Software Requirements Specification

Danyil, Kovalchuk

WSU Cpt S 322

Contents

[**1** **INTRODUCTION** 4](#_Toc92625283)

[**1.1** **Purpose** 4](#_Toc92625284)

[**1.2** **Reference Documents** 4](#_Toc92625285)

[**1.3** **Abbreviations and Acronyms** 4](#_Toc92625286)

[**2** **GENERAL DESCRIPTION** 5](#_Toc92625287)

[**2.1** **Product** **Description** 5](#_Toc92625288)

[**2.2** **Users** 5](#_Toc92625289)

[**2.3** **Operating Environment** 5](#_Toc92625290)

[**2.4** **General Constraints, Assumptions, Dependencies, Guidelines** 5](#_Toc92625291)

[**2.5** **Design and Implementation Constraints** 5](#_Toc92625292)

[**3** **Use Cases** 6](#_Toc92625293)

[**3.1** **Use Case Diagram** 6](#_Toc92625294)

[**3.2** **Use Cases** 6](#_Toc92625295)

[3.2.1 **Use Case 1 – Start Game** 6](#_Toc92625296)

[**4** **REQUIREMENTS** 7](#_Toc92625297)

[**4.1** **Functional Requirements** 7](#_Toc92625298)

[4.1.1 Main Menu 7](#_Toc92625299)

[4.1.2 Game Play and Interaction 7](#_Toc92625300)

[4.1.3 Scoring 7](#_Toc92625301)

[4.1.4 Lives – Game End 7](#_Toc92625302)

[4.1.5 Saving High Scores 7](#_Toc92625303)

[4.1.6 Loading High Scores 7](#_Toc92625304)

[**4.2** **Non-Functional Requirements** 8](#_Toc92625305)

[4.2.1 **Performance Requirements** 8](#_Toc92625306)

[4.2.1.1 Flicking 8](#_Toc92625307)

[4.2.1.2 Requirement 2 8](#_Toc92625308)

[4.2.1.3 Requirement 3 8](#_Toc92625309)

[4.2.2 **Usability** 8](#_Toc92625310)

[4.2.2.1 Requirement 1 8](#_Toc92625311)

[4.2.2.2 Requirement 2 8](#_Toc92625312)

[4.2.2.3 Requirement 3 8](#_Toc92625313)

[**4.3** **Other** **Requirements** 8](#_Toc92625314)

[4.3.1.1 Requirement 1 8](#_Toc92625315)

[4.3.1.2 Requirement 2 8](#_Toc92625316)

[4.3.1.3 Requirement 3 8](#_Toc92625317)

# **INTRODUCTION**

## **Purpose**

The purpose of this document is to specify the requirements for Smart Parking project. It has a general description, where you can find information such as purpose of the product, types of the users, targeted operating systems and some constraints.

It also a section for use cases, which from the perspective of the user show to you how the application can and should be used. And finally it has a requirements section, which tells about various requirements for the software.

## **Reference Documents**

* List any references used (e.g. websites, etc.)

## **Abbreviations and Acronyms**

* Admin – Administrator
* Info – Information

# **GENERAL DESCRIPTION**

## **Product** **Description**

The Purpose of this system is to display all the taken and available parking lots in real time. It will use 4 sensors and Bluetooth beacons to accomplish this goal.

This system will also present some statistical information such as the number of cars in parking lots, and what type of a car is parked in a parking spot. It also will have a login screen, and allow administrator to add users and cars to the system.

## **Users**

There are 2 types of users of this app which are:

* + Regular User – Can see what parking lots are available, and some general statistical info.
  + Administrator – Can add and delete new users, cars, and will have access to more statistics than regular user.

This section provides background to the users of the system (from assignment 1)

## **Operating Environment**

Because firebase is cross system platform, there application will be able to run on almost any system. So the system will be targeted for Windows 10/11, Linux and MacOS.

What is the targeted operating system. For example, what types of computers / platforms are targeted? What version of bash? Web servers ?

## **General Constraints, Assumptions, Dependencies, Guidelines**

List any assumptions that you may make, dependencies the system has etc.

* Should run on elec.tricity.wsu.edu
* Should run on a windows computer with internet.
* There will be 2 types of users.
* The map and dimensions of the parking lot should be provided by the server or client.

## **Design and Implementation Constraints**

List any constraints that are imposed on the design, and implementation. What requirements should be considered.

* To be implemented in C#
* Uses Firebase
* Should be implemented using object-oriented programming.
* Should get a data from the provided server.
* The user should have enough memory to be able to create a local database.

# **Use Cases**

## **Use Case Diagram**

Diagram

Description automatically generated

<UML Use Case Diagram>

You should have at least 5 use cases these should be elements in your UML Use Case Diagram

* Login on the Application
* Load data to start application
* Exit the application
* Review user
* Add user,
* Check Available Parking
* Parking statistics

Each use case should be a table and have a title, include the UML diagram

## **Use Cases**

## **Use Case 1 – Login**

|  |  |
| --- | --- |
| Overview | |
| Title | Login |
| Description | The user enters his credentials to login to the application |
| Actors and Interfaces | User, Computer, Local database |
| Initial Status / Preconditions | The user already has an account created by the admin, in local database |
| Steps | |
| 1. User opens the application and is welcomed with the login screen. 2. User is prompted to enter his credentials. 3. User Enters his credentials. 4. The credentials are compared to other credentials in the local database. | |
| Post Condition | |
| The main screen is showed to the user | |
| Alternative Flow | |
| If the user’s credentials are not in the database, he gets an error, and is prompted to enter credentials again | |

Complete the remaining use cases – 1 table per use case

## **Use Case 2 – Load Data to Start Application**

|  |  |
| --- | --- |
| Overview | |
| Title | Load Data |
| Description | The user loads data such as map dimensions, and location of 4 sensors from the server |
| Actors and Interfaces | User, Computer, Server, local database |
| Initial Status / Preconditions | The user logged into the account |
| Steps | |
| 1. User clicks Start Application Button 2. The users are loaded from the local database. 3. Computers sends request to server. 4. Server responses with all necessary data 5. Simulation updates | |
| Post Condition | |
| User can see all occupied and free parking spots, and statistics | |
| Alternative Flow | |
| If connection to the server failed, user gets an error message and is prompted to retry | |

## **Use Case 3 – Review Users**

|  |  |
| --- | --- |
| Overview | |
| Title | Review Users |
| Description | The admin can choose a user, and change his name tag and type of a car |
| Actors and Interfaces | User, Computer, Local database |
| Initial Status / Preconditions | The user is logged into the administrator account |
| Steps | |
| 1. Admin chooses the user from the list. 2. Window changes 3. The admin modifies one of the values such as name, tag and/or type of the car. 4. The admin clicks “Save” button. | |
| Post Condition | |
| User’s information is saved to the database. | |
| Alternative Flow | |
| If the users information already exists in the database admin gets an error message and is prompted to retry. | |

## **Use Case 4 – Add User**

|  |  |
| --- | --- |
| Overview | |
| Title | Add User |
| Description | The administrator adds users to the local data base |
| Actors and Interfaces | Administrator, Computer, Local database |
| Initial Status / Preconditions | The user is logged into the administrator account |
| Steps | |
| 1. Admin clicks “Add User” button. 2. Screen changes to add user screen. 3. Admin enters information such as name, tag number, and then chooses the car icon from the list. 4. Admin hits “add user button.” 5. The user is added to the local database | |
| Post Condition | |
| User can login to the application, and you can see car associated with him on the simulation | |
| Alternative Flow | |
| If the user is already in the database, the admin is prompted to retry and gets an error message. | |

## **Use Case 5– Check Available Parking**

|  |  |
| --- | --- |
| Overview | |
| Title | Check Available Parking |
| Description | The user looks at the parking spots and finds ones that are not taken |
| Actors and Interfaces | User, Computer, Local database, server |
| Initial Status / Preconditions | The user is logged into his account |
| Steps | |
| 1. The server sends data to client computer, about each user’s car. 2. The client use this data to triangulate positions of each car. 3. Software displays empty and occupied spots. 4. The user observes a screen and sees the empty slots. | |
| Post Condition | |
| User knows which slots are empty and which are taken | |
| Alternative Flow | |
| If the server is not responding, the user gets an error message and is prompted to retry. | |

# **REQUIREMENTS**

## **Functional Requirements**

Complete a list of requirements. This should be a structured format

## Login Interface

* The software SHOULD do y
  + Should be easy to use
  + Should look nice
* The software MUST do x
  + Must prompt the user for the username
  + Must ask for the password

## Main menu

* The software SHOULD do y
  + Should give user access to all the functions of the software, such as profile controls, and statistic tabs.
* The software MUST do x
  + The main menu must include the simulation tab

## User add/ remove interface

* The software SHOULD do y
  + Should be simple
  + Should be self-explanatory
* The software MUST do x
  + Must allow only admin to do changes to other users.
  + Must allow users to do changes to their profiles.
  + Must allow admin and users to change:
    - Username
    - Tag
    - Car type
  + Must allow admin to remove other users.

## Parking Statistics

* The software SHOULD do y
  + Should show the number of occupied spots.
  + Show the number of different cars parked in each spot.
* The software MUST do x
  + Must show the number of available spots.

## Parking Interface Representation

* The software SHOULD do y
  + The interface should look appealing
  + Should show the type of the car parked in the slot
* The software MUST do x
  + Must show all the parking spots.
  + Must show which spots are occupied and which are not

## Update parking data in real time

* The software SHOULD do y
  + Should be reliable.
  + Should update the cars position even if it is not in the slot.
* The software MUST do x
  + Must update data every 500ms
  + Must update each car in the parking lot

## Loading data for parking representation

* The software SHOULD do y
  + Get data about parking lot dimensions from the server.
* The software MUST do x
  + The software must load each user from the local database.

## **Non-Functional Requirements**

## **Performance Requirements**

## Parking accuracy Representation (free/occupied)

## The sensor must be in the 70% if the center of the parking slot, to be considered as occupied.

## Need to use the closest 3 sensors for the triangulation.

## **Real time delay**

## Data Should be updated every 500ms.

## The application should feel “snappy”.

## Transitions between screen should not be longer than 1s.

## **Usability**

## The interfaces must be appealing and simple.

## The interfaces should not have too many buttons.

## The functions and buttons should be combines into groups or menus.

## **Other** **Requirements**

## The parking slots should be numbered.

## There should be 3-4 different screens.

## The project must be submitted through GitHub.