

An aerial photograph of a dense city skyline, likely New York City, featuring numerous skyscrapers and buildings. A semi-transparent geometric overlay is present in the top right corner, consisting of a white triangle and a grey square. The text is overlaid on the left side of the image.

Prasad Prabhu,  
Rayman Thandi, Andy Chung,  
Vincent Boyack, Ronald Her,  
Bryan Burch, Frank Fine

# Dream Team Final Project

Spring, 2021

## Parking Reservation App

## Table of Contents

<b>Project Management/Scrum</b>	<b>4</b>
Sprint 1 (View Sprint assignments for a more detailed record)	4
Sprint 2	5
Sprint 3	6
<b>Handling User Stories/Scrum PBI Preparation</b>	<b>7</b>
Five user stories with unambiguous requirements using Scrum tool (Jira)	7
User Story 12:	7
User Story 6:	7
User Story 7:	8
User Story 16:	8
User Story 20:	9
Two examples of CCC (Card, Conversation, Confirmation)	10
Example 1:	10
Example 2:	11
Story points, prioritization, and task assignment using a tool (Jira)	12
Risk Table on People, Process, and Product / Risk Solutions	13
Burndown Chart: User Stories + Tasks	15
<b>Coding using GitHub</b>	<b>16</b>
Proof of collaboration	16
Written description of High Cohesion and Low Coupling	17
Using JUnit	18
Using EclEmma	19
<b>Four types of UML Diagrams</b>	<b>20</b>
Use Case Diagram	20
Class Diagram	21
Activity Diagram (Menu)	26
Sequence Diagram (Log in)	27
<b>Design Patterns</b>	<b>28</b>
Design A: Singleton	28
Design B: Iterator	29
<b>Architectural Style</b>	<b>30</b>
Pipe and Filter	30

<b>Bonus Points</b>	<b>31</b>	
User Stories		31
Validate User Story with INVEST and MOSCOW principles:		31
User Story Map:		34
Definition of Done with Acceptance Criteria:		35
Additional Design Pattern		36
<b>Interaction Design Document Section</b>		<b>37</b>

## Project Management/Scrum

### Sprint 1 (View Sprint assignments for a more detailed record)

#### Assignment of Roles

- Scrum Master: Bryan
- Product Owner: Prasad

#### Sprint Planning Meeting

- Our initial meeting was mainly about laying the foundations of our project. We discussed what the core features of our project should be, and which features should be prioritized.
- We determined the core classes that our project required (User, Role, Vehicle, Parking Garage, Parking Lot, Reservation, Parking Garage Map). The work for each class was divided up among the team.

#### Sprint Review

- With our core classes now being built, we began discussions on how to tie all of our classes together.
- We discussed the implementation of user stories from Jira.
- We discussed the implementation of design patterns (A singleton design pattern was used for our project).
- An interactive output was discussed (Viewing the map, selecting a parking slot, etc).
- We decided on a simpler approach to ensure the core functionality of our application (for example, printing everything to the console rather than designing a UI) and left the option open to add more complex features later down the line.

#### Sprint Retrospective

- We decided that more frequent meetings were needed to properly communicate our project, so we agreed to have planning meetings at the beginning of the week (Monday) so that everyone was on the same page. Our sprints would take place on Fridays.
- Discord was a very useful tool for our meetings (both for voice chat, and also for writing items down on a to-do list during our meetings). Google Drive also worked well for our group assignments so everyone could easily contribute.

## Sprint 2

### Assignment of Roles

- Scrum Master: Vince
- Product Owner: Ronald

### Sprint Planning Meeting

- With our project foundations largely in place, this meeting primarily focused on the need to tie all of our code together and ensure that our classes all interacted with each other properly.
- We also looked at what classes we needed and what classes we didn't and prioritized the classes and features which were most important.

### Sprint Review

- Adjustments to our class structure were made. Some of our initial classes were removed or reworked. New classes were added (LoginManager, Permit).
- Priority features were discussed (Quick reserve, cancel reservation).
- The name and functionality of ParkingGarageMap were adjusted (now GarageFloorPrinter class).
- A file reader for the login manager was deemed necessary (Storing user login info in a TXT file).
- Implementation of JUnit tests to our code was discussed.

### Sprint Retrospective

- Our communication as a team has markedly improved thanks to having more frequent meetings to discuss our progress.
- Our project is finally starting to come together, but more work still needs to be done in regards to ensuring our classes interact together properly.

## Sprint 3

### Assignment of Roles

- Scrum Master: Rayman
- Product Owner: Andy and Frank

### Sprint Planning Meeting

- The reservation class was a major point of discussion during this meeting, as it still needed more work, and was a vital part of our project's design.
- The team decided to postpone the time slot feature in the reservation class.
- A lot of discussions focused on putting together the final deliverable and PowerPoint presentation for our project.

### Sprint Review

- Progress was made on the reservation class, with the state of our project nearing completion.
- It was decided that the permit class should be added to the user class.
- JUnit tests were implemented for our code (100% green).
- Adjustments were made to the GarageFloorPrinter class.
- Menu options were revised.

### Sprint Retrospective

- With our final due date approaching, we held numerous smaller meetings throughout the week to ensure work was progressing smoothly.
- Some features of our project were not able to be completed on time, so we instead focused on ensuring that the program had core functionality.

## Handling User Stories/Scrum PBI Preparation

Five user stories with unambiguous requirements using Scrum tool (Jira)

### User Story 12:

#### Description

As a student user, I want to be able to use my student email to log into the website so that I can log in without having to remember a separate username and password

MoSCoW: Should Have

#### Functional Requirements

1. The user will have the option to authenticate with their SacState account instead of creating a new unique account or login in with an existing unique one.
  - a. The SacState authentication option should be presented at the bottom of the login prompt and the create new account option.
    - i. The user will be prompted to enter their SacState email.
2. Upon successful login the user will be redirected to the main reservation page.
3. Upon unsuccessful login the user will be asked to reenter their SacState email.

#### Nonfunctional Requirements

1. Any entry within the list of SacState emails shall be accessible at worst in linear time.

### User Story 6:

#### Description

As a administrator user, I want to automatically cancel any students who are late to their reservation so that I can open up a parking space for another individual.

MoSCoW: Could Have

#### Description

#### Functional Requirements

1. Any user's reservation must be revoked if they fail to fulfill it.
2. Parking spots with a late reservation must be made available.

#### Nonfunctional Requirements

1. Reserved parking spots that aren't fulfilled shall be updated within 10 seconds after the time they were reserved for.

## User Story 7:

## Description

As a student user, I want to see a list of available parking space in a specific garage so that I can find a location to park.

MoSCoW: Could Have

## Description

## Functional Requirements

1. All garages shall be listed to users in a short form.
  - a. The short form must display the garage name
  - b. The short form must display the number of available spots for each garage.
2. Selecting a garage will redirect the user to a detailed view of that garage.

## Nonfunctional Requirements

1. All garages shall be stored in a fixed-sized list.
  - a. Accessing an arbitrary garage must be in constant time.

## Description

## Functional Requirements

1. Selecting a parking spot shall display additional details.
  - a. Taken spots will display to the user that it is unavailable.
  - b. Open spots will prompt the user if they wish to take the spot.

## Nonfunctional Requirements

1. Two lists must maintain the parking spots that are taken and open.
  - a. Both lists must have a dynamic size.
2. The status of all parking spots shall preferably be updated in real time, at worst every one minute.

## User Story 16:

## Description

As a student user, I would want to see a 2D map of the parking space to see which parking spaces are available so that I can see which parking space number corresponds to its location in the Garage

MoSCoW: Could Have

## Description

## Functional Requirements

1. The map must replicate the parking space as close to true to life as possible.
2. All parking spots of the garage must be displayed in an aerial perspective.
  - a. Taken spots shall be filled in with the color *red*.
  - b. Open spots shall be filled in with the color *green*.
3. The user shall be able to pan the map.
4. The user shall be able zoom the map.

## Nonfunctional Requirements

1. The map shall update parking spots statuses no longer than every one minute.
2. The map shall load on the user's device within 10 seconds at the most.



## User Story 20:

### Description

As a student user, I would like to be able to link my parking permit to my account so I can validate that I am able to park in designated parking locations.

MoSCoW: Should Have

### Description

#### Functional Requirements

1. During first setup, the user must be prompted to enter their parking permit to their account.
  - a. The user must enter all relevant details pertaining to their permit.
  - b. The user must be warned that if they enter an invalid permit they may be penalized.

#### Nonfunctional Requirements

1. Linking should take no longer than 5 seconds.

## Two examples of CCC (Card, Conversation, Confirmation)

Example 1:

Card:

- As a student user, I want to press a button and find a random open parking spot in a specific garage so that I can quickly have an available spot to park in without spending time searching for a particular location.

Conversation:

1. *Q: What if there are no available spots?*  
A: The application must check for whether there is availability and present an error if there is none. It could do this when selecting the garage.
2. *Q: How quickly can we make this process?*  
A: Make the button available to click (and easy to see) after the user is logged in and authenticated, and they have chosen a specific garage. (Quick Reserve button)
3. *Q: How will the spot be chosen?*  
A: It could select a random spot but go through a list starting with spaces closest to the school first, so people aren't randomly parked far away for no reason.

Confirmation:

- User logs in. User selects the garage. If there is at least one available space, garage choice is accepted. The user clicks the Quick Reserve button and a random space is assigned.

Example 2:

Card:

- As a student user, I would like to be able to link my parking permit to my account so I can validate that I am able to park in designated parking locations.

Conversation:

1. Q: *How will the linking occur?*

A: It should happen during the account creation process. If you do not have a valid parking permit with the school, there is no reason to have an account with the app.

Confirmation:

- The user creates an account with the app. The user enters parking permit information. The information is authenticated and the application checks what parking locations the user is eligible to reserve. The user's parking permit and account are now linked and they may freely use the reservation app.

## Story points, prioritization, and task assignment using a tool (Jira)

Note: View the Jira page for additional details on all user stories

Story points: 10-20

Prioritization: 1-5 (1 meaning most important)

### **User story 12:** “Account Setup”

Point Value: 13

Assigned: Rayman Thandi

Priority: 5

### **User Story 6:** “Reservation”

Point Value: 20

Assigned: Andy Chung and Ronald Her

Priority: 2

### **User Story 7:** “Parking Garage Available Spots”

Point Value: 17

Assigned: Prasad Prabhu

Priority: 3

### **User Story 16:** “Parking Garage Floor Map”

Point Value: 16

Priority: 1

Assigned: Bryan Burch and Frank Fine

### **User Story 20:** “Permit”

Point Value: 11

Assigned: Vincent Boyack

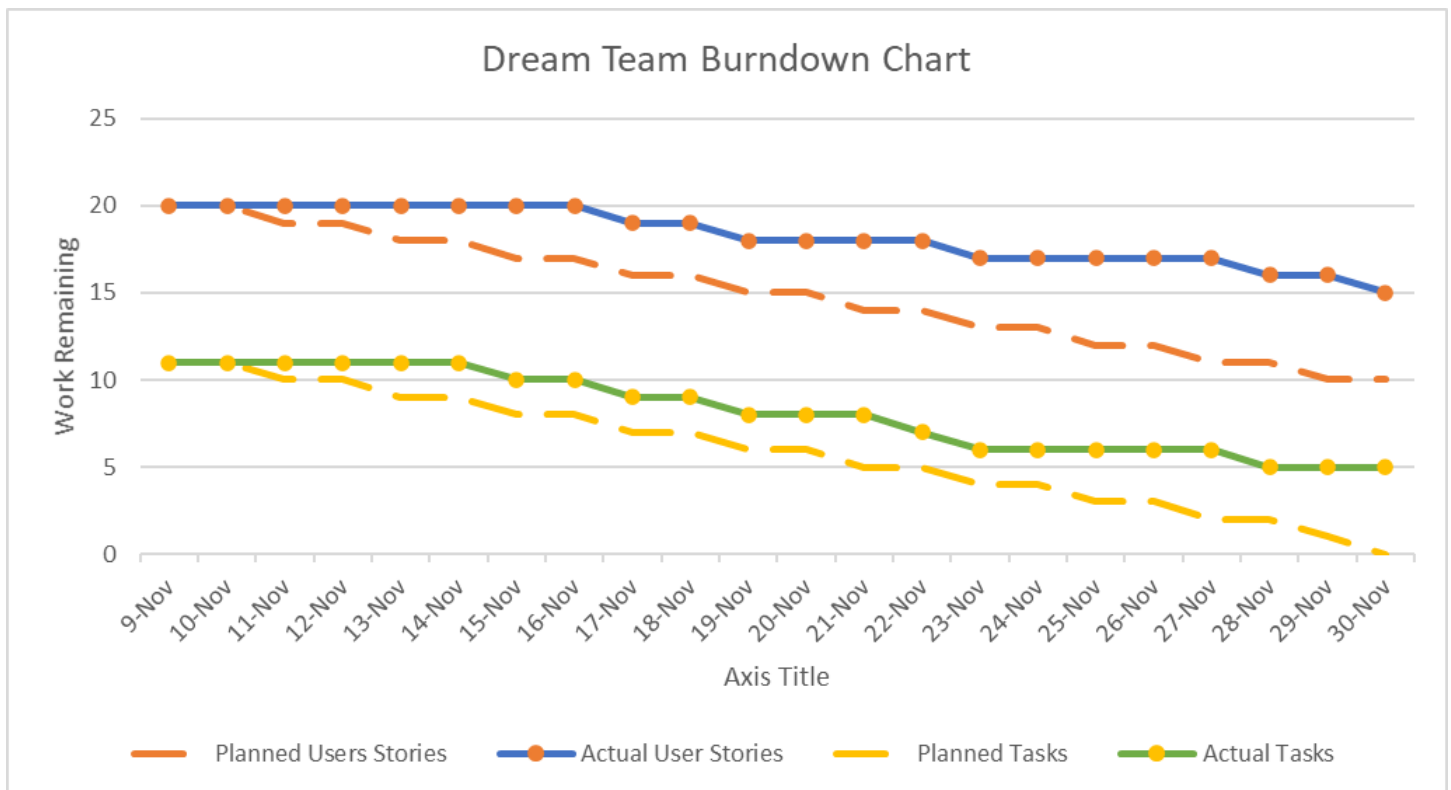
Priority: 4

## Risk Table on People, Process, and Product / Risk Solutions

Risk Outline	Predictable	Solution	Unpredictable	Solution
People risk	<ul style="list-style-type: none"> <li>Given the wrong information</li> <li>Team members developing Application/ website at a different time zone and a different</li> <li>Lack of awareness</li> </ul>	Provide a set schedule and information to all team members. Include the times/ availability to work and set reminders for each member.	<ul style="list-style-type: none"> <li>Locate the wrong parking stop for individual</li> <li>Something bad happens to a team member</li> <li>Logins for team members get hacked</li> </ul>	<p>To address incorrect parking stops, meet with the security where they will provide details to solve the issue.</p> <p>Team members' emergency contact information will be utilized in case of any mishappenings.</p> <p>In case of a team member's account being hacked, the account will be assessed for deletion for any critical information and will be recovered through backup restoration.</p>

Process risk	<ul style="list-style-type: none"> <li>• Server for the software malfunctions</li> <li>• Connection error</li> <li>• Data wiped off the servers</li> </ul>	First, restart the server to see if that is a solution. If not, ensure there is a backup restoration plan in case of emergency with the application.	<ul style="list-style-type: none"> <li>• The users selected/reserved parking doesn't get processed</li> <li>• The application processing slow</li> <li>• Displays the wrong location of the parking area</li> </ul>	If any errors or issues occur when using the application, please proceed to restart and see if any resolving will be needed any further. Should the issues persist, notify the help desk and they'll provide details and ways to resolve the issue.
Product risk	<ul style="list-style-type: none"> <li>• App not responding</li> <li>• User payment doesn't go through</li> <li>• Lack of updates to the product</li> </ul>	Provide details and evidence of the error and send an email to the help desk support. The help desk team will provide the necessary steps to address the issue. Regarding updates, if there are known bugs/defects in our system please notify our team. We will address the issue as soon as possible.	<ul style="list-style-type: none"> <li>• Application crash</li> <li>• GPS system not working for the Map</li> <li>• Server overheat and break</li> </ul>	Proceed to restart the server and analyze if the problem is technical or maintenance.

## Burndown Chart: User Stories + Tasks



# Coding using GitHub

## Proof of collaboration

Note: View GitHub page: <https://github.com/pprabhu2727/DreamTeamFinalProject.git>

master - DreamTeamFinalProject / DreamTeam_FinalProject.java / src / parkingGarageApp /			Go to file	Add file	...
achung99	Update Reservation.java	b5ee854c	10 hours ago	History	
..					
CampusParking.java	Changed default class to parkingGarageApp		5 days ago		
GarageFloor.java	Changed default class to parkingGarageApp		5 days ago		
GarageFloorPrinter.java	Prints floor num now, available spaces now updates correctly		4 days ago		
LoginManager.java	Updated Login Manager		4 days ago		
Main.java	Update Main.java		4 days ago		
ParkingGarage.java	Reverted change to loop starting at 2		4 days ago		
ParkingSlot.java	Changed default class to parkingGarageApp		5 days ago		
Permit.java	Changed default class to parkingGarageApp		5 days ago		
Reservation.java	Update Reservation.java		10 hours ago		
Users.java	Update Users.java		4 days ago		
Vehicle.java	Added getters and setters		4 days ago		

History for DreamTeamFinalProject / DreamTeam_FinalProject.java / src / parkingGarageApp					
Commits on Nov 28, 2021					
Update Reservation.java	achung99	committed 10 hours ago	b5ee854c		<>
Commits on Nov 25, 2021					
quick reserve Pseudocode added	pprabhu2727	committed 3 days ago	b432600		<>
Commits on Nov 24, 2021					
Update Users.java	Rthandi01	committed 4 days ago	4cfa8ac	Verified	<>
Updated Login Manager	Rthandi01	committed 4 days ago	ffdaf62		<>
Update LoginManager.java	Rthandi01	committed 4 days ago	396d2fc		<>
got the filewriter to work	Rthandi01	committed 4 days ago	61f0135		<>
Update Main.java	Rthandi01	committed 4 days ago	6818029		<>
Added Print reader and writer still in progress	Rthandi01	committed 4 days ago	1f74b91		<>
loggedInUser now its own field	bryanburch	committed 4 days ago	19768d2		<>
Added vehicle and reservation	bryanburch	committed 4 days ago	0e1da06		<>
Added new menu options for reservations	bryanburch	committed 4 days ago	a34ebf9		<>
Added getters and setters	bryanburch	committed 4 days ago	dac9def		<>
Added getters and setters	bryanburch	committed 4 days ago	2362fcf		<>
Tweak to map printing	bryanburch	committed 4 days ago	a7df564		<>





## Written description of High Cohesion and Low Coupling

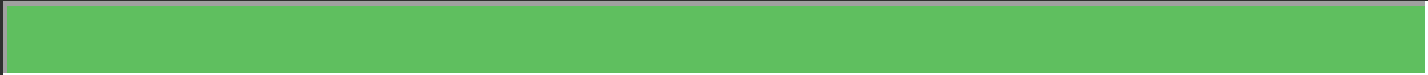











In our parking garage design, most of our classes followed the high cohesion principles. Our design had many classes of their own which allowed our program to be efficient and well-maintained.

- **Reservation class:** The parking garage code creates the number of garages, number of floors, and number of slots on each floor. This class is used by the reservation class in order to reserve a parking spot. The reserve class consists of multiple classes implemented such as Users, Garage floor, Parkingslot, and campus parking. The following classes made the reservation class more efficient and stable. Also, for the low coupling design, most of the classes were separated from the unrelated parts of the codebase. Our parking garage design was able to follow the high cohesion and low coupling principles.
- **LoginManager class:** The login manager class utilized Reservation, Users, and Vehicle. These three classes provided information that allowed the loginmanager to check if the user is the correct account. If all information checks out from the classes, the user may proceed to make a reservation.

Note: View test package within GitHub project.

Finished after 34.696 seconds

Runs: 57/57       Errors: 0       Failures: 0

- 
- >  tests.AdminTest [Runner: JUnit 4] (0.001 s)
  - >  tests.ReservationTest [Runner: JUnit 4] (3.039 s)
  - >  tests.ParkingSlotTest [Runner: JUnit 4] (0.001 s)
  - >  tests.LoginManagerTest [Runner: JUnit 4] (15.735 s)
  - >  tests.VehicleTest [Runner: JUnit 4] (0.001 s)
  - >  tests.MainTest [Runner: JUnit 4] (15.865 s)
  - >  tests.UsersTest [Runner: JUnit 4] (0.002 s)
  - >  tests.CampusParkingTest [Runner: JUnit 4] (0.000 s)
  - >  tests.GarageFloorPrinterTest [Runner: JUnit 4] (0.001 s)
  - >  tests.GarageFloorTest [Runner: JUnit 4] (0.001 s)
  - >  tests.ParkingGarageTest [Runner: JUnit 4] (0.000 s)

## Using EcEmma

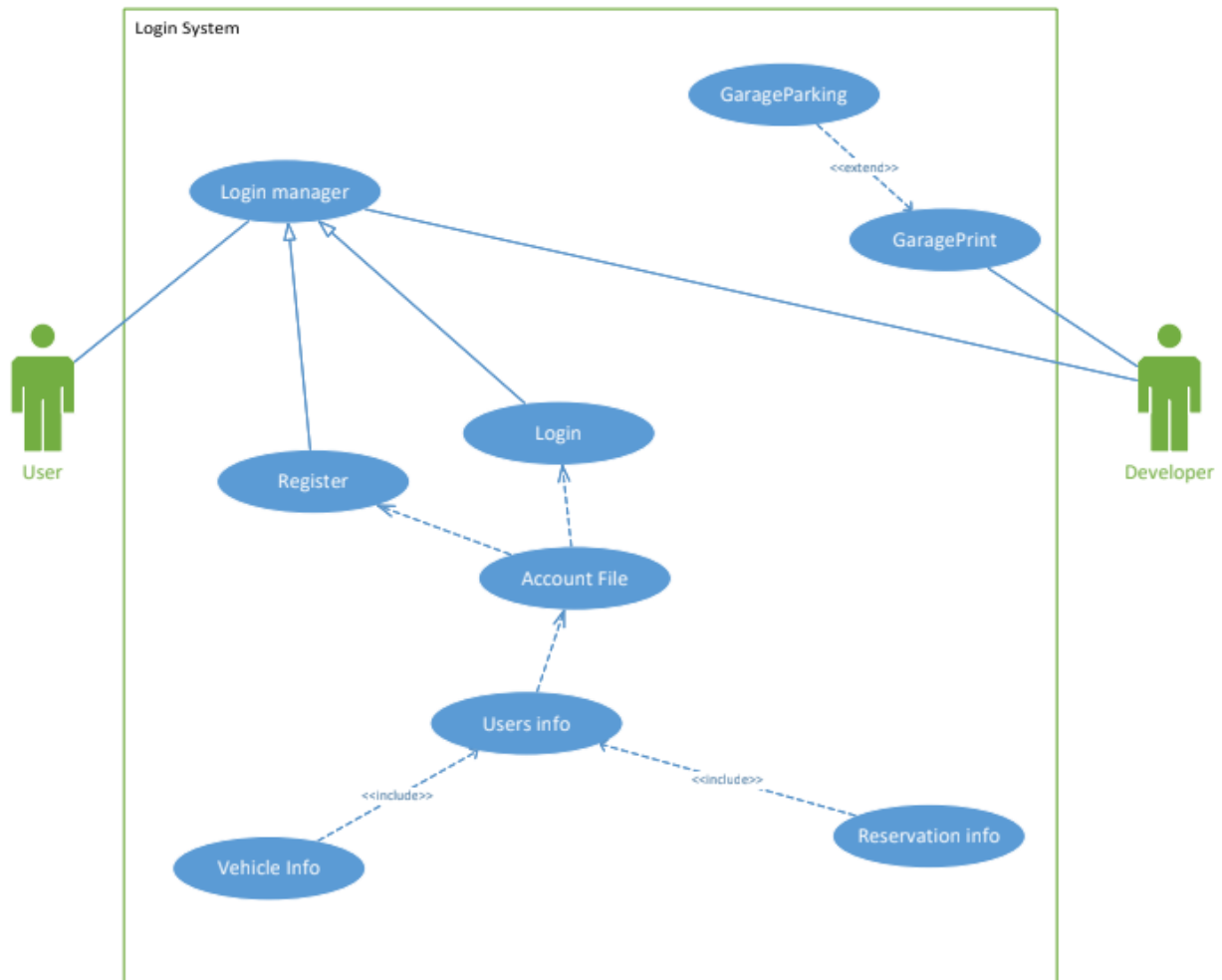
Element	Coverage	Covered Instructions	Missed Instructions ▾	Total Instructions
▾  DreamTeam_FinalProject.java	97.8 %	2,923	66	2,989
▾  src	97.8 %	2,923	66	2,989
▾  parkingGarageApp	96.6 %	1,801	64	1,865
▸  Main.java	74.1 %	126	44	170
▸  LoginManager.java	94.9 %	374	20	394
▸  Admin.java	100.0 %	48	0	48
▸  CampusParking.java	100.0 %	38	0	38
▸  GarageFloor.java	100.0 %	43	0	43
▸  GarageFloorPrinter.java	100.0 %	651	0	651
▸  ParkingGarage.java	100.0 %	45	0	45
▸  ParkingSlot.java	100.0 %	38	0	38
▸  Reservation.java	100.0 %	182	0	182
▸  Users.java	100.0 %	172	0	172
▸  Vehicle.java	100.0 %	84	0	84
▾  tests	99.8 %	1,122	2	1,124
▸  LoginManagerTest.java	95.3 %	41	2	43
▸  AdminTest.java	100.0 %	92	0	92
▸  CampusParkingTest.java	100.0 %	20	0	20
▸  GarageFloorPrinterTest.java	100.0 %	49	0	49
▸  GarageFloorTest.java	100.0 %	41	0	41
▸  MainTest.java	100.0 %	27	0	27
▸  ParkingGarageTest.java	100.0 %	44	0	44
▸  ParkingSlotTest.java	100.0 %	81	0	81
▸  ReservationTest.java	100.0 %	161	0	161
▸  UsersTest.java	100.0 %	388	0	388
▸  VehicleTest.java	100.0 %	178	0	178

## Four types of UML Diagrams

### Use Case Diagram

Drive Link:

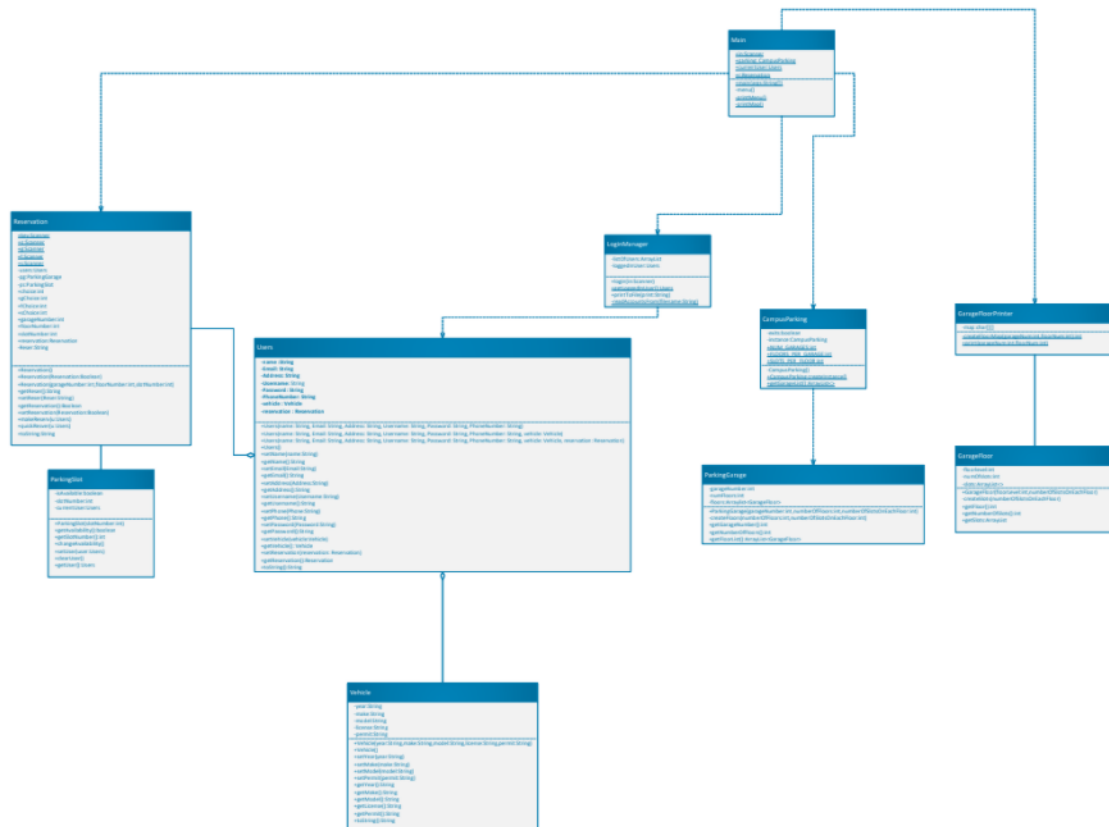
[https://drive.google.com/file/d/1RG5cCyrUeHM\\_ESbYV6EN36X1wiMpz00q/view?usp=sharing](https://drive.google.com/file/d/1RG5cCyrUeHM_ESbYV6EN36X1wiMpz00q/view?usp=sharing)



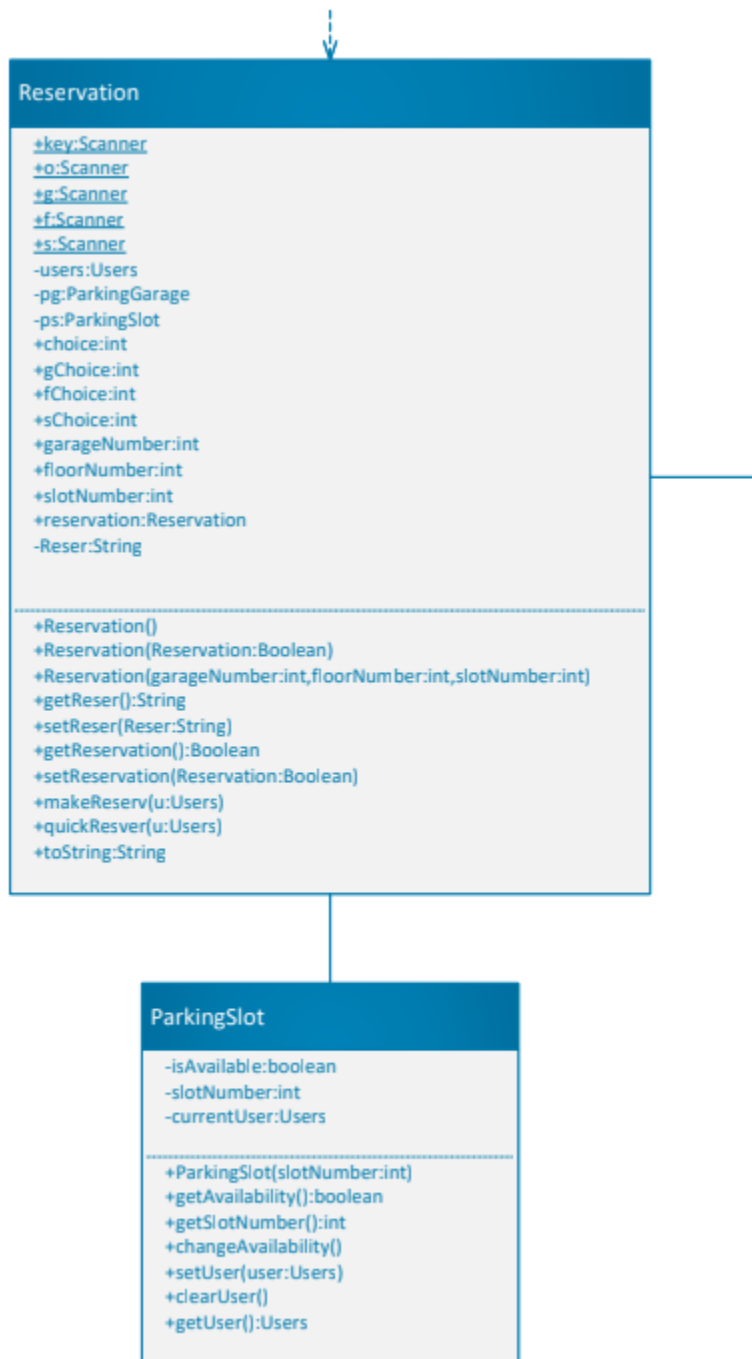
## Class Diagram (Entire Project)

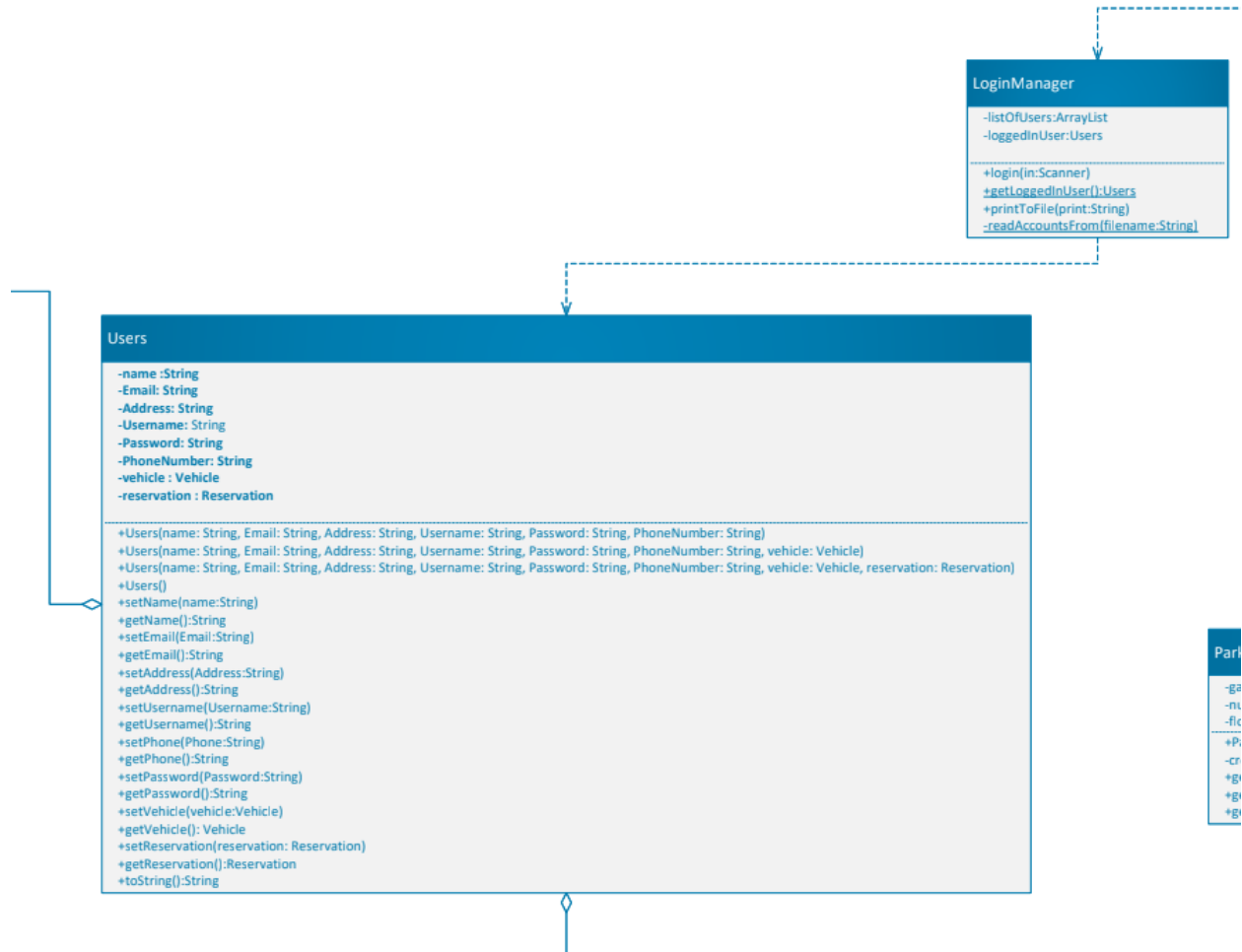
Drive Link:

<https://drive.google.com/file/d/1s6zg1qmVkdMSV8tBJfeWNBaeaWfHqkt5/view?usp=sharing>



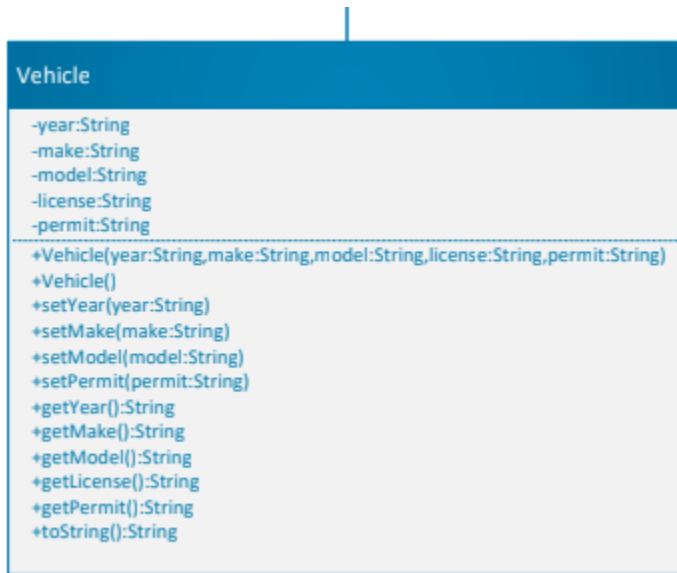
## Breakdown of Class Diagram:



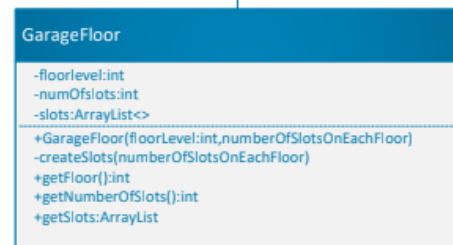
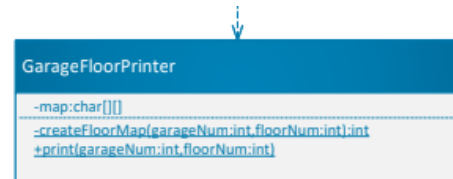
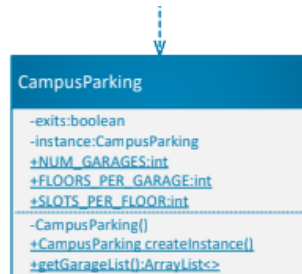


Part

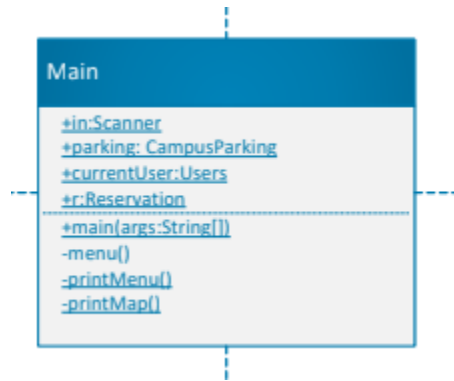
- B
- n
- f
- c
- +P
- c
- +B
- +B
- +B



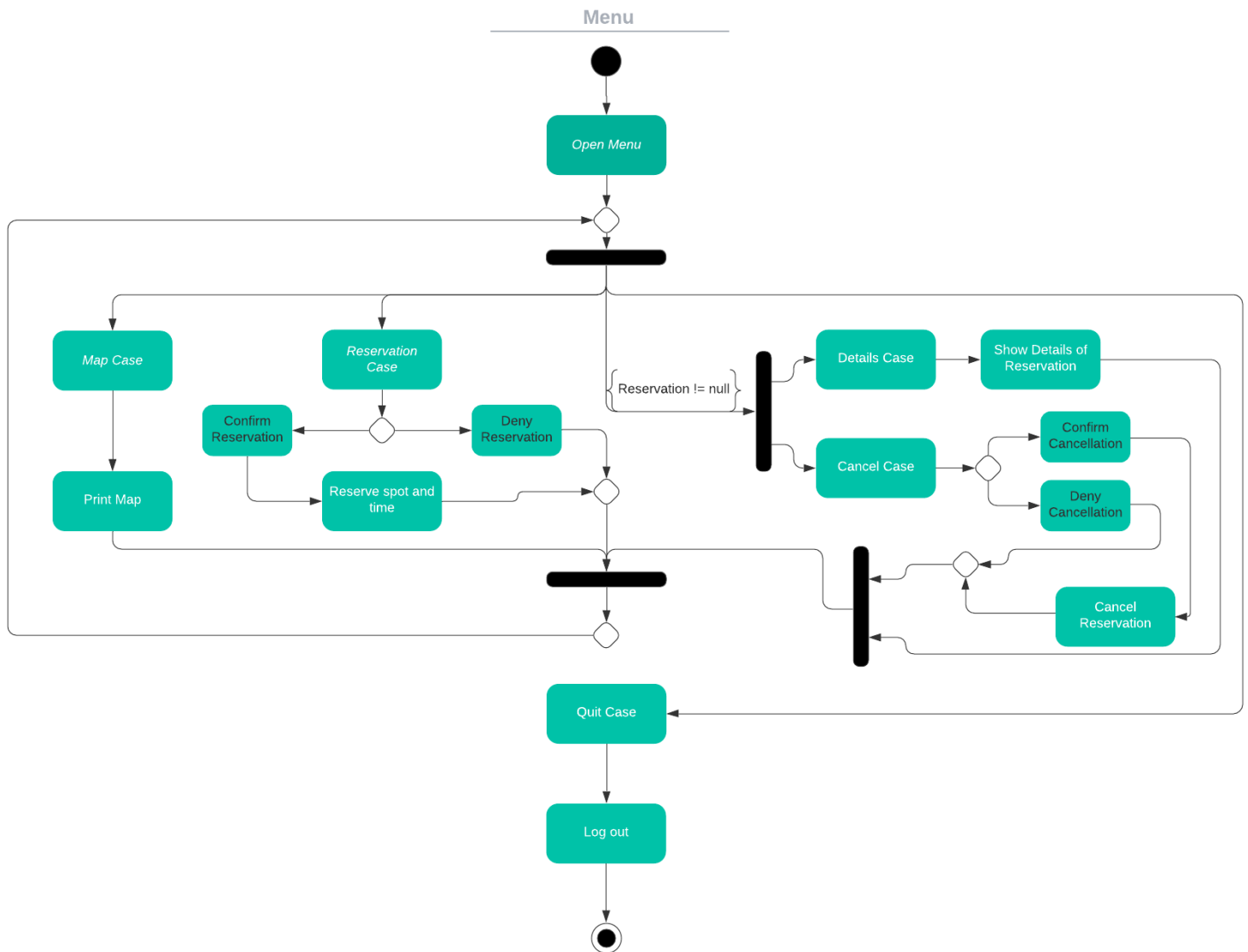
ring)

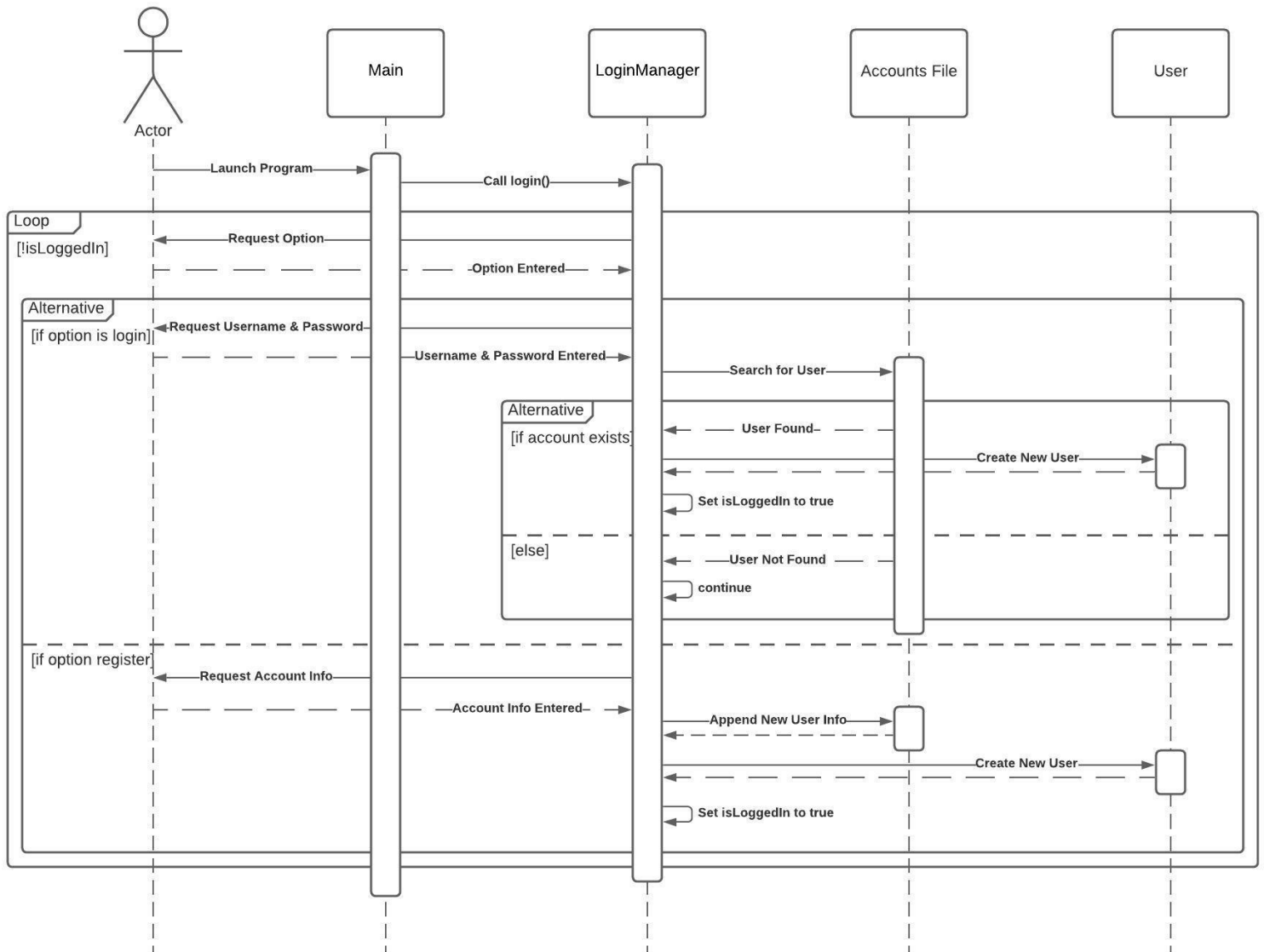






## Activity Diagram (Menu)





# Design Patterns

## Design A: Singleton

```
public class CampusParking {
    private static boolean exists = false;
    private static CampusParking instance;

    // Assuming for now that we have 5 garages with 6 floors each and 30 slots per floor.
    static final int NUM_GARAGES = 5;
    static final int FLOORS_PER_GARAGE = 6;
    static final int SLOTS_PER_FLOOR = 48;

    //private static ParkingGarage[] garages = new ParkingGarage[NUM_GARAGES];
    private static ArrayList<ParkingGarage> garagelist = new ArrayList<ParkingGarage>(); //Creatin

    /*
     * Can't directly create campus parking.
     */
    private CampusParking() {
        for (int i = 1; i <= NUM_GARAGES; i++) {
            //garages[i] = new ParkingGarage(i, FLOORS_PER_GARAGE, SLOTS_PER_FLOOR);
            garagelist.add(new ParkingGarage(i, FLOORS_PER_GARAGE, SLOTS_PER_FLOOR));
        }
        exists = true;
    }

    /*
     * Call this function to construct all campus parking.
     */
    public static CampusParking createInstance() {
        if (!exists) instance = new CampusParking();

        return instance;
    }

    public static ArrayList<ParkingGarage> getGaragelist() {
        return garagelist;
    }
}
```

This is a singleton design pattern of the class CampusParking. This class is a singleton because only one instance of CampusParking can be active at a time and any other attempts to create another CampusParking result in the initial instance is returned. We chose CampusParking to use the singleton pattern because it is our instantiation class and it should only be called once. This class creates the initial list of parking garages so only having a single list to refer to is beneficial because otherwise, we might run into errors if there are two different lists of garages we refer to elsewhere in the code.

## Design B: Iterator

```
// TODO: iterator
Iterator<Users> iterator = listOfUsers.iterator();
System.out.println("Iterator type for the datastructure is: " + iterator.toString());
for (Users user : listOfUsers)
{
    if (username.equals(user.getUsername()) && password.equals(user.getPassword()))
    {
        loggedInUser = user;

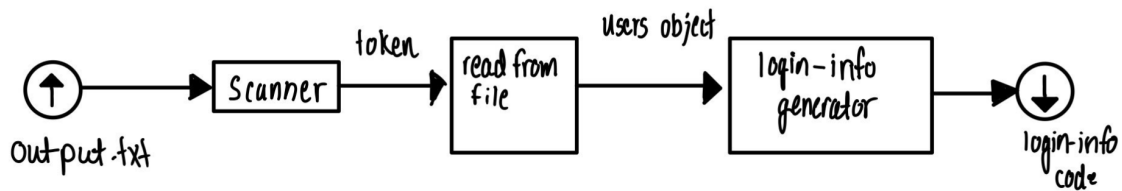
        // when a user is found, "break" stops iterating through the list
        break;
    }
}
```

This is the iterator design pattern of the Login Manager class. This class iterates through the ArrayList to assign the correct user info to the user object. This is an iterator pattern because the code is written in a way that allows us to iterate through an object in a way that makes it easy to change from one aggregate to another. This is an iterator in the loginManager class. We chose this class to include an iterator design pattern because our code was using a loop to iterate through each element in a user's username and password to check if the user is the one who is logged in. An iterator is helpful here because it is easy to change the aggregate in the future if needed.

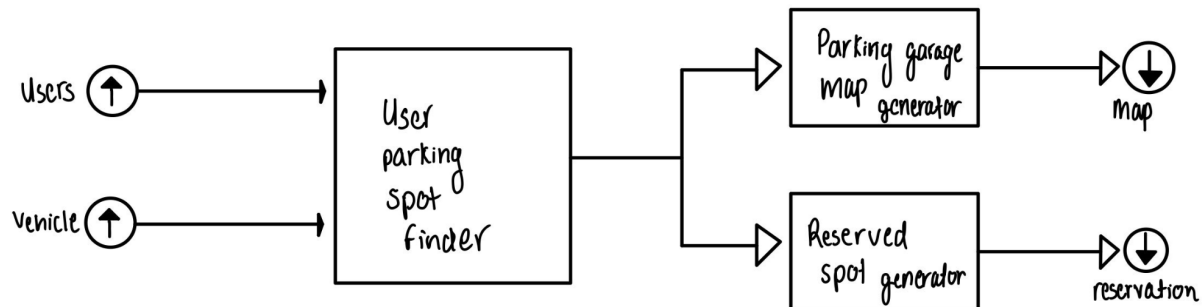
## Architectural Style

### Pipe and Filter

The project includes the pipe and filter architectural style since it inputs and outputs information through a tokenizer and filters them out through the pipe to generate the Reserved spot and the parking garage map.



Reads from the “output.txt” file that contains the user’s information and produces the strings contained in the text file and generates it into the Users object and the ArrayList.



Reads the file that contains vehicle and users information and generates the map of the garage and reserved spot that the user chose. From assigning the spot to that current user.

## Bonus Points

### User Stories

Validate User Story with INVEST and MOSCOW principles:

User Story 12: As a student user, I want to be able to use my student email to log into the website so that I can log in without having to remember a separate username and password

- INVEST
  - Independent: Simply a branch of the login system so it can be implemented separately.
  - Negotiable: There are different implementations, so this can be discussed with the team and project stakeholders.
  - Valuable: This story will be convenient for the user.
  - Estimable: The team will be able to break down this story into subtasks and assign reasonable time estimates.
  - Small: Only a part of the login system so it will be simpler to implement.
  - Testable: This story can be easily unit and acceptance tested.
- MoSCoW: Should Have

User Story 7: As a student user, I want to see a list of available parking spaces in a specific garage so that I can find a location to park.

- INVEST
  - Independent: While the printing of parking spaces will rely on existing spaces and availability, the print function can be developed outside of the main system and integrated later.
  - Negotiable: There are different implementations, so this can be discussed with the team and project stakeholders.
  - Valuable: This story will be beneficial to the user.
  - Estimable: The team will be able to break down this story into subtasks and assign reasonable time estimates.
  - Small: Should only involve implementing a function or two.
  - Testable: This story can be easily unit and acceptance tested.
- MoSCoW: Could Have

User Story 16: As a student user, I would want to see a 2D map of the parking space to see which parking spaces are available so that I can see which parking space number corresponds to its location in the Garage

- INVEST
  - Independent: While this story may seem to clash with User Story 7, it can be developed separately.
  - Negotiable: There are many implementations, so this can be discussed with the team and project stakeholders.
  - Valuable: This story will tailor to users who prefer maps over lists.
  - Estimable: This story can be broken down into smaller parts and assigned time estimates.
  - Small: While this story is on the verge of being too involved, it will likely only involve a new class with a couple functions.
  - Testable: This story can be easily unit and acceptance tested.
- MoSCoW: Could Have

User Story 20: As a student user, I would like to be able to link my parking permit to my account so I can validate that I am able to park in designated parking locations.

- INVEST:
  - Independent: This story is entirely separate from any subsystem.
  - Negotiable: There are ways of implementing the story that can be determined through collaborating with team members and stakeholders.
  - Valuable: This story is necessary for users.
  - Estimable: This story can be broken down into smaller parts and assigned time estimates.
  - Small: Likely cannot be broken down into smaller stories.
  - Testable: Should be testable with unit and acceptance tests.
- MoSCoW: Should Have



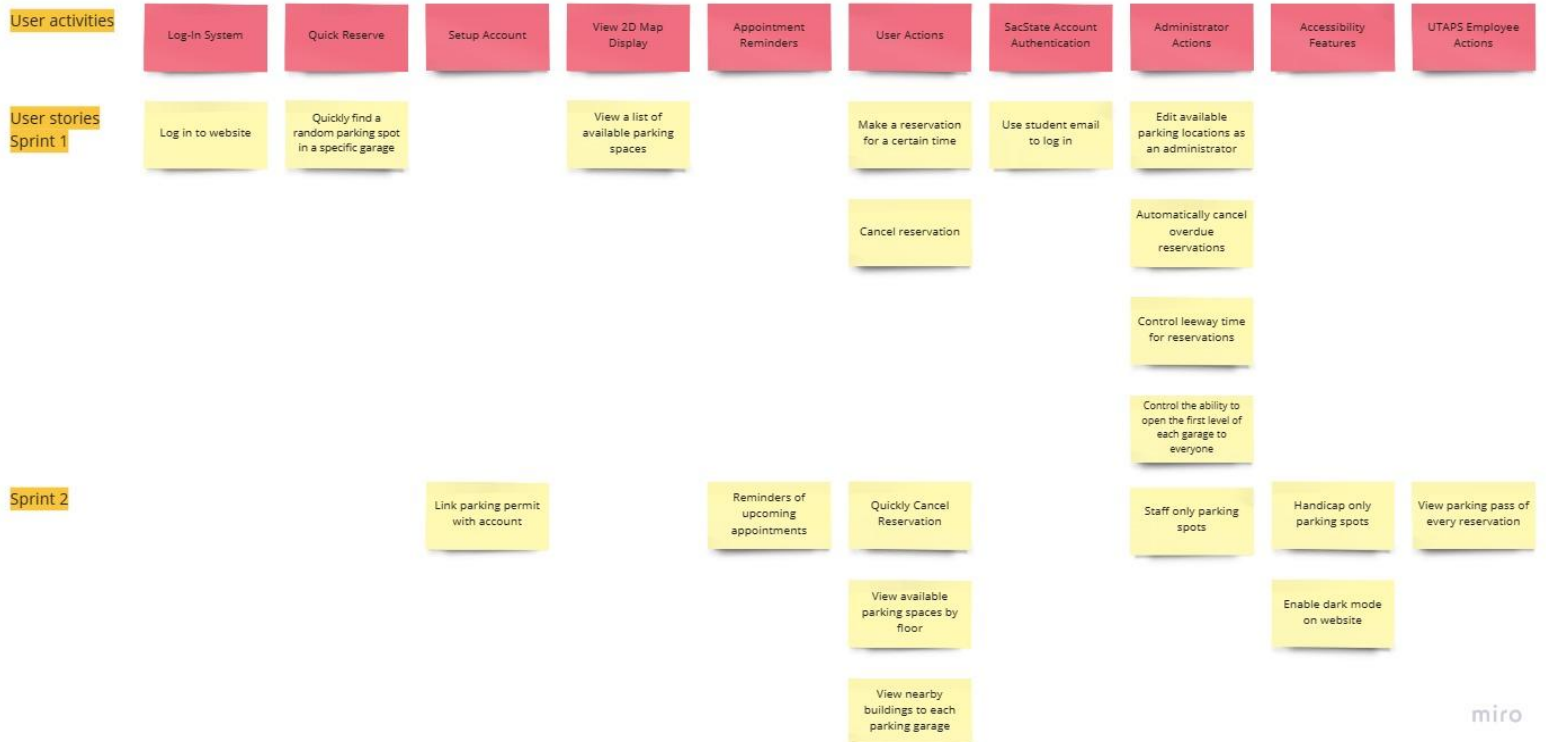
User Story 15: As a student user, I want to be able to quickly tap a button to indicate I am about to leave so that I can open my parking space up for another reservation.

- INVEST
  - Independent: Though associated with reserving, it would only require referencing parking spaces. Which could be implemented without the reservation system in place.
  - Negotiable: There are different implementations, so this can be discussed with the team and project stakeholders.
  - Valuable: This story will save the user time.
  - Estimable: Can be broken down into easily estimable subtasks.
  - Small: simple story that can be implemented in one sprint.
  - Testable: This story can be easily unit and acceptance tested.
- MoSCoW: Should Have

## User Story Map:

Link: [https://miro.com/app/board/uXjVOduNARY=?invite\\_link\\_id=566218871964](https://miro.com/app/board/uXjVOduNARY=?invite_link_id=566218871964)

## Dream Team User Story Map



### Definition of Done with Acceptance Criteria:

A definition of done is a checklist of what needs to be accomplished for a BPI to be considered complete. All of our user stories would need to meet each point in the following checklist before they would be considered “done.”

- The design is complete and reviewed by at least two peers
- There are comments within the code
- The code has passed a visual inspection
- The code has passed JUnit and EcEmma tests
- The code works with existing classes and other sections of the program
- Documentation has been updated
- The acceptance criteria have been met

Each user story has its own acceptance criteria that must be met before a BPI can be considered complete:

- User Story 12: Logging into a website using student email
  - Acceptance Criteria:
    - Users can enter in the necessary information
    - The program allows the user to log in with their student email?
    - The system remembers the account
- User Story 7: View a list of available parking spaces
  - Acceptance Criteria:
    - The system displays the list of parking spaces
    - The displayed list is accurate
    - The user interface can be used to search for parking spaces
- User Story 16: View a 2D map of parking spaces
  - Acceptance Criteria
    - The system displays the map of parking spaces
    - The displayed map is accurate
    - The user interface can be used to search for parking spaces on a given floor
    - Only show available special parking spaces if applicable (handicapped and staff only spaces/elevators and stairs / etc.)
- User Story 20: Link parking permit to account
  - Acceptance Criteria
    - User must create an account
    - The user must have a valid parking pass
    - The user's parking pass must be marked as a special pass in order to use specific parking spaces (staff, handicap, etc)

### Additional Design Pattern

```
1 package parkingGarageApp;
2 import java.io.BufferedWriter;
11
12 public class LoginManager {
13
14     private static boolean exists = false;
15     private static LoginManager instance;
16
17     private static ArrayList<Users> listOfUsers = new ArrayList<Users>();
18     private static Users loggedInUser = null; // Used to hold the instance of a user
19
20     public static LoginManager createInstance() {
21         if (!exists) instance = new LoginManager();
22
23         return instance;
24     }
25 }
```

Singleton Design Pattern: should only have one instance of the LoginManager at a time. This class deals with login and registration and references a file of pre-existing users. We could run into trouble if we had more than one instance as it could have a different user logged in.

## Interaction Design Document Section

### User Interaction and Design:

These are wireframe images of what our “Home” and “Map” page could look like. We applied each of the interaction design principles in creating these wireframes.

#### SAC:

- Simplicity: The buttons are clearly labeled and there aren't any unnecessary buttons or functions which are hard to use.
- Accessibility: There is no use of color to indicate important distinctions. The fonts are large enough to read.
- Consistency: Both pages displayed in the two wireframes follow a similar design. Fonts, colors, and navigation mechanisms are consistent throughout.

#### CAP:

- Contrast: Certain texts are larger, italicized, bolded, or underlined to make different options appear different.
- Alignment: Navigation Menu and Filters page are aligned in a row/column format. The main information box is slightly larger and towards the center-right of the page.
- Proximity: The course navigation menu is placed all together. The same goes for the filters page. Groups of related items are grouped together spatially in tight groups.

#### FeVER:

- Feedback: this isn't shown in the wireframe, but we plan to show visual feedback whenever a user presses buttons.
- Visibility: this isn't shown in the wireframe, but we plan to implement it in the final product
- Error Prevention and Recovery: this isn't shown in the wireframe, but we plan to implement it in the final product. For example, the cancel reservation button should be disabled if the user doesn't have a current active reservation.

Diagram A: Home Screen

On Point Parking

Home Map Info

Quick Reserve Cancel Reservation

Filters:

Garage

Floor

Time

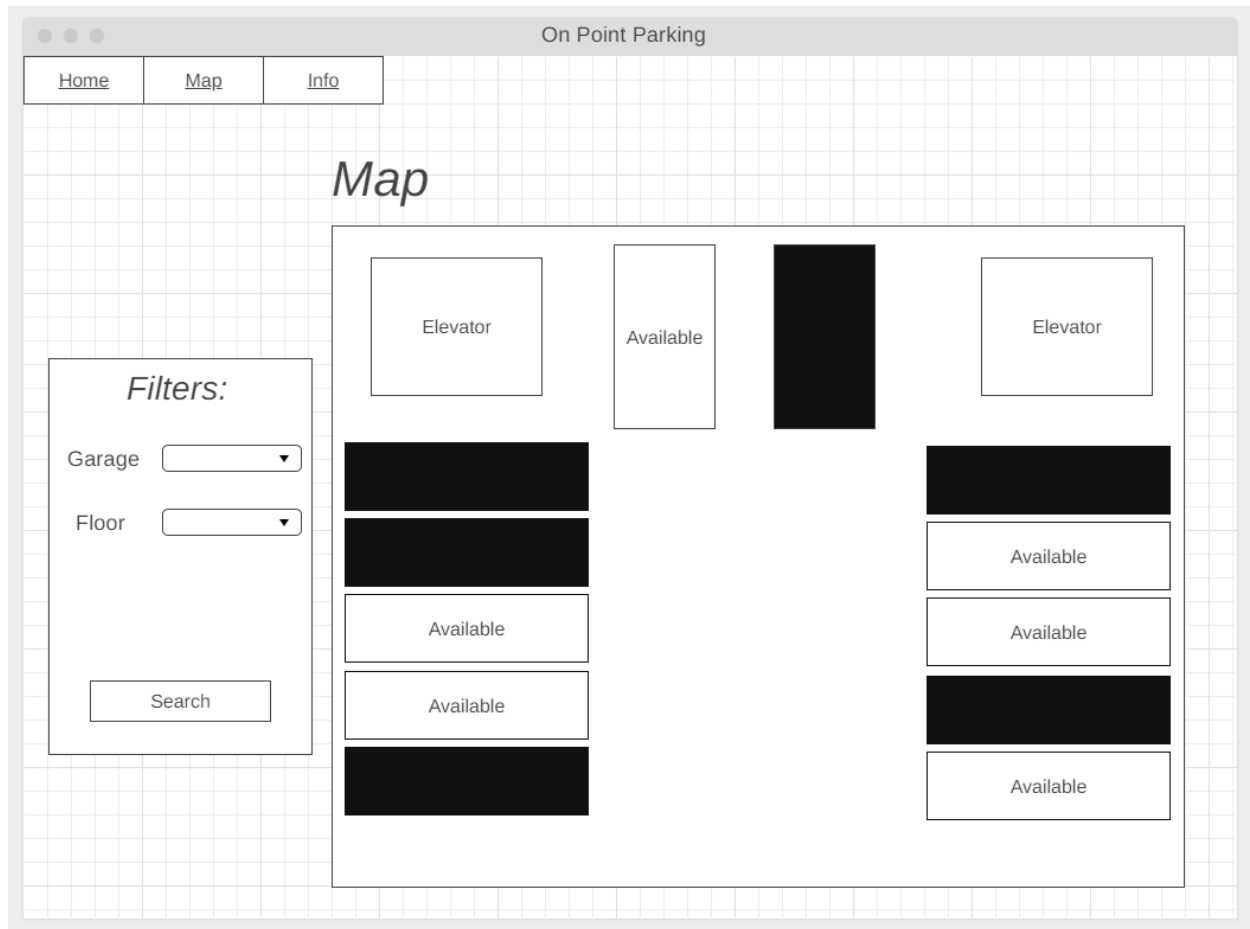
Search

Available Parking Locations

- 
- 
- 
- 
- 
- 
- 
- 
- 
- 

1. When the program starts up the data store (if any) must be loaded and the system must be ready to search.
2. Users should be able to select the filters they want and press the search button. A search must occur and update the available parking location box.
3. Each filter will have a dropdown list of preselected options that the user can click on.
4. Users can press any button on the course navigation and the program should load up the appropriate page.
5. Selected the quick reserve button should automatically find and reserve a parking location with no other input needed from the user.
6. Selecting the cancel reservation button should automatically cancel the user's current reservation and update the available parking locations for other users without any other input needed from the initial user.

Diagram B: Map Screen



1. When the program starts up the data store (if any) must be loaded and the system must be ready to display the map.
2. Users should be able to select the filters they want and press the search button. A search must occur and update the map to show the appropriate parking garage and floor.
3. Each filter will have a dropdown list of preselected options that the user can click on.
4. Users can press any button on the course navigation and the program should load up the appropriate page.
5. The map should update frequently to reflect any changes made by other users' reservations.