Rutgers University

Auto Park

Parking Garage Automation

Report 2

Author:	*Email:*
Chunhua Deng	chunhua.deng@rutgers.edu
Corey Chen	cc1437@scarletmail.rutgers.edu
Jonathan Garner	jonathan.garner@rutgers.edu
Ridhima Sakhuja	rs1425@rutgers.edu
Siddharth Musale	siddharth.musale@rutgers.edu
Siyu Liao	siyu.liao@rutgers.edu
Xianglong Feng	xianglongchunyi@gmail.com

November 11, 2018

Contents

List of Figures

1 Interaction Diagrams

1.1 Use Case 1: Register

We decided to assign the responsibility to register to the website (system), as the website is the main interface through which the customer can make parking reservations. The website has the responsibility of allowing customers to login and make reservations, which ensures that the website has focused specialty and does not have too many responsibilities assigned to it. Even though the customer interacts with the system to for logging in and making reservations, we assigned the database the responsibility to verify and store the data that is being received. In this way the database can easily access information about customer when it is needed for parking.

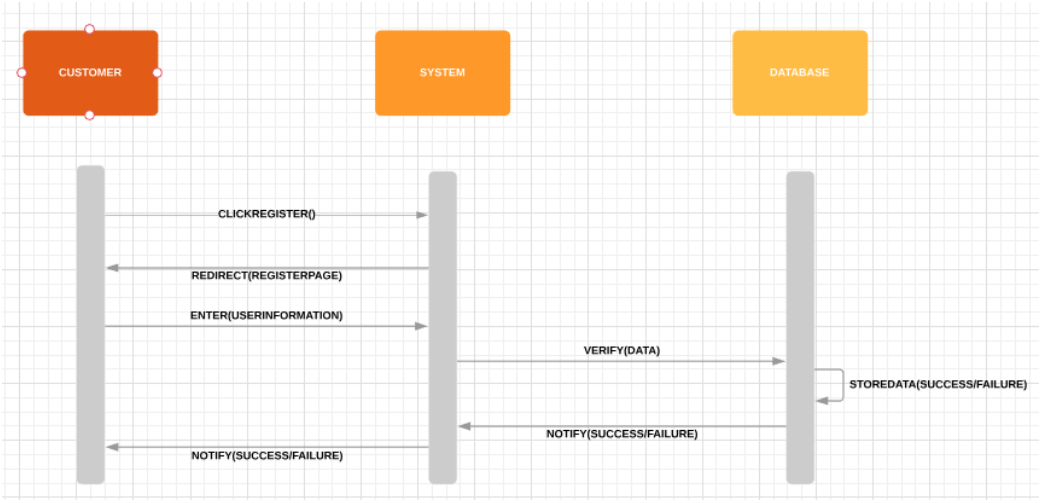

Figure 1: Sequence Diagrams for Use Case 1.

1.2 Use Case 3: Reservation

When a customer attempts to place a reservation, the system attempts to verify their account by querying the database for the correct account info. In this use case, the database's main responsibilities are keeping track of and reporting account information and reservation/spot information. This data is then given back to the system, which is responsible for passing information from the customer to the database and displaying relevant information from

the database to the customer in an easy to understand format. The system is also responsible for processing the customer's action on the website, such as redirecting the customer to different pages on the website based on their requests.

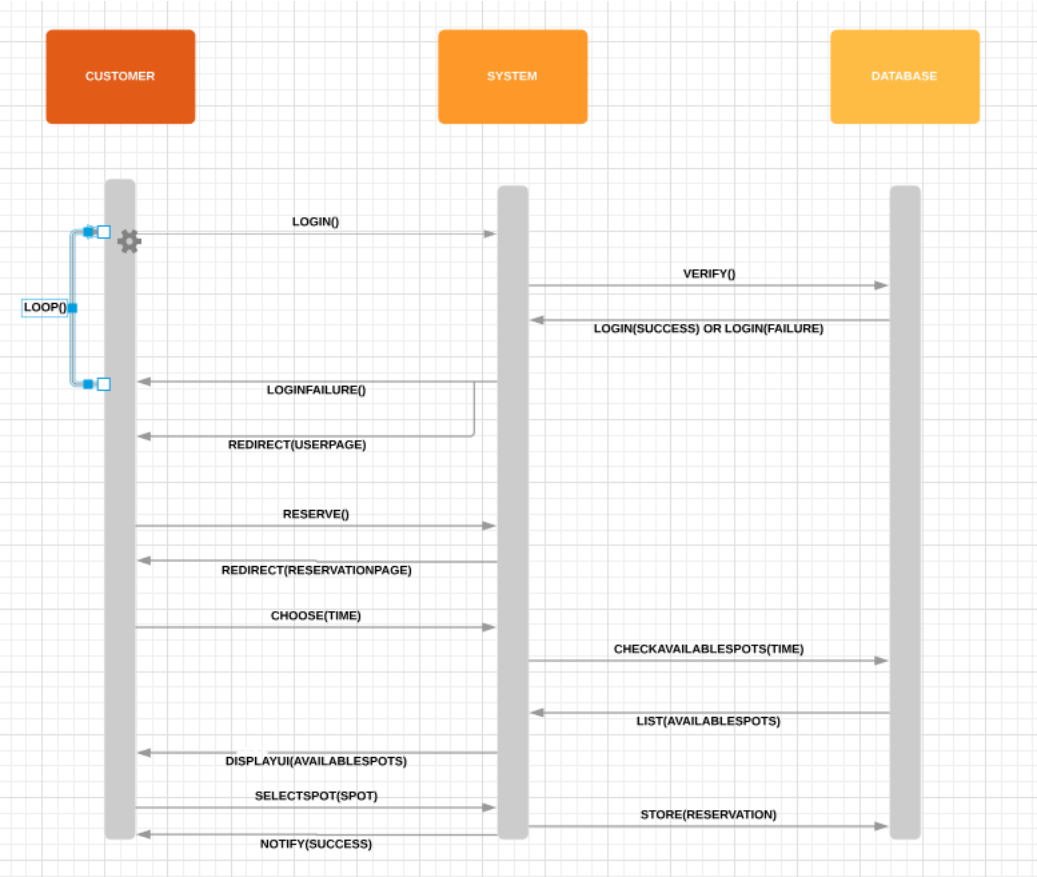

Figure 2: Sequence Diagrams for Use Case 3.

1.3 Use Case 4: Ad-hoc

When an Ad-Hoc user enters the garage, they will interact with a user interface that will be placed at the front of the parking garage. The user interface(denoted as system in the interaction diagram) will be responsible for showing the ad hoc customers all the available parking spots and for

requesting and verifying user information so that the users can enter the garage. Since the user interface is the first system that ad-hoc users interact with upon entering the garage, we decided to assign the previously mentioned responsibilities to the the system as both responsibilities should be completed before the user enters the parking garage.

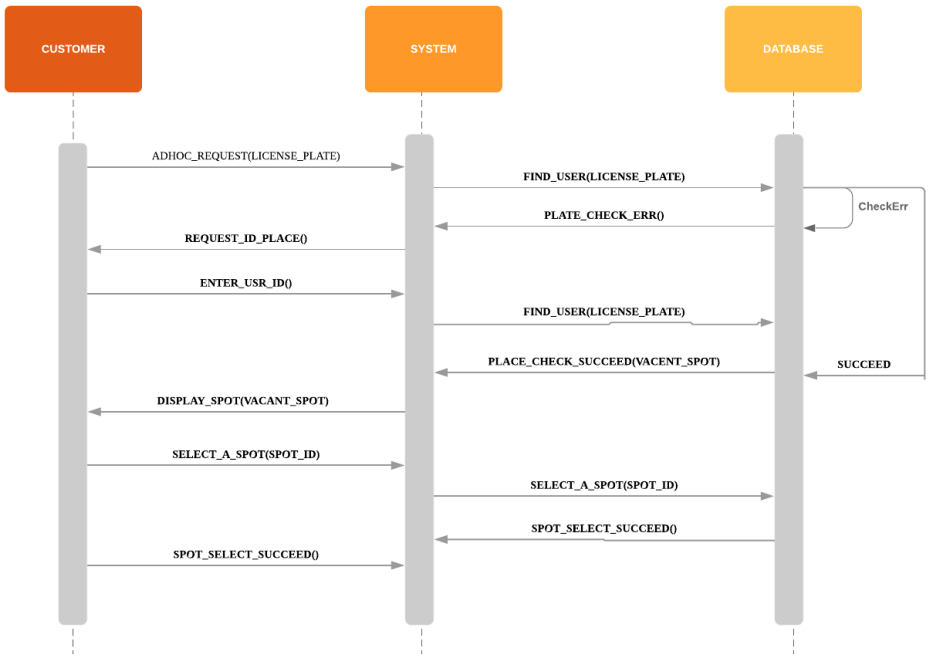

Figure 3: Sequence Diagrams for Use Case 4.

1.4 Use Case 14: Payment

After a user has completed their parking (reservation or adhoc) they will move to the exit. They will interact with the system and request to pay and leave. The system will read the license plate and request parking information from the database. Total cost will be computed and then the system will check the database to see if this is a registered user or not. If the user is a registered user then the credit card associated with the account will be charged and the person will be allowed to leave. If the account is not reg-

istered then payment information and billing information will be requested. Upon verification of payment method, the user will be allowed to leave.

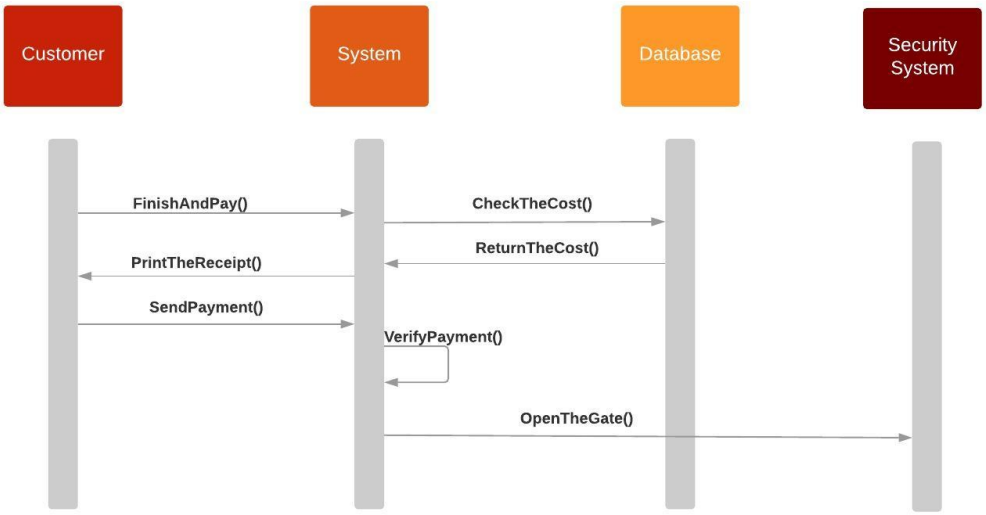

Figure 4: Sequence Diagrams for Use Case 14.

2 Class Diagram and Interface Specification

2.1 Class Diagram

2.2 Data Types and Operation Signatures

1. Manager
   (a) Attributes
   - int customer_id
   - string first_name
   - string last_name
   - string email
   - string phone_number
   - string password

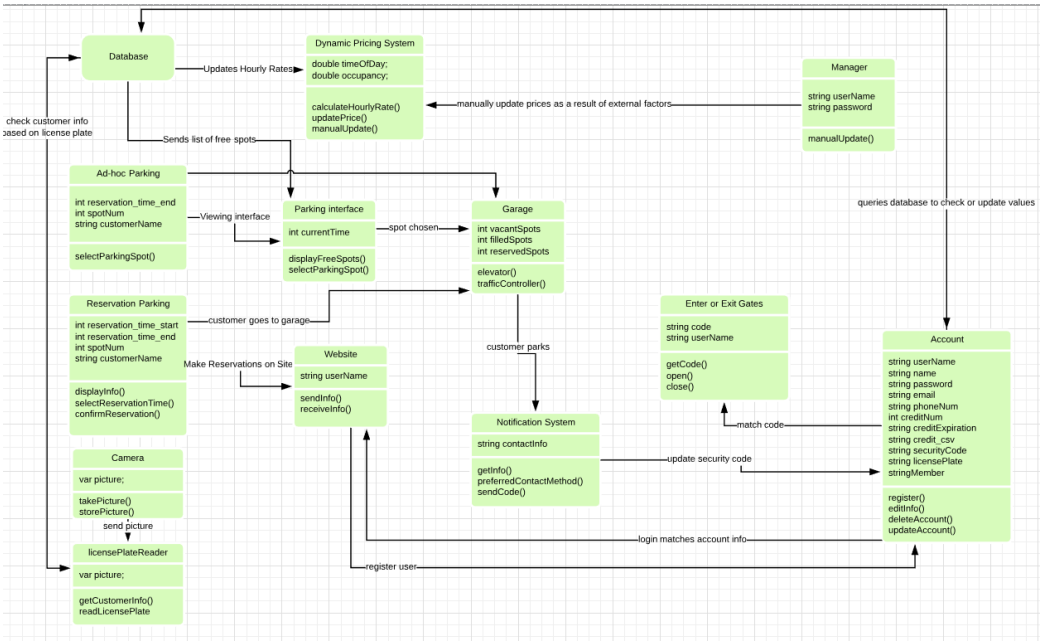

Figure 5: Class Diagram.

- date dob
- string creditcardno
- string csv
- string securitycode
- date dob
- enum Type('Membership', 'Ad-hoc')

(b) Operation: (all operations attach customer to any created objects)

- Register(): Creates a new user and inserts the user info into database.
- addCar() : creates a car and inserts into database
- addCreditCard() : creates a credit card and inserts into database
- newReservation() : creates a reservation and inserts into database

2. Camera

(a) Attributes

- var picture

(b) Operation: (all operations attach customer to any created objects)

- getCustomerInfo(): check the customer information from the database
- readLicensePlate(): Read the license plate.

3. PriceSystem

(a) Attributes

- int currentTime

(b) Operation: (all operations attach customer to any created objects)

- calculateHourlyRate(): calculate the hourly based price
- updatePrice(): update the price to the database
- manualUpdate():manually update the price to database

4. Ad-hoc Parking

(a) Attributes

- int reservation_time_end
- int spotNum
- string customerName

(b) Operation: (all operations attach customer to any created objects)

- selectParkingSpot(): select the Parking Spot with spot id spotNum

5. Reservation Parking

(a) Attributes

- int reservation_time_start
- int reservation_time_end
- int spotNum
- string customerName

(b) Operation: (all operations attach customer to any created objects)

- displayInfo(): display the vacant parking spots to customers
- selectReservationTime(): let customers to select the time spots
- confirmReservation(): confirm the customers' revervation

6. Parking Interface
    (a) Attributes
        - int currentTime
    (b) Operation: (all operations attach customer to any created objects)
        - displayFreeSpots(): display the vacant parking spots to customers
        - selectParkingSpot(): let the customers select the parking spot
7. Garage
    (a) Attributes
        - int vacantSpots
        - int filledSpots
        - int reservedSpots
    (b) Operation: (all operations attach customer to any created objects)
        - elevator(): go to the floors where the selected parking spot on
        - trafficController(): allow only one car entering one floor
8. Website
    (a) Attributes
        - string userName
    (b) Operation: (all operations attach customer to any created objects)
        - sendInfo(): send registering user information to customer module
        - receiveInfo(): receive registered user information from customer module
9. NotificationSystem
    (a) Attributes
        - string contactinfo
    (b) Operation: (all operations attach customer to any created objects)
        - getInfo(): get user information from the dataset

- preferredContactMethod(): check the emailphone information
- sendCode(): generate the info and send it to user

10. Elevator
    (a) Attributes
        - string code
        - string username
    (b) Operation: (all operations attach customer to any created objects)
        - getCode():get the verify code from user and check the code
        - open(): Open the gate
        - close(): close the gate

2.3 Traceability Matrix

The domain concepts were taken from part 5.4 in Report 1. The responsibilities for each of the domain concepts are taken from the class diagram. The classes for each of the domain concepts derived from their responsibilities.

1. Database
    (a) Responsibilities
        - Store customer and reservation information
        - Check to see if incoming customer has a reservation
        - Update hourly rates for dynamic pricing system
2. Dynamic Pricing System
    (a) Responsibilities
        - Set prices for customers based on various conditions
    (b) Classes
        - calculateHourlyRate(): calculate rate per hour for garage parking
        - updatePrice(): update the current price
        - manualUpdate(): in case of system failure, manually update prices

3. Website
    (a) Responsibilities
        - Collect customer information
        - Allows customers to make/update reservations
    (b) Classes
        - sendInfo(): collects customer info from website and sends it to database
        - receiveInfo(): receives responses from the system and possibly display it
4. Notification System
    (a) Responsibilities
        - Notify the customer when they have successfully parked
        - Send alphanumeric security code to customer which they will use to enter and exit the garage
    (b) Classes
        - getInfo(): receive customer information
        - preferredContactMethod(): store customer's preferred contact method (ie. phone or email)
        - sendCode(): send alphanumeric/confirmation code to customer
5. License Plate Reader
    (a) Responsibilities
        - Scan license plate to gather customer information on incoming customer
    (b) Classes
        - readLicensePlate(): query database with license plate to check for customer information
        - getCustomerInfo(): gather info about customer from license plate
6. Visual Interface (Parking Interface)

(a) Responsibilities

- Allow walk-in customers to choose parking spot
- Collect walk-in customers' information
- Allow for manual input in case of system failure

(b) Classes

- displayFreeSpots(): show all available parking spots on visual interface
- selectParkingSpots(): select parking spot

3 System Architecture and System Design

3.1 Architectural Style

In terms of structure of the garage logic modules, the Auto Park system follows a component-based design style. Each task that must be completed within the system is performed by a module dedicated to that task (the notification system handles the sending of messages to customers, the traffic control system handles the amount of cars driving on each floor, etc). Since there are many unique functions which must operate in parallel with one another and process the same information, each component of the system must be able to communicate with the relevant other subsystems. Every component is responsible for independently executing its primary functions while also coordinating with the other components to ensure smooth operation.

As for memory and data sharing, the system uses a database-centric architecture. All information relating to user accounts, parking space status, reservations, and payments are saved in databases which are accessible by the other modules in the system. This database design allows the various components of Auto Park to access, view, and edit the same data, allowing for easier communication between components in many cases.

The direct communication between components in the garage is also heavily based around an event-driven architecture. Each primary function in the system must happen at a specific step during a normal use case. For example, the security system waits for a customer to park their car correctly before generating a code, and the notification system waits for this code to be generated before sending it to the customer. Each step in the system's

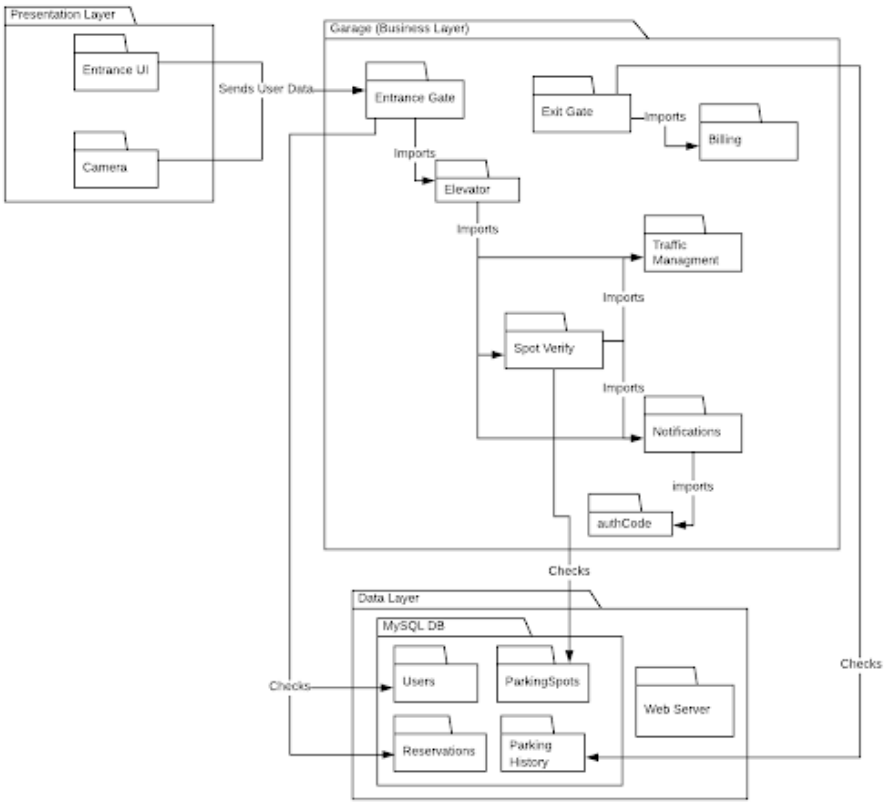

Figure 6: Subsystems in our System.

process is triggered by the completion or progress of a previous step, which is the main philosophy behind an event-driven messaging style.

3.2 Identifying Subsystems

Figure 6 shows all subsystems designed in our system.

3.3 Mapping Subsystems to Hardware

Our website runs using typical server client technology. The server handles all request sent out from clients, which could be cellphone, laptop or anything that can run a browser. Apart from this, our data processing runs on the spark platform. It's configurable for users to run over single machine, several

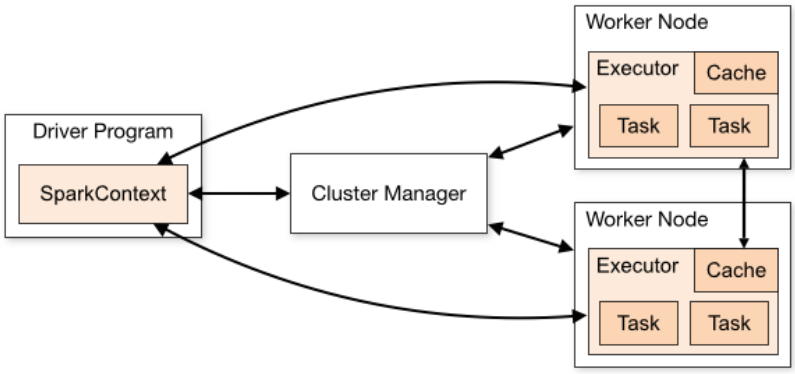

Figure 7: An Overview of the Spark Structure [1].

machines and even hundreds of machines if necessary. The main structure of spark is shown in Figure 7.

We launch the data processing job through the driver program, which creates the sparkcontext according to the configurations. It will convert the job into multiple tasks and schedule them over different executors through cluster manager. Executors starts together when sparkcontext is created. When finishing the task, executors will send results back to cluster manager. In our project, we run the data processing using 32 executors with driver memory 20GB and executor memory 80GB.

3.4 Persistent Data Storage

A MySQL database was used to store the data acquired by the system. The database stores information about the customer's contact/payment information as well as information (ie. date, time, reserved parking spot) about any reservations they may have at the parking garage.In this way, the database is also used to determine if an incoming customer has a reservation or is an ad hoc customer. If a customer has parked in the parking garage, the customer's parking spot location is also stored in the database.

3.5 Network Protocol

For communication between client and server and for the structure of the web applications, our system uses Node.JS. Using node allows the server

to load scripts inline with the web pages and redirect the user to different pages via routes based on requests they send to the server. This way the web page displays as a normal HTML page would, but the server is always "listening" for requests from the user. Using node also allows for many different packages to be utilized in the web applications, such as code which can securely handle and encrypt passwords or code which can be used to display the visual interface of parking spots.

3.6 Global Flow Control

3.6.1 Execution Orderness

The parking garage system is both event driven and procedure driven. The system is procedure driven because when the customer arrives at the parking garage, a sensor will read the car's license plate number and check to see if there is already a pre-existing reservation under that customer's name. Additionally, there are sensors on each floor to ensure that no more than one car is moving on each floor at a time and that each car is parked in its designated spot. The system is event driven because if a customer does not have a reservation, they will be required to input their contact information and choose a parking spot before entering the garage.

3.6.2 Time Dependency

The system uses timers to record the duration of a reservation/parking. The timers will be real time as they will be used to ensure that customers are not staying past their reservations. For reserved customers, the system will also give them a one hour grace period during which the customer can come to the garage to park at anytime with their reservation.

3.6.3 Concurrency

Each car that is currently at the parking garage will have its own thread.

3.7 Hardware Requirements

1. Website Hosting Server
    - 33 MB disk space

- 100 Kb/s connection speed

2. Database Hosting
    - 10 GB disk space
    - 80 GB memory for spark processing
    - 100 Kb/s connection speed
3. Garage Hardware
    - Camera (for license plate scanner)
    - Sensor for each parking spot
    - Keypad for security at each pedestrian entrance/exit
    - Tablet for entrance gate display

4 Algorithm and Data Structure

In the smart pricing system, we collected the data from the government. Those data cover the usage of the parking lots in Seattle from 2012 to 2017. We first analyze the parking lots usage, tying to use logic regression methods to estimate the usage over time (hour, day, month). This could help us define the time based demand function, which represents how the demand varies with the time (hourly, daily, monthly). There are other methods to model the time-based demand function, such as the neural network. However, based on our collected data, we found that the demand function is linear and the polynomial could fit the curve. Then, we try to use linear least squares regression and cubic spline interpolation to calculate the coefficient of the polynomial for the time-based demand function.

Besides the time-based demand function, we also proposed several price-based demand functions. The time-based demand function represents the relation between the parking lot usage and the time. Heuristically, it shows when the parking lot used most often and when the parking lot is used rarely. To design a dynamic pricing system, the goal is using the price to adjust the usage of the parking lot and make the most benefits on the fixed parking lots. When there is huge demand for the parking lots, the price will increase and demand should decrease. When there is small demand for the parking lot, the price should decrease to the minimum. So, based on this fact, we

propose the price-based demand function. However, for different cases, the demand function could be impacted by the price differently. So, we proposed three demand models which is exponential, reciprocal and linear model for different scenario. Finally, we would calculate the real demand function based on the time and price. We calculate the total benefits by integral the demand function over time and price. So the final benefits function is the function of price over time. And, based on which, we could calculate the best price by solving the optimization problem of maximizing the benefits function.

4.1 Curve Fitting

Given the collected parking lot data from the government, we try to use polynomial to model the real data. Then, we leverage the linear least squares regression and cubic spline interpolation to estimate the coefficient.

4.1.1 Linear least squares regression

In linear least squares regression, we got the real data as time (t) and the demand (y). So we got several data point (t1,y1), (t2,y2)...(tn,yn). We want the polynomial fit the data. In the other words, we want to minimus the error.

The polynomial we assumed is $y = a*t^6+b*t^5+c*t^4+d*t^3+e*t^2+f*t^1+g$.

$\frac{\partial Q}{\partial g} = 2\sum_{i=1}^{n}(Y_i - (a*t^6 + b*t^5 + c*t^4 + d*t^3 + e*t^2 + f*t^1 + g)*(1))$

Set the real data is Y. the error is$E = Y - y$; The squared error is$E^2 = (Y-y)^2$.

The total error for all the data set is

$Q = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n}(Y_i - y_i)^2$

The goal is to minimus the error and so we will calculate the partial derivative for a,b,c,d,e,f and g. Calculate the value by set the partial derivative equal to 0.

$\frac{\partial Q}{\partial a} = 2\sum_{i=1}^{n}(Y_i - (a*t^6 + b*t^5 + c*t^4 + d*t^3 + e*t^2 + f*t^1 + g)*(-t^6))$

$\frac{\partial Q}{\partial b} = 2\sum_{i=1}^{n}(Y_i - (a*t^6 + b*t^5 + c*t^4 + d*t^3 + e*t^2 + f*t^1 + g)*(-t^5))$

$\frac{\partial Q}{\partial c} = 2\sum_{i=1}^{n}(Y_i - (a*t^6 + b*t^5 + c*t^4 + d*t^3 + e*t^2 + f*t^1 + g)*(-t^4))$

$\frac{\partial Q}{\partial d} = 2\sum_{i=1}^{n}(Y_i - (a*t^6 + b*t^5 + c*t^4 + d*t^3 + e*t^2 + f*t^1 + g)*(-t^3))$

$\frac{\partial Q}{\partial e} = 2\sum_{i=1}^{n}(Y_i - (a*t^6 + b*t^5 + c*t^4 + d*t^3 + e*t^2 + f*t^1 + g)*(-t^2))$

$\frac{\partial Q}{\partial f} = 2\sum_{i=1}^{n}(Y_i - (a*t^6 + b*t^5 + c*t^4 + d*t^3 + e*t^2 + f*t^1 + g)*(-t^1))$

$\frac{\partial Q}{\partial g} = 2\sum_{i=1}^{n}(Y_i - (a*t^6 + b*t^5 + c*t^4 + d*t^3 + e*t^2 + f*t^1 + g)*(1))$

4.1.2 Cublic spline interpolation

In numerical interpolation, the Runge's phenomenon is the oscillation at the edge of intervals when higher-degree polynomials are used in interpolation. It was first discovered by Carl David Tolmé Runge when he explored the behavior of errors with polynomial interpolation to approximate certain functions [2]. Fig. 1 shows the example of Runge phenomenon [3], in which we can see that the higher the polynomial, the larger the oscillation. This is why we use cubic spline interpolation. The interpolation function is piece-wise smooth function, which could avoid the global high-degree interpolation function. The main idea behind cubic spline interpolation is that we utilize low-order polynomials to approximate the function, and let the function be smooth at the intersections, that is second-order derivatives continuous.

Let interval [a,b] has some interpolation points $x_0, x_1, x_2, ...x_n$ where $a = x_0 < x_1 < x_2 < ... < x_n = b$, and each x corresponding a y. So we have $(x_0, y_0), (x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ pairs of points. If function $S(x)$ satisfies the following three conditions:

- At each interval $[x_{k-1}, x_k]$, where $k = 1, 2, ..., n$, $S_k(x)$ is a polynomial less then 3-order.
- At each interpolation point x_i, $S_i(x_i) = y_i$, where i=0,1,2,...,n
- $S_i(x)$ is a second-order derivative continous function at interval [a,b].

We call function S(x) the spline function. If the function S(x) is three-order, that is $S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$, it is called cubic spline function.

To solve the cubic spline interpolation, we need to substitute all the known conditions into the problem to solve the linear equations. For example, we need to fit a curve which pass $(x_0, y_0), (x_1, y_1), (x_2, y_2)$ points. So we have two spline functions with 8 independent variables to solve. The 8 equations

are:

$$
\begin{aligned}
S_0(x_0) &= y_0 \\
S_0(x_1) &= y_1 \\
S_1(x_1) &= y_1 \\
S_1(x_2) &= y_2 \\
S_0^{'}(x_1) &= S_1^{'}(x_1) \\
S_1^{''}(x_1) &= S_1^{''}(x_1) \\
S_0^{''}(x_0) &= 0 \\
S_1^{''}(x_2) &= 0
\end{aligned}
$$

where the last two equations are boundary condition. Therefore, the 8 equations can solve the equation array with 8 independent variables.

4.2 Demand Function

Let $f(t)$ represent the time based function learned from real world data, where t indicates current time. Denote the price function at current time as $p(t)$. Although $f(t)$ can indicate the overall parking demand, we need to figure out the change of demand $d(t) = h(p(t), f(t))$ when given the dynamic price $p(t)$. This actually depends a lot on the driver's strategy. For simplicity, we are aimed at a simple relation such that lower price results in high parking demand while higher price gets low parking demand. Here we list two types of relations:

- Linear relation. $d(t) = \log_{10} f(t) - k * p(t)$, where k is a constant coefficient. The logarithm of $f(t)$ is simply because that $f(t)$ can be very large values according to our fitted results.
- Expoential relation. $d(t) = f(t) * \exp\{-\lambda p(t)\}$, where λ is a constant coefficient. As $p(t)$ increases, the $d(t)$ decreases and vice versa.

4.3 Price Optimization

The profit gain is defined as a function $g(T)$ that means the profit gained from the beginning to some time T, which could be described as an integral

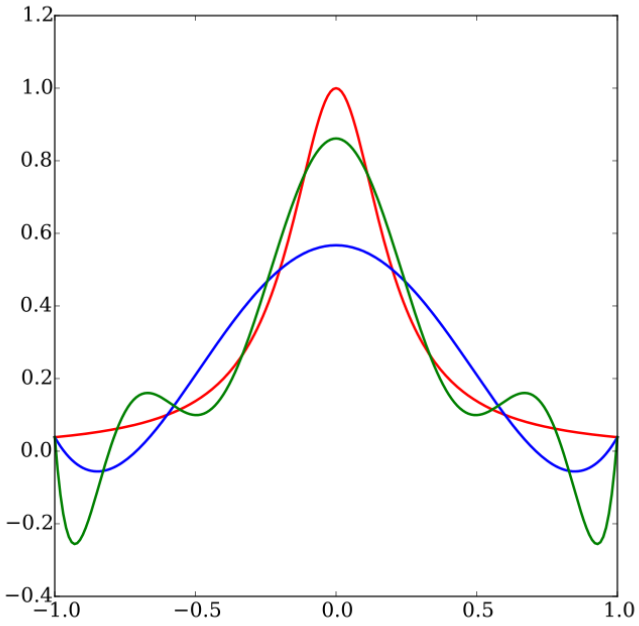

Figure 8: Example of Runge phenomenon. The red curve is the desired function. The blue curve is that interpolated with 5-th polynomial, while the blue curve is interpolated with 9-th polynomial.

process. The final objective function is as follows:

$$\max_{p(t)}\{g(T)\} = \max_{p(t)}\{\int_0^T p(t)d(t)dt\}, \tag{1}$$

where we propose to solve by simply taking the $\frac{\partial g(T)}{\partial p(t)} = 0$ as in [4].

4.4 Data Structure

The system is not using any data structure. Instead, all of the information regarding the user accounts on the website and information about the current state of the parking garage is being stored in a mysql database.

5 User Interface Design and Implementation

5.1 UC-1: Register

Navigation: 1 total click from the main page (click the register button)

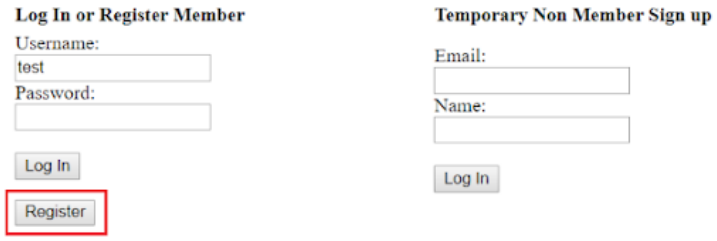

Figure 9: Register system.

Data Entry: total is indeterminant as keystroke per user will be different, 7 clicks

- On Register page enter the follow fields and click to next field
  - Name
  - Username
  - Email
  - Password
  - Confirm Password
  - Phone
- Select the submit button after all fields are entered

There have not been any significant changes we have made to this simple registration process. It has already been implemented on our website.

5.2 UC-3: Reservation

Navigation: total 3 clicks, indeterminant keystrokes as follows

- Enter Login information as follows
  - Enter Username

Figure 10: Register Page.

  - Enter Password

- Click Login button (server sends user to account main page) shown below figure
- On account page click Make a Reservation Page(user sent to make a reservation page) shown below

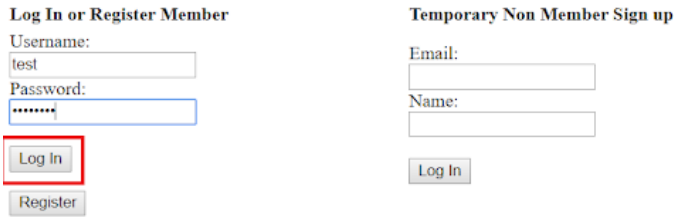

Figure 11: Register Page.

Data Entry: total is 5 keystrokes, 2 - 7 clicks

- Select date of reservation by using calendar tool (number of clicks depends on date)

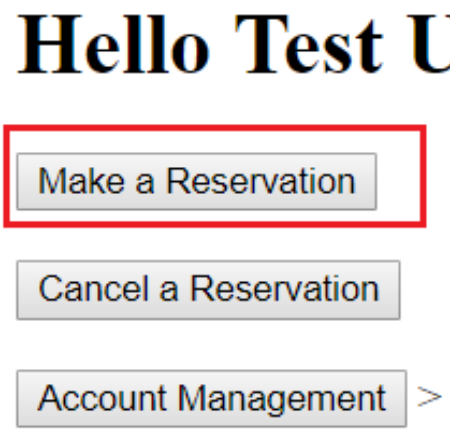

Figure 12: Welcome Page.

- Enter time of reservation in 24 hour time (5 key strokes → xx:xx)
- Select parking spot choice from the parking garage spot map tool (preliminary design shown below as the GUI has not been fully implemented yet)

There have not been any significant changes to our plan of utilizing this reservation GUI, but it has not been implemented yet. As of now, we have a field where customers can enter a date and select an available spot from a list of free spots (indeterminate keystrokes and 3 clicks to enter a time block as well as confirm a spot to reserve). However, we plan on lessening the user effort by having the user select a date (2 clicks: one for the date and one for the month) and an available spot (2 clicks: one to select a spot and one to confirm the reservation).

5.3 UC-3: Login

Navigation: total 2 clicks, indeterminant keystrokes as follows

- Enter Login information on homepage (shown above) as follows

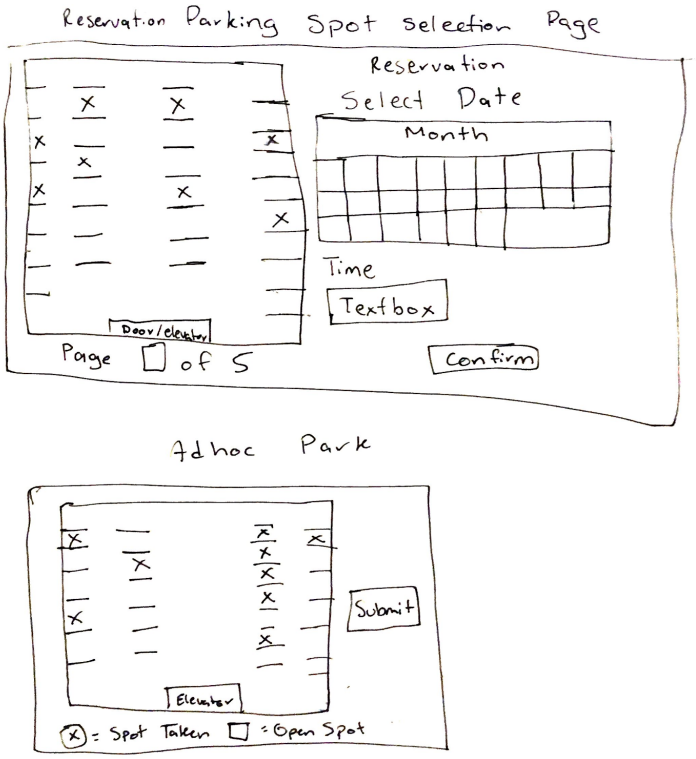

Figure 13: Login Page.

  - Enter Username
  - Enter Password

- Click Login button (server sends user to account main page)
- On account page click Make a Reservation Page(user sent to make a reservation page)

Data Entry: total is 5 keystrokes, 2 - 7 clicks

- Select date of reservation by using calendar tool (number of clicks depends on date)
- Enter time of reservation in 24 hour time (5 key strokes → xx:xx)

- Select parking spot choice from the parking garage spot map tool

This login process part of the website is already up and running. There have not been any significant changes besides ones intended to make the site more aesthetically pleasing.

6 Design of Test

6.1 Web Form Test Cases

1. Login: A registered user can input their username and password to log into their account

2. Registration: A new user can input their information and register for an account. All the data must be collect via a web form. Testing can be done via querying of database for the new user

3. Reservation: A logged in user can input time, day, and parking spot into a web form to reserve a spot. Users cannot reserve a spot that is already reserved for that same day. Testing can be done via querying of database for new reservations with correct user, date, time, and spot information

4. Cancel Reservation: A logged in user can cancel existing reservations. Testing can be done by checking database and validating that that the correct reservation has been cancelled

5. Update Information: A logged in user can update their user info via a webform. Users have to make their changes and click save to update their user info. Testing can be done via query of database and validating that the user data is up to date and the changes made via webform are reflected in the database

6.2 Garage Test Cases

6.2.1 Entrance gate

1. Main: main execution loop, this method should be able to continually accept customers coming in. Also it should send new and registered

users to the correct handling function (handleNew or handleExisting). Testing can be done by hard coding a license string and testing to see if the function sends the user with the associated license string to the correct handling function

2. checkResTime: checks a list of reservations and determines if any are within the acceptable checkIn times for a reservation. Testing can be done via giving a list of reservations in the form of tuples (reservationID (int), userID (int), DateTime (datetime), CheckIn (boolean), ParkingSpot (int)). Then, by inspecting the result and making sure that either no reservation is returned in the case of no reservation check in allowed at the current time or a reservation is returned if the user has a reservation made for the current time

3. handleNew: handling function for new users. This function gets new user data, inserts into db and allows the new user to select a spot to do ad hoc parking. This function can be tested by sending in an unregistered license string and starting the function. After input of data the tester can query the database to see if the new user data has been inputted. Also the test should check to see if the function sends the correct spot based on user choice to the elevator handlePerson function

4. handleExisting: handling function for registered users. This function should allow users to choose adhoc or check in to a registration. The test should have an existing user come in and try both adhoc and reservation check in and see if the correct spot is sent to the elevator handling function

5. scanPlate: This function reads the license.jpg file in the project folder and determines the license plate. This function is easy to test as all it does is return the license string with the highest confidence for the machine learning algorithm. The test should have the function called on many pictures and see if the license string return is correct

6.2.2 AuthCode

1. generateCode: This function just generates a random 8 character string. It can be called and the output just has to be an 8 character string

6.2.3 Billing

1. Email: this function determines the amount of time parked and bills the user likewise. The bill is sent as an email. The function takes in user email and hours parked so the test should pass these values in. The test can be considered successful if the email is sent successfully with the correct charge

6.2.4 Elevator

1. handlePerson: this function takes in (spot, resID, plate) and extracts the floor from the parking spot and "brings" the person to the floor if the floor is ready. To test this we have to first send a floor that is ready to be parked and see if the function calls the bringCar method to the floor. Then, to test the waiting part, the function should be called again while the first car is still parking and see if the wait functionality is operating. If the first park is successful and the second is queued then the function is operating correctly

2. Bringcar: this function spawns a thread to perform spot verifcation. The only test required is to make sure this function spawns a thread and then returns to the main

6.2.5 Notifications

1. sendAuth: This function send a text to the specified user phone with the 8 character code. This function can be easily tested by supplying a Twilio registered number and seeing if the function call results in a text from the system

2. sendWrongSpot: This test is the same as the one above however the message should just be different ("You are in the wrong spot")

6.2.6 TrafficManagement

1. isFloorReady: this function just returns a boolean based on the global list floorsReady. It can be easily tested by ensuring that the floor (index) supplied is the correct value based on the floorsReady boolean list

2. readyFloor: this function takes in a floor as a parameter and sets the index in floorsReady for the specified floor to True. This can be easily tested by checking the state of the floorsReady list after the function is called. If the specified floor index is now true the function is successful

3. Unreadyfloor: this function takes in a floor as a parameter and sets the index in floorsReady for the specified floor to False. This can be easily tested by checking the state of the floorsReady list after the function is called. If the specified floor index is now false the function is successful

6.2.7 ExitGate

1. scanPlate: same test as the function in entrancegate

2. Get_datetime_hours: this function queries the database and determines how long a user has been parking for. It returns a double value representing how many hours a user has parked. Testing this function is a bit more complicated as it requires interactions with the database. The function looks for the first occurrence in the ParkingHistory table of the user's license where the ending time is not specified yet. So to test this the program code to test the function must generate some synthetic start time for parking in the database and see if the function returns the correct elapsed time.

3. Main: this function is the main execution loop and controls when the user leaves the garage. It automatically bills registered users and asks unregistered/new users for their email for digital receipt. To test this function, new and registered user data has to be synthesized. To test registered users allow the function to scan a registered license plate and if the function sends the correct bill then the test is successful. To test unregistered users, give an unregistered license plate string and enter a valid email. If the function sends the correct bill then the test is successful.

6.2.8 Spot Verification

1. checkParking: this function takes in (spot,resID,floor,plate) and checks if the correct spot has been occupied. This function behaves differently based on if the spot checking is adhoc or reservation. For adhoc the

function just determines where the person parked on the floor they went to since adhoc users are allowed to park wherever they want to park. For reservations it makes sure the user parks in the specified spot and texts the user if the spot is not correct. There are 3 situations to test: adhoc parking, reservation successful parking and reservations unsuccessful parking. To test adhoc the occupy function in the simulation file can be used to simulate a sensor so if the function correctly watches a floor and determines that the adhoc user parked then the test is successful. The reservation success works the same way. For reservation unsuccessful the occupy function can be used to occupy the wrong spot and if a wrong spot text is sent then the test is successful

2. getUnoccupiedList: this function takes in the floor and returns a list of all unoccupied spots. This function is easily tested as the function returns data straight from the database so the data can be validated by comparing it to the database

3. unReserve: this function sets the reserved flag for a specified spot to 0. This function can be tested by checking the database after the function call

4. findSpotOccupied: this function takes two occupancy lists and determines what spot is occupied. It can be tested by sending occupancy lists and seeing if the result is the missing element

5. getPhonefromPlate: this function returns a phone number string based on the plate parameter. This function can be tested by a quick database check

7 Project Management

7.1 Merging the Contributions from Individual Team Members

Siyu Liao will work to format and compile the final version of Report 2. He will utilize Latex to format the report into a cleaner version that we can submit. In addition, we need to think about the uniformity of the diagrams in our report in order to appear more professional. We also need to uniformize the report formatting in order to maintain consistency between reports.

7.2 Project Coordination and Progress Report

Currently, we have implemented a website with basic functionality (register, login, and reservation). We have implemented most of the garage logic except for the user interface screen that customers see when they enter the garage. We have a license plate reader system that is currently in place that can also recognize ad-hoc users. Both ad-hoc users and registered users can park in the garage and are also able to switch spots if necessary. There is also a system in place that simulates entering and exiting the garage, as well as a way to allow users to manually input their license plate number in the interface at the front of the garage in case the license plate reader does not work properly. We also have a notification system in place so that customers can be sent confirmation codes, security codes, and payment information based on their preference of SMS or email notification.

We currently have a flat-rate payment system in place; however, we are working on implementing the dynamic pricing system we have completed a lot of research on already. The algorithm will be based on demand and time of day. In addition, we are working on implementing a manager account for the garage so that owners or managers can manually input information and alter some settings in the garage such as price for special circumstances. We are also working on creating the visual interface that will allow users to see all of the available spots in the parking garage. Even though we have a system in place that allows users to enter and exit the parking garage and switch spots, we are currently working on integrating and refining it.

7.3 Plan of Work

Below is a task list and Gantt diagram showing the estimated start date, duration, and end date of the upcoming tasks for the project. It is obvious that the task flow is heavily modeled off of the software development lifecycle. Descriptions of each of the tasks are also listed below.

Task	Start Date	Duration	End Date
Review Feedback	31-Oct	1	1-Nov
Redesign & Detail	1-Nov	4	5-Nov
Development	5-Nov	14	19-Nov
Testing	19-Nov	4	23-Nov
Final Development Cycle	1-Dec	7	8-Dec
Review & Bug Fixes	8-Dec	4	12-Dec
Demo2	12-Dec	0	12-Dec
Review Feedback & Bug Fixes	12-Dec	3	15-Dec

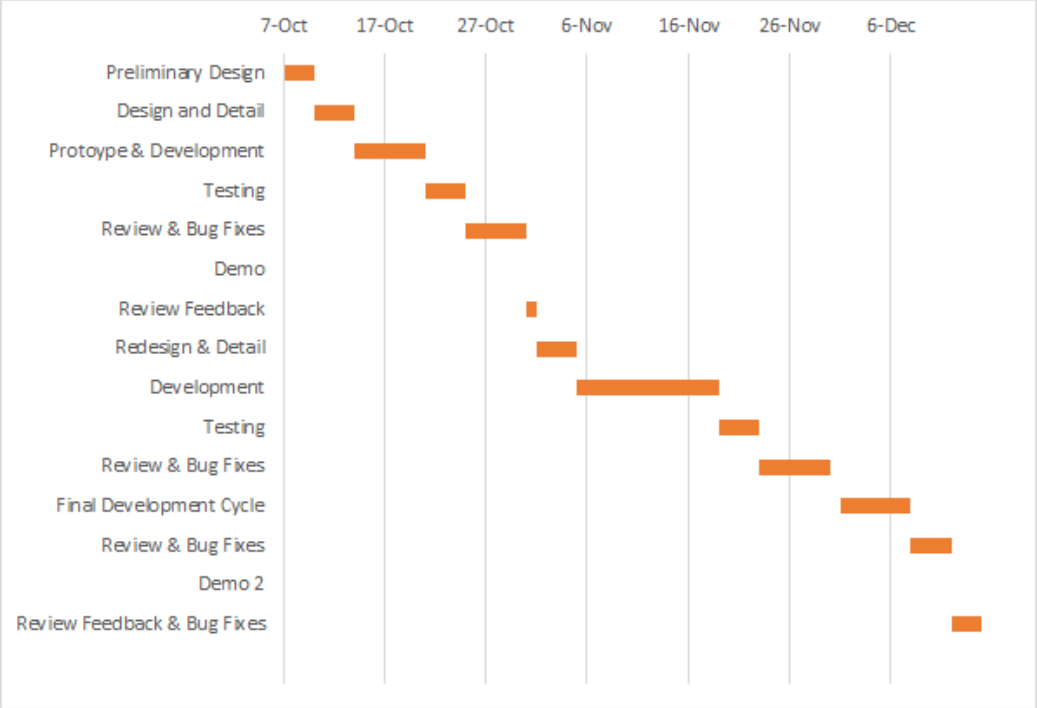

Figure 14: Our development plan.

7.4 Breakdown of Responsibilities

All members are contributed equally to the report.

Corey Chen:

- Database Functionalities

- CSS and HTML formatting for the website
- Login and Register modules
- Integration Testing
- Website Navigation
- Website/Database Communication

Chunhua Deng:

- Email notification module
- Exit garage module
- Cubic spline interpolation for fitting curve based on the data.
- Propose the price-based demand function.
- Total benefits Optimization.
- Final dynamic pricing system optimization
- Integration of the report

Jonathan Garner:

- SMS Notification Module
- Authentication Module
- Integration Testing
- Database Interactions

Siyu Liao:

- Data Retrieval Module
- Data Cleaning Module
- Data Processing Module
- Data visualization

- Calculate the price based demand function with multiple model.
- Profit optimization based on the demand function.

Siddharth Musale

- Database Creation and Management
- Entrance Gate
- Exit Gate
- Notification and Email module integrations
- Traffic Management
- Elevator
- Spot Verify
- Reservation page
- Open ALPR api integration
- Simulation
- All unit testing

Ridhima Sakhuja:

- Login and Registration (Website) Modules
- CSS and HTML formatting for website
- User info/password input validation
- Integration Testing
- Website Navigation/Database Communication

Xianglong Feng:

- Data cleaning
- Logic regression for fitting curve based on the data.

- Propose the price-based demand function.
- Total benefits Optimization.
- Final dynamic pricing system optimization

References

[1] "Spark overview." https://spark.apache.org/docs/latest/cluster-overview.html. Accessed: Nov. 4, 2018.

[2] C. Runge, "Über empirische funktionen und die interpolation zwischen äquidistanten ordinaten," *Zeitschrift für Mathematik und Physik*, vol. 46, no. 224-243, p. 20, 1901.

[3] Wikipedia contributors, "Runge's phenomenon — Wikipedia, the free encyclopedia," 2018. [Online; accessed 12-November-2018].

[4] Q. Tian, L. Yang, C. Wang, and H.-J. Huang, "Dynamic pricing for reservation-based parking system: A revenue management method," *Transport Policy*, vol. 71, pp. 36–44, 2018.