Parking Database Management System report

Qingbo Lai

Dec, 2018

Contents

1. Project Description

Link to project requirement:

https://github.com/qingbol/CUParkit/blob/master/ProjectRequirment/CpSc%2046206620%20Project%20Requirements.pdf

1.1. Problem Statement

On Clemson's campus, parking lots often fill up, which causes employees and students to weave in and out of parking lots in search of an empty spot. This can waste a lot of time and require the employees or students to try multiple parking lots. Another issue is that students have odd parking rules such as being able to use employee parking during certain hours and only being able to park in certain student parking lots designated for their parking pass' s classification. Visitors who come to campus on busy days may also have difficulty finding parking. Also, it would be nice for a visitor to view how much time he has left in his parking spot. If the visitor is not near his vehicle, but needs to extend his allotted parking time, there should be a way to do so without having to revisit the station near his vehicle.

1.2. Project Goal

An app that could communicate which lots are available at the current time for the person's parking pass restrictions, which spots (if any) are available in those lots, and how much time a vehicle has left in a visitor spot would be useful on a daily basis. Additionally, enabling a visitor to remotely extend his parking time would be beneficial.

With the time limit of one semester in mind, this project made simplifying assumptions to the above problem statement and does not contain all of the features and functionality that it ideally should.

1.3. Survey of Related Work

Finding a vacant parking spot can be a challenging and frustrating task sometimes. Many applications have been developed for and much research has been reported on parking manage systems. Sites such as Parkopedia and ParkingSpotter aim to help people find an available parking spot.

Two research papers were investigated: "Spot and Park: Parking management system" and "A Public Parking Management System for Zurich".

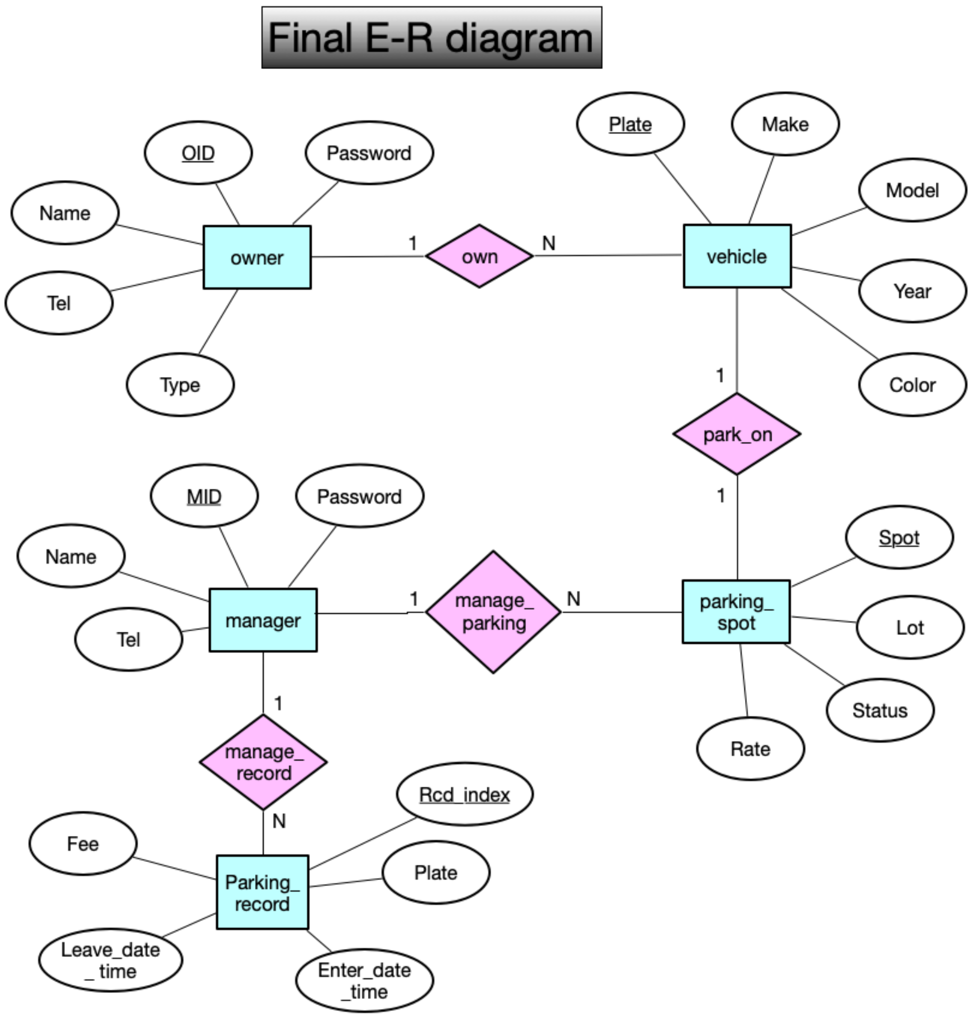
"Spot and Park: Parking management system" described an application called Spot and Park that aims to solve the problem of finding parking. Spot and Park software architecture consists of 3 components: a user facing mobile optimized website, an admin facing website, and a MySQL database. What is new in this system is that it can let individuals publish their parking data easily and share with their users. CUParkit used some ideas from this project in the development of its code.

Many jobs such as adding, removing or suspending parking spaces are performed manually without the support of an information system. In order to solve this problem, "A Public Parking Management System for Zurich" proposes an information system to support Parking Authorities with the management of public on-street parking spaces. The system include three components: the Parking Space Marking System, the Parking Area Management System and the Parking Inspector' s controller application, these components increase the performance and e□ciency of the parking management process.

CUParkit seems to fill a need that has not been addressed yet.

2. Design and Implementation of Database

Thinking through a basic design for the data resulted in an entity-relationship scheme shown in the following diagram.



An employee, student, or visitor can be classified as an owner of a vehicle. An owner can own multiple vehicles.

A vehicle parks on a parking spot and is assumed to be owned by one owner.

A manager (human or an automated system with sensors) is assigned to a set of parking spots (called a parking lot). Each manager observes his set of parking spots for the event of a vehicle parking or leaving a parking spot. When either of these events occurs, the manager records the event in a parking record.

Careful thought was given to the selection of the keys of the relations.

An owner ID (“OID”) can be something like a student ID (C12895968) or a unique username (Bob1).

A vehicle’s plate number (“Plate”) is not necessarily unique because there could be a duplicate one in another state. For now, our database assumes it is unique, though.

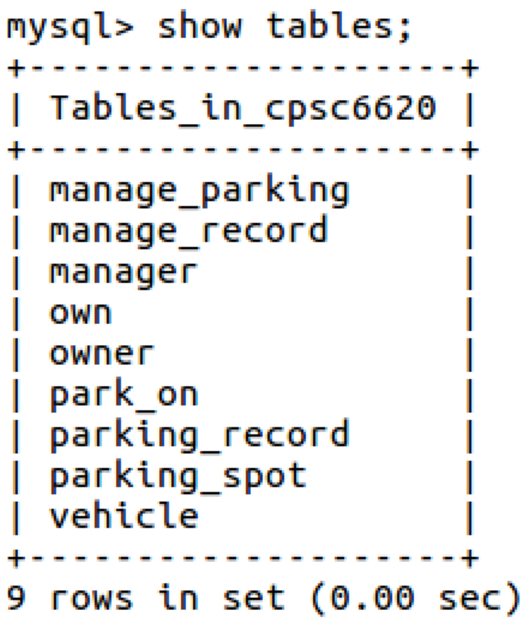
The parking spot's ID ("Spot") is unique because it has an identifier for which lot it belongs to. For example, a spot labelled as "E104" would be the 104th spot in the Employee lot. Since we assumed there is only one employee lot, one visitor lot, and one student lot, this spot is guaranteed to be unique.

The manager ID could be automatically incremented it is assigned to each manager and can be a short number or something like "M100" .

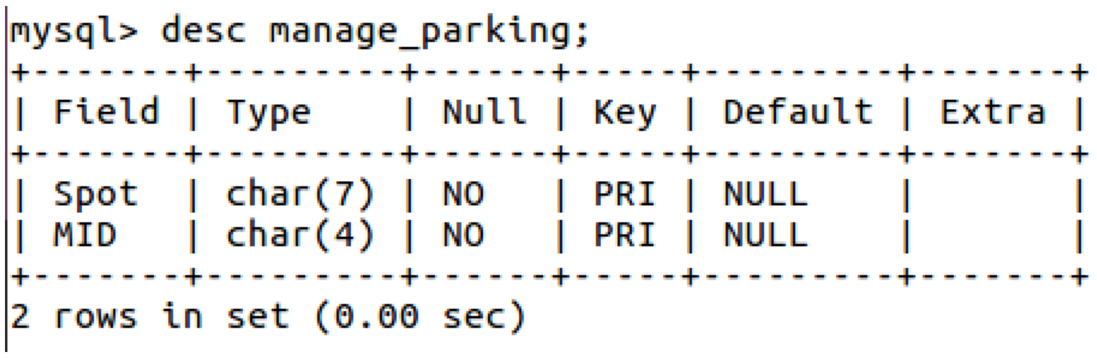
Since the same car could park in a spot, leave it, and come back to that same spot, the vehicle cannot be used to as a key for the record of which vehicle parked in which spots. Instead, an automatically incremented record index ("Rcd_index") is used.

The relations are stored in a MySQL database with the following structures. The MySQL database is hosted on Clemson's buffet server.

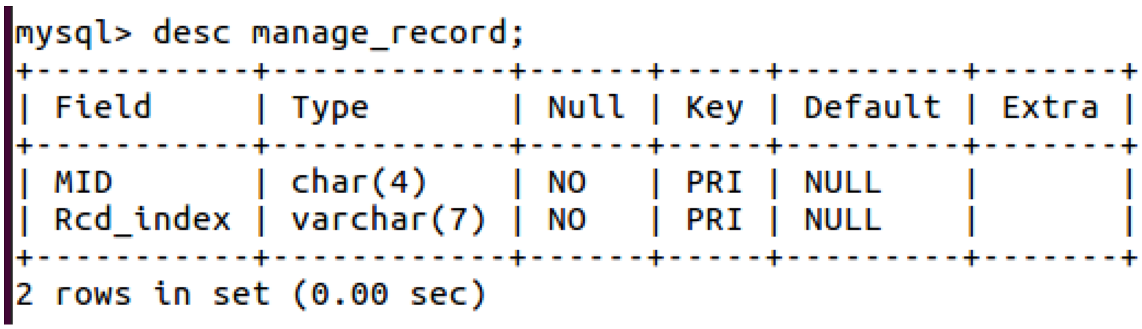
2.1. Tables structure in mysql server:



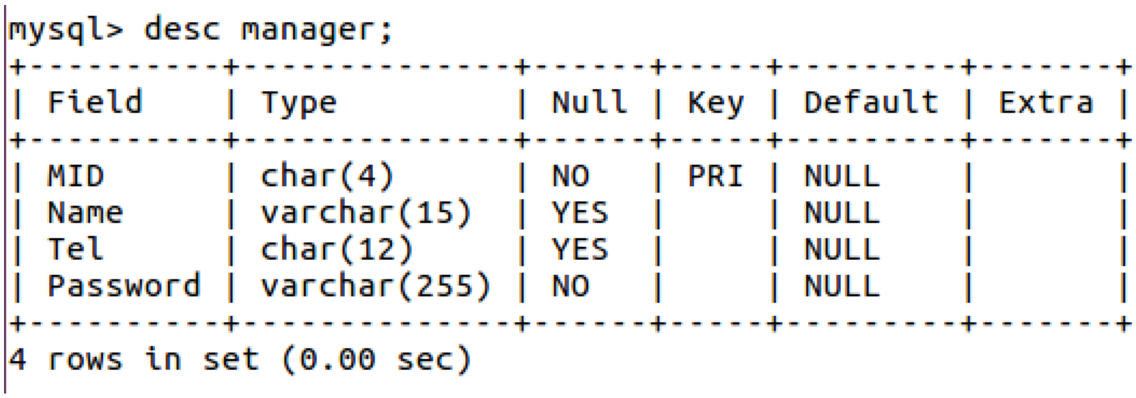
```
mysql> show tables;
+--------------------+
| Tables_in_cpsc6620 |
+--------------------+
| manage_parking     |
| manage_record      |
| manager            |
| own                |
| owner              |
| park_on            |
| parking_record     |
| parking_spot       |
| vehicle            |
+--------------------+
9 rows in set (0.00 sec)
```



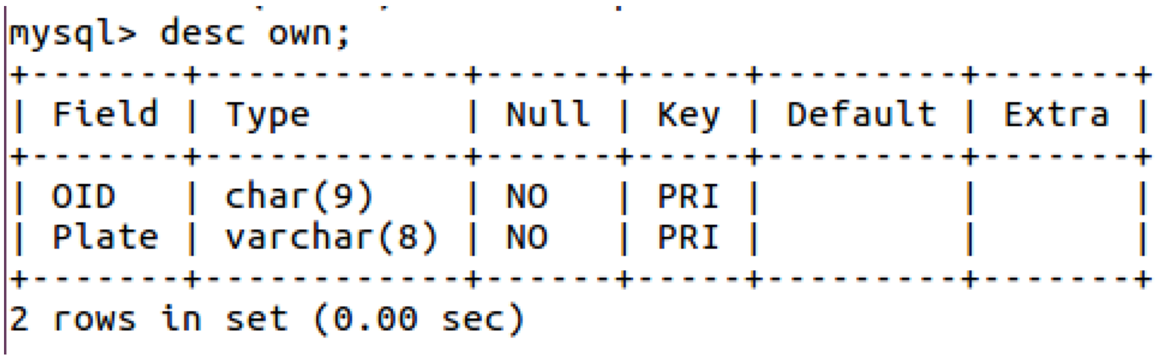
```
mysql> desc manage_parking;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| Spot  | char(7) | NO   | PRI | NULL    |       |
| MID   | char(4) | NO   | PRI | NULL    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```



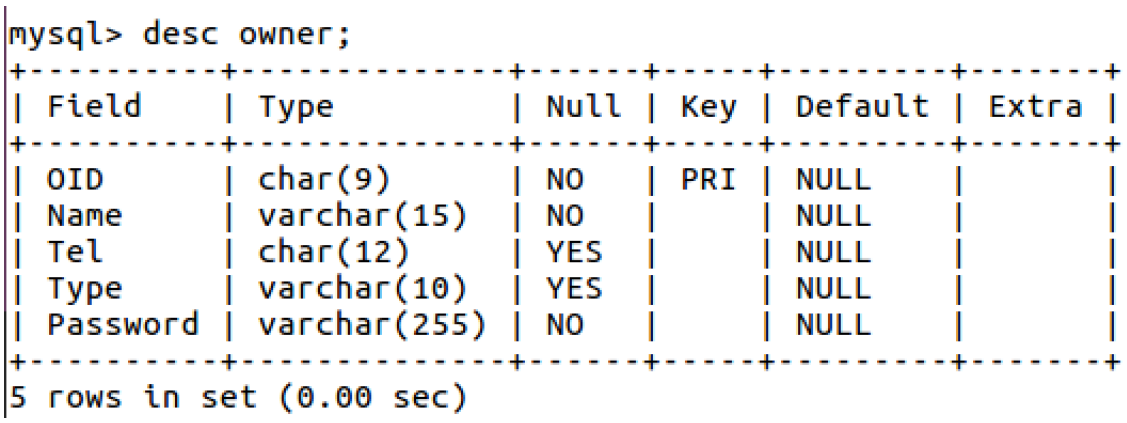
```
mysql> desc manage_record;
+-----------+------------+------+-----+---------+-------+
| Field     | Type       | Null | Key | Default | Extra |
+-----------+------------+------+-----+---------+-------+
| MID       | char(4)    | NO   | PRI | NULL    |       |
| Rcd_index | varchar(7) | NO   | PRI | NULL    |       |
+-----------+------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```



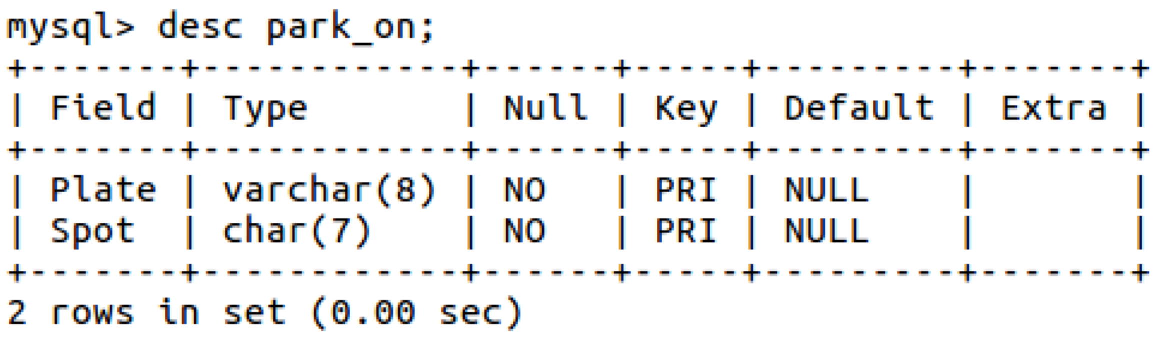
```
mysql> desc manager;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| MID      | char(4)      | NO   | PRI | NULL    |       |
| Name     | varchar(15)  | YES  |     | NULL    |       |
| Tel      | char(12)     | YES  |     | NULL    |       |
| Password | varchar(255) | NO   |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```



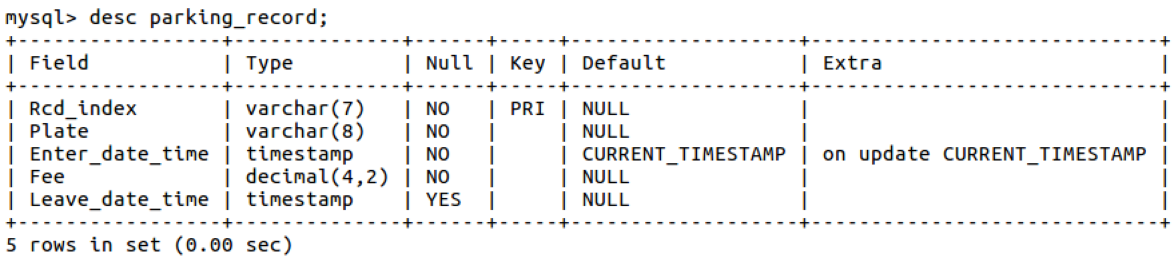
```
mysql> desc own;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| OID   | char(9)    | NO   | PRI |         |       |
| Plate | varchar(8) | NO   | PRI |         |       |
+-------+------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```



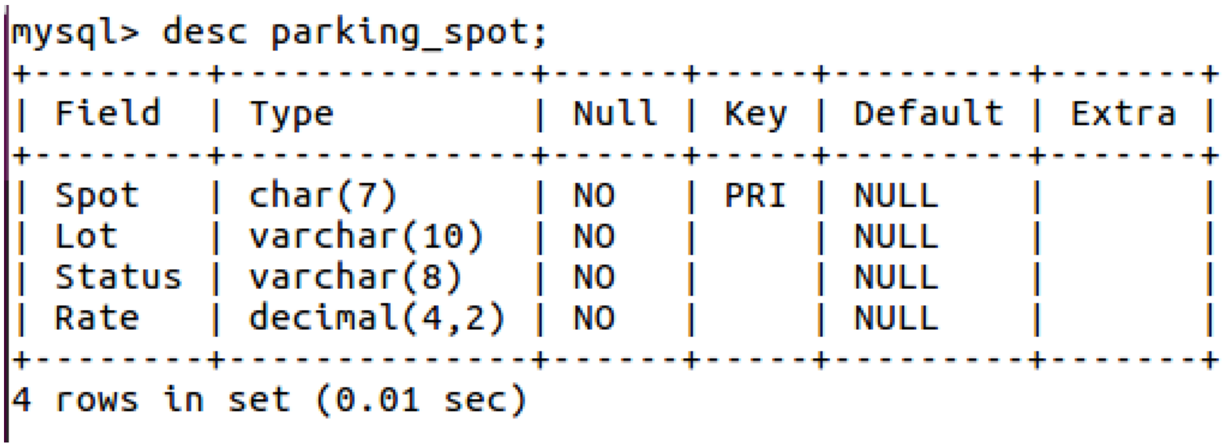
```
mysql> desc owner;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| OID      | char(9)      | NO   | PRI | NULL    |       |
| Name     | varchar(15)  | NO   |     | NULL    |       |
| Tel      | char(12)     | YES  |     | NULL    |       |
| Type     | varchar(10)  | YES  |     | NULL    |       |
| Password | varchar(255) | NO   |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```



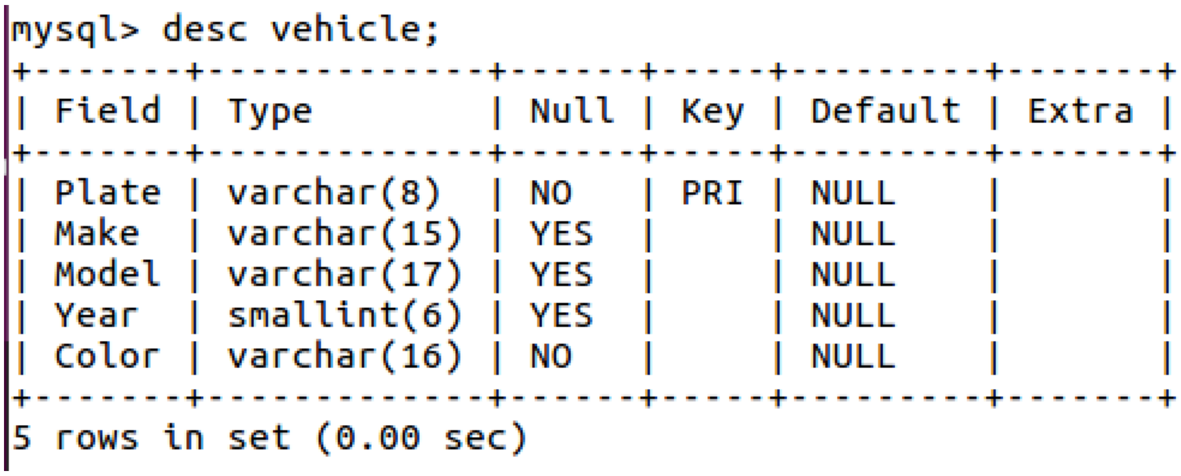
```
mysql> desc park_on;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| Plate | varchar(8) | NO   | PRI | NULL    |       |
| Spot  | char(7)    | NO   | PRI | NULL    |       |
+-------+------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```



```
mysql> desc parking_record;
+-----------------+--------------+------+-----+-------------------+-----------------------------+
| Field           | Type         | Null | Key | Default           | Extra                       |
+-----------------+--------------+------+-----+-------------------+-----------------------------+
| Rcd_index       | varchar(7)   | NO   | PRI | NULL              |                             |
| Plate           | varchar(8)   | NO   |     | NULL              |                             |
| Enter_date_time | timestamp    | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
| Fee             | decimal(4,2) | NO   |     | NULL              |                             |
| Leave_date_time | timestamp    | YES  |     | NULL              |                             |
+-----------------+--------------+------+-----+-------------------+-----------------------------+
5 rows in set (0.00 sec)
```



```
mysql> desc parking_spot;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| Spot   | char(7)      | NO   | PRI | NULL    |       |
| Lot    | varchar(10)  | NO   |     | NULL    |       |
| Status | varchar(8)   | NO   |     | NULL    |       |
| Rate   | decimal(4,2) | NO   |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

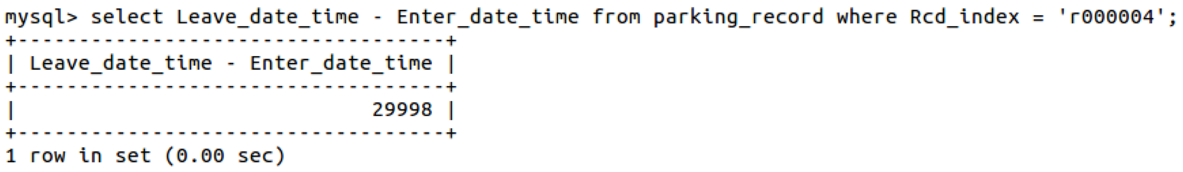


```
mysql> desc vehicle;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| Plate | varchar(8)  | NO   | PRI | NULL    |       |
| Make  | varchar(15) | YES  |     | NULL    |       |
| Model | varchar(17) | YES  |     | NULL    |       |
| Year  | smallint(6) | YES  |     | NULL    |       |
| Color | varchar(16) | NO   |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

2.2. Some MySQL queries

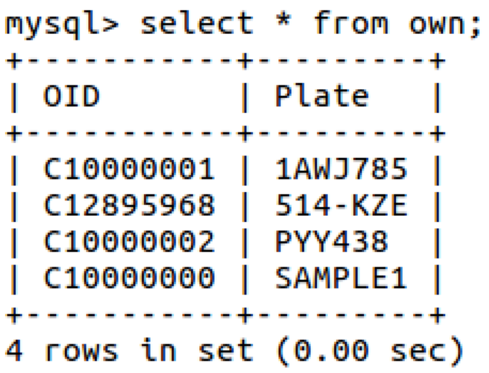
The following MySQL queries were developed. Some of them have not been implemented in the web portal yet. The section discussing what is implemented so far identifies the specific queries that are implemented.

• Calculate the parking time(in seconds).



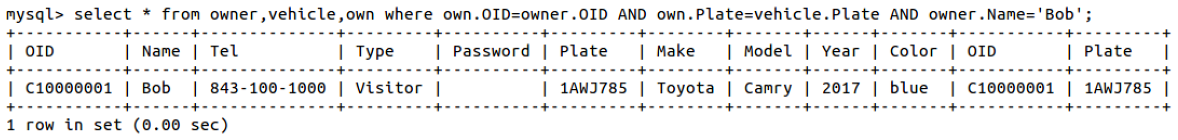
```
mysql> select Leave_date_time - Enter_date_time from parking_record where Rcd_index = 'r000004';
+-----------------------------------+
| Leave_date_time - Enter_date_time |
+-----------------------------------+
|                             29998 |
+-----------------------------------+
1 row in set (0.00 sec)
```

• List all the information of vehicle which park on the parking lot.



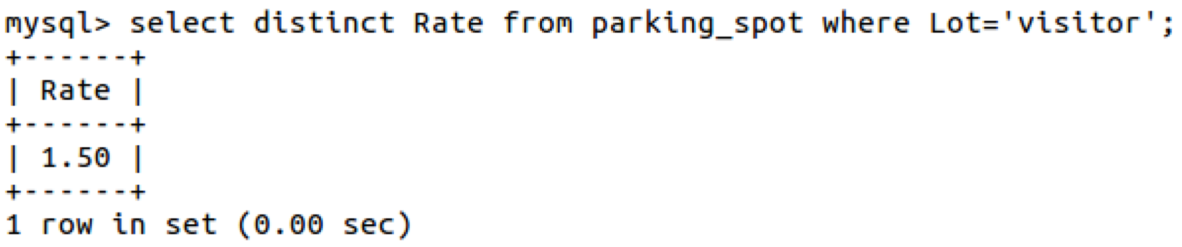
```
mysql> select * from own;
+-----------+---------+
| OID       | Plate   |
+-----------+---------+
| C10000001 | 1AWJ785 |
| C12895968 | 514-KZE |
| C10000002 | PYY438  |
| C10000000 | SAMPLE1 |
+-----------+---------+
4 rows in set (0.00 sec)
```

• List all the vehicles which own by Bob.



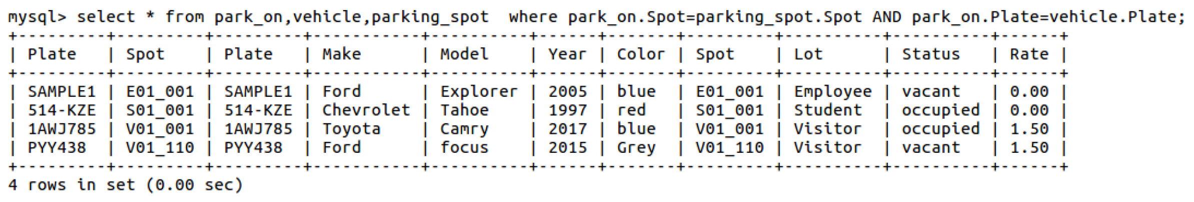
```
mysql> select * from owner,vehicle,own where own.OID=owner.OID AND own.Plate=vehicle.Plate AND owner.Name='Bob';
+-----------+------+--------------+---------+----------+---------+--------+-------+------+-------+-----------+---------+
| OID       | Name | Tel          | Type    | Password | Plate   | Make   | Model | Year | Color | OID       | Plate   |
+-----------+------+--------------+---------+----------+---------+--------+-------+------+-------+-----------+---------+
| C10000001 | Bob  | 843-100-1000 | Visitor |          | 1AWJ785 | Toyota | Camry | 2017 | blue  | C10000001 | 1AWJ785 |
+-----------+------+--------------+---------+----------+---------+--------+-------+------+-------+-----------+---------+
1 row in set (0.00 sec)
```

• List the charge rate of visitor parking lot.



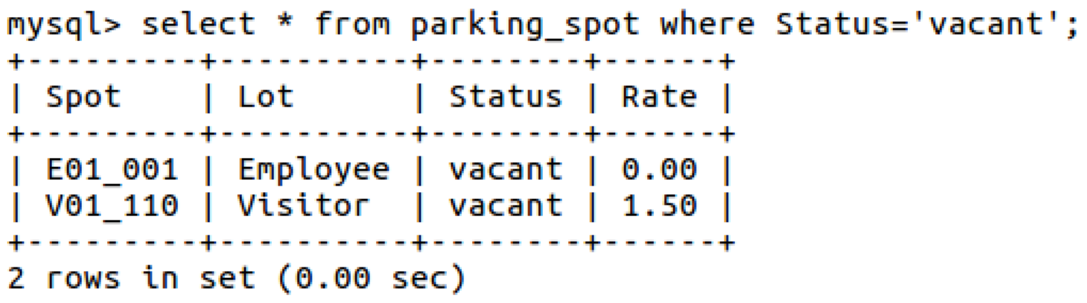
```
mysql> select distinct Rate from parking_spot where Lot='visitor';
+------+
| Rate |
+------+
| 1.50 |
+------+
1 row in set (0.00 sec)
```

• List all the available parking spots.



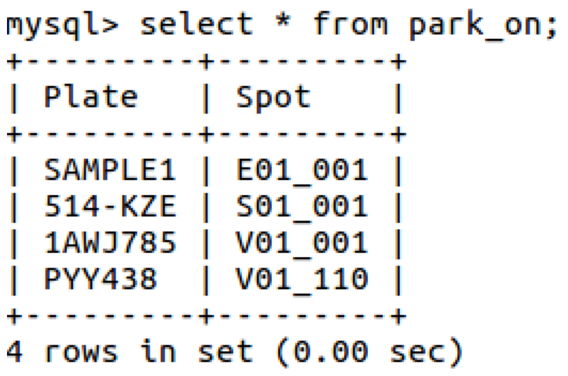
```
mysql> select * from park_on,vehicle,parking_spot  where park_on.Spot=parking_spot.Spot AND park_on.Plate=vehicle.Plate;
+---------+---------+---------+-----------+----------+------+-------+---------+----------+----------+------+
| Plate   | Spot    | Plate   | Make      | Model    | Year | Color | Spot    | Lot      | Status   | Rate |
+---------+---------+---------+-----------+----------+------+-------+---------+----------+----------+------+
| SAMPLE1 | E01_001 | SAMPLE1 | Ford      | Explorer | 2005 | blue  | E01_001 | Employee | vacant   | 0.00 |
| 514-KZE | S01_001 | 514-KZE | Chevrolet | Tahoe    | 1997 | red   | S01_001 | Student  | occupied | 0.00 |
| 1AWJ785 | V01_001 | 1AWJ785 | Toyota    | Camry    | 2017 | blue  | V01_001 | Visitor  | occupied | 1.50 |
| PYY438  | V01_110 | PYY438  | Ford      | focus    | 2015 | Grey  | V01_110 | Visitor  | vacant   | 1.50 |
+---------+---------+---------+-----------+----------+------+-------+---------+----------+----------+------+
4 rows in set (0.00 sec)
```

• List the quantity of the available parking spots.



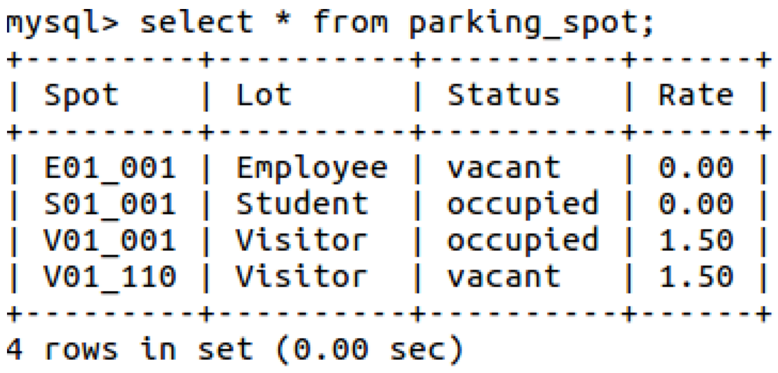
```
mysql> select * from parking_spot where Status='vacant';
+---------+----------+--------+------+
| Spot    | Lot      | Status | Rate |
+---------+----------+--------+------+
| E01_001 | Employee | vacant | 0.00 |
| V01_110 | Visitor  | vacant | 1.50 |
+---------+----------+--------+------+
2 rows in set (0.00 sec)
```

• List the vehicles's plate and occupied spot number



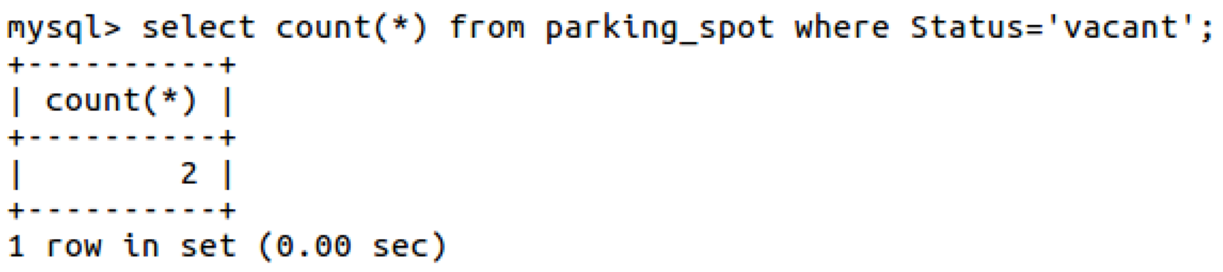
```
mysql> select * from park_on;
+---------+---------+
| Plate   | Spot    |
+---------+---------+
| SAMPLE1 | E01_001 |
| 514-KZE | S01_001 |
| 1AWJ785 | V01_001 |
| PYY438  | V01_110 |
+---------+---------+
4 rows in set (0.00 sec)
```

• List all the parking spot information



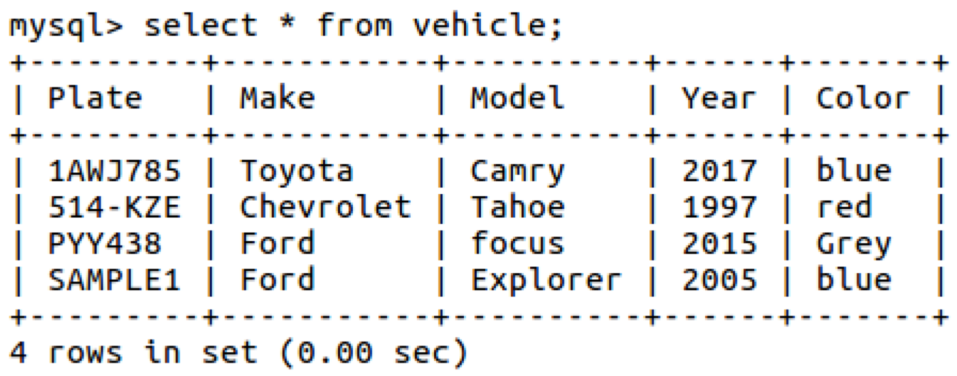
```
mysql> select * from parking_spot;
+---------+----------+----------+------+
| Spot    | Lot      | Status   | Rate |
+---------+----------+----------+------+
| E01_001 | Employee | vacant   | 0.00 |
| S01_001 | Student  | occupied | 0.00 |
| V01_001 | Visitor  | occupied | 1.50 |
| V01_110 | Visitor  | vacant   | 1.50 |
+---------+----------+----------+------+
4 rows in set (0.00 sec)
```

• List all the owner's vehicle plate .



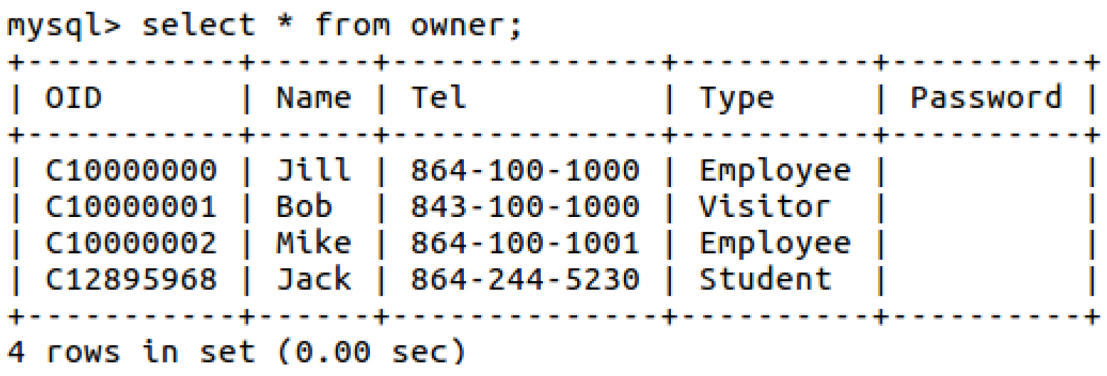
```
mysql> select count(*) from parking_spot where Status='vacant';
+----------+
| count(*) |
+----------+
|        2 |
+----------+
1 row in set (0.00 sec)
```

• List all the vehicle's information



```
mysql> select * from vehicle;
+---------+-----------+----------+------+-------+
| Plate   | Make      | Model    | Year | Color |
+---------+-----------+----------+------+-------+
| 1AWJ785 | Toyota    | Camry    | 2017 | blue  |
| 514-KZE | Chevrolet | Tahoe    | 1997 | red   |
| PYY438  | Ford      | focus    | 2015 | Grey  |
| SAMPLE1 | Ford      | Explorer | 2005 | blue  |
+---------+-----------+----------+------+-------+
4 rows in set (0.00 sec)
```

• List all the owner's information



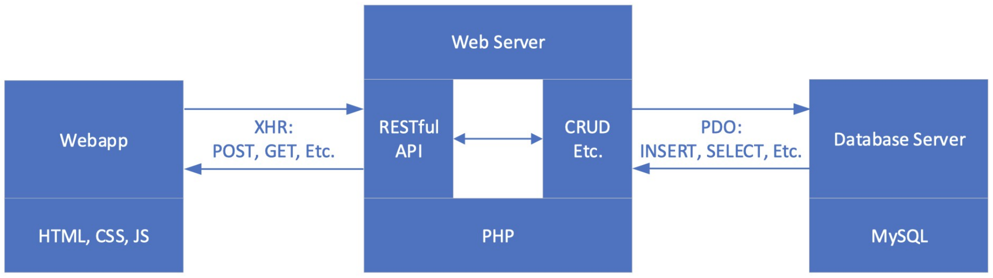
```
mysql> select * from owner;
+-----------+------+--------------+----------+----------+
| OID       | Name | Tel          | Type     | Password |
+-----------+------+--------------+----------+----------+
| C10000000 | Jill | 864-100-1000 | Employee |          |
| C10000001 | Bob  | 843-100-1000 | Visitor  |          |
| C10000002 | Mike | 864-100-1001 | Employee |          |
| C12895968 | Jack | 864-244-5230 | Student  |          |
+-----------+------+--------------+----------+----------+
4 rows in set (0.00 sec)
```

3. Design and Implementation of Web Server

Much effort has been put into designing and implementing a strong architecture for the backend code that runs on the web server. The web server is being hosted on Clemson's webapp server.

The following figure depicts the structure that we have designed and implemented. The webapp's HTML communicates to the webapp's javascript with ajax. The webapp's javascript then uses XHR to communicate with the RESTful API, which is written in PHP. XHR stands for XmlHttpRequest, which can handle sending and receiving JSON. The webapp and web server communicate to each other with JSON and XHR. The specific libraries we are using to do this in javascript are axios and jquery (and jquery ajax).

Note that it is possible to send and receive HTTP packets without using the webapp / web portal. Postman was used extensively in the development and testing of this api, and can still be used to access it. Other webapps could also



communicate with this api.

CRUD stands for Create, Read, Update, and Delete. These are basic functionalities and queries for a database.

PDO stands for PHP Data Objects, which is a set of functionality built into PHP. PDO communicates SQL queries from the PHP code on the web server to the MySQL on the database server. It also takes the responses (errors, successes, etc) from the database server and converts it into PHP-readable stuff.

There is one PHP Class in a script that, when instantiated, can use PHP Data Objects (PDO) to connect from the web server to the MySQL database that is running on the database server. For each table in the MySQL database, there is one PHP file containing a PHP class that defines a data model that implements basic Create, Read, Update, and Delete (CRUD) functionality. All of these files are in the models folder and implement an interface so they have the same structure. The RESTful API is built upon this CRUD functionality and is located in a folder named api. For each data model, there is a subfolder within the api folder. For each function in the data model, there is a PHP script located in the subfolder.

3.1. Description of Project Structure

The **config** folder houses Database.php, which contains a class that connects to the MySQL database via PDO.

The Owner.php file in the **models** folder is a data model class that has fields corresponding to the Owner table in the database.

It implements the interface in model_interface.php, so it is guaranteed to have certain functions.

The **bin** folder is used with Composer, a package manager for PHP. This project makes use of a library that implements ‘zxcvbn’, a password strength checker.

```
|---- models
|    |---- ManageParking.php             |---- bin
|    |---- ManageRecord.php              |    |---- composer.json
|    |---- Manager.php                   |    |---- composer.lock
|    |---- Own.php                       |    |---- composer.phar
|    |---- Owner.php                     |    !--- vendor
|    |---- ParkOn.php                    |---- config
|    |---- ParkingRecord.php             |    !--- Database.php
|    |---- ParkingSpot.php               |---- manage_tables.php
|    |---- Vehicle.php                   |---- cpsc6620.sql
|    !--- model_interface.ph
```

Using an instance of Database.php, it can send SQL queries by using the PDO functionality of the Database object.

The update.php file in the Owner subfolder in the **api** folder is a RESTful API endpoint.

It is accessible to the webapp.

It instantiates the Database class in Database.php. Then, it constructs an Owner object with the Database object. So, it is able to fill data in the Owner data model object and call its member function named “update” that contains an SQL query to be sent using the Database object.

Authentication is handled with the HTTP basic authentication scheme. The api creates a session when a user logs in. Authorization is implemented in some of the api endpoints, such as delete. In such a case, the api file checks for proper credentials in the session received from the client making the call to the api. Passwords are stored only after being salted and hashed.

The backup and restore functionality is also in the api. The database dump file is stored in api/backup.

The **app** folder contains the HTML, CSS, and JS files that form the webapp. This folder could theoretically be run on a separate web server from the web server that hosts the RESTful API.

```
|---- api
|    |---- AccessRecord
|    |---- Authentication
|    |---- ManageRecord
|    |---- ManageSpot
|    |---- Manager
|    |---- Own
|    |---- Owner
|    |---- ParkOn
|    |---- ParkingRecord
|    |---- ParkingSpot
|    |---- Vehicle
|    |---- backup
```

```
|---- app
|    |---- images
|    |---- index.html
|    |---- login.html
|    |---- lot_view.html
|    |---- query.html
|    |---- register.html
|    |---- scripts
|    |---- settings.html
|    !--- styles
```

Outside of any folder is the manage_tables.php file.

It is a script that runs in the command line on the web server.

It was used to fill in the tables and test the data models.

Its functionality will eventually be placed in the API and available to administrative users via the web portal.

The Readme file is also outside of any folder. It contains similar text to these bullet points describing the structure and usage of the files.

The following is an example to help show the logic flow of the project structure.

If a user wanted to register himself, he would fill in his details and press a submit button on the web portal. The webapp would then send a HTTP POST request to be handled by the API at webapp.cs.clemson/ jbtabb/api/owner/create.php. So, the POST request containing JSON data would be sent from the webapp to the web server’s create.php file. The create.php file would instantiate an owner from the models/owner.php file. Then, the create.php file would take the JSON data and use it to fill in the corresponding PHP variables in the owner object from the owner.php file.

Then, the create.php file would call the owner object’s create() function. The owner.php file would instantiate a connection to the MySQL database and use PDO to check/clean and submit a query with the relevant data and parameters to the database server. Next, the web server would receive a response from the database server and communicate that response back to the webapp via JSON. Finally, the webapp would display a success message or an error message.

One simplification that should be pointed out is that for now, a Manager is considered an Administrator. The two roles will be separated in the future. There have been more developments in the data model and api that go beyond CRUD functionality. Unfortunately, due to time constraints, the RESTful architecture was somewhat broken with in the list_all.php file and others like it because it does server-side rendering and returns HTML instead of sending just the data to the client and allowing it to do the rendering.

4. Design and Implementation of Web Portal / Web Application

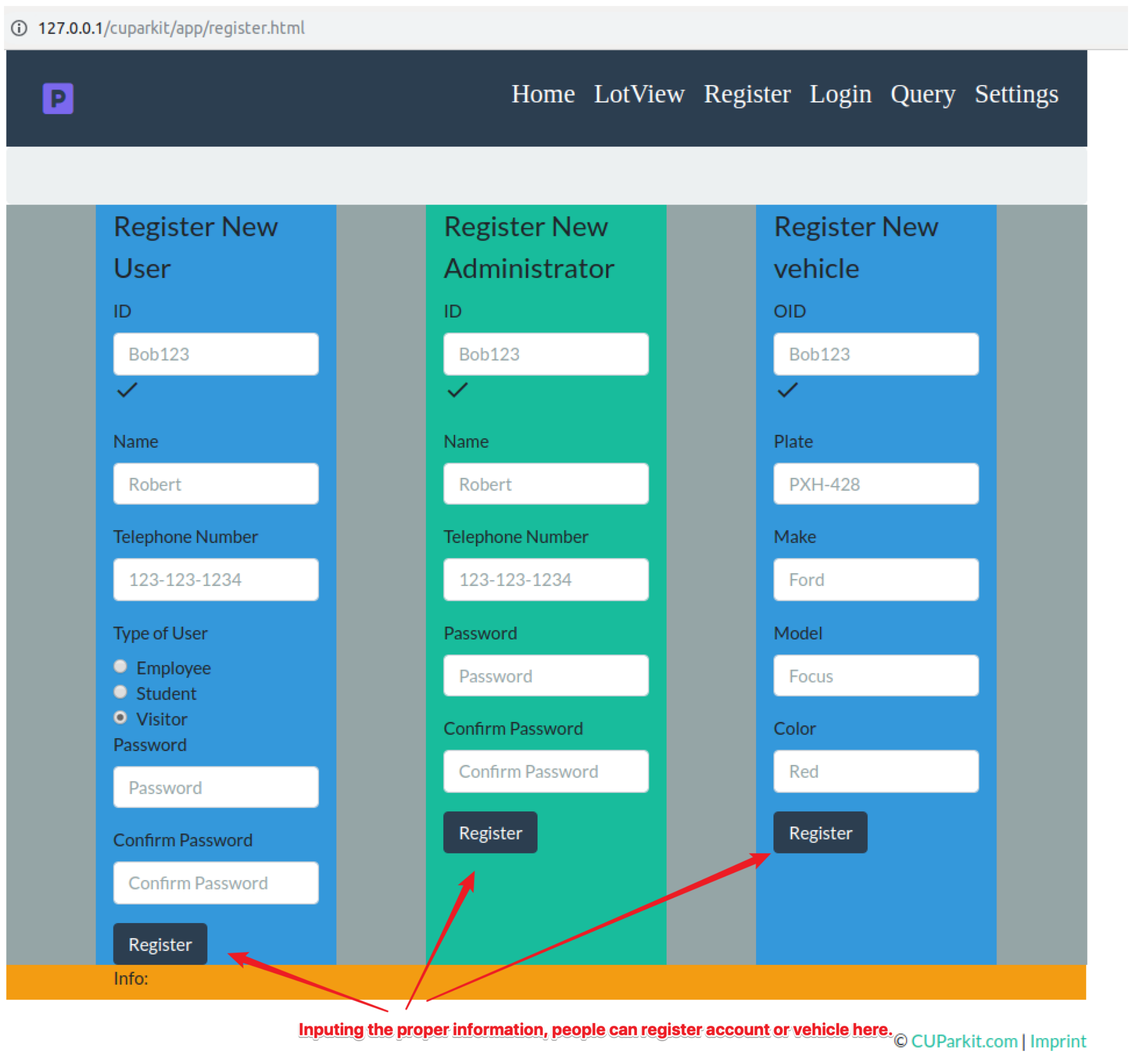
The web portal has a welcoming home screen. From there, the user can navigate to the LotView page, Registration page, Login page, Query page, or Settings page.

- The Register page allows a new owner to register himself. It also lets an owner register a vehicle to himself. Lastly, the Register page allows a user to register a new manager (admin). This is for testing purposes only. Obviously, a production app should not allow anyone to create a manager or admin account.
- The Login page allows an owner or a manager (admin) to authenticate. Once authenticated, the user will be authorized for the appropriate functions for the type of user.
- The query page has a dropdown menu to select “Parking Report” or “Occupied Spots”. Parking Report queries “parking_record” and shows all records of vehicles entering and exiting spots, with the date and time of the events (and the fee). Occupied Spots queries the “park_on” table and shows the state of the parking lot (i.e. which spots are open or not). The text box allows the user to filter the results of the “Occupied Spots” query to show only the spots that are vacant or only the spots that are occupied. The filter for occupied spots shows more information such as which car is occupying the spot.
- The Settings page allows a logged-in manager (admin)
- The LotView page is still not fully implemented, but is meant to provide a graphical user interface to allow an owner to park a vehicle into a parking spot. The LotView page also lets the manager see what the parking lot looks like in case he wants to add or remove spots from certain lots. (For instance, giving more lots to the students and less to the visitors, etc.)

Functions Implemented for Project Requirements

1. User account management

• New user registration



127.0.0.1/cuparkit/app/register.html

Register New User

ID: Bob123 ✓

Name: Robert

Telephone Number: 123-123-1234

Type of User: Employee / Student / [✓ Visitor]

Password: Password

Confirm Password: Confirm Password

Register

Register New Administrator

ID: Bob123 ✓

Name: Robert

Telephone Number: 123-123-1234

Password: Password

Confirm Password: Confirm Password

Register

Register New vehicle

OID: Bob123 ✓

Plate: PXH-428

Make: Ford

Model: Focus

Color: Red

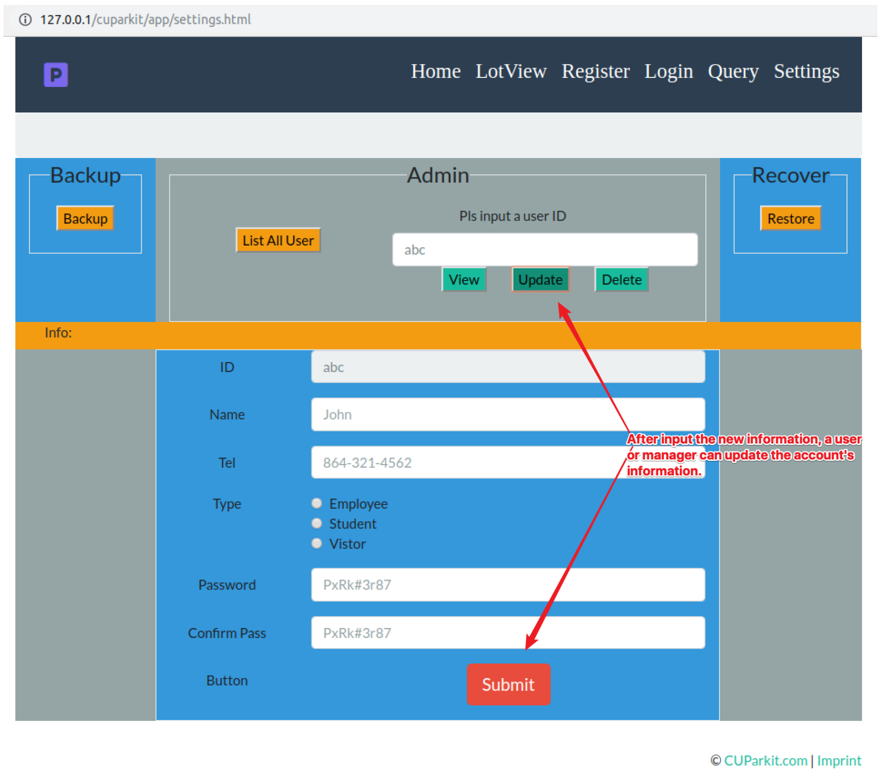
Register

Info:

Inputing the proper information, people can register account or vehicle here.

© CUParkit.com | Imprint

• User profile update



127.0.0.1/cuparkit/app/settings.html

Home LotView Register Login Query Settings

Backup: Backup

Admin: List All User | Pls input a user ID: abc | View | Update | Delete

Recover: Restore

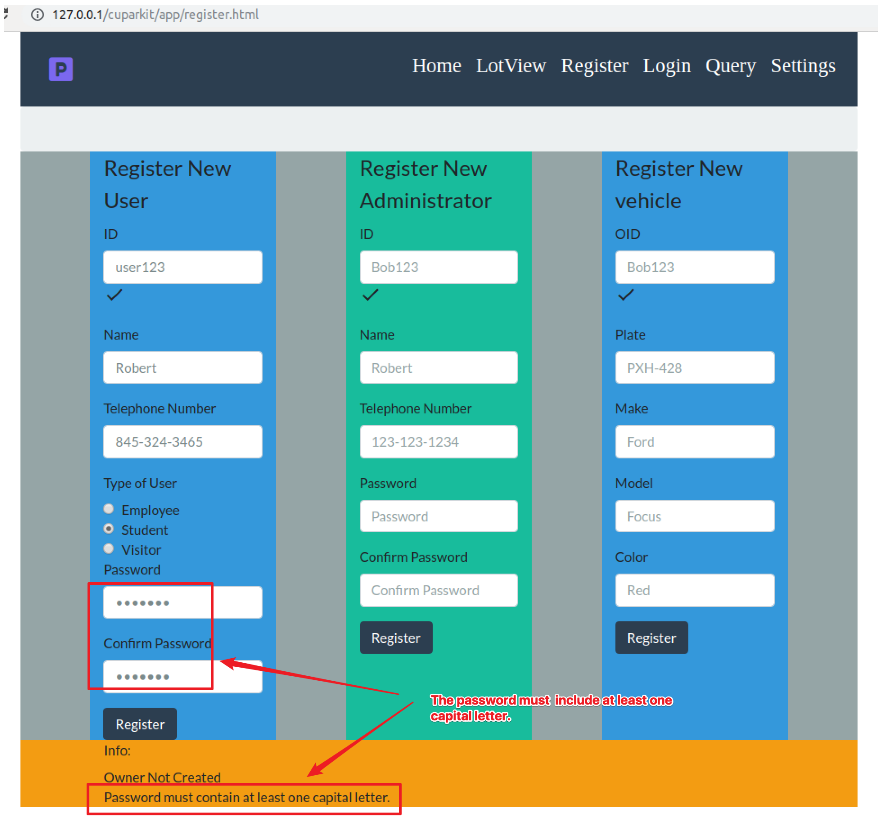
Info:

ID	abc
Name	John
Tel	864-321-4562
Type	Employee / Student / Vistor
Password	PxRk#3r87
Confirm Pass	PxRk#3r87
Button	Submit

After input the new information, a user or manager can update the account's information.

© CUParkit.com | Imprint

• Password must contain at least one capital letter



127.0.0.1/cuparkit/app/register.html

Home LotView Register Login Query Settings

Register New User	Register New Administrator	Register New vehicle
ID: user123 ✓	ID: Bob123 ✓	OID: Bob123 ✓
Name: Robert	Name: Robert	Plate: PXH-428
Telephone Number: 845-324-3465	Telephone Number: 123-123-1234	Make: Ford
Type of User: Employee [✓ Student] Visitor	Password: Password	Model: Focus
Password: •••••••	Confirm Password: Confirm Password	Color: Red
Confirm Password: •••••••	Register	Register
Register		

The password must include at least one capital letter.

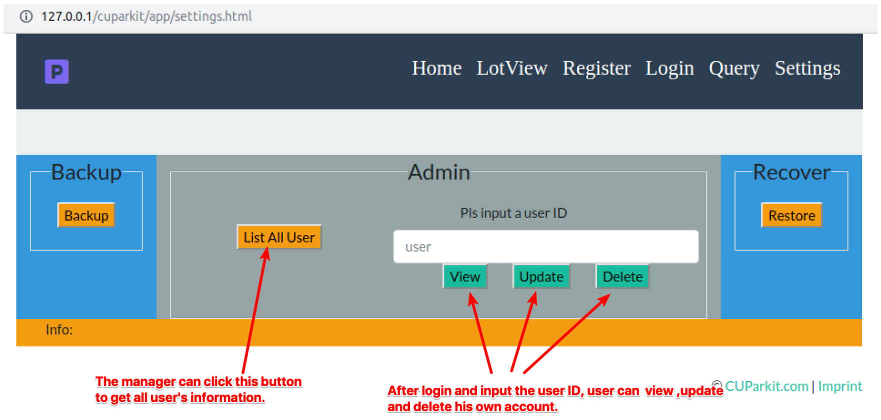
Info:

Owner Not Created

Password must contain at least one capital letter.

2. Data management

• Owners can view, query, update ···


127.0.0.1/cuparkit/app/settings.html
P
Home LotView Register Login Query Settings
Backup
Backup
Admin
Pls input a user ID
List All User
user
View
Update
Delete
Recover
Restore
Info:
The manager can click this button to get all user's information.
After login and input the user ID, user can view ,update and delete his own account.
© CUParkit.com | Imprint

• The portal splits large query results into multiple pages.

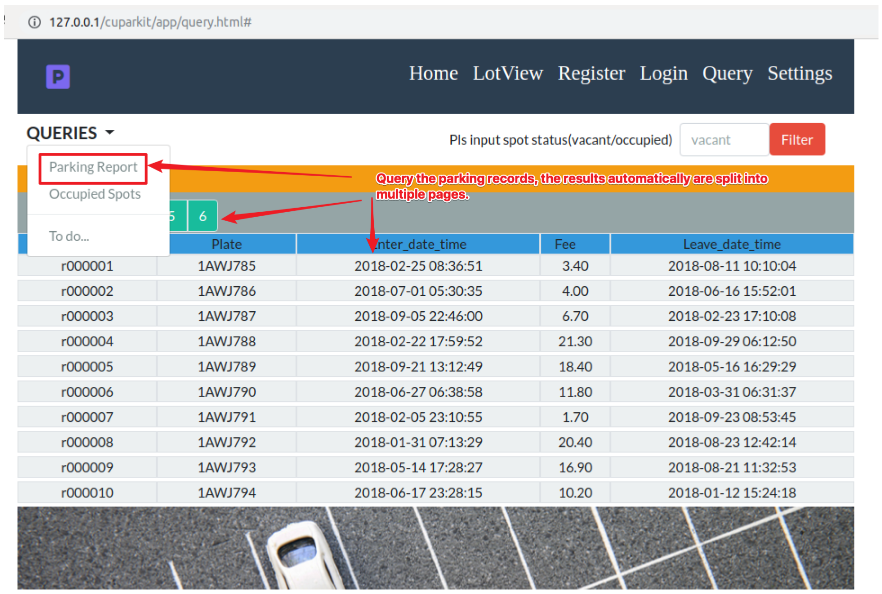
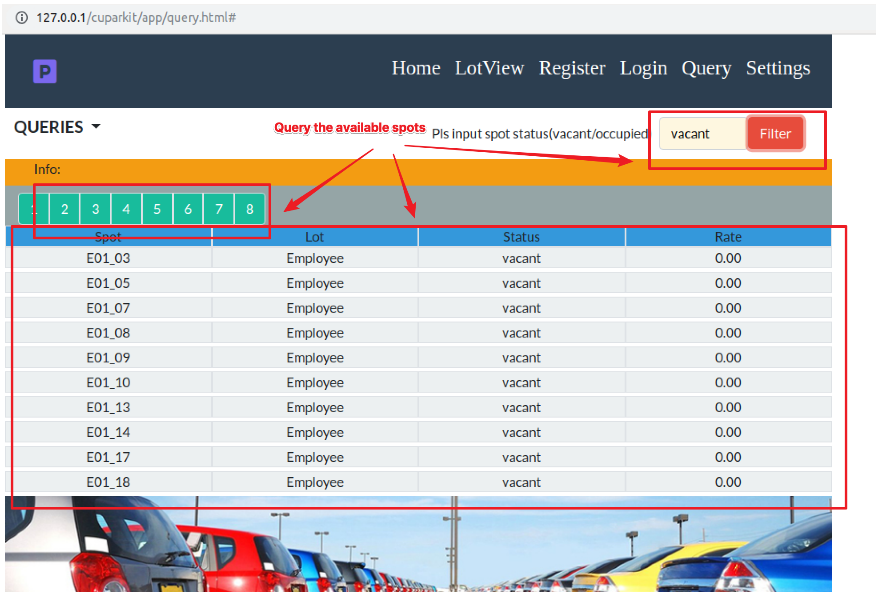

127.0.0.1/cuparkit/app/query.html#
P
Home LotView Register Login Query Settings
QUERIES
Parking Report
Occupied Spots
To do...
Pls input spot status(vacant/occupied)
vacant
Filter
Query the parking records, the results automatically are split into multiple pages.
5
6

	Plate	Enter_date_time	Fee	Leave_date_time
r000001	1AWJ785	2018-02-25 08:36:51	3.40	2018-08-11 10:10:04
r000002	1AWJ786	2018-07-01 05:30:35	4.00	2018-06-16 15:52:01
r000003	1AWJ787	2018-09-05 22:46:00	6.70	2018-02-23 17:10:08
r000004	1AWJ788	2018-02-22 17:59:52	21.30	2018-09-29 06:12:50
r000005	1AWJ789	2018-09-21 13:12:49	18.40	2018-05-16 16:29:29
r000006	1AWJ790	2018-06-27 06:38:58	11.80	2018-03-31 06:31:37
r000007	1AWJ791	2018-02-05 23:10:55	1.70	2018-09-23 08:53:45
r000008	1AWJ792	2018-01-31 07:13:29	20.40	2018-08-23 12:42:14
r000009	1AWJ793	2018-05-14 17:28:27	16.90	2018-08-21 11:32:53
r000010	1AWJ794	2018-06-17 23:28:15	10.20	2018-01-12 15:24:18

• The portal can filter the results of parking spots and show only the vacant ones.

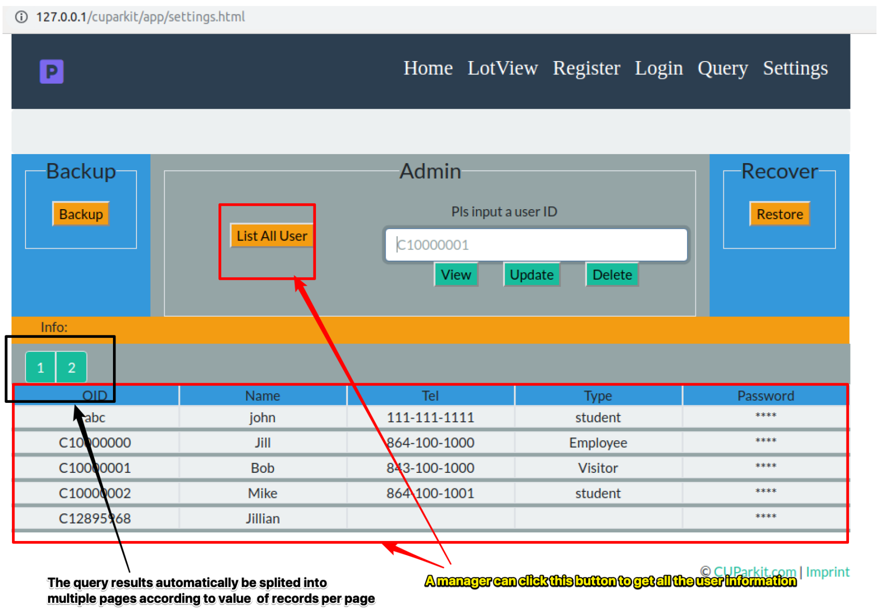

127.0.0.1/cuparkit/app/query.html#
Home LotView Register Login Query Settings
QUERIES
Pls input spot status(vacant/occupied)
vacant
Filter
Info:
1 2 3 4 5 6 7 8

Query the available spots

Spot	Lot	Status	Rate
E01_03	Employee	vacant	0.00
E01_05	Employee	vacant	0.00
E01_07	Employee	vacant	0.00
E01_08	Employee	vacant	0.00
E01_09	Employee	vacant	0.00
E01_10	Employee	vacant	0.00
E01_13	Employee	vacant	0.00
E01_14	Employee	vacant	0.00
E01_17	Employee	vacant	0.00
E01_18	Employee	vacant	0.00

3. Database administration

• Managers (admins) can view, query, update···

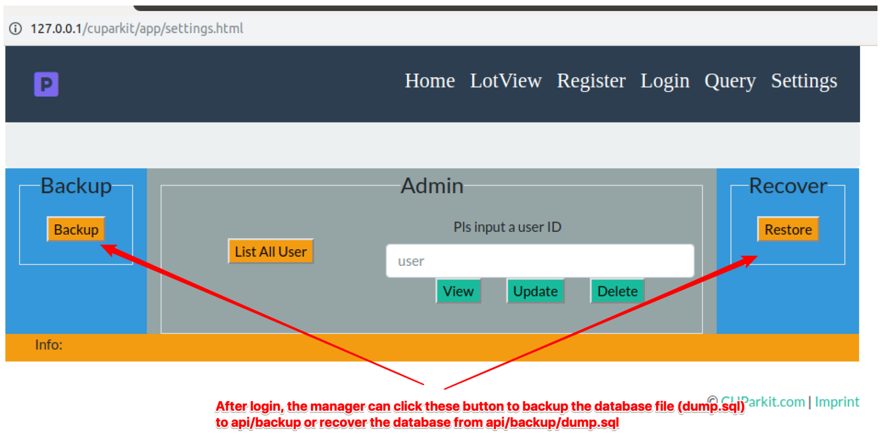


OID	Name	Tel	Type	Password
abc	john	111-111-1111	student	****
C10000000	Jill	864-100-1000	Employee	****
C10000001	Bob	843-100-1000	Visitor	****
C10000002	Mike	864-100-1001	student	****
C12895768	Jillian			****

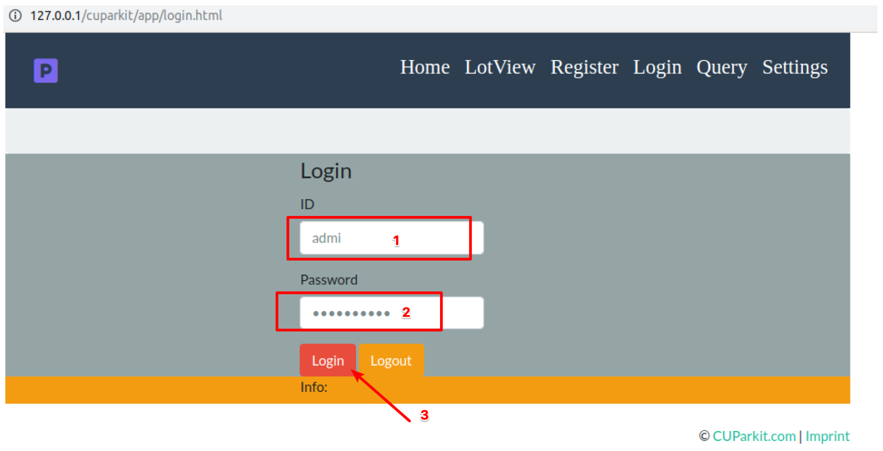
The query results automatically be splited into multiple pages according to value of records per page

A manager can click this button to get all the user information

• Managers (admins) can backup / recover a database dump file onto / from the web server in ‘api/backup/dump.sql’


127.0.0.1/cuparkit/app/settings.html
Home
LotView
Register
Login
Query
Settings
Backup
Backup
Admin
List All User
Pls input a user ID
user
View
Update
Delete
Recover
Restore
Info:
After login, the manager can click these button to backup the database file (dump.sql) to api/backup or recover the database from api/backup/dump.sql
© CUParkit.com | Imprint

• Login and logout module



Some functions are limited to anyone who is logged in. Some functions only allow an owner to change the data that is specific to him and not any other owners. Some functions only allow the manager (admin) to view / change all things.

There are some functionalities that were implemented in the api, but that were not implemented in the web portal. These can be tested with Postman to send custom-made HTTP packets to the api.

5. Limitations in Current Implementation and Future Fixes

Many points addressing the limitations of this project have been addressed throughout this report. A summary of them and some additional points will be made here. It' s assumed that a vehicle is owned by one owner. In the future, allowing multiple users to own the same vehicle should be implemented. It' s assumed that a vehicle' s plate number is unique, even though there could be a duplicate one in another state. Combining more details about the vehicle with its plate number should make an actually unique key. It' s assumed that there is only one employee lot, one visitor lot, and one student lot. In the future, there should be several of each. The simplification that a Manager is an Administrator will be removed in the future. Also, the ability for a new user to register as a manager / admin will be removed. The LotView page needs to be greatly improved and implemented. More advanced queries, sorting, and filtering should be implemented. The files that break the RESTfulness of the API should be reworked so that they are not sending

HTML, but just data. A lot of comments are erroneous because they were copied and pasted from other parts of code. Eliminating redundant code and fixing the comments is a necessary future task.

There are many more things that can be and / or should be done for this to be a functioning application. This project will probably be developed beyond the scope of this course’s project requirements. If it does get that far, then implementing the hardware such as sensors, etc. will need to be thought out, too.

6. Web Portal and Source Code Access

The web portal can only be accessed through the Clemson network. The Clemson VPN does allow access if connecting from off campus.

Link to web portal: http://webapp.cs.clemson.edu/ qingbol/6620/CUParkit/app/index.html

Link to source code:https://github.com/qingbol/CUParkit

Note: The source code will have the main branch for the TA / Instructor to view for this deadline. We will continue to update on another branch that is also in this repo on this link.

Note: Our project allows the user to register a new Manager / admin, but the following one already exists.

Administration (Manager) username: dbms

Administration password: Dbms#6620

Normal User (Owner): user

Normal User pass: Dbms#6620