# Report # 2

**Course Title:** Software Engineering Spring 2019

**Group Number:** 19

**Project Title:** Rutgers Parking System

**URL of Project Website:**

https://sites.google.com/a/scarletmail.rutgers.edu/parking-garage-se/

**Submission Date:** March 17, 2019

**Team Members:**

Suva Shahria, Krithika Uthaman, Andrew Schneeloch, Josh LoGiudice, Gabriel Shen, Anthony Lau, Jahidul Islam, Yu Liu & Max Davatelis

# Table of Contents

**Section 1: Interaction Diagrams** **3**

**Section 2: Class Diagram and Interface Specification** **6**

**Section 3: System Architecture and System Design** **11**

Architectural Styles 11

Identifying Subsystems 11

Mapping Subsystems to Hardware 12

Persistent Data Storage 12

Network Protocol 12

Global Control Flow 13

Execution Orderness 13

Time Dependency 13

Hardware Requirements 13

**Section 4: Algorithms and Data Structures** **14**

Algorithms 14

Data Structures 14

**Section 5: User Interface Design and Implementation** **15**

User Effort Estimation 15

**Section 6: Design of Tests** **16**

**Section 7: Project Management and Plan of Work** **22**

Merging the Contributions from Individual Team Members 22

Project Coordination and Progress Report 22

Plan of Work 22

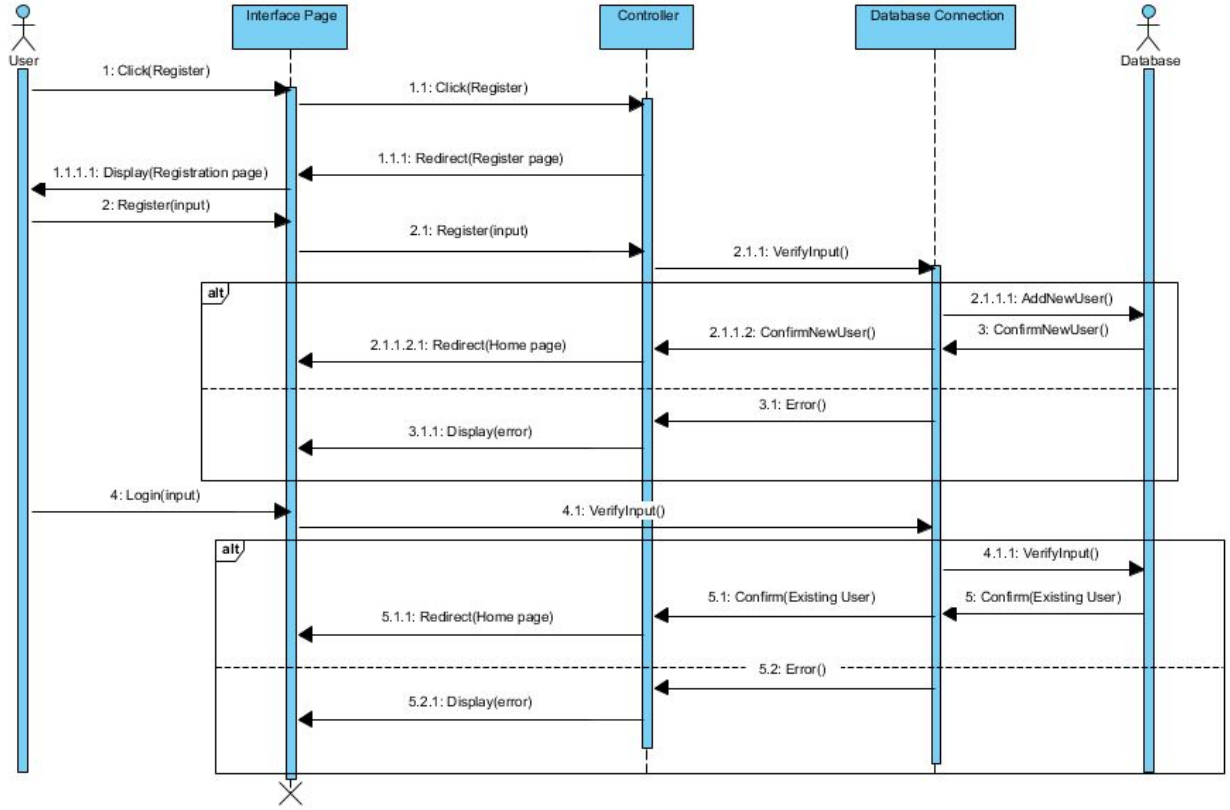
Breakdown of Responsibilities 23

**Section 8: References** **24**

**Section 9: Effort Breakdown** **25**

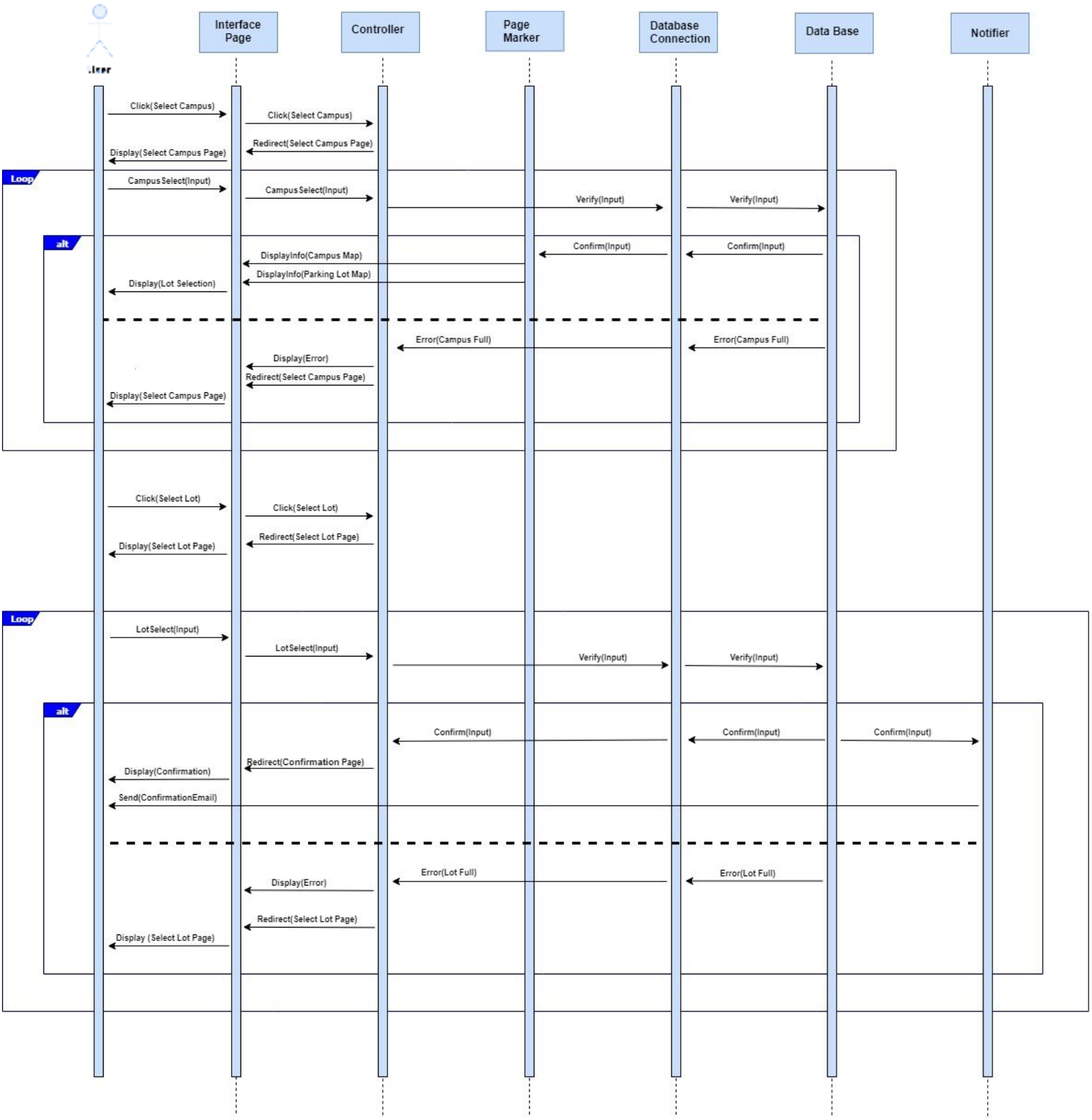
# Section 1: Interaction Diagrams

**UC - 1 Register + Login**




Here much of the responsibility is assigned to the controller to communicate between the interface page and the database connection.

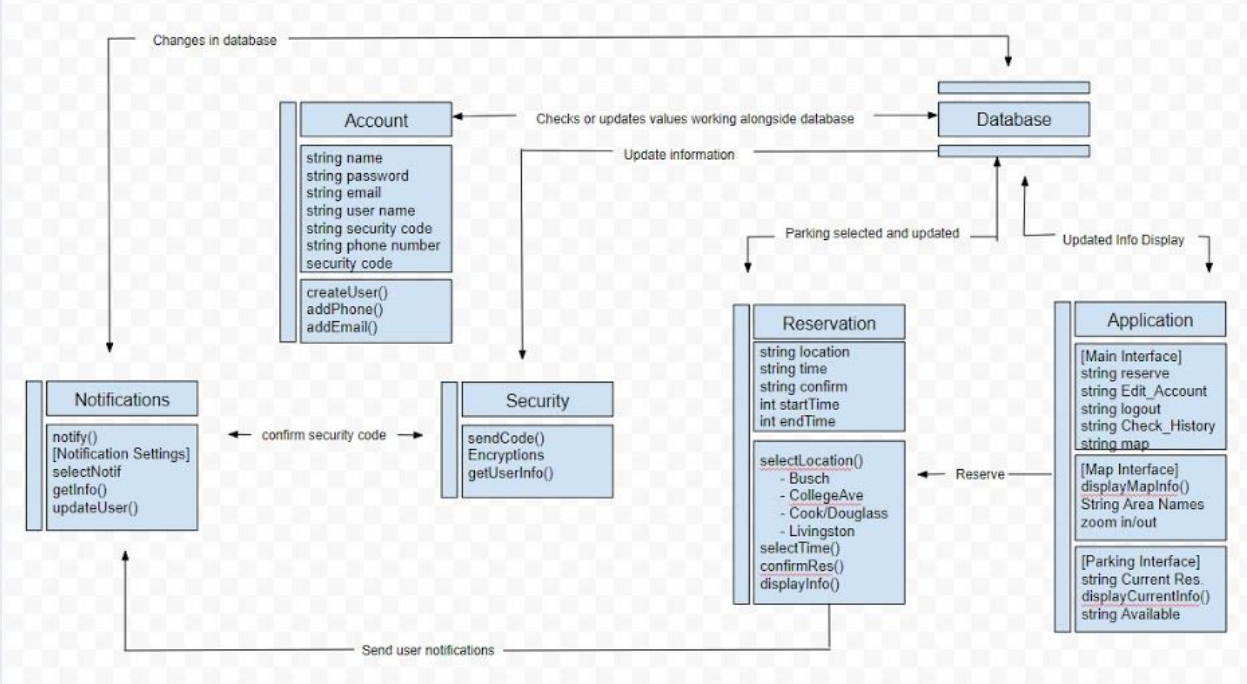
## UC - 5 Pick Spots



After the user logs in successfully. The user will now attempt to select the preferred campus and parking lot. The Database has the most important responsibility of keep track the availability of the lots. So after each selection input the database will verify both the campus and parking lots' availability. Then the data will be retrieved by the Page Maker in which its job is to convert the most updated information on the status of the lots into a map for user's convenience and display it for on the Interface Page. Meanwhile, the Controllers will direct the data to its corresponding locations. When successful, the user will be taken to the confirmation page and receives a confirmation email by the Notifier. However, when the lot is taken/full, the user will simply restart the process and will be directed to either Campus Selection or Lot Selection.

# Section 2: Class Diagram and Interface Specification

## 2.1 Class Diagram




## 2.2 Data Types and Operation Signatures

1) Account

(A) Attributes

string name : user's full name

string firstName : account user's first name

string lastName : account user's last name

string phone number : user's phone number

string password : user selects a password

string date_of_birth : user's date of birth

string security code : security code sent for confirmation to the account

string forgot_password : When password is forgotten

int userID : identification number assigned to the user

(B) Processes

createUser( ) : creates the new user with the inputted information and adds it to the database

addPhone( ) : binds phone number to user account for notifications

addEmail( ) : used as an identifier as well as backup for sending notifications

2) Application

I. Main Interface

(A) Attributes

string Reserve

string Account

string Logout

string History

(B) Processes

userInterface( ) : takes user input and moves throughout the screen

II. Map Interface

(A) Attributes

string Names : Displays names of nearby areas of interest

string zoom_In : Allows user to zoom in on the map

string zoom_Out : Allows user to zoom out of the map

(B) Processes

clickLocation( ) : Allows user to click a location to focus on it and that gives user ability to get more info on the location

getLocationInfo( ) : Allows user can get more info on that location

zoomIn( ) : Zooms in on the map

zoomOut( ) : Zooms out of the map

III. Parking Interface

(A) Attributes

string Current_Reservation : Shows what the user's current reservation is

string Available : Allows user to see what parking spots are currently available

(B) Processes

displayCurrentInfo( ) : Shows current info of user reservation

displayCurrentEvents( ) : Info on current events that might affect parking

3) Reservation

(A) Attributes

string location : Allows user to view available locations

string time : The time that the user selects

int start_time : Start time of reservation

int end_time : End time of reservation

string busch : Info for Busch campus

string college_ave : Info for College Ave campus

string cook_douglass : Info for Cook/Douglass campuses

string livingston : Info for Livingston campus

(B) Processes

selectLocation( ) : Sets the user selected location that they what to reserve at

selectTime( ) : Sets the time user selects to reserve during

confirmReservation( ) : Finalizes the reservation and prepares notifications

displayInfo( ) : Shows the user more information on selected area and events

4) Notifications

(A) Attributes

string notification_settings

string by_phone

string by_email

(B) Processes

notify( ) : Notifies the user about parking reservation

getInfo( ) : Get more info on what parking spot was registered

updateUser( ) : Can update phone number or user email for notifications

selectNotificationType( ) : Allows selecting notification by phone or email or both

5) Security

(A) Attributes

encryption : Encrypts data

(B) Processes

sendCode( ) : Sends user confirmation codes during important instances like creating an account or when password is forgotten.

getUserInfo( ) : Can display user info only when properly logged in

## 2.3 Traceability Matrix

| | Concepts | | | | | |
|---|---|---|---|---|---|---|
| Classes | Database | Security | Main Interface | Map Interface | Parking Interface | Notification Controller |
| Registration | **X** | **X** | **X** | | | **X** |
| Login | **X** | **X** | **X** | | | |
| Notifications | | **X** | | | | **X** |
| Parking Availability | **X** | | | **X** | **X** | |
| Reservations | **X** | | **X** | | | **X** |

| | | | | | | |
|---|---|---|---|---|---|---|
| Location Info | **X** | | | **X** | | |
| Parking updates | **X** | | | | | **X** |
| Area Map | **X** | | | **X** | | |
| Database Connection | **X** | **X** | **X** | **X** | **X** | **X** |

1) Database
    (A) Responsibilities
        - Holds customers' registration information as well as any other info added to the account such as where to send notifications
        - Holds map and event data to display to the user and updates data with recent events
    (B) Classes
        - displayAvailableParking( ) : uses the database information to display the current available parking information to the user upon request
        - displayAreaMap( ) : uses database information on the local area to generate a map of the area and display to the user parking lots nearby and other areas of interest
        - registerUser( ) : allows user to register and account and add it to the database
        - login( ) : when a user enters credentials, it is confirmed with the database information to identify the user and allow them to login
2) Main Interface
    (A) Responsibilities
        - Displays options to the user to navigate the application that includes: Edit Account, Check History, Make Reservation, Display Map, and Logout.
    (B) Classes
        - changeAccount( ) : allows user to make account changes and updates database
        - displayMap( ) : shows the map of the local area based on database information
        - makeReservation( ) : allows user to make a parking reservation
        - checkHistory( ) : allows user to check their own current or previous registration
        - logout( ) : logs out of the account
3) Map Interface
    (A) Responsibilities
        - Displays to the user the local map of selected location and shows parking lots in the area with the option to select one to get more info on the selected location

(B) Classes

- getLocationInfo( ) : user chooses a location on the map where they would like to receive more information on
- zoom( ) : allows user to zoom in and out of the map
- selectParking( ) : user can select a parking location directly from map view and then proceed to reservations

4) Parking Interface

(A) Responsibilities

- Gives the user the options to properly select the location they wish to park at and then updates database on location availability

(B) Classes

- getAvailabilityInfo( ) : allows user to attain info on parking availability in locations such as Busch, Livingston, College Ave, and Cook/Douglass based on information that is available in the database
- makeReservation( ) : users can move directly to making reservation after seeing the available information on the map

6) Notification Controller / Security

(A) Responsibilities

- Sends user notifications for reservations as well as codes to confirm account during registration or when password is forgotten
- Sends user updates regarding location if any changes were made with the area or an event is going on affecting parking availability
- Encrypts user data and uses authentication codes when needed

(B) Classes

- parkingReserveNotif( ) : sends user notification regarding their selected reserve time and location
- confirmationCode( ) : sends user code to confirm either account creation or for other security reasons and confirms with database
- eventUpdate( ) : sends user information if any changes were made with the are they selected for parking

# Section 3: System Architecture and System Design

## Architectural Styles

Overall, the architectural style our system follows is the layered architecture. There are three different layers: the view layer, application layer, and database layer. The top most layer is the view layer, which is what enables the user to access the application layer. The application layer then interacts with the database layer and finally, the database layer is where all the data regarding user information and parking lot information is stored.

The database and phone application will follow a client-server architectural style, because we need something to maintain and retrieve data, such as login information or parking lot information. The application will request a service from the server, and the server will access the information and display the requested information to the user.

The event driven architecture is also implemented for displaying the status of each parking spot, as well as sending information directly to the user. When a spot in a parking lot is taken, the status of that spot will go from vacant to occupied and vice versa. Additionally, if there are any lots that are full or closed due to events happening on campus, a message will be sent to the user.

## Identifying Subsystems

## Mapping Subsystems to Hardware

Web Servers:
The web server will contain the main database in which all of the user's information will be stored on the net. Its primary function is store, process, and deliver specific information to the clients. Parking lot info and statistics will also be located here in which it will be sent to the client upon request. It will also maintain most of the codes for the application to run through clients' phones. Any new information will be updated immediately on the server so the client will be notified with the most accurate parking lot info.

Client Server:
The client will be using their mobile devices to utilize the parking app's feature. The interface will be provided for the user to request services from the web server. Obviously, Internet is required to access the server, and once connected, the client is able to set up accounts, upload class schedules, and search for parking lots. For example, the client will send requests such as parking lot info or registers in which the web server retrieves the info from the database and display it through the client server.

## Persistent Data Storage

The system requires a relational database to store key data about the users and the lots for the system to run correctly.

## Network Protocol

The system is going to use HTTP communication protocol because of the fact that it is a stateless protocol so the people using the app can make a request to get some data and the server will return that required data. We could use other protocols but this will be the easiest to use for an app.

# Global Control Flow

## Execution Orderness

Our service is mainly procedural. The order of actions the user will have to take will always be linear. For instance, the user will always need to verify log-in before making any other actions. And before choosing a lot, the user will need to select a campus. In between these steps, the application will make requests to the server to populate user interface with data selections.
Our service also employs some event driven systems. This is used mainly for notifications i.e. if a reserved lot becomes full before the user get there, a message from the server will trigger a notification event that requires the user to pick a new lot. Another case is when the user swipes into a lot, they will receive a notification confirming their vacancy as an event from the database.

## Time Dependency

The service will not rely on any timings as requests for data are made at user input. However, time will be used by the app on entry of a lot and sent to the database as part of the entry log.

# Hardware Requirements

Users need a working phone in order to use our app.
Garage will need a scanner to scan license plates as they come in.
We will need 10gb of hard disk space to host the database.
As we code and implement our design more hardware requirements may arise, in such a case we will add them.

# Section 4: Algorithms and Data Structures

## Algorithms

## Data Structures

# Section 5: User Interface Design and Implementation

As of now, our user interface implementation is very similar to our initial screen mock-ups and we plan to keep it alike. So far, we have the login page implemented using Android Studio. The user interface for this page looks exactly the same as our screen mock-up. In order to reduce the user effort and create better "ease of use," we propose to remove the "Choose Campus" page on the application. By immediately showing all available parking without choosing a campus reduces a keystroke. This also allows the user to see all available parking spots much quicker and reducing keystrokes by a minimum of two if the user wanted to view two campuses. If the user wished to see more spots on more than two spots, the ease of use is greatly improved. We hope to make the application very easy to use by implementing few pages on the application. That way the user should not feel lost in the app and are able to navigate through the app with ease. In addition, this reduces the worst case scenario user efforts.

## User Effort Estimation

Login:

1. Data Entry: total 1 mouse click and more than 2 keystrokes
    a. Click the "Username" text field
    b. Press the keys for your "Username"
    c. Hit the "tab" key to move to the "Password" text field
    d. Press the keys for your "Password"
    e. Press "Enter" to finish

Getting Parking Spot:

1. Navigation: 1 mouse click
    a. Click which lot you would like to park at
    ---Enter Data---

Register:

1. Navigation: 1 click
    a. Click "Sign up" from first screen
    ---Enter Data---
2. Data Entry: 1 click, 3 text fields, 3 key presses minimum
    a. Click "scarletmail" text field
    b. Fill in text field
    c. Press the "tab" key to move to the next text field
    d. Fill in the "Password" text field
    e. Press the "tab" key to move to the "Confirm Password" field

f. Fill in the “Confirm Password” text field
g. Press the “Enter” key to finish

# Section 6: Design of Tests

| | |
|---|---|
| **Test-Case Identifier:** TC-1 | |
| **Use Case Tested:** UC-1 | |
| **Pass/Fail Criteria:** The test case passes if the user can create an account with non-existing credentials | |
| **Input Data:** New user email, chosen password, confirm password | |
| **Test Procedure:** | **Expected Result:** |
| Step 1: Select a unique email and password you wish to associate with newly created account and press “Register” | System recognizes unique email and password and allows account creation by confirming with database that email is unique |
| Step 2: Enter previously created account info into registration process for a new account | System matches email with email from existing account and thus returns “error email already in use” |
| Step 3: Enter unmatching passwords for “confirm password” | System responds with “error, passwords do not match” |
| Step 4: Enter only email, or password, or confirm password (only one) | System responds with “error, complete all required fields” |

| | |
|---|---|
| **Test-Case Identifier:** TC-2 | |
| **Use Case Tested:** UC-2 | |
| **Pass/Fail Criteria:** The test case passes if the user can login to the account with existing email and password with less than max number of allowed attempts. | |
| **Input Data:** Existing user email, existing password | |
| **Test Procedure:** | **Expected Result:** |

| | |
|---|---|
| Step 1: Enter proper credentials on the very first attempt | System will successfully allow user to enter account |
| Step 2: Enter proper email and password of an existing account previously created within maximum allowed attempts | System will state incorrect information but allow user to attempt login again |
| Step 3: Incorrectly enter password for existing email exceeding or equaling maximum allowed attempts. | System will restrict access to account for a predetermined amount of time |
| Step 4: Enter either only the email or the password (only one) | System will respond with “error, complete all required fields” |

| | |
|---|---|
| **Test-Case Identifier:** TC-3 | |
| **Use Case Tested:** UC-3, UC-4, UC-5, and UC-11 | |
| **Pass/Fail Criteria:** The test case passes if the user can select a particular campus, followed by being able to view and select an available lot on that Campus which then displays information on the lot | |
| **Input Data:** Chosen campus, chosen lot | |
| **Test Procedure:** | **Expected Result:** |
| Step 1: Select from one of the four Rutgers campuses then select an available parking lot that shows up as available on the info screen of the lot | System will successfully allow user to proceed with selected parking spot and display information that it is currently available |
| Step 2: Select one of the campuses then choose a lot that shows up as unavailable on the information screen | System will state in the info section that parking is currently unavailable in the selected spot. For example the College Ave parking deck becomes available to anyone after 5:00 pm. |

| |
|---|
| **Test-Case Identifier:** TC-4 |

| | |
|---|---|
| **Use Case Tested:** | UC-5, UC-7, and UC-8 |
| **Pass/Fail Criteria:** | The test case passes if the user can select a parking spot and set up the arrival and exit time for their parking |
| **Input Data:** | Chosen lot, arrival time, exit time |

| **Test Procedure:** | **Expected Result:** |
|---|---|
| Step 1: Select a parking spot and then select an arrival and exit time that is permitted for the chosen parking spot based on rules and availability | System will successfully allow user to proceed with selected parking spot |
| Step 2: Select a parking spot and an arrival and exit time that breaks a rule for the particular lot (Ex: Reserving College Ave Parking Deck after 5:00PM but with an exit time of noon the next day) | System will state that the selected times are invalid and direct user to info regarding the restrictions of their selected parking spot |
| Step 3: Select a parking spot and select either arrival time or exit time (only one) | System will respond with "error, complete all required fields" |

| | |
|---|---|
| **Test-Case Identifier:** | TC-5 |
| **Use Case Tested:** | UC-11 Main information, UC-12 |
| **Pass/Fail Criteria:** | The test case passes if the user can select map view and have basic interactions such as moving around while names of areas of interest, parking lots, are displayed on the map and can be clicked for more information |
| **Input Data:** | Map view, map interactions (move, zoom, etc.) |

| **Test Procedure:** | **Expected Result:** |
|---|---|
| Step 1: Select map view on the app and click a parking lot to view more information | System will successfully allow user to select map view opening up interactable map with marked areas of interest that can be selected to view more information |
| Step 2: Select map view and click on locations that are not parking lots or areas of interest | System will do nothing as only areas of interest are available to click for more info |

| | |
|---|---|
| Step 3: Attempt to zoom and move around with registered inputs<br><br>Step 4:Attempt to interact with the map with unregistered inputs | System will respond accordingly (“zoom in” will zoom into the map, etc.)<br><br>System will do nothing with unregistered inputs (clicking randomly at nothing will do nothing) |

| | |
|---|---|
| **Test-Case Identifier:** | TC-6 |
| **Use Case Tested:** | UC-9, and UC-10 |
| **Pass/Fail Criteria:** | The test case passes if the user can edit account information such as change password, change username, or even delete the account |
| **Input Data:** | Change email, change password, view account information |
| **Test Procedure:** | **Expected Result:** |
| Step 1: After having already logged in select the “view account information” option and choose change email | System will allow the user to change their email to a different email with verification of account currently as account has already been successfully logged on to; A notification of change is also sent out |
| Step 2: After already being logged in the user navigates to change password | The user can change their password and receive a notification of a password change |
| Step 3: The user attempts relogging in with updated email and password | The database already updated the user’s info during change and so the user should be able to log in with their new credentials |
| Step 4: The user attempts logging in with their old credentials | The system notifies the user of incorrect email or password |

| | |
|---|---|
| **Test-Case Identifier:** | TC-7 (Integration Tests) |
| **Pass/Fail Criteria:** | The test case passes if the user can login after registering and proceed to make a reserve or view information about selectable parking lots. The user must then be notified of important information like selected parking spot. |

| | |
|---|---|
| **Methodology:** | Integration will be done by combining previously separated coding by identifying the important factors of each code's variables and functions and proceeding to use that information to update the other existing code portions. For example, the functions defined for making a reserve will be noticed and used when developing the notifications portion of the code to ensure code unity. |

| **Integration Types:** | **Description:** |
|---|---|
| A) Interactive Map ←→ Database/UI | The Interactive Map will be integrated with the database to ensure proper map display while the UI functionalities must also be considered for interactions |
| B) Login ←→ Database ←→ Security | Login must be properly secured and also connected to the database for authorization through correct credentials |
| C) Register ←→ Database ←→ Security | Similar to login, registering must update the database regarding the new account and be properly secured |
| D) Notifications ←→ Database/Reservation | Notifications must be sent when database changes in account occur or a reservation is made |
| E) Reserve ←→ Database | When reserves are made database must track reserve and update info on user parking spot |

**Test-Case Identifier:** TC-8

**Use Case Tested:** UC-1, UC-2, UC-10

**Pass/Fail Criteria:** The test passes if the user inputted password matches with the stored hashed and salted password after it's been hashed and salted.

**Input Data:** The user inputs a password

| **Test Procedure:** | **Expected Result:** |
|---|---|

| | |
|---|---|
| Step 1: Enter in user's password | System will hash and salt the password and then check to see if it matches with the one in database. |
| Step 2: Incorrect Password | System will state incorrect information but allow user to attempt login again |

# Section 7: Project Management and Plan of Work

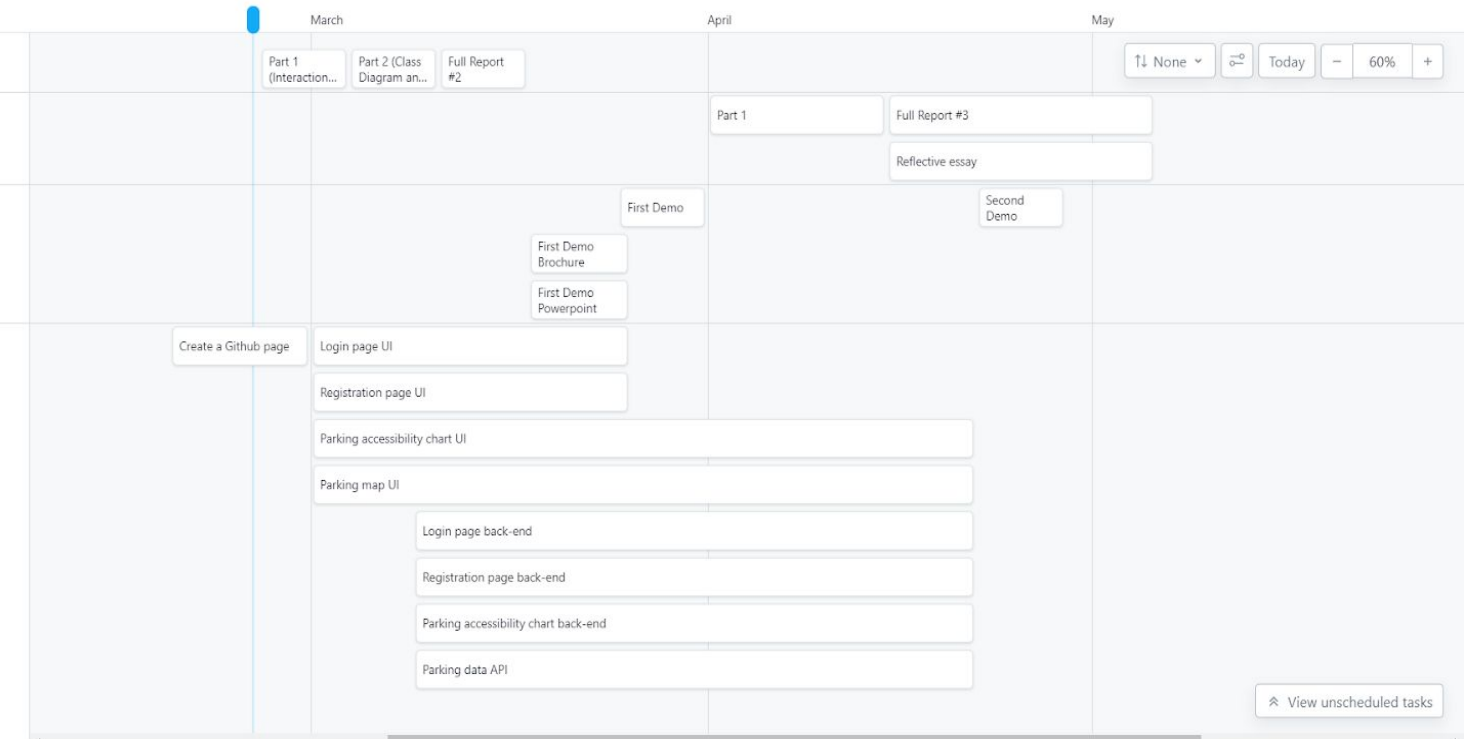
## Merging the Contributions from Individual Team Members

To ensure that the final copy of the report is consistent we need to establish a set format for the various diagrams and chart. We must also agree on more minute details of the formatting like the use of indents and so on.

## Project Coordination and Progress Report

We are currently tackling the project by splitting the programming of the use cases, UI and backend development amongst the team members.

## Plan of Work



# Breakdown of Responsibilities

- App Development
    - Andrew
    - Gabriel
    - Suva
- UI
    - Anthony
    - Max
    - Josh
- Contact User
    - Jahidul
- Database
    - Krithika
    - Anthony
    - Josh
    - Yu
- Security
    - Suva
- Integration
- Testing
    - Jahidul

# Section 8: References

"An Overview of HTTP." *MDN Web Docs*, developer.mozilla.org/en-US/docs/Web/HTTP/Overview.

# Section 9: Effort Breakdown

| | Suva Shahria | Jahidul Islam | Krithika Uthaman | Josh LoGiudice | Gabriel Shen | Yu Liu | Andrew Schneeloch | Lau, Anthony | Max Davatelis |
|---|---|---|---|---|---|---|---|---|---|
| Interaction Diagrams (30 points) | | | 50 | | | 50 | | | |
| Classes + Specs (*10 points*) | | 100 | | | | | | | |
| Sys Arch & Design (*15 points*) | 14.29 | | 14.29 | 14.29 | | 14.29 | 14.29 | 14.29 | 14.29 |
| User Interface (*11 points*) | | | | | 100 | | | | |
| Testing design (*12 points*) | 30 | 70 | | | | | | | |
| Project management (*18 points*) | 50 | | 50 | | | | | | |

Suva = 15*.1429+12*.3+18*.5 = 14.74
Jahidul = 10 *1 +12 *.7 = 18.4
Krithika = 30*.5+15*.1429+18*.5 = 26.14
Josh = 15*.1429 = 2.14
Gabriel = 11 * 1 = 11
Yu = 15*.1429 + 30 * .5 = 17.14
Andrew = 15*.1429 = 2.14
Anthony = 15*.1429 = 2.14
Max = 15*.1429 = 2.14


Effort Breakdown
30
28
26
24
22
20
18
16
14
12
10
8
6
4
2
0
14.74
18.4
26.14
2.14
11
17.14
2.14
2.14
2.14
Suva
Jahidul
Krithika
Josh
Gabriel
Yu
Andrew
Anthony
Max