Parking Garage/Lot Automation

Software Engineering Second Report

Group 14

Tingcong Jiang, Chenyu Cao, Shijie Xu, Yiran Tan, Zhuohuan Li, Buyuan Lin, Samuel Cho, Christopher Cheng

https://cc1539.github.io/SE-project/

March 3th, 2019

Contribution Matrix

	Tingcong Jiang	Chenyu Cao	Shijie Xu	Yiran Tan	Zhuohuan Li	Buyan Lin	Samuel Cho	Christopher Cheng
Section Project Management	8%	7%	7%	8%	6%	6%	29%	29%
Section 1	17%	17%	17%	11%	17%	11%	8%	2%
Section 2	13%	13%	13%	13%	13%	13%	11%	11%
Section 3	13%	13%	13%	13%	13%	13%	11%	11%
Section 4							30%	70%
Section 5						35%	15%	50%
Section 6	20%	15%	15%	20%			15%	15%
Section References		25%						75%

Table of Contents

Section 1: Interaction Diagram	4
Section 2: Class Diagram	8
Section 3: System Architecture	17
Section 4: Algorithms and Data Structures	21
Section 5: User Interface Design and Implementation	22
Section 6: Design of Tests	25
Project Management	30
References	36

Section 1: Interaction Diagrams

Use Case 1: Registration

The registration is the most important idea since our system features (reservation, status checking, and etc.) are entirely based on registered account. We provide unregistered customers the services that nothing different from regular parking lot. So, to ensure that those unique features can be experienced by our registered customers, we need to make sure that our customers can easily use our system, by successful registration at first. For our website, it has the responsibility that allows customers to modify their accounts, such as registration, login, make reservations, and etc. Here it comes the first things first, registration.

For use case 1 Registration, the user who does not have an account for our website needs to enter our website first and click on the Register bottom to create an account. By entering the basic information of the user and the user's car, the user can create an account successfully and the user will be notified by our system that the account has been created successfully or not. User's information will be stored in our database, also the database will show the data stored successfully or not. Database can access the data that input by the user or the system to make a better communication between the user and the manger. Then the user can log in his account and make a reservation which is use case 2.

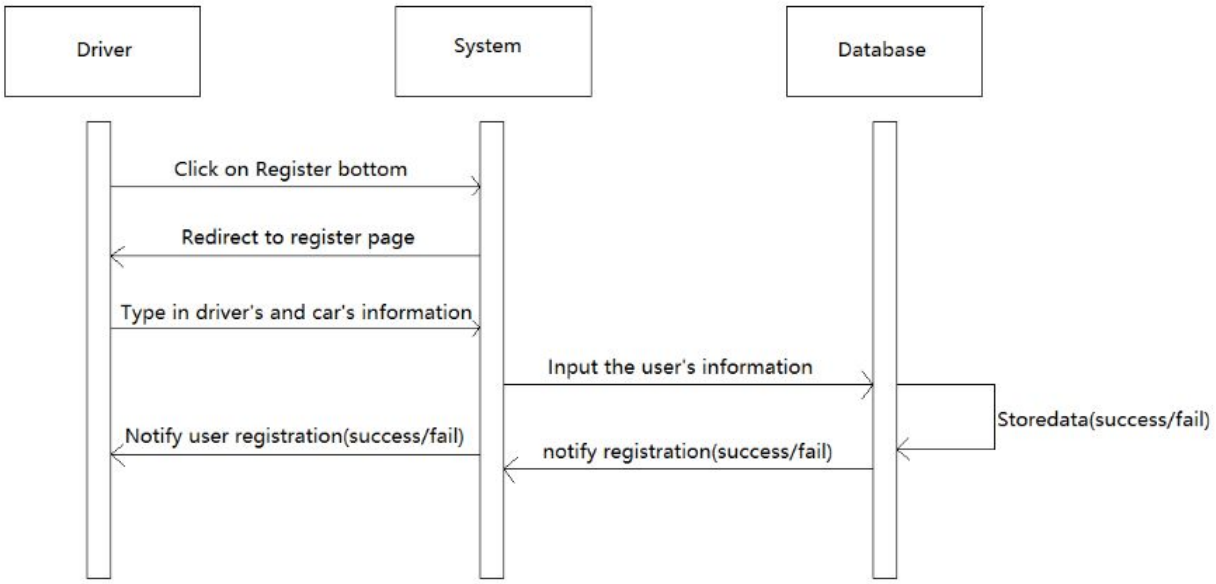


Figure 1: Sequence Diagrams for Use Case 1.

Use Case 2: Reservation

The assignment of responsibilities of each role is based on privacy and safety. All parking details and users information is unknown to the website. The only data is known to the website is the result of each test, and the website takes action based on each test's result. In other words, the website is a mediator. Since the parking information is changed rapidly, the parking spots information is not stored in the database for space and time concern. The detailed procedure of reservation is stated below.

User sends a page request by clicking the reservation button. By examining cookie data in the user's browser, the website will either return a reservation page or login page depending on whether the user is logged in or not. In the case of that user is not logged in, the user needs to log in or register an account in the returned registration page. If the user chooses to create an account, they will send their data to the website with a UserInfo container. The website will call an availability test to the Database. If the test passes, the user will be redirected to the reservation page, and the database will store user info into. If the test failed, the website will send the registration page to the user. Once the user is in the reservation page, they will send their reservation preference within a ResInfo container to the website. The website will call an availability test to the database. The database will request all available parking spots at that specific time from Garage. If the test fails, the website returns the reservation page to the user. If the test passes, the website returns a reservation success page, and send all reservation detail to the user via email or text based on the user's preference.

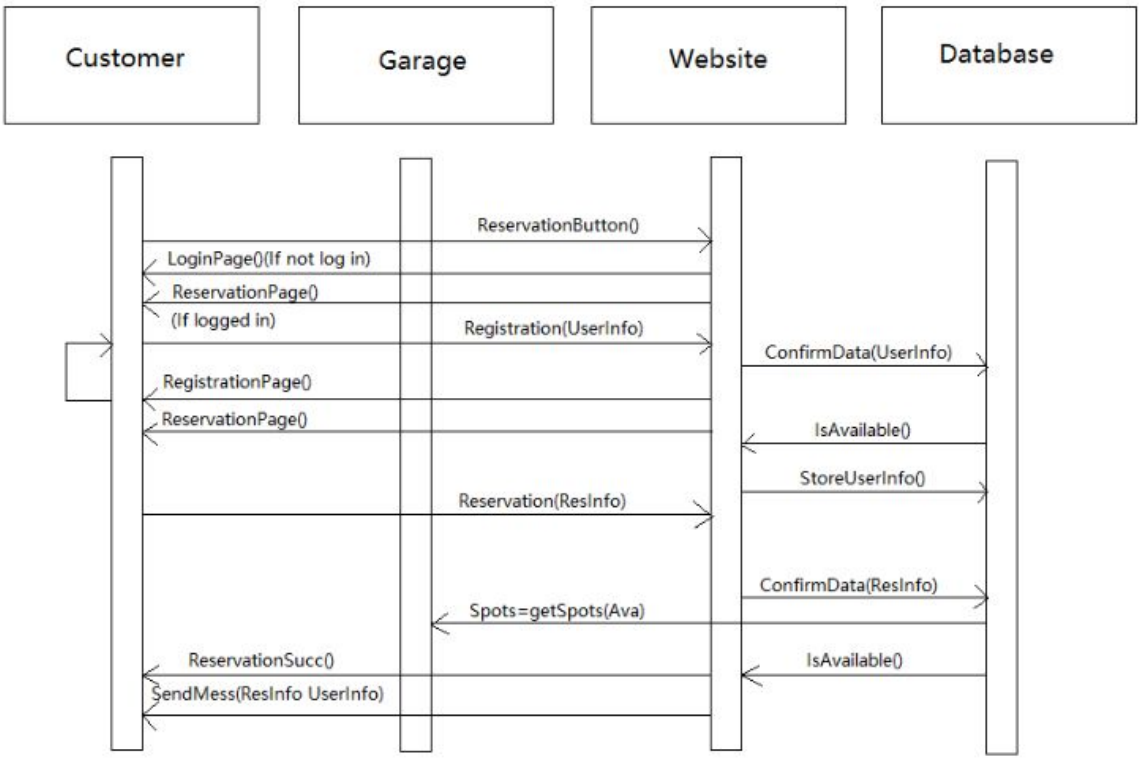


Figure 2: Sequence Diagrams for Use Case 2.

Use Case 3: Entering the garage

The reason we pick it as our fully dressed use case is because of its significance. This is the use case that providing basic functions as parking garage, while also very different from regular parking lots. The goal of our design (garage service) should be intuitive, fast and convenient. Therefore, it should be responsible for the basic parking function for unregistered customers, and the intuitive parking experience for our reserved customers.

For use case 3, The user drives to the gate of parking lot. The the License Plate Recognize System Scan the car plate. System first check customer if they have reservation. If they have reservation, LED screen will directly guide him to his reserved parking lot and Gate will open. If the customer does not have reservation, system will connect to database to search if any available spots. Database will feedback the searching results. If garage still have available parking spot, gate will open and system will guide customer to nearest spot. Else, parking spot is insufficient, gate will not open and the customer has to leave garage. After customer successfully park their car, system will mark the spot as occupied and update information on database.

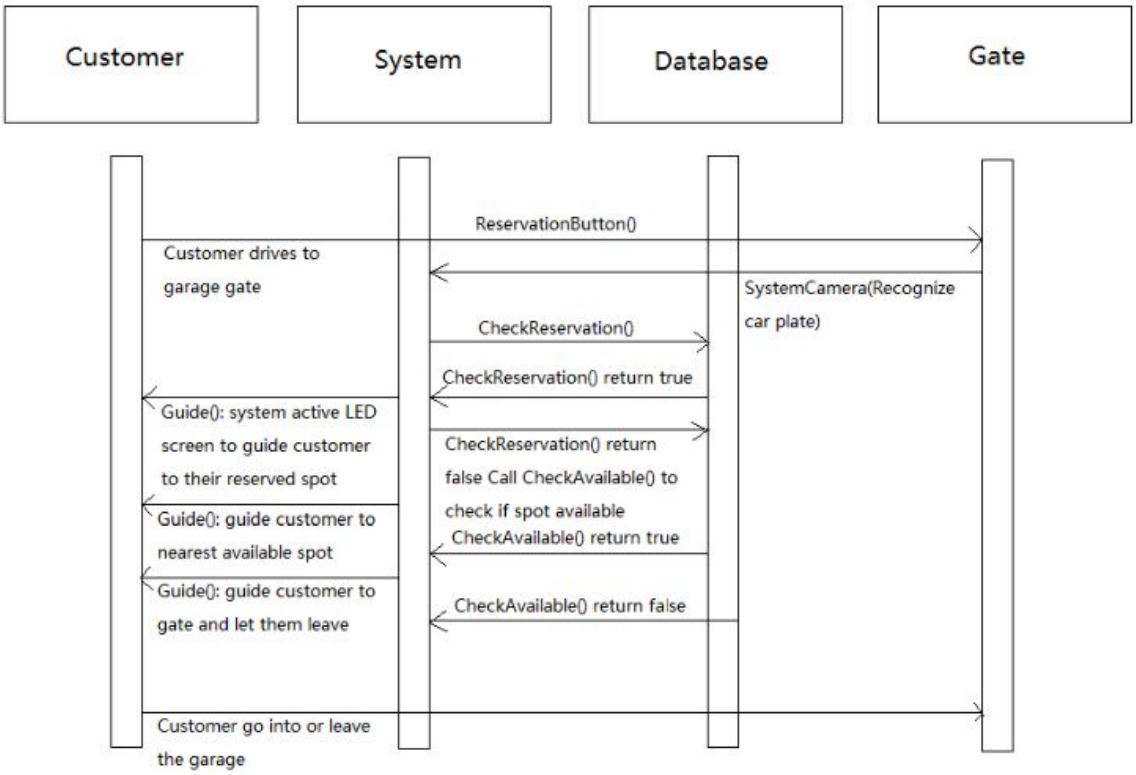


Figure 3: Sequence Diagrams for Use Case 3: Entering the garage.

Use Case 4: Online Payment

Our principle of designing this garage is to save as much time for the customers as possible. The automatic payment method is how we are going to achieve that. For registered customer, the parking fee will be automatically deducted from their account. In this use case, the data base plays the most important role. Entering and leaving the garage will trigger the camera, sending a signal to the database to start the time counting. When the customer leaves the garage, certain fee will be deducted from their balance. The receipt and rates will be available. When it comes to payment, security is our first concern. All the payment information are stored in the database and can only be altered by the customer, and can only be accessed by the database. The manager cannot alter or change the payment information without the customer's permission.

For use case 4 Online Payment, the customer will trigger the camera when leaving. The camera will get the license plate information and check if the customer is registered. If not then it will notify the user to use a valid payment method. If the user is registered the system will make the payment for the customer automatically. If the payment is done successfully, the gate will open. If not it will notify the customer to use a valid payment method. After that, the database will update the customers' information.

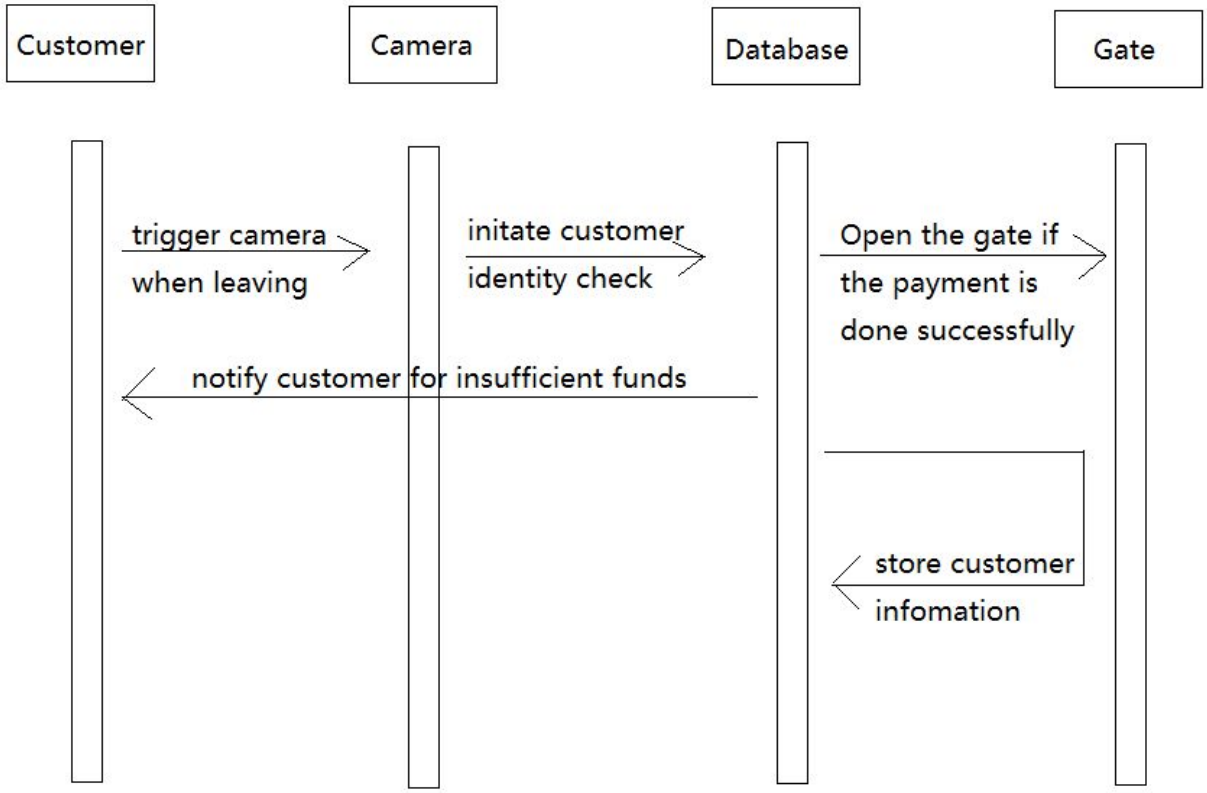
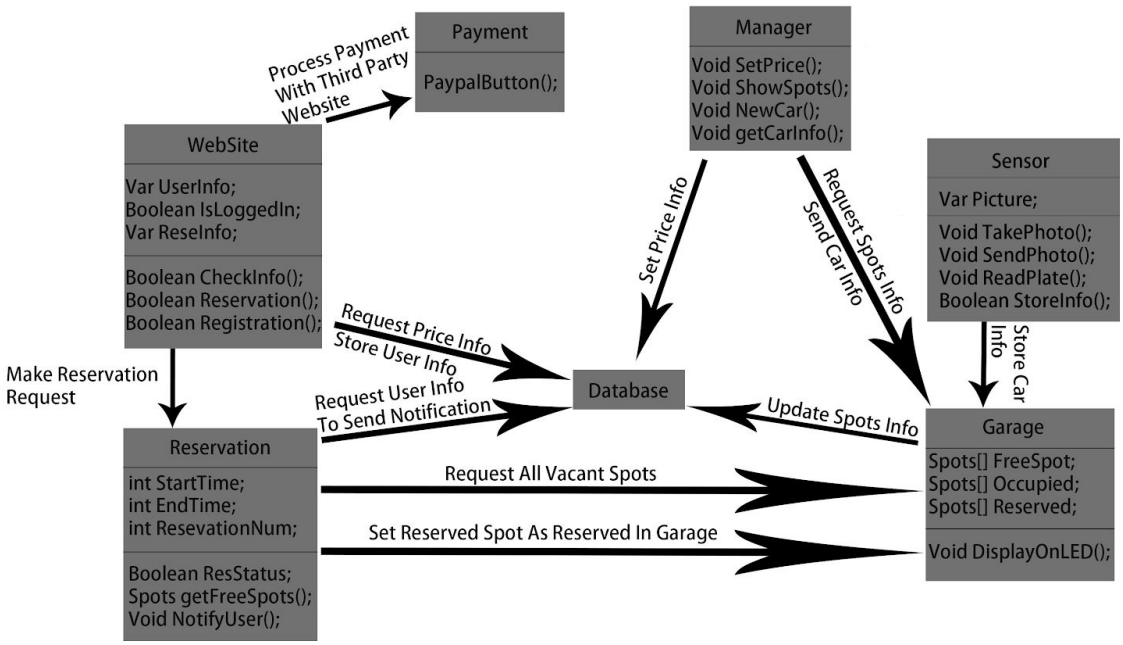


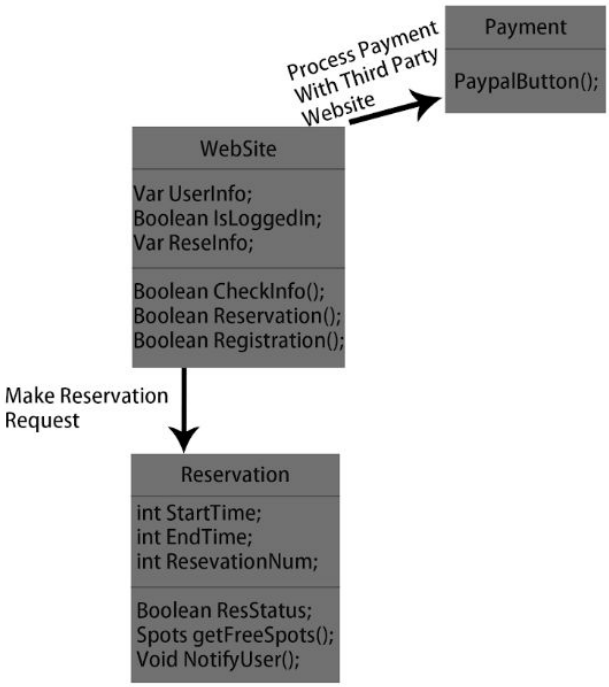
Figure 4: Sequence Diagrams for Use Case 4.

Section 2: Class Diagram

a) Class Diagram

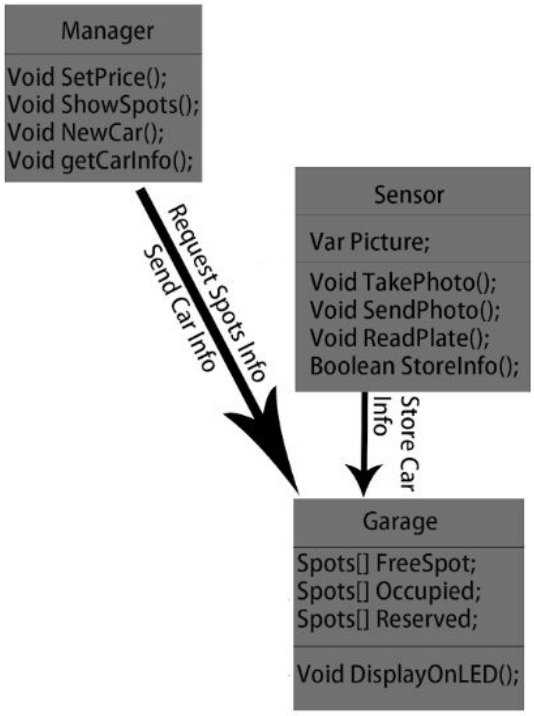


Subgroup #1: Payment, WebSite, & Reservation



This subgroup encompasses the frontend features of our system and defines our customer-facing functionality. The “WebSite” class interfaces with the “Payment” and “Reservation” classes, and allows customers access to the main features of our system.

Subgroup #2: Manager, Sensor, & Garage



This subgroup encompasses the backend features of our system and defines functionality available to administrators and parking lot owners. This group takes information from hardware to maintain and provide an updated representation of the parking lot.

b) Data Types and Operation Signatures

1. Manager

(1) Attributes

(a) String email: customer's email address for register.
(b) String first_name: customer's first name
(c) String last_name: customer's last name
(d) String phone_number: customer's phone number
(e) String account_number: customer's account number for login
(f) String password: customer's customized password

(g) Int customer_age: customer's age
(h) String creditcard_number: customer's credit card number
(i) String expire_date: expire date of customer's credit card
(j) String CVV: three digits of CVV code from customer's credit card

(2) Operation:
(a) Register(): Creates a new user account and put the information into the database.
(b) addCar(): Creates a car linked to an user account and put car's information into the database.
(c) addCreditCard(): Creates a credit card number linked to an user account and put card's information into the database.
(d) Reserve(): Creates a reservation linked to an user account and put reservation's information into the database.

The manager should have control over all the accounts and their information.

2.Camera and sensor

(1) Attributes
(a) Var picture: picture of customer's vehicle when the vehicle is entering the garage.
(b) Boolean occupied: Tell whether a spot is occupied or not

(2) Operation:
(a) CheckVehicleInfo(): check the vehicle's information with the the owner's information from database
(b) ScanVehiclePlate(): Scan the vehicle plate with picture photoed by the camera
(c) SenseOccupied(): Sense whether a car is on a spot

The camera sensor system should be able to recognize cars and determine if spaces have been occupied.

3.Timing System

(1) Attributes:
    (a) Int total_parking_time: total time after a vehicle entering the garage
(2) Operation:
    (a) Counting_time(): count the time after a vehicle entering the garage

The timing system should record how long a user has been in the garage based on their enter and exit times.

4.Ad-hoc Parking

(1) Attributes:
    (a) String Vehicle_Plate: ad_hoc's vehicle plate
    (b) Int spot_Num: the spot number of the customer's vehicle parked
(2) Operation:
    (a) SelectParkingSpot(): select the parking spot with spot id and spot number.

Allow walk in customers to use the parking lot.

5.Reserved Parking

(1) Attributes:
    (a) Int reservation_start_time: the start time of the reservation
    (b) Int reservation_end_time: the end time of the reservation
    (c) Int reservation_num: the reservation number generated when the customer make a reservation
    (d) Int spotNum: spot number of the spot reserved by the customer
    (e) String user_account: customer's account number
(2) Operation:
    (a) confirmReservation(): send successful reservation notification
    (b) pick_time(): let the customers to pick the time of reservation

Allow users to reserve a spot in the garage.

6.Garage

(1) Attributes:
    (a) Int num_of_vacant_spots: number of vacant spots
    (b) Int num_of_occupied_spots: number of occupied spots
    (c) Int num_of_reserved_spots: number of reserved spots
(2) Operation:
    (a) Direct_Vehicle(): direct the entering vehicles to their optimized spots.

The garage should keep track of the status of each spot to help users find an empty spot.

7.Website

(1) Attributes:
    (a) String Username : used by a customer to login to the website.
    (b) String Password : customer's account password linked with account number in the database
    (c) Integer AccountID : Uniquely identifies a customer using a number.
    (d) String SessionID : Identifies a customer's login session.
    (e) String AccountType : Determines an account's privileges.
    (f) String EmailAddress : The customer's email address.
    (g) String HomeAddress : The customer's mailing address.
    (h) String LicensePlateNum : The customer's car's license plate number.
(2) Operation:
    (a) send_information(): send customer's information to database
    (b) receive_information(): receive information from database

The website will allow users to create an account and use it to look at their parking information.

c) Traceability Matrix

1. Database

(a) Responsibilities

(i) Store customer and reservation information

(ii) Store price information

(iii) Store parking spots information

2. Payment

(a) Responsibilities

(i) Process payment via a third party API

(b) Classes

(i) PayPalButton();

3. Manager

(a) Responsibility

(i) Set parking price

(ii) Add new car into database

(iii) Show all spots and car information to administrator

(b) Classes

(i) SetPrice();

(ii) ShowSpots();

(iii) NewCar();

(iv) getCarInfo();

4. Website

(a) Responsibilities
   (i) Acquire customer information
   (ii) Acquire reservation information
   (iii) Showing garage information

(b) Classes
   (i) requestPriceInfo()
   (ii) storeUserInfo()
   (iii) processPayment()
   (iv) makeReservation()

5. Sensor

(a) Responsibilities
   (i) Take photo when pressure sensor is triggered
   (ii) Send photo to manager software
   (iii) Send photo to car plate reader

(b) Classes
   (i) TakePhoto();
   (ii) SendPhoto();
   (iii) ReadPlate();
   (iv) StoreInfo();

6. Garage

(a) Responsibility
   (i) Show all essential information on LED monitor

(b) Classes
   (i) DisplayOnLED();

7. Reservation

(a) Responsibility

(i) Retrieve all available spots from Database

(ii) Send notification to user regarding reservation status

(b) Classes

(i) getFreeSpots();

(ii) NotifyUser();

	Domain Concepts						
Classes	Database	Payment	Manager	Website	Sensor	Garage	Reservation
Customer Registration	✔	✔		✔			
Customer Login/out	✔			✔			
Garage Gate	✔				✔	✔	✔
License Plate Scanner	✔				✔	✔	✔
Notification System	✔		✔				
Security System	✔		✔		✔	✔	
Spots sensor	✔				✔	✔	
Database Managing	✔		✔			✔	

Database Connection	✔	✔	✔	✔	✔	✔	✔

Section 3: System Architecture

a) Architectural Styles

The Inspired Auto Parking System will follow a Master-Slave pattern. This pattern consists of two parties: the master and the slaves.The master component distributes work to each slave component and slave components return results to the master component. The Backend management system is our Master component. Every simple task distributed by the Backend management system will be handled by each slave component (like the camera, parking sensor, or LED screen). The Master will manage and distribute tasks to the slaves and the slaves send feedback results. There will be a lot of independent features which will be accomplished simultaneously so the bus is very important. The bus will connect each slave component to the master component. As the data sharing and component connection, the bandwidth of the bus is critical. All data (including customer's information, account info, parking status) are stored on our SHA-256 encrypted database on our server. So our server is our Master and all other sensors or monitors etc. are slave components.

Our service lives on the Internet which can be accessed through Inspire Auto Parking's website. Parking lot owners can register as administrators by providing parking lot proof-of-ownership and a valid key to access the backend management toolset through the website or application. Therefore, our working style is a Service-Oriented Architecture (SOA).

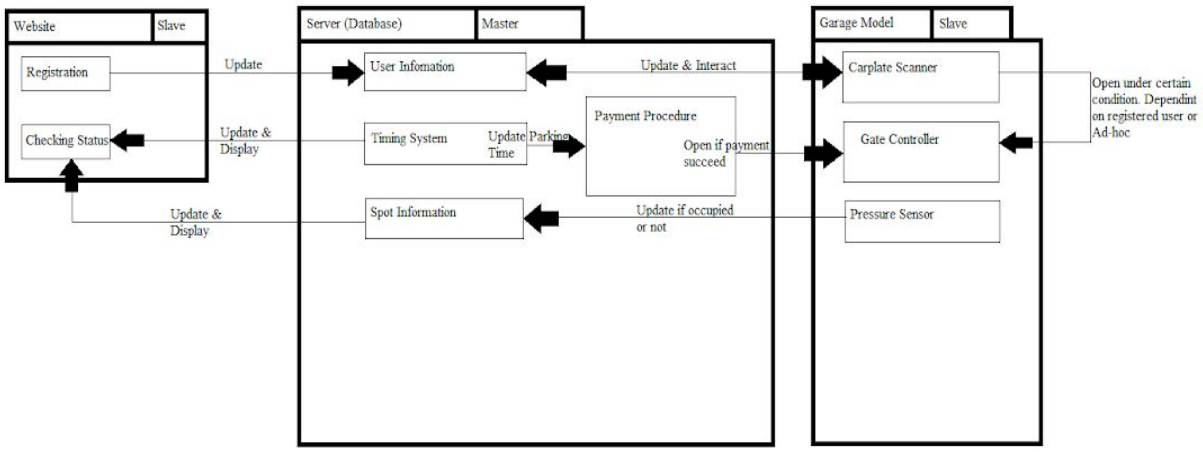
To reduce energy using and cost, server will take charge primary component functional. Even though make every slave component solve some simple task will reduce server load but this will increase cost. Our goal is to be economic. So we choose to use algorithm to optimize stability and reliability.

b) Identifying Subsystems

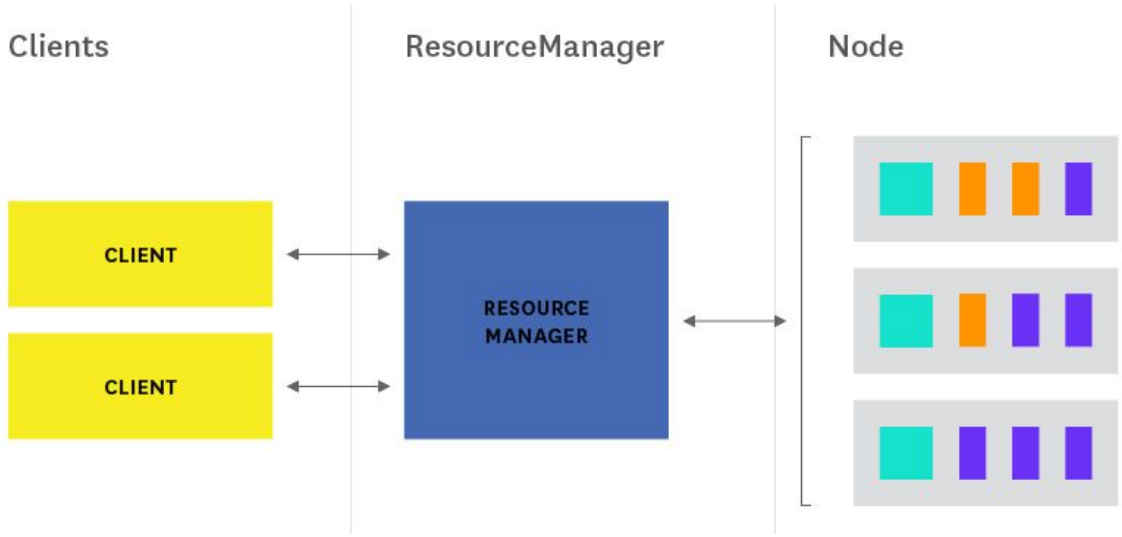
In our "Master-Slave" structure, there will be three main subsystems: The server (database); the website (User interface) and the hardwares (the garage demo model).

The “Master” in this structure is the server, both the website and the hardware will take order and report to the servers.

c) Mapping Subsystems to Hardware



The master server should be able to collect data correctly from multiple devices. The hardware includes a car plate scanner and a pressure sensor. Beside those hardware devices, the server should also be able to interact with the website, providing the desired and correct data to the website for the user to check their status.



d) Persistent Data Storage

A database will be used to store the information of the users and spots in our system. The users' payment/contact/car information will be stored. Also, the current status of spots will be always updated and stored for the reservation. Moreover, the history of spots will be recorded(Time,Occupied/Reserved or not,User,Payment)

e) Network Protocol

We decided to use TCP/IP as our network Protocol because TCP/IP is wide-used mature protocol. Our server will use ECS which is safer and convenient and costless. Our website will obey TCP/IP protocol which is a universal Internet protocol so that different computer could access to our website. Our website will be a simple HTML page which will accomplish several "manage" behavior.

f) Global Control Flow

Execution orderliness:

The system is both event driven and procedure driven. The sensor is waiting for the "Car Enter" event to wake the sleeping camera to take a picture and recognize the license plate which means the system is event driven. Moreover, the system is procedure driven because after recognizing the license plate, the system will check the users' information and decide the next step.

Time dependency:

The system has a reservation and payment feature. Therefore, we need a timer to determine the cost and update the status of the spots. Also, the system will notify the user based on real time to make sure they will not park overtime unintentionally.

Concurrency:

All parked cars share the same server. All updates and commands transfers simultaneously. Our service is completely web-based. Any “threads”, if any, are handled by Apache, MySQL, PHP, or any other server-side services that already exist and not by the developer.

g) Hardware Requirements

1. Website Host Server (Planning to use ECS)

- 1GB RAM
- 10GB Disk Space
- Firewall Applicable
- 100Mbps Connection Speed

2. Database

- 4GB RAM
- 64GB Disk Space
- 1Mbps Wired Connection through USB2.0+

3. Garage Hardware

- Camera for License Plate Recognition
- Sensor for detecting parking spot occupation
- Small LED Screen for displaying instructions

Algorithms and Data Structures

Algorithms

In order to implement Use Case #3, “Entering the Garage”, we must select a shortest-path algorithm that allows use to quickly determine the most efficient path from the entrance to an available space. Each parking lot will be modeled by an undirected graph, where each edge begins with a certain weight depending on the length of the path segment it represents. These weights may be modified accordingly to reduce congestion as traffic increases. Since the graph contains rings, we will implement Dijkstra’s algorithm, which deals with rings sufficiently well. Since our edge weights can mutate as traffic changes, we cannot use a precomputed table of paths for each parking space.

Data Structures

Our system uses a multidimensional array to represent each space. We chose to use a multidimensional array because parking lots are often spatially organized in a grid-like manner. In the case that a parking lot is more unconventionally designed (e.g. it is shaped like an L or like a circle) we believe that the speed advantage of accessing an array outweighs the losses incurred by the space wasted by unused entries in the multidimensional array.

Our system utilizes an undirected graph to represent the paths that can be taken around the parking lot. We chose this representation over alternatives such as trees because a graph is more versatile and easily lends itself to representing multidimensional information. We chose to make the graph undirected, since this reflects that fact that a car could be moving in either direction along a particular path segment.

User Interface Design and Implementation

Our initial user interface designs depict a website composed of various separate pages, each reserved for their own special function. Our current design and implementation instead consists of an easily accessible page that itself offers access to the full functionality of our service to the customer. This reduces the amount of clicks taken to access a certain feature and the time taken to become adjusted to the structure of a website, thus decreasing overall user effort. To organize our features and help customers compartmentalize them in their minds, we modularize our user page in the form of "tabs" that help a customer quickly access the desired feature.

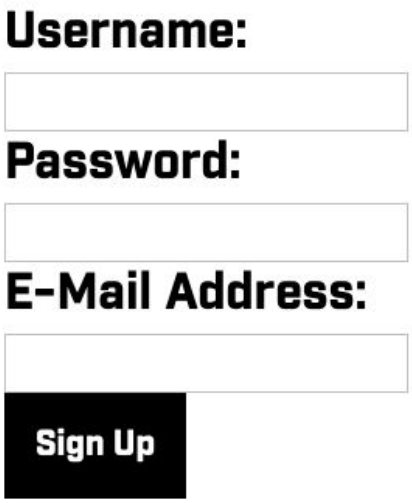
Our website is designed to be highly streamlined and efficiency-oriented. We take pointers from what seems to be a modern trend of minimalist UIs, which are defined by simple blocks of solid color and only the occasional outline. This increases the "ease-of-use" of our website and further reduces user effort.

UC-1 Register

When clicking on the Sign in button, the user will enter an interface like this:

Like other common websites, for unregistered users, there is a register option on the sign in page.

After clicking on the register button, the user will be directed to the registration page:



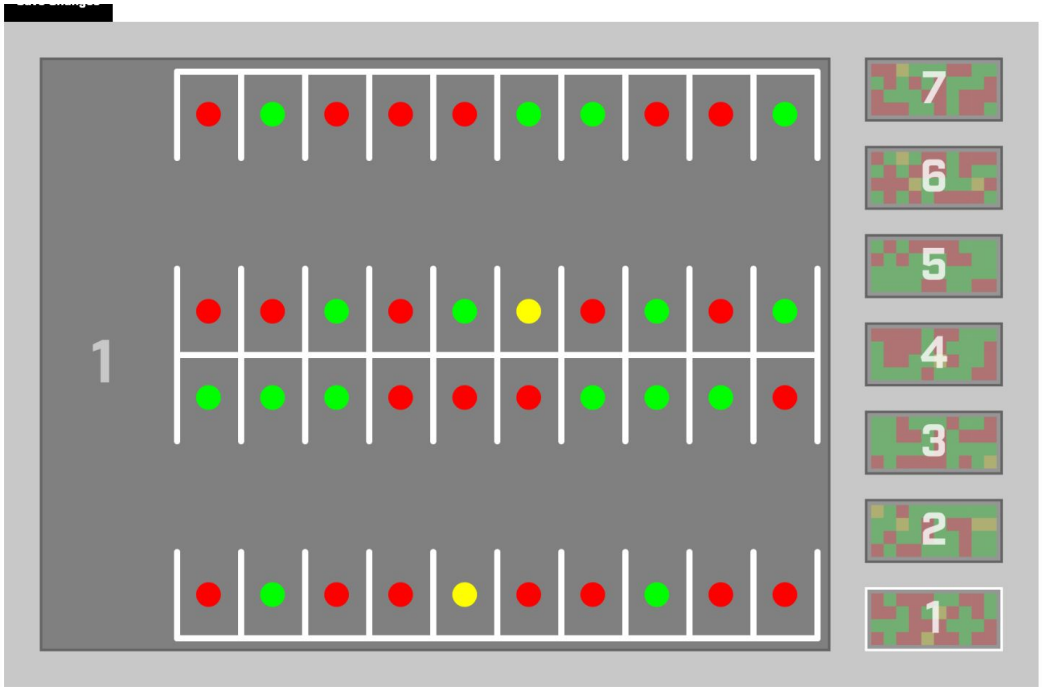
After filling in all the information, click the Sign up button and that's the end of the registration process.

UC2- Reservation

After the registration process, the user can sign in to their account on the sign in page. After doing that, the user will be navigated to their account information page, where they can add more detailed information.

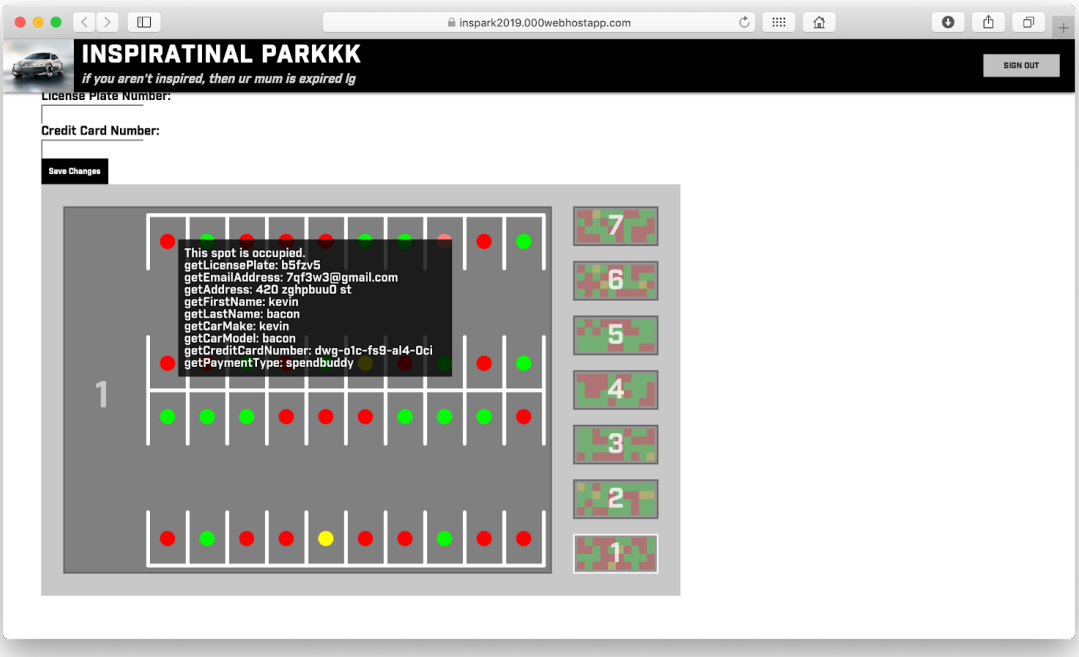
This page allows the user to add more detailed information so that they can use the reservation and auto pay function.

Our website contains the map of the garage and the status of all parking spots (red means its occupied, yellow means its reserved and green means it's vacant.)

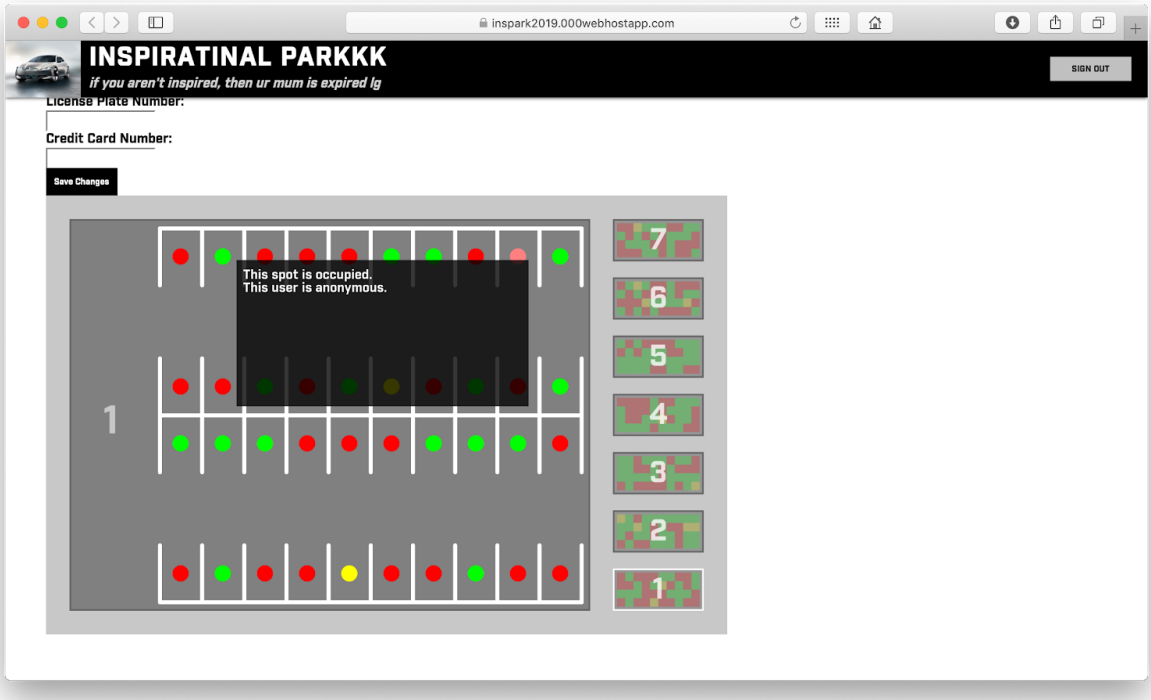


And for the manager, if the manager puts the mouse on each parking spot, the website will show the information of the vehicle that's parked there.

For registered user:



And for walkin-customers, there's no information available:



Design of Tests

For our first demo, we will only implement and test the use cases most critical to the functioning of our service, which happen to be our first five use cases:

UC#1 - Registration

A user should be able to register for an account by providing a unique username, password, and a valid e-mail address. If an account already exists with the given username or e-mail address, registration should fail. Otherwise, the account is created and the user should be able to login using the same credentials.

Test coverage: For the user name, we will proceed this test with three different inputs which are new username, duplicate user name, and invalid user name. For the password, we will proceed this test with two kinds of inputs which are valid password, and invalid password. For email address, we will proceed this test with the 29 test cases provided by the cited website.

UC#2 - Reservation

A user should be able to submit a reservation for a particular parking lot. An ordinary user should only be able to reserve one spot ahead of time; should they attempt to reserve a different spot, they will be asked whether or not they want replace their last reservation with their current one.

Test coverage: The accuracy of the reservation time and user name will be covered. Also, users cannot reserve a spot already be reserved at the same period. Moreover, the old reservation will be cancelled if the user successfully reserve another spot.

UC#3 - Entering the Garage

A user should be able to enter the garage when there is available spot in the garage or the user has a valid reservation for that specific period of time. The garage door should be opened when the user's vehicle is stopped at specific spot for scanner to run validity test, and the user's vehicle passes the validity test. If the user's vehicle did not pass the validity test, the garage gate should be remain closed. All relevant information should goes to the LED screen in front of garage gate.

Test coverage: We will simulate a small parking lot with four parking spots, and 2 car models. For the entering garage, the car should orient to a different angle from the camera. There are three situation we have considered. The first situation is when the

garage is empty. The second situation is when there is a car with no reservation entering a full garage. The third situation is where there is a car with reservation entering a full occupied garage.

UC#4 - Status Checking

A user should be able to see his/her parking status, including the current parking time, parking spots and parking fees. If the user is currently not parking, then the user should see nothing with status checking. When the user leaves the spot, the sensor should notify the database and change the status checking site with updated information.

Test coverage: Parking status, parking time parking spot and fees of customer will be tested through data change in database. Status check interface will be tested through different customer accounts. Also, the parking sensor will be tested using car models.

UC#5 - Online Payment

Users should be able to set up online payment options for their accounts. User should be able to choose between Paypal and Credit Card for online payment. If the user don't want to link any account, they should be able to make deposits to their balance as online payment funds.

Test coverage: The amount of the funds should be accurate after payment finished or successfully recharged. The online payment should not charge the same order twice. Also, the payment should be done for the user whose account the charge is tied to.

7.3 Integration testing strategy We will act as a normal user, and follow the procedures below.

Test Case ID	Test Case Objective	Test Case Description	Expected Result
1	Register an account	Enter User name and Password to create an account.	For Invalid input, the account will not be successfully created. And the user will be notified. For valid input, the account

			will be created successfully and Customer Information will be stored in database
2	Reservation	Select a period of time on the reservation website.	Get a notification regarding reservation from website.
3	Enter garage	Drive the car to the front gate of the garage and enter.	The gate should open once the car arrives the front gate with no other problem.
4	Status Checking	After parking at the reserved spot, user wants to check parking records on the phone.	The website will show the reservation information and a text message will be sent to the user's phone.
5	Online Payment	Pay for current parking fee or add balance into account	Current parking fee successfully paid. Customer Status show paid and gate opened. If customer choose to add balance into account, then database should update the new balance and account should present the accurate balance.

Project Management

a) Merging the Contributions from Individual Team Members

Describe what issues were encountered and how they were tackled.

An issue we encountered was properly communicating with all the team members and being transparent about what was being worked on by each member of the team. Some issues that arose from this were that different team members were working with different types of UML software. We decided to keep this layout for now in order to distinctly identify the differences between the 4 use cases we have fully dressed. Since then, we have standardized them to look the same.

b) Project Coordination and Progress Report

Use Cases

A prototype of UC#4, “Status Checking”, has been implemented. For now, the virtual map shows a mock representation of a seven-story parking lot where each spot is either occupied, unoccupied, or reserved. Hovering the mouse over a spot marked as occupied or reserved shows information about a fake, procedurally generated account.

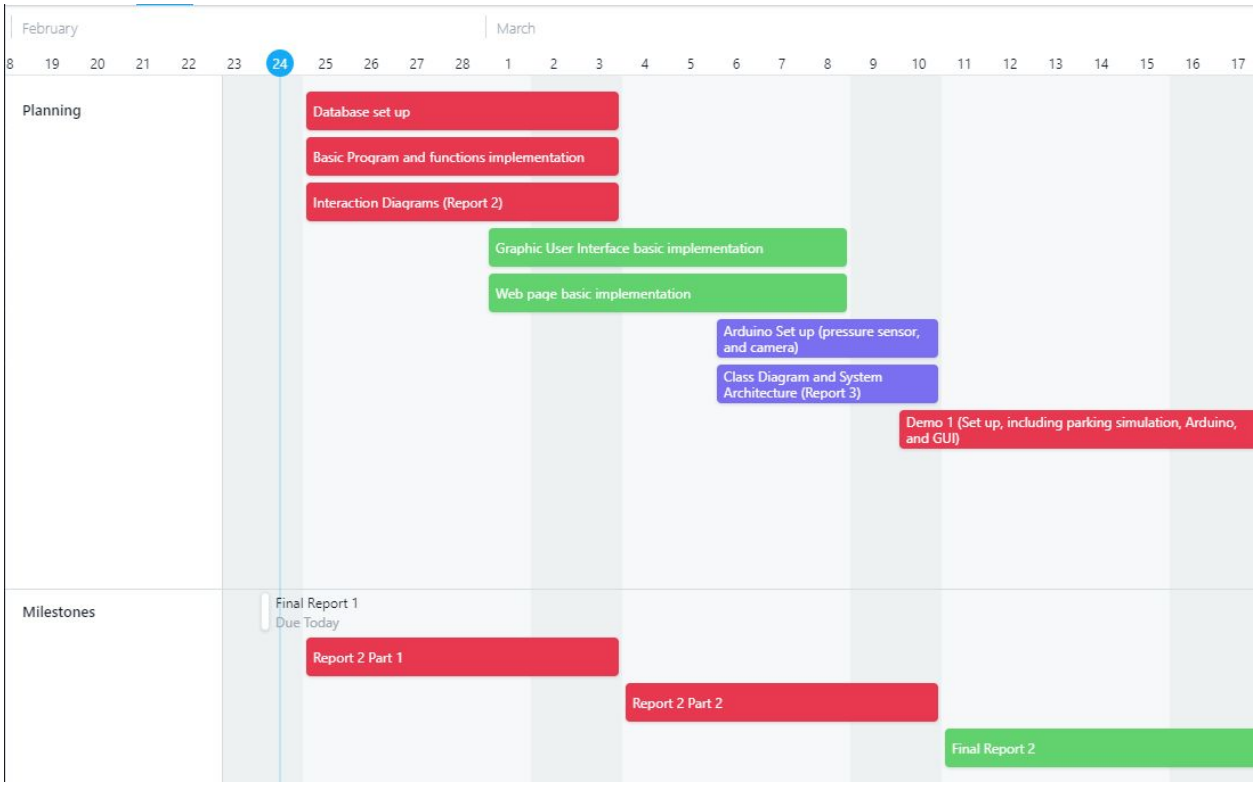
Implemented Functionality

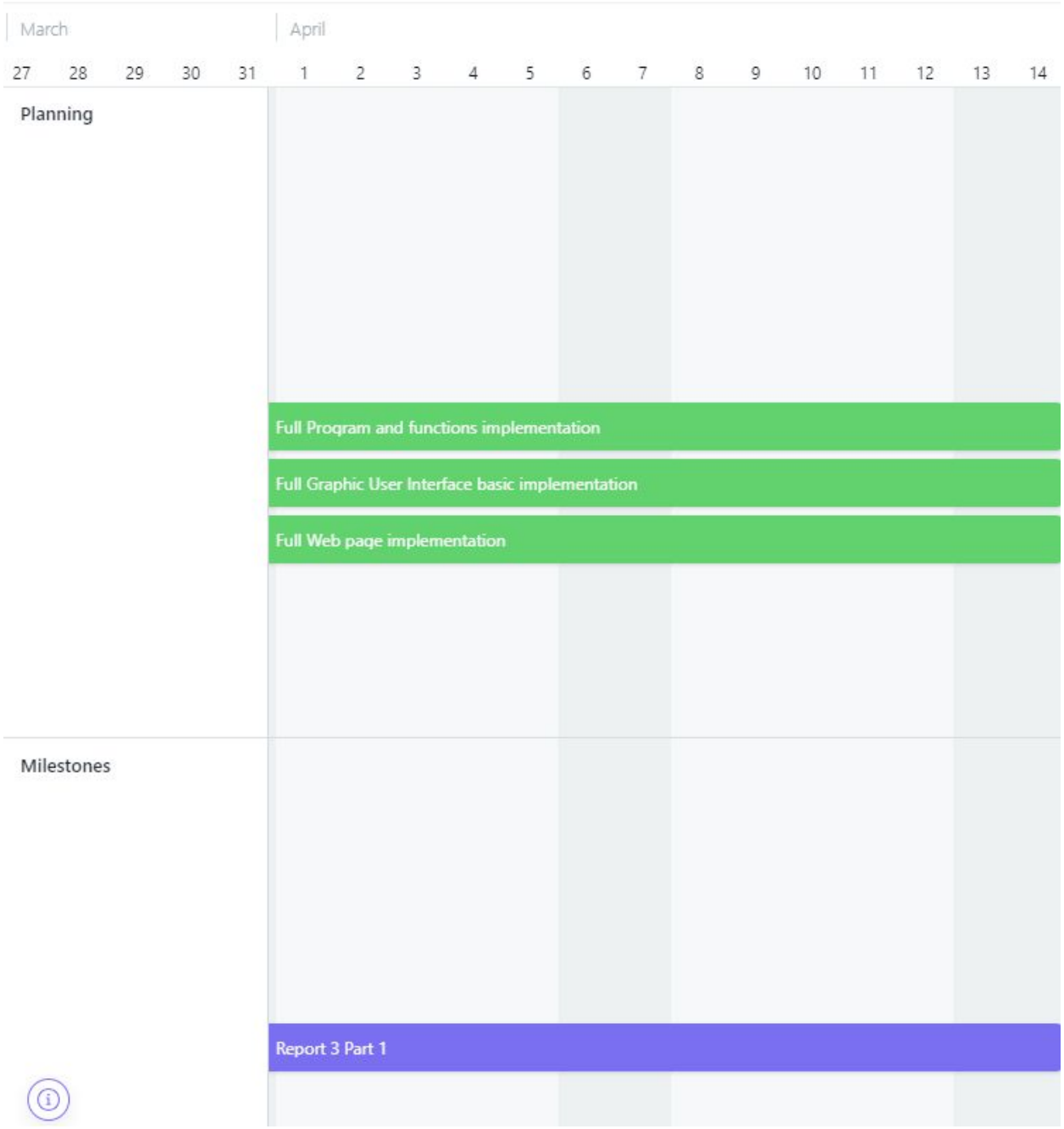
A demo website is up and running on Github pages. Currently, the only thing a user can do on the site is view the current implementation of UC#4 in the form of an interactive, virtual map, but only after logging in using predetermined, hardcoded credentials.

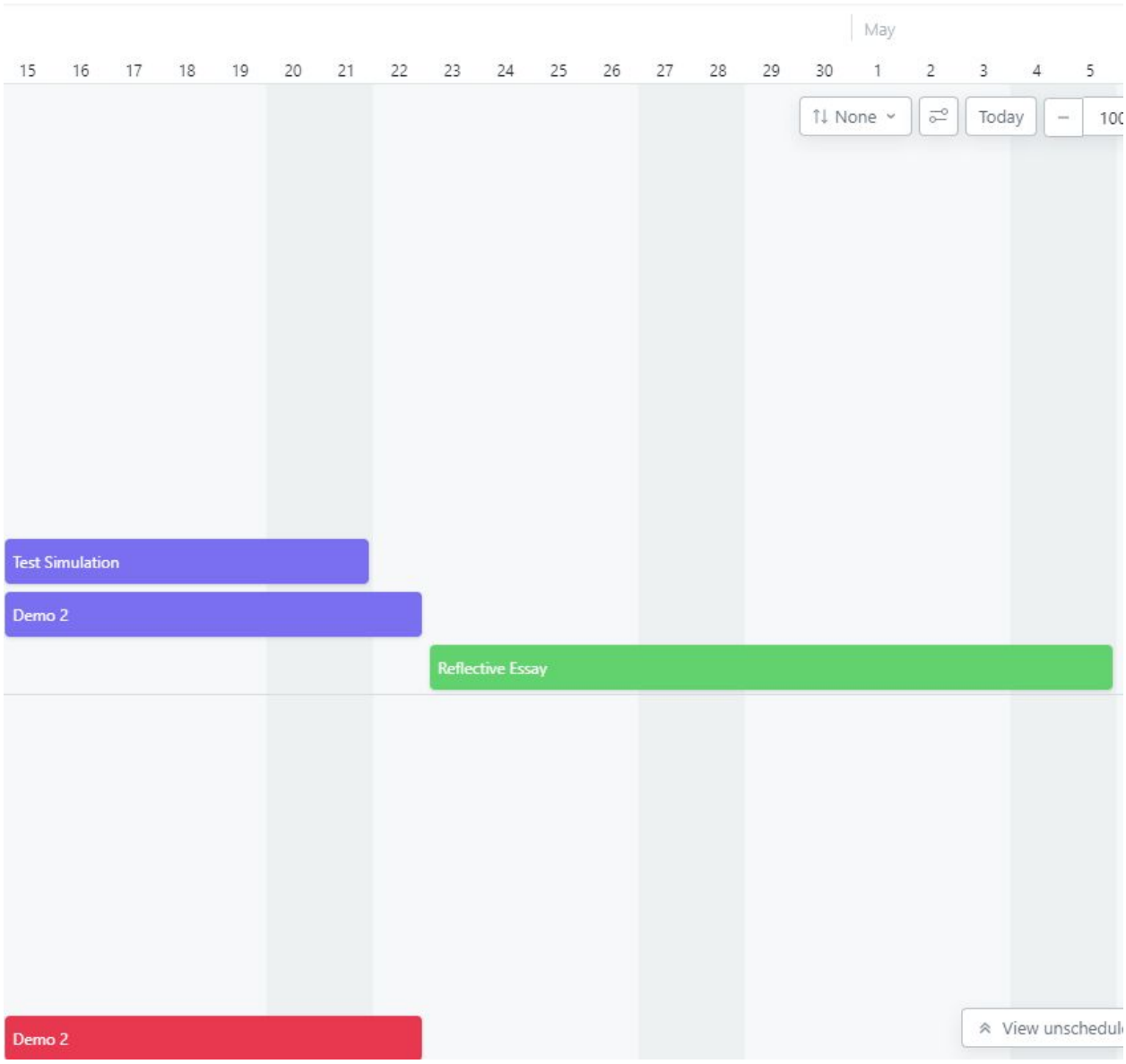
Problems

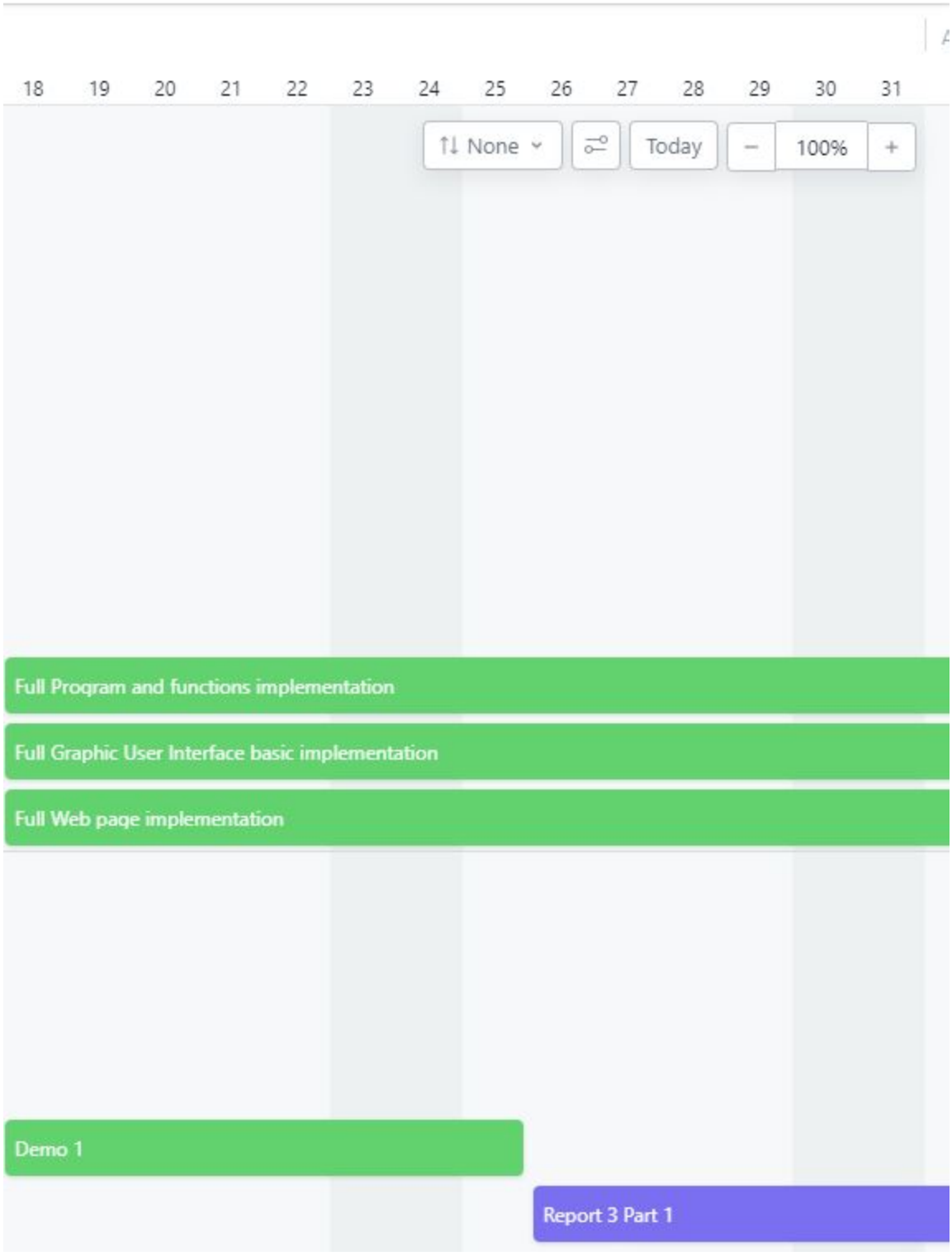
One of the reasons why the registration system hasn’t been implemented yet is that Github pages doesn’t support “dynamic” websites, so it is impossible to save any information server-side using SQL or PHP. Therefore we will eventually have to either find an alternative hosting site or host it ourselves locally. We may opt to implement a client-hosted database first, for demo purposes.

c) Plan of Work




March
April
27
28
29
30
31
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Planning
Full Program and functions implementation
Full Graphic User Interface basic implementation
Full Web page implementation
Milestones
Report 3 Part 1


May
15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5
None
Today
Test Simulation
Demo 2
Reflective Essay
Demo 2
View unschedul


18 19 20 21 22 23 24 25 26 27 28 29 30 31
None
Today
100%
Full Program and functions implementation
Full Graphic User Interface basic implementation
Full Web page implementation
Demo 1
Report 3 Part 1

d) Breakdown of Responsibility

Subgroup #1 - (Software & Communication)

Members:

- Christopher Cheng
- Samuel Cho

Contributed Functionality:

- Parking Spot Reservation
- Parking Lot Visualization

Contributed Qualitative Property

- Develop clean and easy-to-use online UI

Subgroup #2 - (Direction and Account System)

Members:

- Zhuohuan Li
- Tingcong Jiang

Contributed Functionality:

- Account-User Information Binding
- Customer Account Registration

Subgroup #3 - (Hardware + Arduino)

Members:

- Buyuan Lin
- Yiran Tan

Contributed Functionality:

- Parking Lot Status Acquisition
- Parking Lot Management

Subgroup #4 - (Test & Development)

Members:

- Chenyu Cao
- Shijie Xu

Contributed Functionality:

- Online Payment
- Data Submission

References

Roser, M. and Ortiz-Ospina, E. (2019). *World Population Growth*. [online] Our World in Data. Available at: https://ourworldindata.org/world-population-growth [Accessed 6 Feb. 2019].

Hedges & Company. (2019). *US VIO Vehicle Registration Data 2018, Fast Quote on Car Data*. [online] Available at: https://hedgescompany.com/automotive-market-research-statistics/auto-mailing-lists-and-marketing/ [Accessed 7 Feb. 2019].

Rasmussen, I. (2016). *A Major Parking Garage Problem*. [online] Strong Towns. Available at: https://www.strongtowns.org/journal/2016/11/21/major-parking-garage-problem [Accessed 8 Feb. 2019].

https://www.quora.com/What-is-Hadoop-YARN-Resource-Manager/ [Accessed 10 Mar. 2019]

https://blogs.msdn.microsoft.com/testing123/2009/02/06/email-address-test-cases/ [Accessed 17 Mar . 2019]