license plate, the application should discard the information). In addition, the information about issued tickets can be used by SafeStreets to build statistics.
  1. Acronyms and Abbreviations
* DD: Design Document.
* RASD: Requirement Analysis and Specification Document.
* API: Application Programming Interface.
* DB: Database.
* TDD: Test Driven Development.

1.4 Document’s Structure

This *DD* project is composed by 6 chapters and appendix:

1. *Introduction* contains the description of the given problem and basic information about this

document in order to provide better understanding of the next chapters.

1. *Architectural Design* contains an overview, high level architecture description, component view, class diagram, deployment view, sequence diagrams, component interfaces description and definitions and design patterns.
2. *User Interface Design* directs the user to the *RASD* file, where the user interfaces are presented.
3. *Requirements Traceability* relates the requirements and goals described in the *RASD* file to components presented in this file.
4. *Implementation, IntegrationandTestingPlandescribestheneedapproachtodevelopthedescribed* problems. Including implementations and testing description and needed integration.
5. *Effort Spent* contains the relation of time spent in the project for each member of the group.

2. Architectural Design

2.1 Overview

A 3-tier architecture is a type of software architecture which is composed of three "tiers" or "layers" of logical computing which we use it for designing the architecture of our application. They are using in applications as a specific type of client-server system. 3-tier architectures provide many benefits for production and development environments by modularizing the user interface, business logic, and data storage layers. Doing so gives greater flexibility to development teams by allowing them to update a specific part of an application independently of the other parts. That is why we choose this method

• **Presentation Tier:** The presentation tier is the front-end layer in the 3-tier system and consists of the user interface. This user interface is often a graphical one accessible through a web browser or web-based application and which displays content and information useful to an end user. This tier is often built on web technologies such as HTML5, JavaScript, CSS, or through other popular web development frameworks, and communicates with other layers through API calls.

• **Application Tier:** The application tier contains the functional business logic which drives an application’s core capabilities. It’s often written in Java, .NET, C#, Python, C++, etc.

**• Data Tier:** The data tier comprises of the database/data storage system and data access layer. Examples of such systems are MySQL, Oracle, PostgreSQL, Microsoft SQL Server, MongoDB, etc. Data is accessed by the application layer via API calls.

**2.2 High level architecture**

A screenshot of a computer

Description automatically generated

Figure 1: High Level Architecture Diagram.

The figure 1 shows the general architecture of the system. Both Registered User and municipal agent nodes are connected to the Internet in order to communicate with the External Server and Application Server nodes. Registered User and municipal agent are registered to the application in order to communicate with the application server. Application Server is the part of the program that encodes the business logic that determine how data can be created, stored, and changed. On the other way it is the logical part of the system. Application server uses the services which External Server provides. Application Server is connected to the Database Server to have access to the data. In the Database server all the data, information, pictures and reports which used in the system will be stored, such as the personal information of Registered User.

2.3 Component view

Figure 2: High Level Components Diagram.

2.4 Deployment View

A close up of a sign

Description automatically generated

Figure 5: Deployment View Diagram.

This view describes the environment into which the system will be deployed, including the dependencies which the system has on its runtime environment which can be seen in the figure 5.

The five-dimensional boxes represent nodes, either software or hardware. Physical nodes should be labeled with the stereotype device, to indicate that it’s a physical device such as a computer or smart phone.

User: This node consists of one node, Smartphone. Smartphone is a device which used by user. Smartphone is a device which is used by the user and communicates directly to the application layer but since the information is critical, a firewall between the Application Server.

Municipality Agent: Agents are able to use web application by their web browser. Agents are able to use external server which includes Google Maps API and AmbulanceService that are connected to the third party. another server

that is connected to third parties is the application server which includes business logic. Consequently, application server sends the data to DB server.

Application server: It has relation with all other nodes of the system. In this node, all the computation of the system will be done, hence this node is logical part of the server. Application server obtains data from the external services of the system and stored information which are in database server and manages data requests.

External server: This node equips the system with extra services which are essential for the application. For instance, maps service are two nodes which are present in the External servers. Between external server and application server we have firewall since the data which shared are really critical for the system. This means that these types of data is really substantial such as the location of the people, so there should not be any access to the data from outside.

Database server: All the data and information which used in the system will be stored in the Database server such as the personal information of users. Between Application Server and DB Server, a firewall will be used to ensure the integrity of the data transfers.

2.7 Selected architectural styles and patterns 2.7.1 Overall Architecture

As it has been explained in high level architecture, application will use three tier architecture ap- proach. This choice will improve the applications maintainability and complement teamwork in high number of people.

Presentation Layer will be the most lightweight tier to ensure that mobile devices and smartwatches which have limited resources will not have any performance issues. Some of the most recent frameworks for both web applications and mobile applications are an excellent choice for this tier. Angular and NativeScript will be used as the main front-end frameworks for both interfaces.

Application layer will handle the core functionalities of this project. Without compromising security in mind, this layer will be able to handle high number of requests and responses with the help of Node.js and Express.js frameworks. With asynchronous task management, the application layer will not require a high specced hardware. This will reduce the maintentance costs significantly as well.

Data Layer is the data storage and data management tier. PostgreSQL database server will be used since it is completely free and this is a big plus for the reduced costs of the project.

Of course the seperation of the layers will introduce performance drawbacks according to the data transfer between layers. But for maintainability and reusability in mind, this layered architecture is a much better choice in the whole development and runtime lifecycle.

**2.7.2 Design Patterns Client-Server Architecture**

This seperation of client and server will improve the performance of the client side. Since all of the business computations will be run on server, client will have a lightweight application. Also this decoupling also improves development process with letting developers work without knowing what is the business logic running on server side.

MVC pattern in Angular

Angular framework introduces a MVC pattern which is very useful for decoupling of the components and request services. Three modules can be used for the development process:

* Components:This is the module which will mostly contain the data which is presente data certain time. These components can also reused and extended according to the needs of the application. TypeScript is used to develop these modules.
* Views: This type of module is responsible for handling the presentation of the data to user. HTML and CSS is used to develop these modules which is mostly related to design for the user interface.
* Services: This modules are handling the request-response cycle of the web application. All data requests from the components are sent to these modules and data transfer is always going through these modules. The design pattern of this module is Observable. These services can be injected to other components or even other services. For example; client side authentication will be done

through authenticaton service and this service will be injected to every service which handles the data requests from the application server.

Node.js and Express.js

Node.js is a cross platform engine which can be developed with JavaScript language. Express.js is a REST server implementation. With these two tools, business logic of the application will be deveoped. Since Node.js has a single non-blocking thread design, asynchronous development is necessary for the performance of the application server. With properly architectured code, this server is able to handle very high numbered requests and responses with respect to a very low-end hardware.

Mobile Application

NativeScript framework will be used for developing both of mobile applications. Since components and services are developed the same way as Angular, only Views have to be coded with NativeScript. This is an excellent use of reusability of the codebase. After the compilation of the application, all of the views are transofrmed to native components of the corresponding operating systems. Native components are the best choice with respect to both performance and user experience.

3. User Interface Design

All of the user interface mockups are present in the RASD file. There are no further detailed mockups needed in this section but for coherence of this document to be an independent from RASD here we depict the main UX design of the application. Here we present some UX diagrams to show the flow which the Customer will follow to navigate inside the application, in accordance with the mockups contained in the RASD. It should also be noted that here we only depict main tabs and pages on the application.

A close up of text on a white background

Description automatically generated

4. Requirements Traceability

The whole design has been thought to guarantee that the system is able to enforce the requirements defined in the RASD (and, as a consequence, to achieve the prefixed goals). Here a mapping between those requirements and the design components in the application server that will ensure their fulfillment in a direct way (some other components that are indirectly needed to enforce some requirements are not mentioned in the list, but their role has been made clear through some comment) is shown:

[R.1]:  Individuals and third parties can register to Data4Help services by providing identifiable information. (Goals G.1)

* Authentication Service

[R.2]: Allow users to report a violation. In specific Giving Access to use camera of his/her smart mobile phone, Providing a form for some additional information, such as the license plate number (it would be better to has this item which user can write license plate number in form), type of violation, details about location and vehicle’s position, and other description. also Give access to users the location (it is possible by employing Map service to find the location of user and violation on the map and sending it to authorities).

(Goal G2.1 - G2.2 - G2.3)

* Report Service

[R.3]:  Allow users to get useful information about general information and news about streets situation and Highlighting and warning most accident potential areas. (goal G.3)

* Statistics service

[R.4]:  Give access the municipality agents the detail of violations. (goal G.4)

* Third-party service

[R.5]: Give access to municipal agent a panel for general announcements. (goal G.5)

* Third-party service

[R.6]: The system must reject the violation report if the place is not exact or if the photo is in poor quality. (goal G.9)

* Report validation service

[R.7]: The system must communicate to the municipality system. (goal G.12)

* Third-party service

[R.8]: The system must be able to show zone with highest rate of violation based on type. (goal G.13)

* Statistics service

[R.9]: The system must be able to show car number with high rate of parking violation. (goal G.14)

* Statistics service

5. Implementation, Integration and Testing Plan

5.1 Implementation Plan

5.1.1 Agile Development Approach

This is a very popular approach for most of the software developed in recent years. Iteratively developing the functionalities of the software helps developers to focus only the important parts of the development lifecycle and improves the efficiency.

Agile approach will complement the development process since required features can be developed in each agile cycle. After each cycle, other requirements and any missing features can be determined and applied in the next cycle.

5.1.2 Server-Side First

The core business logic of the project needs to be implemented at first to ensure that presentation layer of the software will receive the correct representation of the data. Front-end layer will have to make API calls to get the data for presentation hence relevant back-end REST API has to be implemented first. After each completion of a component implementation, front-end part of the software will be implemented as well. This way developers can work simultaneously and reduce the development time significantly.

5.1.3 Top-Down Approach

In the implementation process of the core logic of the software, both Top-down approach will be used in the development process. Report-violation is the core component and other components rely on this component and services it includes. As Top-down approach in mind, Report-violation component will be developed at first to ensure that relying components will have a strong foundation. The following order of services will be implemented:

1. Report-violation
2. Reports validation service
3. Database Service
4. Statistics service
5. Third-party service
6. Authentication Service

Report-violation will be implemented first since most of the services will be using this service. Next report validation has to be developed since it is critical for both performance of the app and achieving desired goals, to have reliable and correct input data from users.

Database service has to be deployed in early stages as well since very important part of the application heavily relies solely on this service and that is statistical services which aim to give insights to both users and third parties.

Authentication Service is left to last since it is the last service to be implemented for individuals and third parties to register and use the services provided.

Below, the figure illustrates the decided approach.

A screenshot of a social media post

Description automatically generated

5.2 Implementation Testing Plan

Test Driven Development (TDD) approach will be used for testing of the implementation and integration of the software. For every service to be implemented, unit tests will be written first to ensure that implementation of the actual services will function properly. This approach also reduces any maintenance costs because of wrong implementation or addition of new features in the future. Also, this approach is excellent for continuous integration process for the software. For every feature to be developed, unit tests will be run before release and this will reduce the bugs which might not be encountered in development process.

For the components which require other components on development time, mock mechanism will be used to test the behavior of the required components, stubs will be used to test the state verification. These methods will reduce the development time since not all of the components will be developed at the same time and we can use mocks or stubs to replace the required components.

In case of any other functionalities are required by the stakeholders or any functionality must be added to ensure the requirements are fulfilled, same testing and implementation strategy will be applied. Firstly, tests of the functionalities will be written, then they will be implemented.

5.3 Integration Plan

5.4 Integration Testing Plan

Same as the implementation testing plan, TDD method will be used for integration of the components and external services. Before any integration of the components or services, integration test will be written to ensure that integration works properly. Even though integration tests consume more time because of the setup of required components and writing the tests for them, it is much safer and efficient in the long time.

For the integration of the external services, there won’t be any extensive testing because these external components have their own internal tests. Very simple integration tests will be written to ensure that we have connection with these external services, and they are responding to the requests made by the internal services.