**CSE 550 HW2**

**Software Design Document (SDD)**

**Team 8**

**[Cameron Vincent, Charles Weiss, Makayla White, Stone Barrett, Lauren Mikula, Karthik Malyala]**

SDD Version 1.2

April 24, 2022

This Software Design Document was prepared and provided as a deliverable for the University of Louisville, Software Engineering Class, CSE 550, for Spring of 2022, by Group 8 in CSE 550. This document is based on the IEEE Standard 1016-2009, IEEE Recommended Practice for Software Design Descriptions.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

XXXX,

Project Manager

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

XXXX,

Repository Expert

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

XXXX,

Project Leader

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

XXXX,

Technical Leader

**Change History**

| **Revision** | **Date** | **Author** | **Section/Pages Affected** | **Notes** |
| --- | --- | --- | --- | --- |
| Draft 1: 1.0 | February 16, 2022 | Cameron Vincent, Makayla White | All initial sections | Initial creation and framework for design document |
| Draft 2 (Submission HW 2): 1.1 | February 24, 2022 | Stone Barrett,  Lauren Mikula,  Karthik Malyala | Cleanup/organization initial sections |  |
| Draft 3 (Final Version): 1.2 | April 26, 2022 | Cameron Vincent, Stone Barrett, Karthik Malyala, Lauren Mikula, Makayla White, Charlie Weiss | Cleanup and additions or revisions to previous sections | Made some corrections and changes that occurred throughout the project lifecycle |

**Table of Contents**

**Change History …………………………………………………………………………………..3**

**Table of Contents ……………………………………………………………………………..…4**

**1. Introduction ………………………………………………………………………………...…5**

1.1 Purpose ……………………………………………………………………………………...5

1.2 Scope ………………………………………………………………………………………..5

1.3 Definitions and Acronyms ………………………………………………………………….5

1.4 Summary ……………………………………………………………………………………5

**2. References ……………………………………………………………………………...……...7**

**3. Decomposition Description …………………………………………………………………..8**

3.1 Feature 1:Portable & Efficient Hardware Setup ……………………………………….…...9

3.2 Feature 2: Parking Vacancy Detection using Computer Vision …………………..…...…9

3.3 Feature 3: Transfer Vision Data into Data Structure……………………………………9

3.4 Feature 4: Database Storage & Cloud Setup …………………………………………..9

3.5 Feature 5: Delivery of Data onto Web App …………………………………………..11

**4. Interface Description ………………………………………………………..………………12**

**5. Detailed Design ………………………………………………………………………………15**

1. **Introduction**

**1.1 Purpose**

This document outlines the Software Design Document Specifications as part of the design plan for creating the *ParkSense* software system. This document also seeks to expand on the functionality described by the features in the *ParkSense* Software Requirements Specification (SRS) document. Each feature covered will describe the functionality of *ParkSense* and also describe what classes, attributes, etc. to be implemented.

**1.2 Scope**

This SDD will take the features outlined in the *ParkSense* SRS document and expand upon them with some more details, pseudo code, and diagrams. The main topics that will be discussed in this SDD include the class and data diagrams that will detail the software systems,some of the functions, and data that is needed for the intended use of the *ParkSense* software product.

**1.3 Definitions and Acronyms**

SDD: Software Development Document

SRS: Software Requirement Document

DFD: Data-Flow Diagram

**1.4 Summary**

In short, the purpose of the *ParkSense* Software Design Document is to further detail some of the required features so that developers can determine how the software system should be constructed and what needs to be included for the overall system to work. This document will help developers in finding some of the specifications and expectations for what they will be working on and will also allow people other than developers to get a basic understanding of some of the *ParkSense* details.

1. **References**

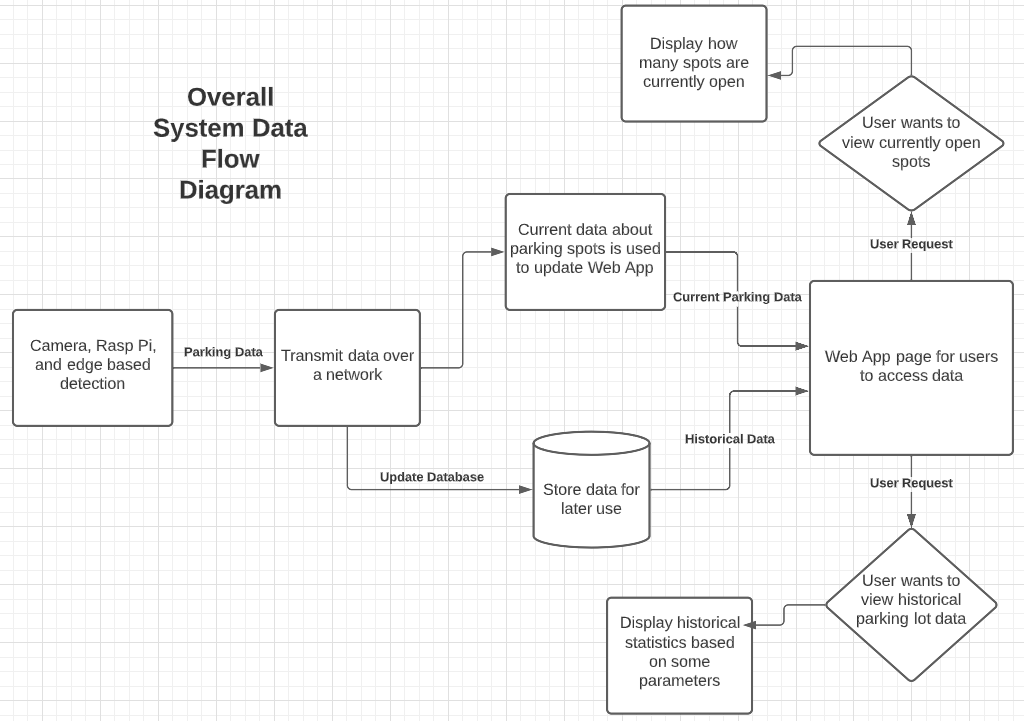
* IEEE Standard for Information Technology - Systems Design - Software Design Descriptions, IEEE Std 1016-2009

**3. Decomposition Description**

This section of the SDD will focus on decomposing multiple features for *ParkSense* that are outlined in the SRS document into data flow diagrams and also go into details like what sections of the overall software package will be responsible for. This section will also include some pseudocode for parts of the software system and lay out some basic design principles.

Below is a simple overarching DFD for the entire *ParkSense* system, showing how data will travel and be used by the system:

**Figure 1: *ParkSense* overarching Data Flow Diagram**

****

The decomposition section will break down the features outlined in the *ParkSense* SRS to give more developer oriented information.

**3.1 Feature 1: Portable & Efficient Hardware Setup**

Given that the Raspberry Pi 3 is a microcomputer that fits in a user’s palm, its power-packed portability feature enables us to mount the technology wherever needed without having to worry about sufficient power or mounting issues. The external webcam can be of any type that allows for clear pixel data for processing and it is not restricted to any specialized camera. The software can be handled efficiently on the Pi without having to install any extra dependencies. Python is one of the most popular programming languages that is enabled on the Pi and OpenCV is the top supporting library that enables for object detection and analysis on platforms as small as a Pi. The additional libraries, NumPy and Pandas, will be utilized to export the gathered data in the form of a 2D array data structure for data extraction and further analysis of historical trends in a parking lot.

**3.2 Feature 2: Parking Vacancy Detection using Computer Vision**

The second part of this project involves the implementation of Computer Vision using OpenCV on the Raspberry Pi 3+ and a 1080P external webcam. OpenCV will be the main driving library for this project as it will enable us to process the image frames gathered by the external camera/webcam. To implement edge detection, we will be mainly focusing on the Canny Algorithm to analyze pixel density in a given slot after applying a Gaussian Blur in grayscale for efficient pixel reading. In the grand scheme of this project, if a parking space has a high pixel density with a lot of edges being present, this signifies that the spot has been occupied. On the contrary, if there is a minimal to no pixel density, this means that the spot is vacant.

**3.3 Feature 3: Transfer Vision Data into Data Structure**

After retrieving the vision data from the OpenCV Edge Detection results, it is then transformed into a Pandas Dataframe in the form of Numpy Arrays for easier parsing and data analysis on the database end. Pandas and Numpy are two of the most popular Python libraries used for parsing data into various sections. While Numpy assists in giving each timestamp data its own list/array of components, Pandas enables data to be transformed into a Dataframe. This dataframe can easily be translated to a 2D array using its Pandas function for data transfer onto the database that is used.

**3.4 Feature 4: Database Storage & Cloud Setup**

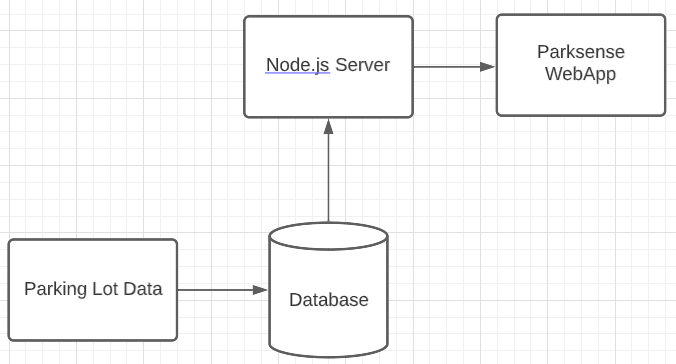
The database component of *ParkSense* will primarily be used to store specific historical data of the parking lots so that users will be able to have useful information about parking lots over time such as how many parking spots are typically available at certain times and days. The data used for this project will be sent across devices from the data structures using a server. This server will be hosted on Amazon Web Services. Specifically, the server will run an instance of Linux that will run multiple instances of different services. One service will be MongoDB. The MongoDB server will be hosted on this Linux server and will host the database for our system. MongoDB has everything needed for remote clients to create database entries on it using standard APIs. Data will be sent over HTTP to and from the server. The following ***Table 1*** lists the SRS requirements dealing with the database portion of the software product that will be expanded upon.

**Table 1: Related SRS Database Requirements**

| **SRS Requirement #** | **Requirement Description** |
| --- | --- |
| **3.4.1** | **Data will be received and translated into useful information** |
| **3.4.2** | **Database will update at regular intervals** |
| **3.4.3** | **Database will be accessed to provide historical data in the Web App** |
| **3.4.4** | **Old database data will be deleted to keep memory needed lower** |

**3.4.1 Database DFD for *ParkSense* Web App**

**Figure 2: *ParkSense* Web App DFD**

****

**3.4.2 DataBase Tables and Table Attributes**

In production *ParkSense* will have multiple tables set up in a similar way to a Python dictionary since we are using MongoDB. Each table will correspond to separate parking lots if multiple parking lots are connected to the system. If requested by the user to include multiple parking lots when viewing historical data, it will be possible by joining the different tables together. The basic layout of each table will remain similar to the below example:

**Table 2: *ParkSense* Database table: {ParkingLotName}**

**\* datetime is used as a key and is automatically filled with the current dateTime when an entry is created. Size is the size of the parking lot and is entered in 2 dimensions. Spots is where the values are stored and updated for if a particular parking spot is empty or full. Name is where the name of the parking lot is entered \*  
  
\_id:ObjectId(“GUID”)  
date:dateTime**

**Size: Array**

**0: 2  
 1: 2**

**…**

**N: 2**

**Spots: Array**

**0: Array**

**0: “empty”**

**1: “empty”**

**1: Array**

**0: “empty”**

**1: “full”**

**…**

**N: Array**

**0: “full”**

**1: “full”**

**Name: “demolot”**

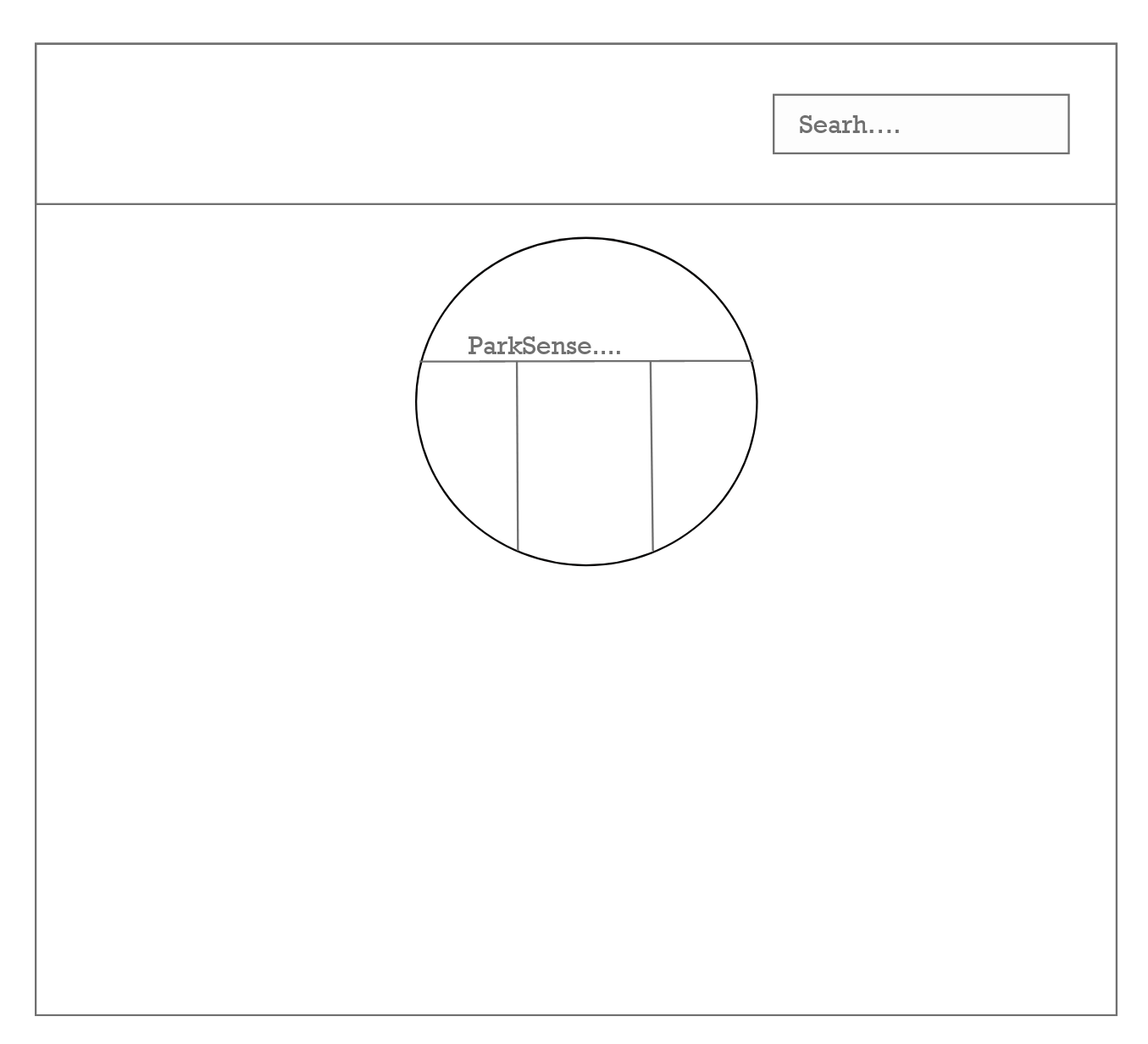
**3.5 Feature 5: Delivery of Data onto Web App**

Since the database will be hosted on a server, which will be connected to the internet, it will be fairly simple to access the database from any web connected device. The database will host the status of each parking spot across multiple time periods. This will allow the web application to grab the most recent parking spot information and display it in a user-friendly format. Further, the web app will be able to query the parking spot status across a timeframe, which will allow the web application to suggest to the user future predictions on parking spot availability.

**4. Interface Description**

This section of the SDD seeks to lay out the fundamentals for the *ParkSense* Web Application UI that users will see and interact with. The main content covered here will include mockups of the UI and explanations of the user interactable elements and information that is displayed.

***4.1 Search Feature - GUI Interface***

****

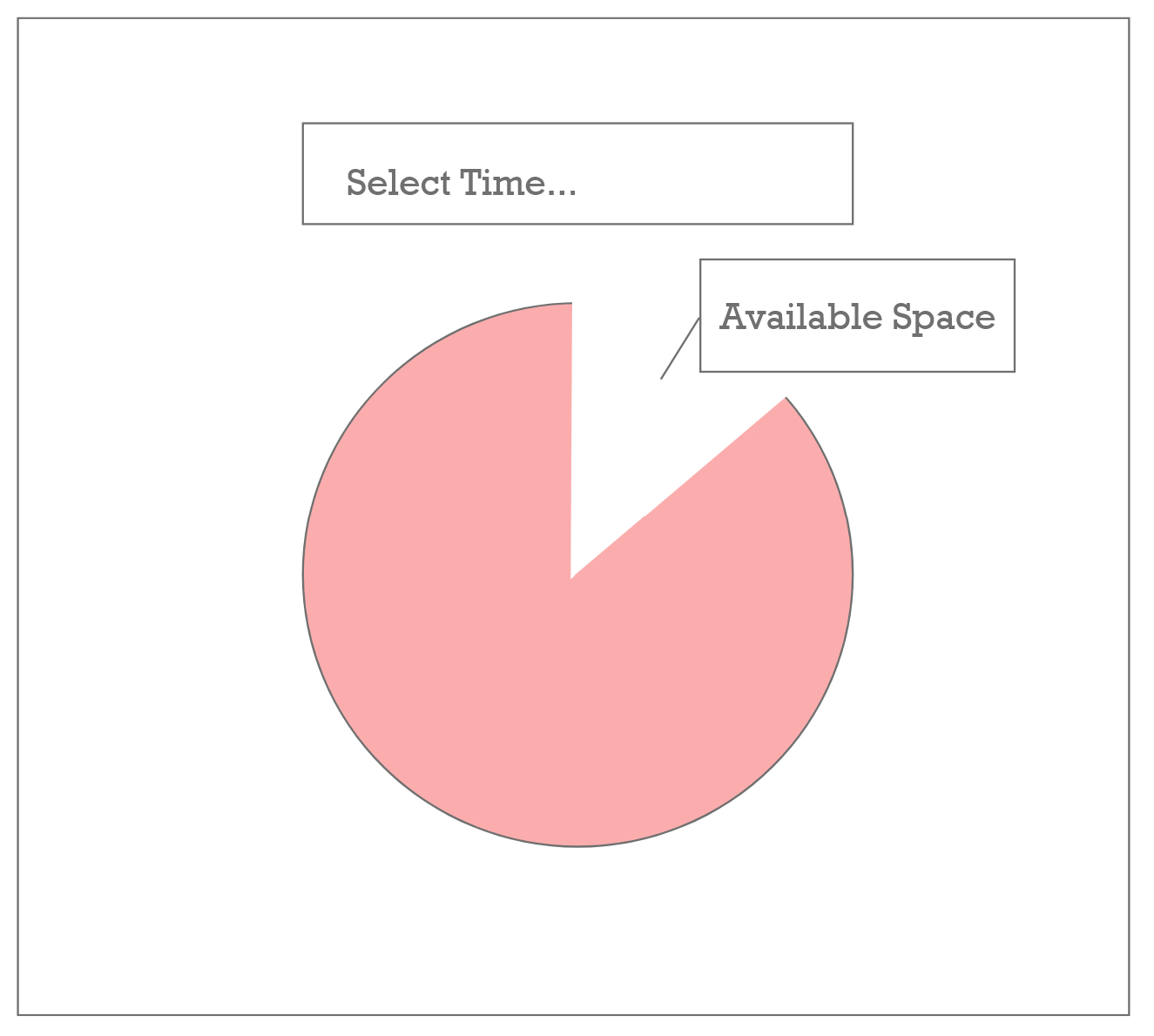
**Figure 3: Mockup of home page**

This page will display the *Parksense* logo and will give the user the option to search for what parking lot they want to use.

****

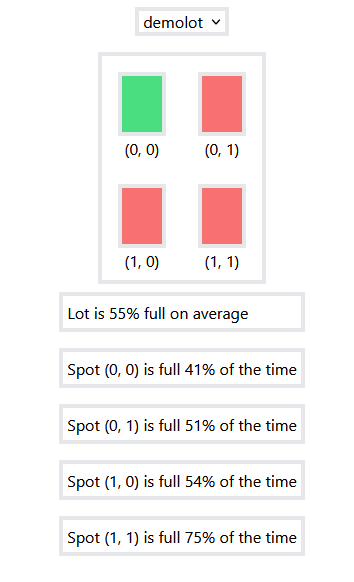
**Figure 4: Mockup of lot search page**

This page has a drop down search bar that allows the user to select which parking lot they would like to view.

****

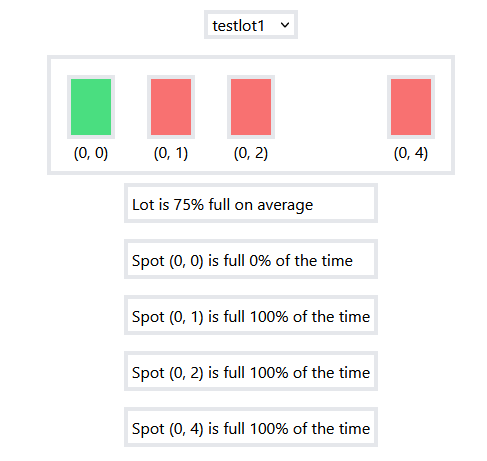
**Figure 5: Mockup of statistics display page**

This page will update in real time with the capacity statistics of the selected parking lot.

****

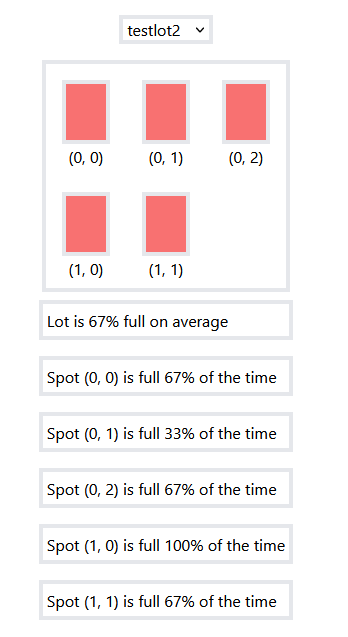
**Figure 6: Final UI Design**

This figure demonstrates the final design of the user interface. The Design includes the name of the parking lot, a general layout of the parking lot, colored blocks to represent each spot's availability, and some general historical data about each spot and the lot in total. The final design only includes one main “Home” page which displays all necessary data. This page changes depending on which lot the user selects as demonstrated in Figures 7 and 8 below.

****

**Figure 7: UI For testlot1**

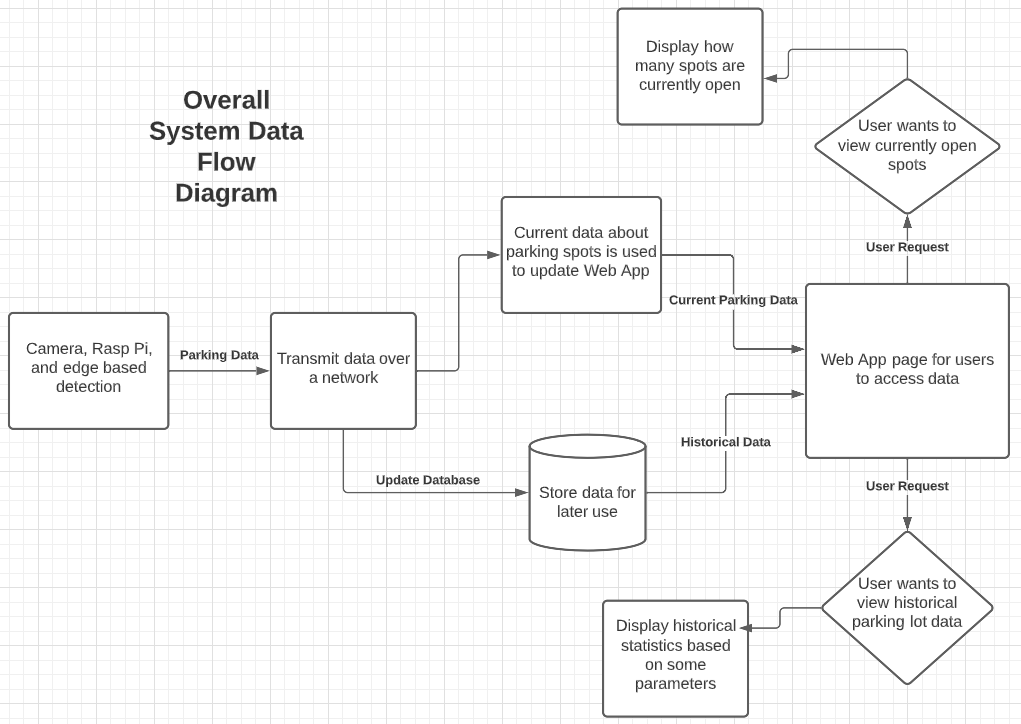
This figure demonstrates how the UI adapts to each lot’s particular layout.

****

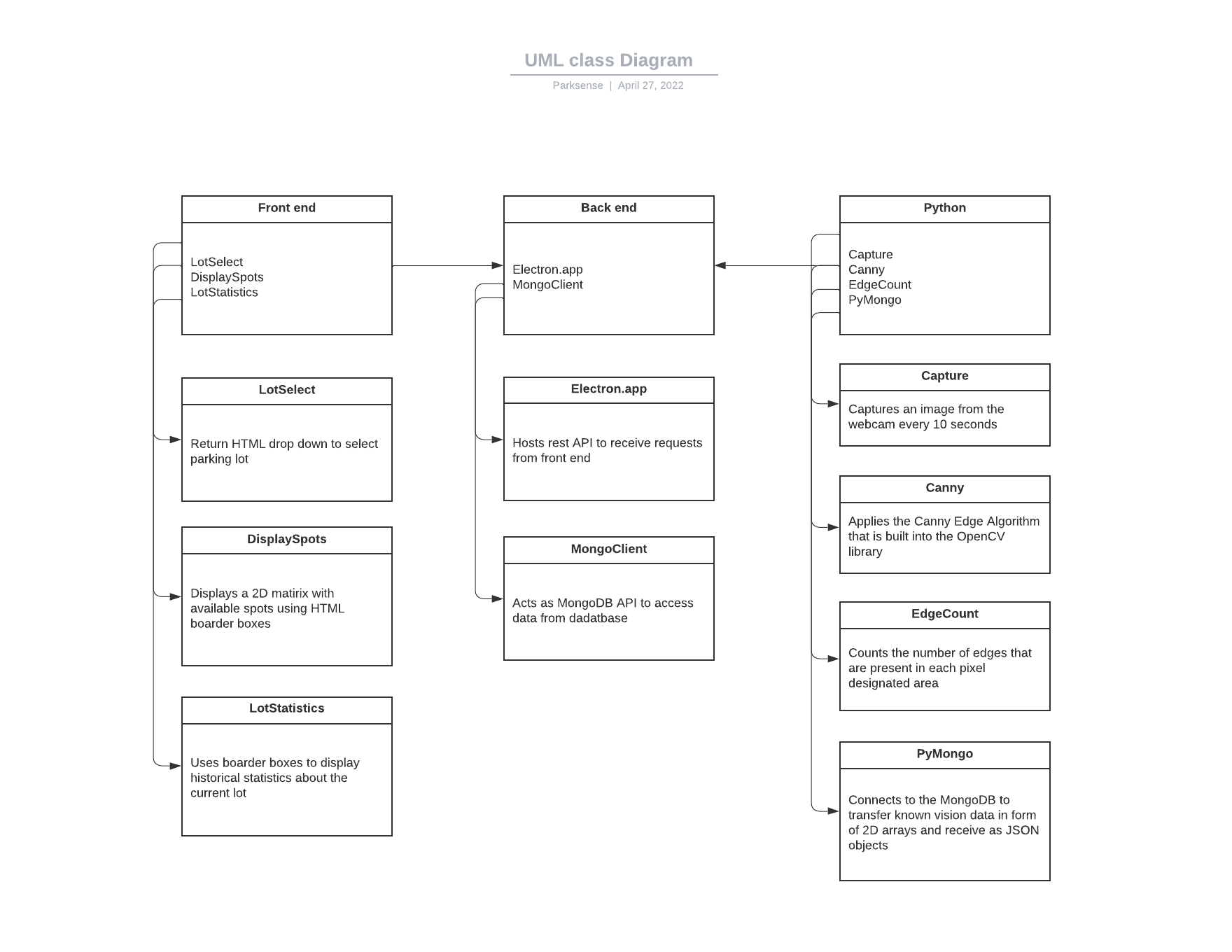
**Figure 8: UI For testlot2**

This figure also demonstrates the adaptability of the UI.

**5. Detailed Design**

****

**6. Class diagram**

****