**CptS322**

**Software Design Document**

**Format and Content Guide**

Software Design

Document

for

Desktop Smart Parking Simulator System

Name: Caitlyn Powers

Project: Desktop Smart Parking Simulator System

Last Updated: [Date and time of this document version goes here]

# Table of Contents

[Update the Table of Contents]

[Table of Contents 3](#_Toc133754109)

[Document Revision History 5](#_Toc133754110)

[List of Figures 6](#_Toc133754111)

[1. INTRODUCTION 7](#_Toc133754112)

[1.1 Purpose 7](#_Toc133754113)

[1.2 Scope 7](#_Toc133754114)

[1.3 Overview Simply 7](#_Toc133754115)

[1.4 Reference Material 7](#_Toc133754116)

[1.5 Definitions and Acronyms 8](#_Toc133754117)

[1.6 GitHub Link 8](#_Toc133754118)

[2. SYSTEM OVERVIEW 9](#_Toc133754119)

[2.1 Story Board and Wireframes 9](#_Toc133754120)

[3. Finite State Automata 11](#_Toc133754121)

[3.1 App States 11](#_Toc133754122)

[3.1.1 Waiting for data update 11](#_Toc133754123)

[3.1.2 Login State 11](#_Toc133754124)

[3.1.3 Admin Settings State 11](#_Toc133754125)

[3.1.4 User Settings State 11](#_Toc133754126)

[3.1.5 Set Up Display 11](#_Toc133754127)

[3.2 State Diagram 12](#_Toc133754128)

[4. DATA DESIGN 13](#_Toc133754129)

[4.1 Data Description 13](#_Toc133754130)

[4.2 Data Dictionary 13](#_Toc133754131)

[4.3 COMPONENT DESIGN 16](#_Toc133754132)

[5. SYSTEM ARCHITECTURE 21](#_Toc133754133)

[5.1 Architectural Design 21](#_Toc133754134)

[6. Structural Diagrams 24](#_Toc133754135)

[6.1 Decomposition Description 24](#_Toc133754136)

[7. BEHAVIOURAL DIAGRAMS 28](#_Toc133754137)

[7.1 UML Sequence Diagrams 28](#_Toc133754138)

[7.1.1 App login and logout 28](#_Toc133754139)

[7.1.2 parking slot status update 28](#_Toc133754140)

[7.2 App Flow Chart 29](#_Toc133754141)

# Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision Number | Revision Date | Description | Rationale |
| 1.0 | 4/30 | Document submitted with final code | Initial document |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# List of Figures

Figure # Title Page #

2.1 -1 Wireframes 10

3.2‑1 State Diagram 12

5.1 1 Call and return diagram 22

5.1 2 data centered diagram 23

6.1‑1 Data classes 25

6.1‑2: User Class Diagram 26

6.1 3 Interface diagram 27

7.1‑1 App login and logout sequence diagram 28

7.1‑2 Parking lot update sequence diagram 29

7.2‑1 App flow chart 30

## INTRODUCTION

## Purpose

This software design document describes the architecture and system design of Desktop Smart Parking Simulator System. The purpose of this document is to allow both parties to clarify how the system will work, ensure that both parties understand how the system works, and to streamline maintenance and updates of the software in the future.

## Scope

The purpose of Desktop Smart Parking Simulator System is to track and monitor what parking spots are occupied by cars, in order to use the space more efficiently. This product includes 4 main pages, which are the login screen, the parking lot display screen, the settings screen and the administrator settings screen. The parking lot display screen shows a real time simulation of the parking spaces contained within the area made by the four sensors. The information for this is retrieved from the clients firebase database which gathers data from sensors. In order for this to work their must be beacons in the cars, that the sensors can sense.

The settings screen is for regular users and contains the change password feature. The administrator settings screen contains the change password feature, as well as the ability to manipulate and view users, cars, and beacons. The administrator settings screen is only available to the administrator. The benefits of this software is the ability to manage the databases of users, cars and beacons in the same place that you can view the status of the parking lot.

## Overview Simply

This document will include a storyboard of the interface wireframes showing the flow between the different wireframes and what event caused the change in wireframe. It will also include the description of the different states of the system, and a state diagram. The next section contains information on the data flow of the system, so what is passed between objects, a dictionary of the different pieces of data, and a overview of the different components and their functionalities. The last two sections are class diagrams and sequence diagrams as well as a focused state diagram for the firebase data update. Those two sections will also include any descriptions necessary to understand the diagrams that are not in other parts of the document.

## Reference Material

This section is optional.

List any documents, if any, which were used as sources of information for the test plan.

## Definitions and Acronyms

* SPSS – Desktop Parking Simulator System

## GitHub Link

<https://github.com/CaitlynCagaaa/cpts322>

## SYSTEM OVERVIEW

## Story Board and Wireframes

Diagram

Description automatically generated

Figure 2.1‑ Wireframes

Press settings button as a admin user

Press logon button with valid user credentials



Press setting button as a normal usercredentials

## Finite State Automata

## App States

## Waiting for data update

The waiting for data update state, is a state on the parking lot display screen where the system is doing nothing other than waiting for any changes in the data for the beacons on firebase.

## Login State

The system is on the login screen waiting for the user to enter their credentials and press the login button.

## Admin Settings State

The system is displaying lists of the users and cars, as well as a button back to the parking lot display. The system is also displaying textboxes and buttons to add, and remove users, cars and beacons, to change the password of the current user and to modify the connection between cars and beacons.

## User Settings State

The system is displaying a screen that has input text boxes and button for changing password. It also has a button back to the display page.

## Set Up Display

The system is setting up the parking lot display, so it is retrieving all the relevant information, drawing the parking lot, and determining the initial positions of the beacons.

## State Diagram

Diagram

Description automatically generated

Figure 3.2‑1 State Diagram

## DATA DESIGN

## Data Description

Explain how the information domain of your system is transformed into data

structures. Describe how the major data or system entities are stored, processed and

organized. List any databases or data storage items.

* Storage database
  + Contains Beacon information, Car information, and User information, may also contain Parking Lot information and Sensor information. IT should be noted that this database is not connected to the database so the distances of the beacons may not be accurate.
* Location database
  + Contains Beacon information, Parking Lot information and Sensor information, this database should be connected to the sensors the distance information is accurate.

The information persistent data in the system is stored in two different firebase real time databases, one database holds the information on users, cars and beacons, this is called the storage database in this document. The storage database may also have parking lot and sensor information for testing purposes. The other real time database is connected to the sensors and has the real time location of the beacons, as well as the parking lot information. This database is called the location database in this document.

The system reads in information and updates information in the database as is relevant. So when someone attempts to login the database reads in user information form the storage database and stores it in a instance of the Users class. During the parking lot setup state the program reads in the sensor information, the parking lot information and the initial beacon information, and stores it as appropriate in one instance of the Beacons class, the Parking class and the Sensors class. It gets this information form the location database. Then if you go the admin settings page the system reads in the information on beacons, cars and users from the storage database and stores it into the appropriate instance of the Users class, the Beacons class, and the CarList class. It will then take the instance of the user class and display a table using a instance of the ListView WinForms class to display each users username and privilege level. It will also do something similar for the instance of CarList and the instance of Beacons, except they are displayed in the same table. If you use one of the buttons to modify the instances of CarList, Beacons or Users, the system will update the firebase storage database.

## Data Dictionary

Alphabetically list the system entities or major data along with their types and

descriptions. If you provided a functional description in Section 3.2, list all the

functions and function parameters. If you provided an OO description, list the objects

and its attributes, methods and method parameters.

* Beacon
  + Attributes
    - double D1
    - double D2
    - double D3
    - double D4
    - long Id
    - long Time
    - string connected
      * The plate number of the connected car.
    - int inside
      * The parking space the beacon is inside if any
  + Methods
    - Point trilateratetion(Sensors location)
* Beacons
  + Attributes
    - int Total
      * The total number of beacons
    - List<Beacon> data
      * A dynamic list of beacons
        + It is a List<T> instead of an array since arrays are static in c#.
  + Methods
    - int addBeacon(long id)
    - int removeBeacon(int id, CarList list)
    - int modify(long id1, long id2, CarList list)
* Car
  + Attributes
    - long connected
      * The Id of the connected beacon
    - string plate
    - string owner
    - string color
* CarList
  + Attributes
    - List<Car> data
    - int Total
  + Methods
    - int addCars(string owner, string color, string plate )
    - long removeCars(string plate, Beacons list)
    - int connect(Beacons list, int id, string plate)
    - int modify(Beacons list, string plate1, string plate2 )
* Parking
  + Attributes
    - List<Slot> data
    - int Total
  + Methods
    - int checkSlot(Point pt, long ID )
* Point
  + Attributes
    - double x
    - double y
* Sensor
  + Attributes
    - Point position
* Sensors
  + Attributes
    - List<Sensor> data
    - int Total
* Slot
  + Attributes
    - List<Point> position
    - int Total
    - double lowX
    - double lowY
    - double highX
    - double highY
    - long spot
      * The Id of the beacon in the parking space
    - long prev
      * The Id of the previous beacon in the space if it needs to be kept.
        + Is only here in case two beacons are somehow in the same spot.
    - char availability
      * The state of the slot, ‘o’=occupied, ‘a’ =available ‘d’=delinquent and ‘b’ = boundary issue.
  + Methods
    - void changeColor(int inside70, long ID)
    - void clear(long ID)
* User
  + Attributes
    - string Password
    - string UserName
    - int level
      * refers to the level of privilege the user has, and is 0 for regular user and 1 for admin.
  + Methods
    - int changePassword(string old, string newP)
* Users
  + Attributes
    - int Total
    - List<User> data
  + Methods
    - int validate(string name, string psw)
    - int addUser(string name, string psw)
    - int removeUser(string name)

## COMPONENT DESIGN

In this section, we take a closer look at what each component does in a more

systematic way. If you gave a functional description in section 3.2, provide a

summary of your algorithm for each function listed in 3.2 in procedural description

language (PDL) or pseudo code. If you gave an OO description, summarize each

object member function for all the objects listed in 3.2 in PDL or pseudocode.

Describe any local data when necessary.

* Point trilateratetion(Sensors location)
  + Create instance of Point pt
  + if any of the distances is the sensors is negative
    - return (-1,-1) to indicate beacon is not in parking lot
  + use trilateration algorithm using all four sensors
    - since it is four sensors instead of three use it twice with 2 separate sets of variables
  + average together the two calculations to get the position of the beacon
  + return the position
* int addBeacon(long id)
  + loop over beacons list to see if beacon of id already exists
    - return -1 if it does
  + create new beacon zeroing out distances and connected attributes and add it to list if id doesn’t already exist
* int removeBeacon(int id, CarList list)
  + create a int return varible and set it to -1
  + loop over beacon list to find beacon with id
    - change int return to 0
    - if that beacon is connected to a car
      * loop over list parameter to find the car
        + disconnect the car form the beacon
        + change int return to -2
    - remove beacon from list
    - return int return variable
  + return int return variable ;
* int modify(long id1, long id2, CarList list)
  + loop over beacon list to find the beacon with id1
    - if found
      * loop over beacon list to find id2
        + if found swap their connected attribute
        + loop over car list to find the car that was connected to beacon with id1

change cars connected to id2

* + - * + loop over car list to find car that was connected to beacon with id2

change cars connected to id1

* + - * + return 0
      * return -1
  + return -2
* int addCars(string owner, string color, string plate )
  + loop over cars list to see if car of plate already exists
    - return -1 if it does
  + create new car with owner, color and plate attributes equal to the the parameters, make connected -1 and add it to list if car of plate doesn’t already exist
* long removeCars(string plate, Beacons list)
  + create a int return varible and set it to -1
  + loop over carn list to find car with plate
    - change int return to 0
    - if that car is connected to a beacon
      * loop over list parameter to find the beacon
        + disconnect the car from the beacon
        + change int return to -2
    - remove car from list
    - return int return variable
  + return int return variable
* int connect(Beacons list, int id, string plate)
  + create a int check variable and set to -1
  + loop over car list
    - if car plate equals parameter plate
      * change check to 1
      * if the car isn’t already connected to a beacon
        + change connected to parameter id
      * else
        + return -3 which means the car or beacon are already connected to something
    - else if car’s connected equals id
      * return -3
  + if check equals -1 which means car doesn’t exist
    - return -1
  + change check to -1
  + loop over beacon list parameter
    - if beacon of id found
      * check =1
      * connected of beacon equals plate
      * return 0
  + loop over car list
    - if car is found
      * disconnect car from beacon because apparently beacon doesn’t exist
  + return -2
* int modify(Beacons list, string plate1, string plate2 )
  + loop over car list to find the car with plate1
    - if found
      * loop over car list to find car with plate 2
        + if found swap their connected attribute
        + loop over beacon list to find the beacon that was connected to car with plate1

change beacons connected to plate2

* + - * + loop over beacon list to find the beacon that was connected to car with plate2

change beacons connected to plate1

* + - * + return 0
      * return -1
  + return -2
* int checkSlot(Point pt, long ID )
  + create slot number and set to -1
  + create counter varible and set to 1
  + loop over list of slots
    - if the point pt is inside of the inner 70% of the slot
      * use slots changeColor method to set it to slot to occupied
        + give 1 as inside70 and the ID
      * set slots spot to ID
      * slotnum=i
    - else if the pt is inside the slot but is to far in either direction of the x value
      * use slots changeColor method to set it to slot to boundary issue
      * change slots spot id to ID
      * change the slot next to it that is closest to the beacon to also have a boundary issue
      * set slot num to -2
    - else if slot spot or slot prev equals the ID
      * use the slots clear method giving it ID
    - increment i
  + return slotnum
* void changeColor(int inside70, long ID)
  + if the beacon was inside the inner 70% of the slots area
    - if either the same beacon was in the spot or no beacon was in the spot
      * change availability to o for occupied
    - else there are two beacons in the spot
      * change the avilaiblity to d for delinquent
      * set prev to spot and set spot to ID
  + else the beacon wanst in the inner 70%
    - change availability to boundary issue
    - set prev =spot
    - set spot =ID
* void clear(long ID)
  + if availability was delinquent
    - change availability to occupied
      * if ID is prev
        + change prev to -1
      * else
        + change spot to prev
        + change prev to -1
      * return
  + else
    - change availability to available
    - set spot and prev to -1
* int changePassword(string old, string newP)
  + if old parameter equals the password attribute
    - change password to newP parameter
    - return 1
  + return -1 otherwise
* int validate(string name, string psw)
  + loop over user list
    - if user with UserName name and Password psw
      * return user level
  + return -1
* int addUser(string name, string psw)
  + loop over user list to see if user of name already exists
    - return -1 if it does
  + create new normal user with UserName, and Password attributes equal to the the parameters, and add it to the user list
* int removeUser(string name)
  + loop over user list
    - if users UserName is name and the user is a admin
      * return -2 since cant delete admin
    - if UserName is name but is regular user
      * remove user form the list
      * return 0
  + return -1 if user doesn’t exist

## ****SYSTEM ARCHITECTURE****

## Architectural Design

In this program there are 3 interface pages the login interface, the parking lot interface and the settings interface. The seetings page will depedning on the privelege level of the user either display everything or only display what is necessary to change the users passoword and go back to the Parking Lot interface. The login interfaces purpose is to provide a place for the user to login and to validate their credentials. It does this by retrieving the user infirmation from the stroage database, storing that information in a instance of a users object and using the validate method in users to see if the credentials are valid. If they are valid the login interface goes to the parking lot interface.

The parking lot interface’s purpose is to show a real time display of the parking lot. It does this by creating the layout of the parking lot display and retireving all the information stored in the locations database and stores it in instances of Beacons, Parking, and Sensors. It will after retrieving the data trilaterate the beacons location using the beacon trilateration method, then give that information to the Parking checkSlot method. Which finds the slot that the beacon is in if any and uses the slots methods to update the avialibility. The Parking lot interface will then use an internal repaint method to update the parking lot display. It will do the trilateration and further steps every time Beacon information changes. The software can also at any time move to the settings interface, by clicking the settings button.

The purpose of the settting interface is to display the inforamtion in the storage database and allow for its modifiaction. If the current user is a normal user it will not display anything and only allow for the user to change their passoword, which means reading in the user inforamtion from the stroage database and updating that information as well as using the changePassword method in user. When the user is an admin user the settings page will allow for a host of feutures explained in the requirments document. Overall the admin setting will retrieve the infromation on cars, beacons, and users from the storage database and sotre them in thea instance of the correlating class . It will then display this information in tables. If the user tries to add, remove or swap any of the beacons, cars and users the interface will use the respective method in each of the Beacons, CarList and Users classes. If the user is trying ot do the initial conenction of the cars and beacons the interface will use the connect method in CarList. If any of the modifications succeed then the settigns interface will update the information in the storge database. If the parking lot display button is pressed the settings interface will close and give control back to the Parking lot interface.

The architecure of the system can be easily modeled using the call and return style and the data centered style. It should be noted that the datacentered style has two databases, and that the location database should only have inforamtion read form it by the clent software while the sensors write to it. The storage database has information written and read to it by the SPSS software.

Diagram

Description automatically generated

Figure 5.1 call and return diagram

Diagram

Description automatically generated

Figure 5.1 data centered diagram

## Structural Diagrams

## Decomposition Description

Graphical user interface, application

Description automatically generated

Figure 6.1‑1 data class

A picture containing arrow

Description automatically generated

Figure 6.1‑2: User Class Diagram

It should be noted that a lot of the buttons labels and textboxes were cut as well as some methods that had a cosmetic purpose were cut from the settings interface on the interface diagram.

Graphical user interface, text, application

Description automatically generated

Figure 6.1 Interface diagram

## BEHAVIOURAL DIAGRAMS

## UML Sequence Diagrams

## App login and logout

Create a UML Sequence Diagram that captures the Events for login, going into the main page, and logout

Diagram

Description automatically generated

Figure 7.1‑1 app login and logout sequence diagram

## parking slot status update

Create a UML Sequence Diagram that captures receive the data from firebase, make the calculations and change the status of a parking slot.

Diagram

Description automatically generated

Figure 7.1‑ parking lot update sequence diagram

\*changeColor and clear can happen multiple times depending on the prior state of the parking lot. It should also be noted that onchildchange is continuously waiting for beacons that have changed so the location database is constantly sending updates which start the trilateration.

## App Flow Chart

Diagram

Description automatically generated

Figure 7.2‑1 app flow chart