



TROT RIGHT IN

Technical Design Document

BAM! Software

Frontend: Evan Kohn. James Lomeo. Justin Trantham.

Backend: Jordan Kayse. James Lomeo. Jessica Yeh. Story Zanetti.

Introduction

We can use technology to solve the problems around us, but if the user interface is too complicated, the problem is not solved. We provide a simple yet elegant solution to fix parking on campus. Our solution allows users to rate how much parking is available and provides that data to others users. While the solution may seem simple to the user, the implementation process is much more complex. To help aid in the implementation, we have developed a design for how the outcome should be achieved. This document includes designs from the front-end and the back-end of the software.

Software Requirements

Visitor

- Visitors should be able to view garage information in list form.
- Visitors should be able to view garage information in map form.
- Visitors should be able to view top ten contributing users.
- Visitors should be able to scan QR codes to navigate to the website page for particular garages.
- Visitors should be able to report the availability of the garage, but only if they login.

User

- Users should be able to access all functionality that visitors can.
- Users should be able to create accounts using Facebook.
- Users should be able to create accounts using Google.
- Users should be able to create generic accounts using their username, email, and password.
- Users should be able to delete their own accounts.
- Users should be able to add their phone numbers for notification purposes.
- Users should be able to report the availability of garages. Each of these ratings gives the user a point toward being a top contributing user.
- Users should be able to input their favorite garages.
- Users should be able to delete their accounts.
- Users should be able to view when a garage is least and most crowded.
- Users should be able to input their commute times.
- Users should be able to be notified about the status of their favorite garages before their commute times.
- Users should be able to request garages to be added.

Admin

- Admins should be able to access all functionality that users can.
- Admins should be able to approve/decline requests for parking garages.
- Admins should be able to modify garages (adding levels, removing garages, etc.).

Use Case Diagram

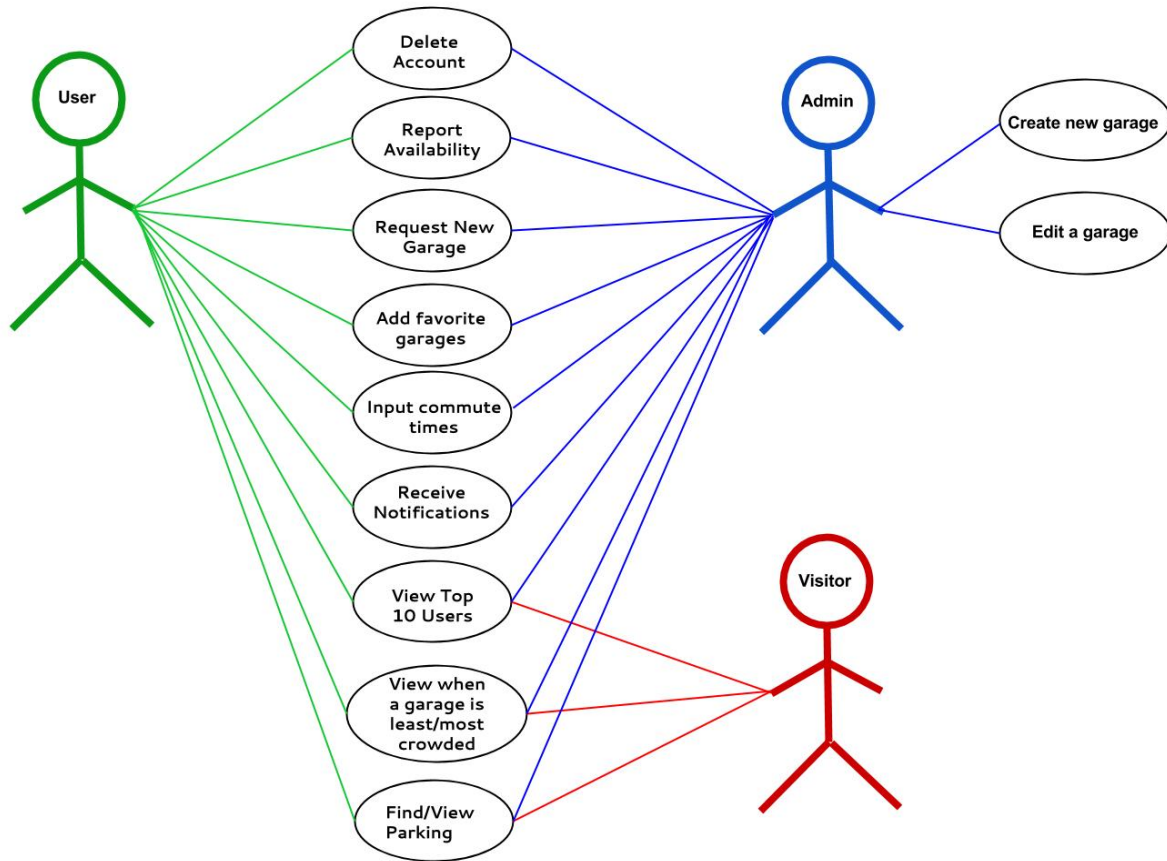


Figure 1: A Use Case diagram of our software.





The above Figure 1 is a diagram of what each different kind of user is able to do. This is a visual representation of the requirements our software will have to meet to satisfy the needs of all types of users.

Database Model

Legend :	Primary key 👑	Foreign Key ⚡	Can be Null ☐	Can't be Null ■
Entity	Attributes			
ParkingLocations	👑 ParkingID INT			
	■ Name VARCHAR(50)			
	■ Address VARCHAR(50)			
	☐ Cost INT			
	☐ Comments VARCHAR(250)			
	☐ NumberOfLevels INT			
Ratings	👑 RatingID INT			
	⚡ ParkingID INT			
	⚡ UserID INT			
	■ Timestamp DATETIME			
	■ Rating INT			
	☐ Level INT			
Users	👑 UserID INT			
	■ Username VARCHAR(50)			
	■ Email VARCHAR(50)			
	☐ Password VARCHAR(64)			
	☐ PasswordSalt VARCHAR(50)			
	☐ PhoneNumber VARCHAR(20)			
	■ UserType INT			
	■ ExternalType VARCHAR(10)			
CommuteTimes	☐ ExternalID VARCHAR(64)			
	👑 CommuteID INT			
	⚡ UserID INT			
	■ Time TIME			
	■ Day INT			
	☐ WarningTime TIME			
	☐ TimeOfNotification DATETIME			

Table 1.a: A list of our tables and their attributes.

Database Model Continued

Legend :	Primary key 	Foreign Key 	Can be Null 	Can't be Null 
----------	---	---	---	---













Entity	Attributes
Requests	 RequestID INT
	 UserID INT
	 Name VARCHAR(50)
	 Address VARCHAR(50)
	 Cost INT
	 NumberOfLevels INT
	 Comments VARCHAR(250)
	 Status INT
FavoriteGarages	 FavoriteID INT
	 UserID INT
	 ParkingID INT
	 Priority INT

Table 1.b: A list of our tables and their attributes.

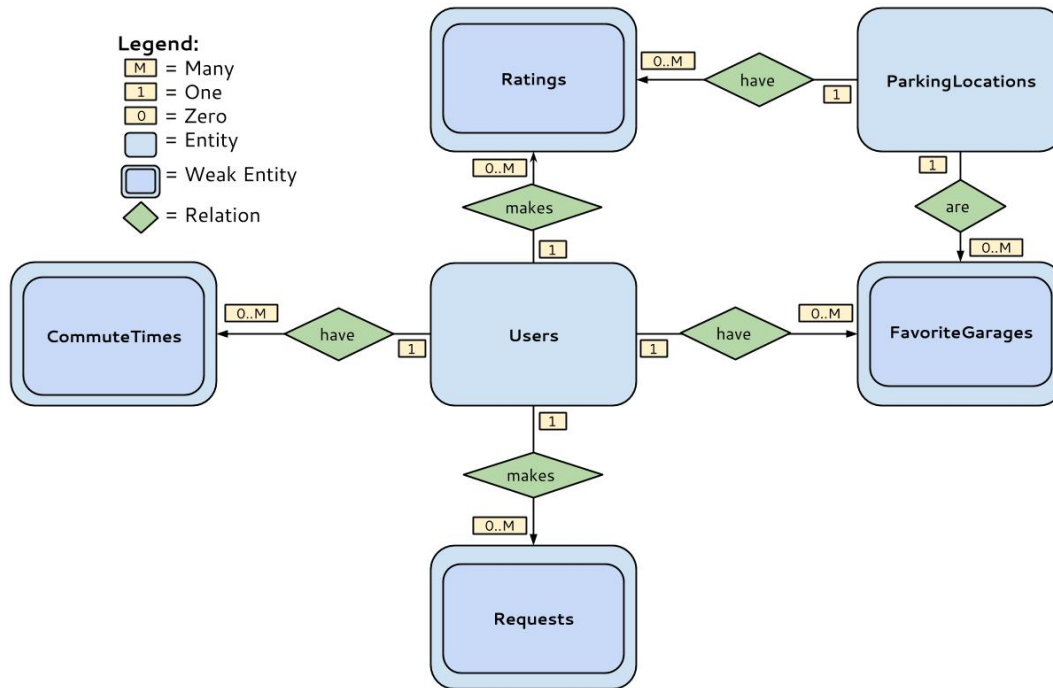


Figure 2: A ER diagram of our database.

Explanation of Requirements Satisfied by the Database Model

The below data is referenced from the above Table 1 and Figure 2, which are representations of our database. The distinct entities and their attributes are viewable in Table 1, while the relationships between these entities are viewable in Figure 2.

Registering/Logging In

The Users table handles all of the registration and login functionality from both the native system and external (Facebook, Google) systems.

When a user registers through the native system on the website, he or she provides a username, password, email, and (optionally) a phone number, and the user is assigned a UserID that is a primary key. The username is stored in the Username attribute, and the password is salted with a salt stored in PasswordSalt and then hashed. The resulting hash is stored in the Password attribute. The email is stored in the Email attribute, and phone number is stored in the PhoneNumber attribute. The ExternalType is set to "native" when signing up through this system.

When a user signs in through Facebook or Google, the ExternalType is set to either "Facebook" or "Google", and the ExternalID is set to the unique ID that is provided to us from the Facebook/Google API.

An admin has his or her UserType attribute set to a certain value, while a regular user has this attribute set to a different value.

Queries to add new users:

Using the native system:

```
INSERT INTO Users (Username, Password, PasswordSalt, Email,
PhoneNumber, UserType) VALUES ('$username', '$password',
'$passwordSalt', '$email', '$phoneNumber', '$type');
```

Using Facebook:

```
INSERT INTO Users (Email, PhoneNumber, UserType, ExternalType,
ExternalID) VALUES ('$email', '$phoneNumber', '$type', 'Facebook',
'$externalID');
```

Using Google:

```
INSERT INTO Users (Email, PhoneNumber, UserType, ExternalType,
ExternalID) VALUES ('$email', '$phoneNumber', '$type', 'Google',
'$externalID');
```

Queries to check login information:

Using the native system:

```
SELECT * FROM Users WHERE (Username = '$user' AND Password =
'$pass');
```

Using Facebook/Google:

If a user logs in through either Facebook or Google, the validation will occur throughout his or her servers.

Request New Garages/Lots

When a user requests a new parking garage, we take the values he or she provides and submit those values into the Requests table in the database. We also have a status attribute that is set to 0 if the request hasn't been reviewed, 1 if it has been rejected, or 2 if it has been approved.

Query to add a request:

```
INSERT INTO Requests (UserID, Name, Address, Cost, Comments) VALUES
('$userID', '$name', '$address', '$cost', '$comments');
```

Report Availability of Parking

A user is allowed to submit a rating for each parking garage and even for a certain level, which will be stored in Level. If the user does not want to rate a specific level the Level value is automatically set to NULL.

The ParkingID is a foreign key that is retrieved from the page of the parking lot that the user is rating. The UserID is also a foreign key, and we get that from the account the user is logged into. The Timestamp is taken from the current time at which the user sends the rating. Finally, the Rating is a number from one to five in the database, which the website displays as: full, 1–5 spots (scarce), empty, etc.

Query to report the availability of parking:

```
INSERT INTO Ratings (ParkingID, Level, Timestamp, UserID, Rating)
VALUES('$parkingID', '$level', '$timestamp', '$userID', '$rating');
```

Favorite Garages

FavoriteGarages is a table that has its own ID, a UserID, ParkingID, and a Priority. Both UserID and ParkingID are foreign keys from the User table and ParkingLocations table, respectively. The FavoriteGarages table uses its own FavoriteID as a primary key. The UserID indicates which user has added a favorite garage, and the ParkingID indicates which garage he or she favorited. Finally, Priority is used to rank the user's favorite garages from most to least important.

Query to get the list of favorite garages of a certain user:

```
SELECT ParkingLocation.Name, FavoriteGarages.Priority FROM ParkingLocations
JOIN FavoriteGarages ON ParkingLocations.ParkingID =
FavoriteGarages.ParkingID WHERE FavoriteGarages.UserID = '$user' ORDER BY
FavoriteGarages.Priority;
```

Query to add a users favorite parking garage:

```
INSERT INTO FavoriteGarages( UserID, ParkingID, Priority) VALUES ('$userID',
'$parkingID', '$priority');
```

Notifications/Commute Times

Every user can manage a list of commute times for when he or she has to be at school and/or work. In the CommuteTimes table, a UserID foreign key references the Users table in order to match up users with their commute times. A user can add a new commute time by specifying a time (Time), a day of the week (Day), and how long before the specified time he or she wants to receive a notification (WarningTime).

Users can also choose not to be notified for particular commute times. In this case the WarningTime is set to NULL in the database. Whenever it is time for a user to be notified, he or she is sent an email and/or text (if the user has provided a phone number in his user info) with information about the parking availability of his favorite parking locations.

Query to add a new commute time:

```
INSERT INTO CommuteTimes (UserID, Time, Day, WarningTime) VALUES
```


('userID', 'time', 'day', 'warningTime');

Query to retrieve a user's commute time list, ordered first by day of the week and then by time:

```
SELECT Day, Time, WarningTime FROM CommuteTimes WHERE  
UserID='userID' ORDER BY Day, Time;
```

Query to retrieve the current capacity (an average of recent ratings) of a user's favorite garages, ordered by the priority of favorite garages, to be sent to the user through email/text:

```
SELECT ParkingLocations.Name, avg(Ratings.Rating) FROM Ratings NATURAL  
JOIN ParkingLocations NATURAL JOIN FavoriteGarages WHERE  
Ratings.ParkingID IN (SELECT ParkingID FROM FavoriteGarages WHERE  
UserID='userID') AND Ratings.Timestamp>DATE_SUB(NOW(), INTERVAL 1  
HOUR) GROUP BY Ratings.ParkingID ORDER BY FavoriteGarages.Priority;
```

List of the Least and Most Active Times for Each Garage

For each garage, we average the ratings made during each hour of the day. This allows us to report the typical occupancy of a garage at each hour of the day. We make use of the Rating and Timestamp attributes in the Ratings table to obtain the average for each hour. We join the ParkingLocations table with the Ratings table through the ParkingID so that we can get the ratings for each specific garage in ParkingLocations.

Query:

```
SELECT HOUR(TIME(Timestamp)) AS 'hour', avg(Rating) FROM Ratings WHERE  
ParkingLocations.ParkingID = 'parkingID' GROUP BY 'hour' ORDER BY Rating  
DESC;
```

List View of Parking Locations

For each parking location, we store the parking location's name, address, cost (if it is known), the number of levels (if it is a parking garage), and any additional comments. When we join this ParkingLocations table with the Ratings table, we are able to provide all relevant information about each parking location. For each parking location, we can show basic information such as the name, location, and cost. We can also show what the average capacity during the past few hours. This average capacity can be either for the whole parking lot/garage, or for a specific level of a parking garage. A user is able to search for parking locations by either name or location.

Get a list of all the parking locations, with basic info on each location as well as a recent average rating:

```
SELECT ParkingLocations.Name, ParkingLocations.Address, avg(Ratings.Rating)
```

```
FROM ParkingLocations NATURAL JOIN Ratings WHERE  
Ratings.Timestamp>DATE_SUB(NOW(), INTERVAL 1 HOUR) GROUP BY  
Ratings.ParkingID;
```

Map View of Parking Locations

We are going to store the address of each parking location, so we can use the Google Maps API to view each parking location on the map.

QR Code

Each parking location has a ParkingID primary key, which is a unique integer to identify each location. Each QR code contains a link that has the ParkingID. An example of such a link is: **www.ponypark.com/location/48364**

When a person scans the QR code, they are taken to the link, which shows all the information about the parking location associated with the ID in the link.

Top 10 Contributors

Using the Ratings table and Users table, we are able to find out how many ratings a user has contributed, and we are able to rank the ten users who have contributed the most ratings.

Query:

```
SELECT Username, count(Ratings.Rating) FROM Users JOIN Ratings ON  
Ratings.UserID = Users.UserID GROUP BY UserID ORDER BY  
count(Ratings.Rating) DESC;
```

Detailed Software Architectural Diagram

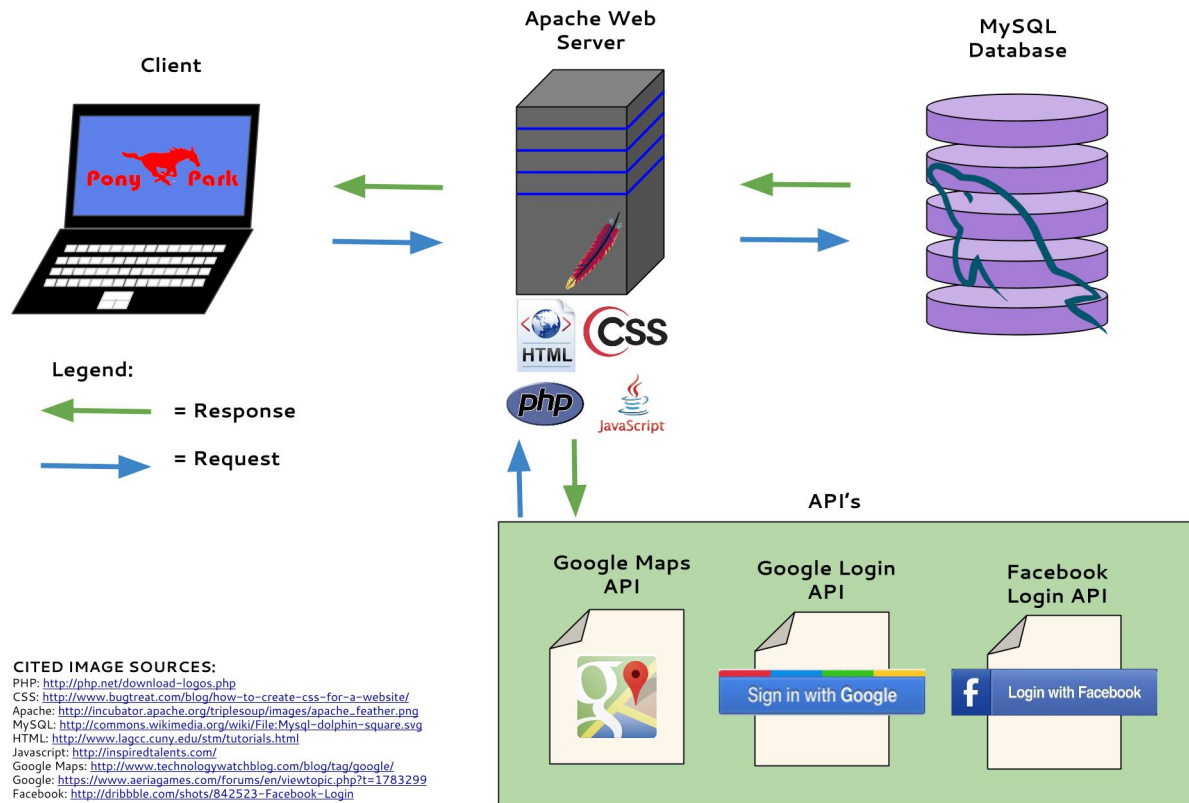


Figure 3: Detailed Software Architecture Diagram.

Figure 3 shows our Detailed Software Architecture Diagram which represents all the different components we utilize when running our program. It includes how requests and responses are made in our system.

Paper Prototypes

Figures 4–12 show the web page paper prototype

Figures 13–19 show the paper prototypes for the Android Application

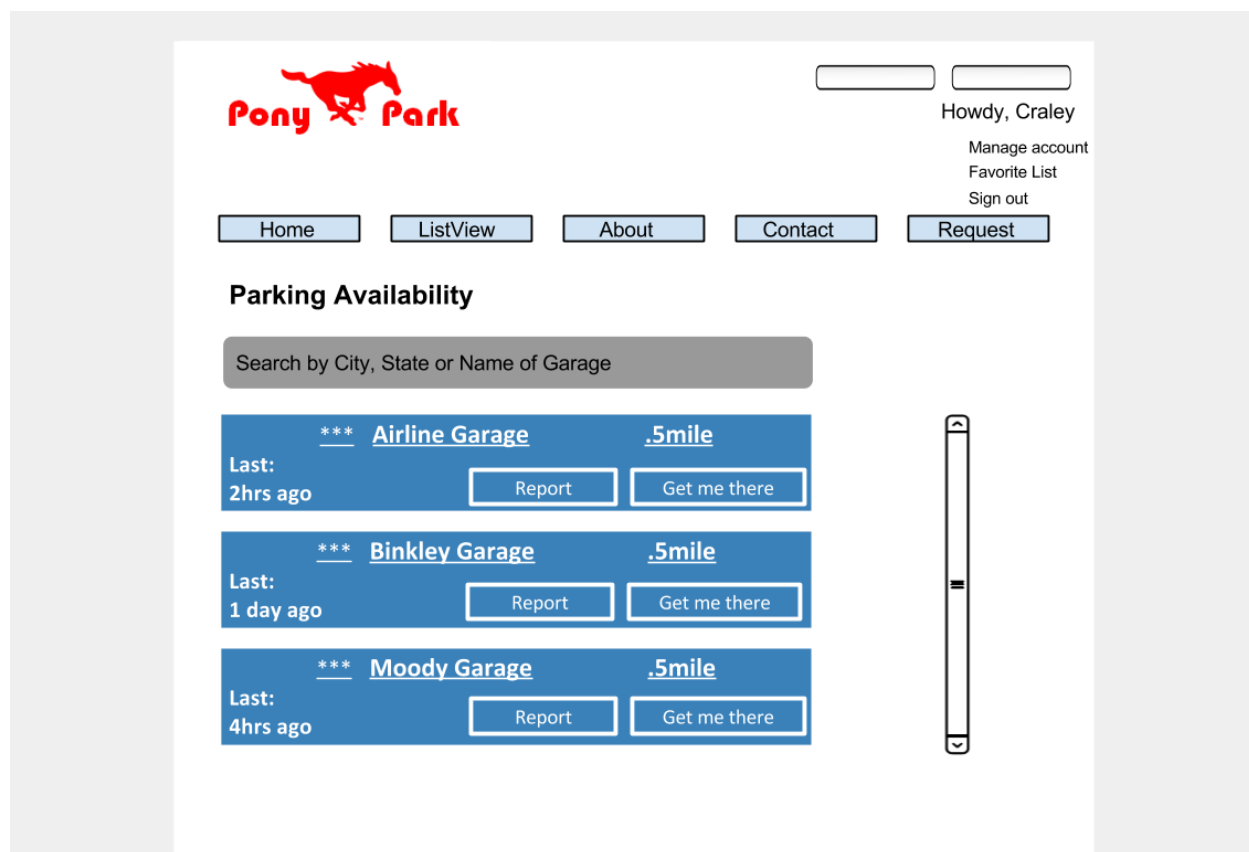


Figure 4

Figure 5

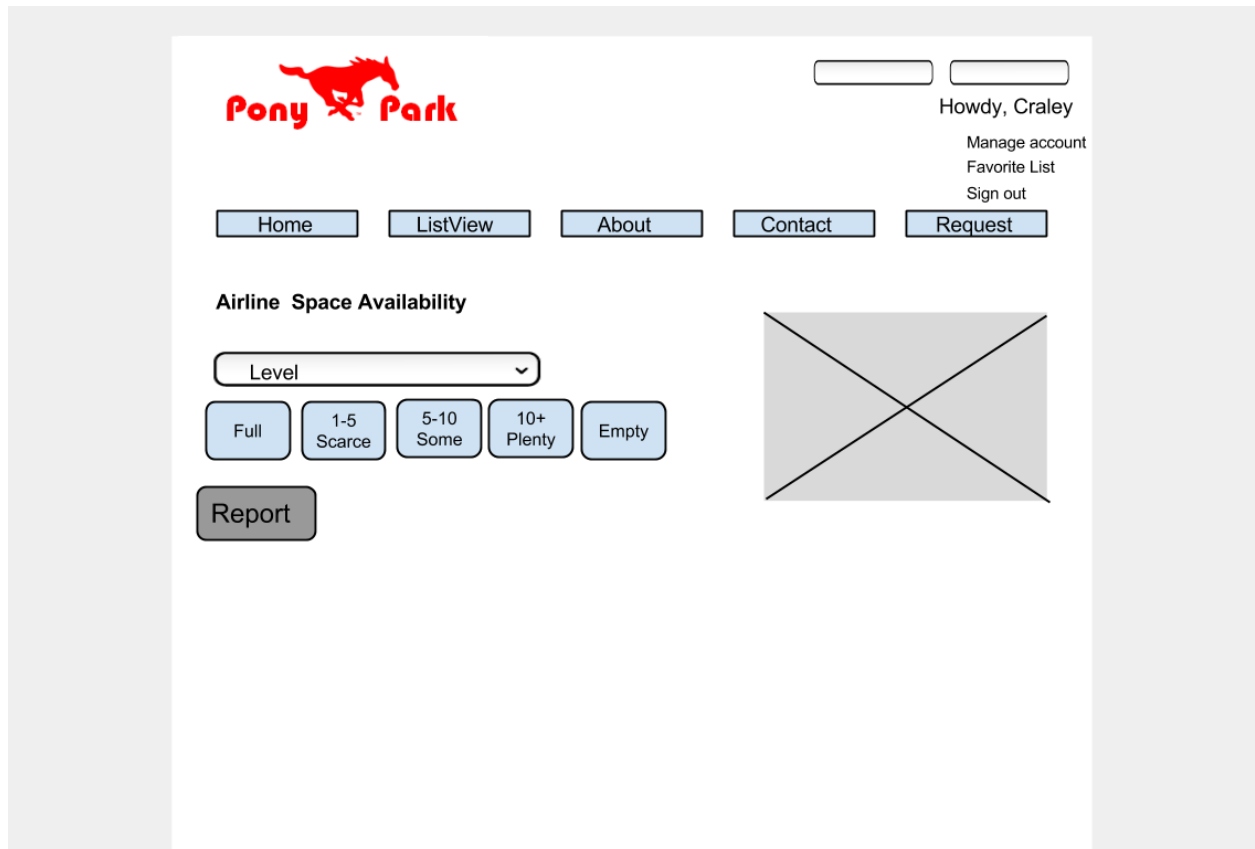



Figure 6



Howdy, Craley

Manage account

Favorite List

Sign out

Home

ListView

About

Contact

Request

Request Form

Garage Name

Cost

Address

Levels

Message

Send

Cancel

Garage Photo





Figure 7

The image is a wireframe of a web page. At the top left is the 'Pony Park' logo, which includes a red silhouette of a running horse. To the right of the logo are two empty rectangular input fields, each followed by the text 'Link'. The main content area is a light blue rectangle with a large, rounded, light gray modal box in the center. The modal box is titled 'Sign Up' and contains three input fields labeled 'Username:', 'Email:', and 'Password:'. Below these fields is a 'Sign Up' button. At the bottom of the modal box is a paragraph of placeholder text: 'Lorem ipsum...Lorem Ipsum....Lorem Ipsum...lorem ipsum....lorem ipsum'. The modal box is flanked by two light blue vertical bars, each with a 'Link' label at the top. The entire page is set against a light gray background.

Figure 8

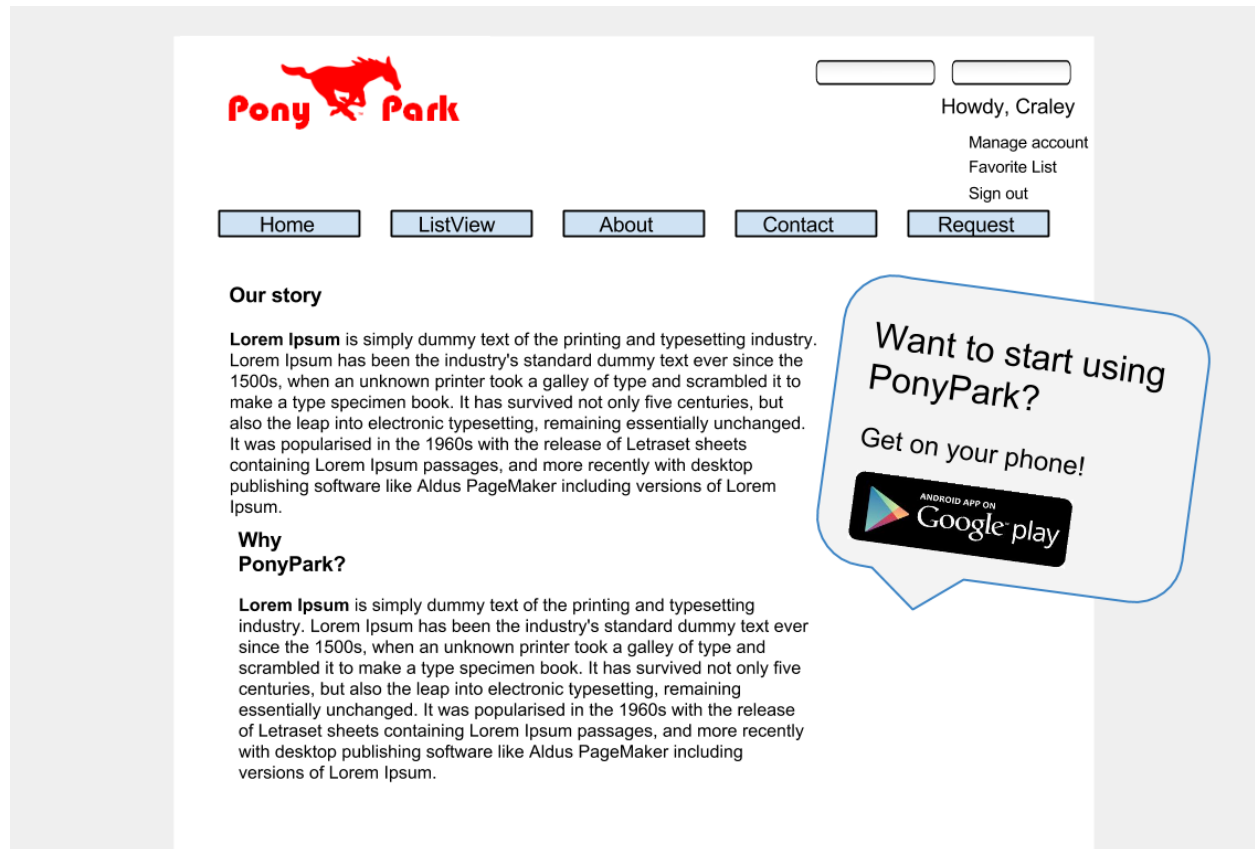



Figure 9



Howdy, Craley

Manage account

Favorite List

Sign out

Home

ListView

About

Contact

Request

Contact Us

Your Name

Email

Phone

Message

Submit

Cancel

Map of business

BAM Software
111 SMU Blvd.
Dallas, TX
888-888-8888

Figure 10

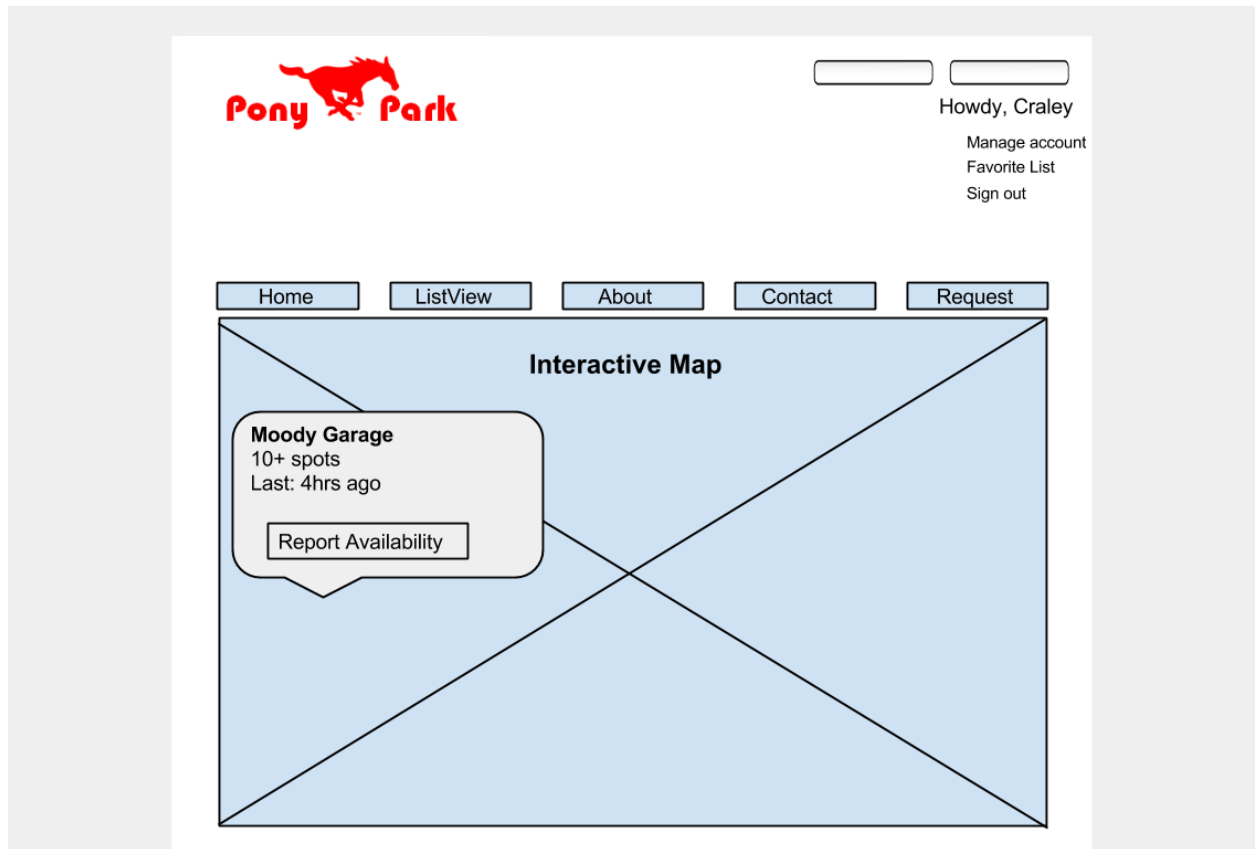



Figure 11



Sign Up

Login

Manage account
Favorite List
Sign out

Home

ListView

About

Contact

Request

Contact Us

Your Name

Email

Phone

Message

Submit

Cancel

Map of business

BAM Software
111 SMU Blvd.
Dallas, TX
888-888-8888

Figure 12

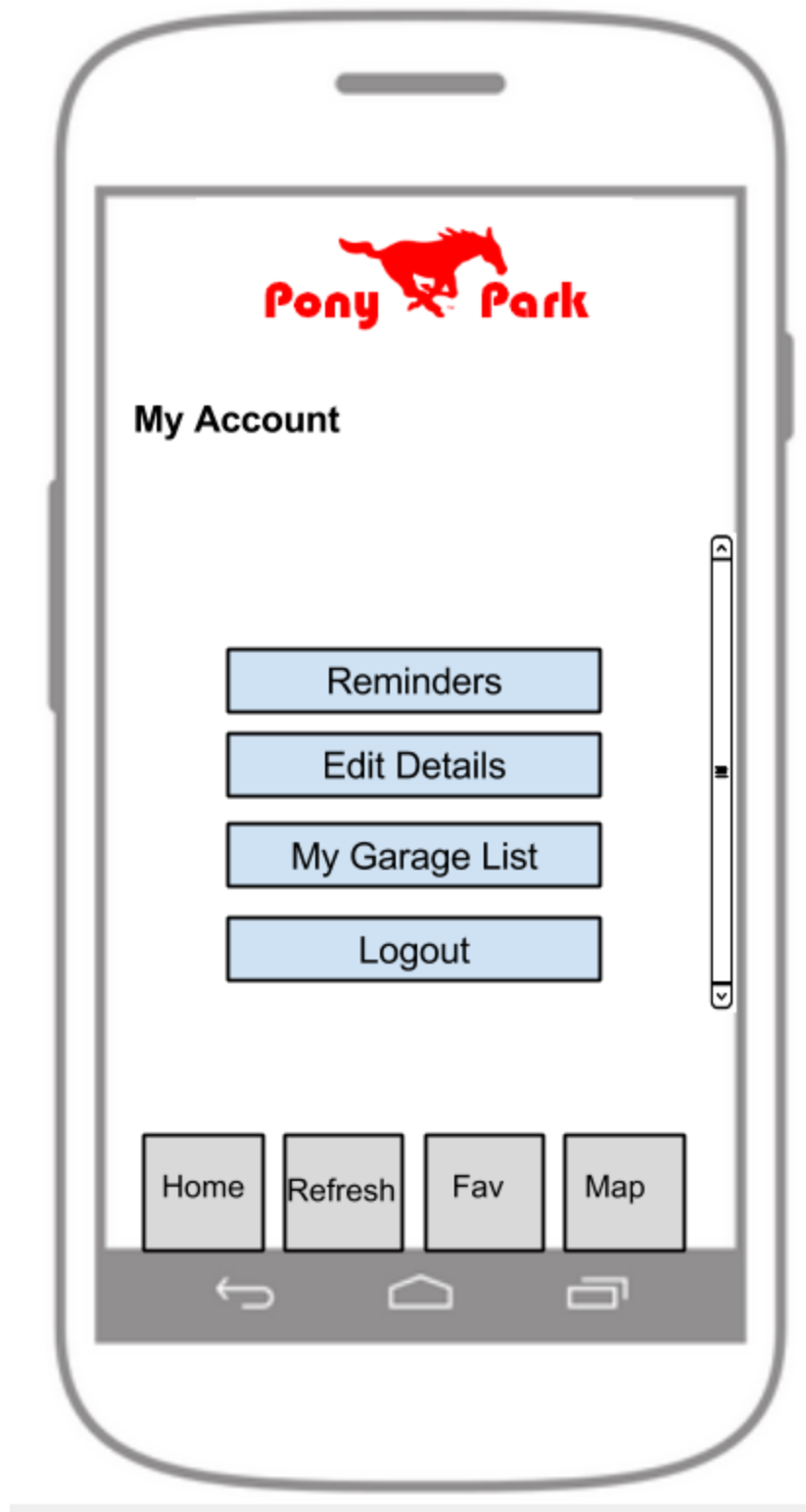


Figure 13

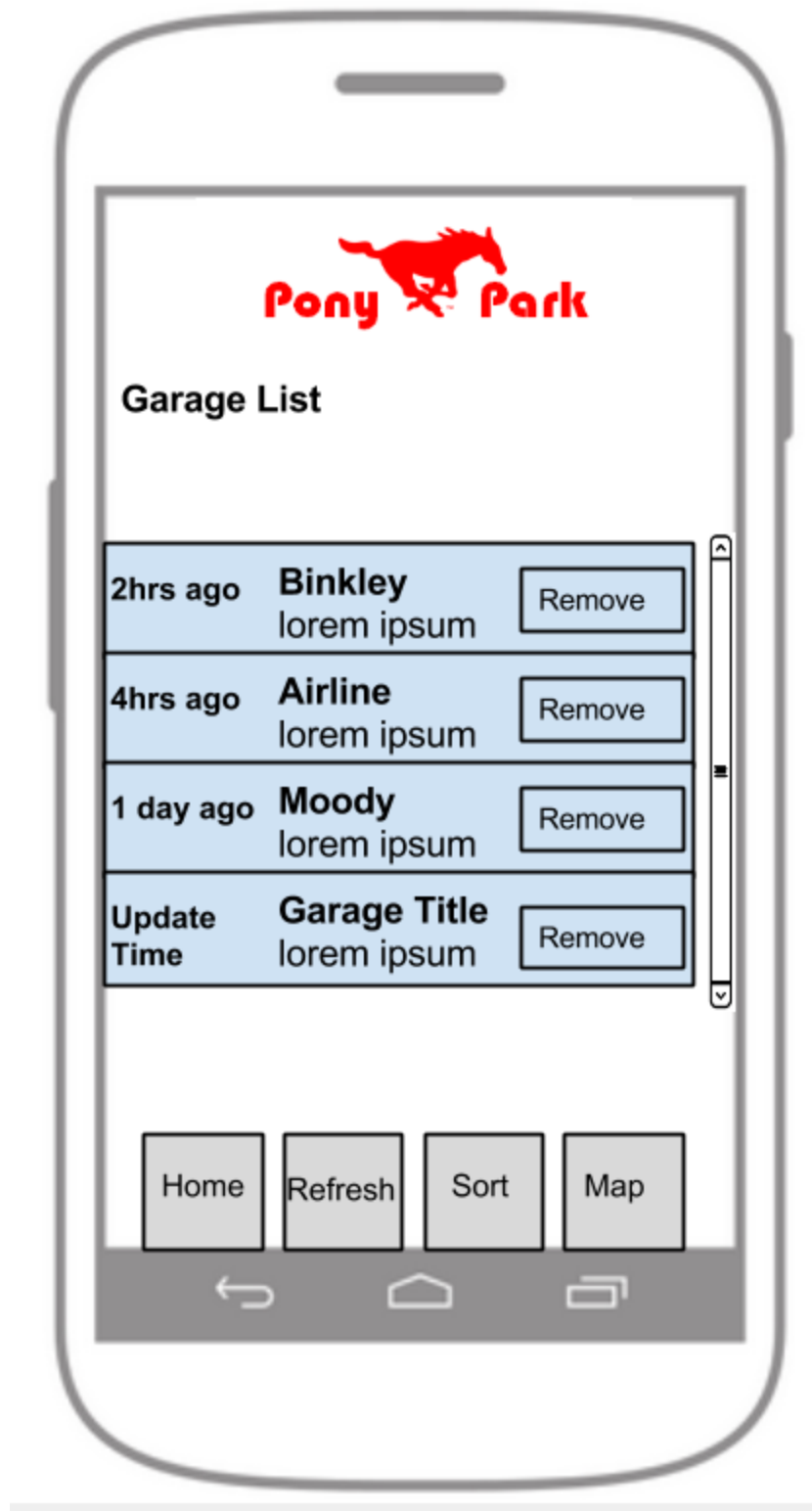


Figure 14

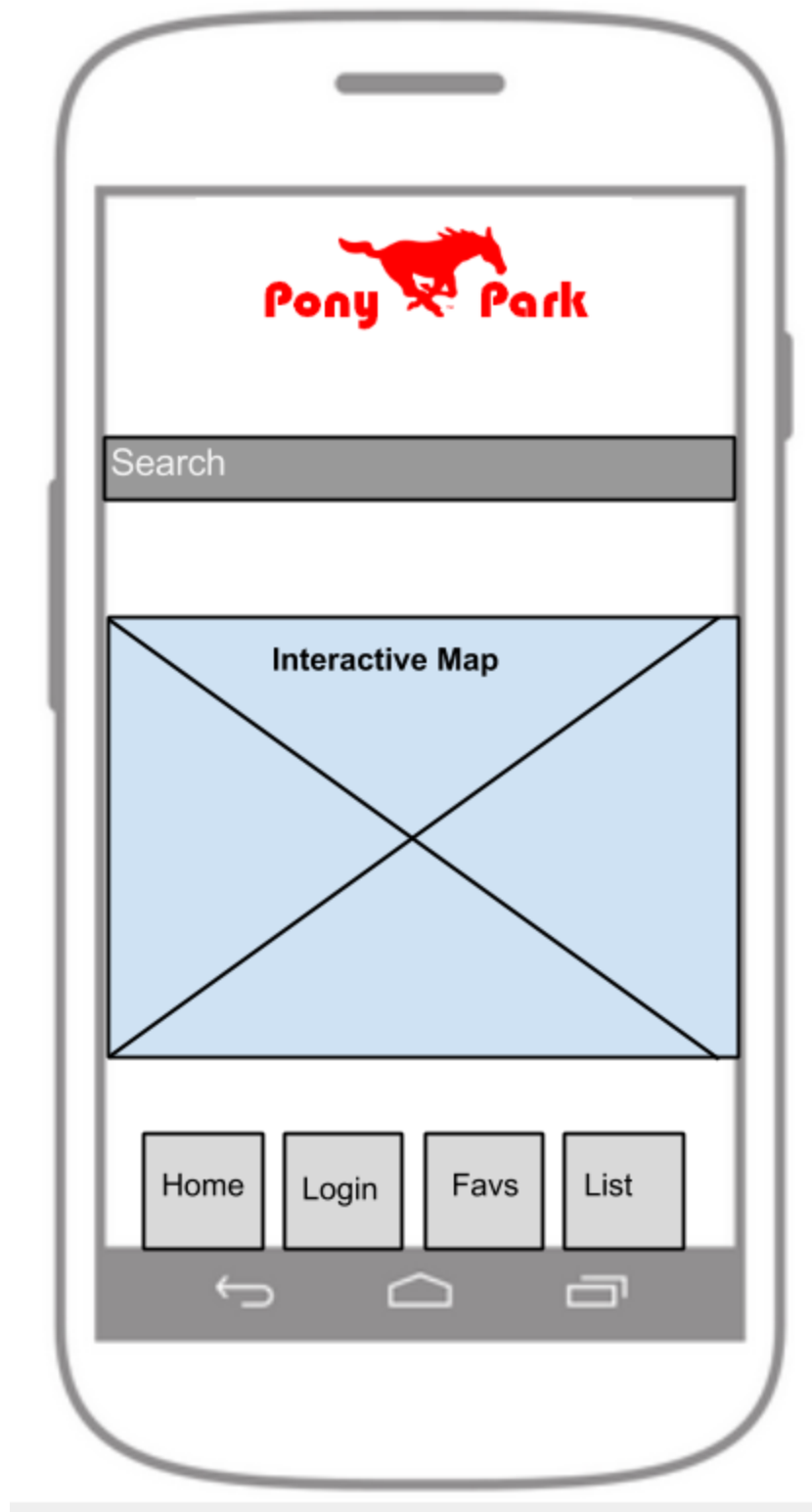


Figure 15

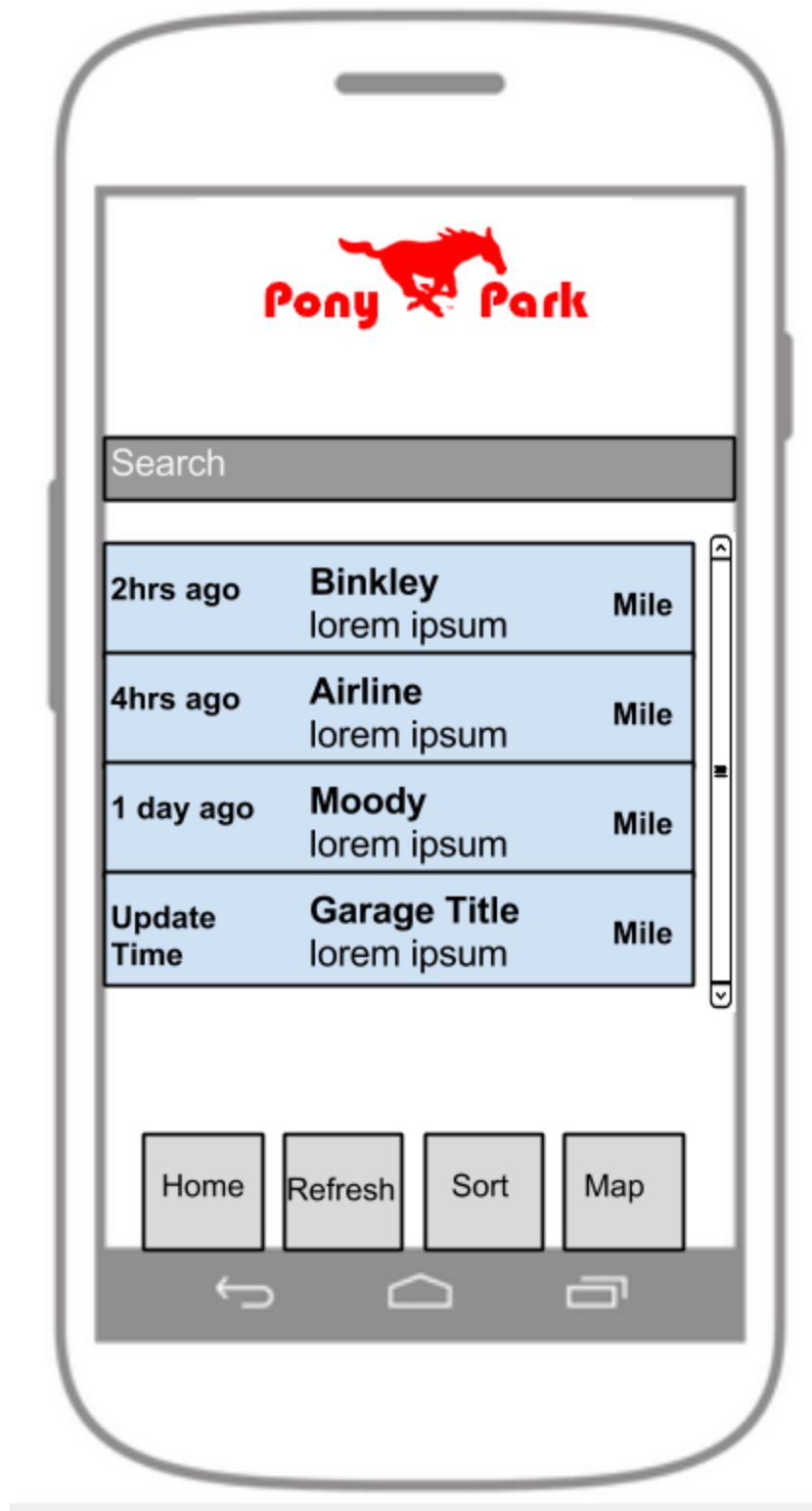


Figure 16

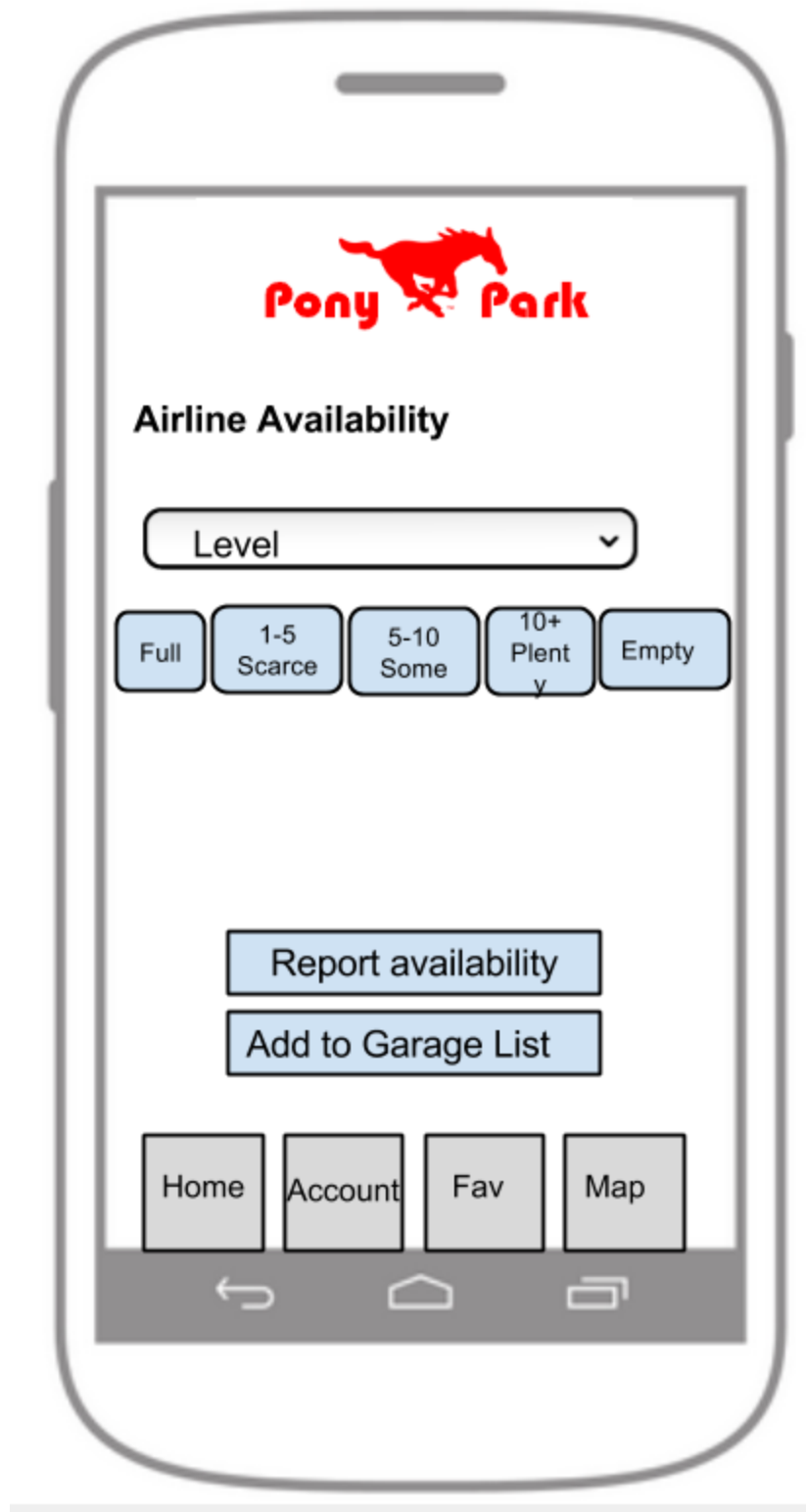


Figure 17

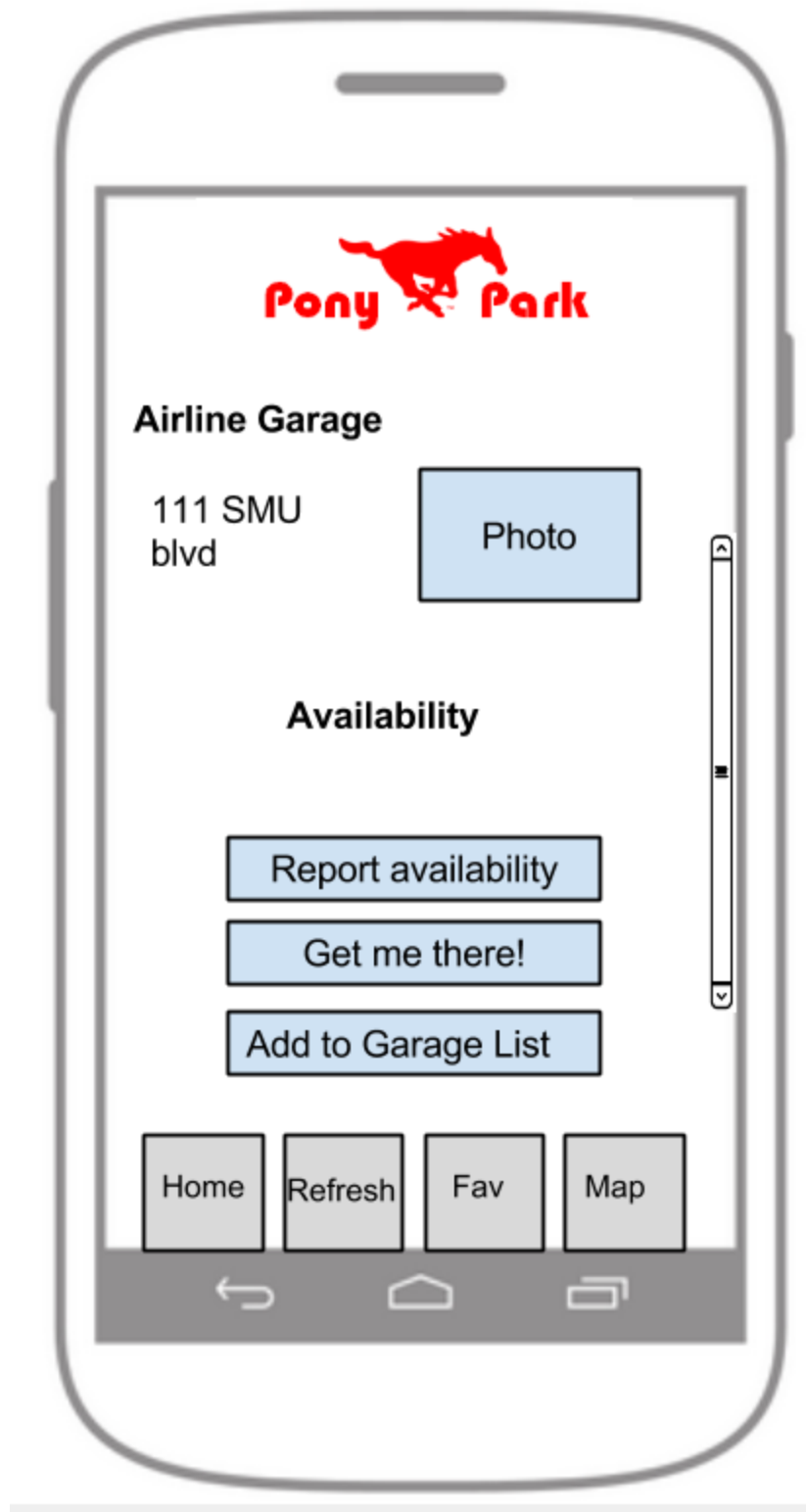


Figure 18

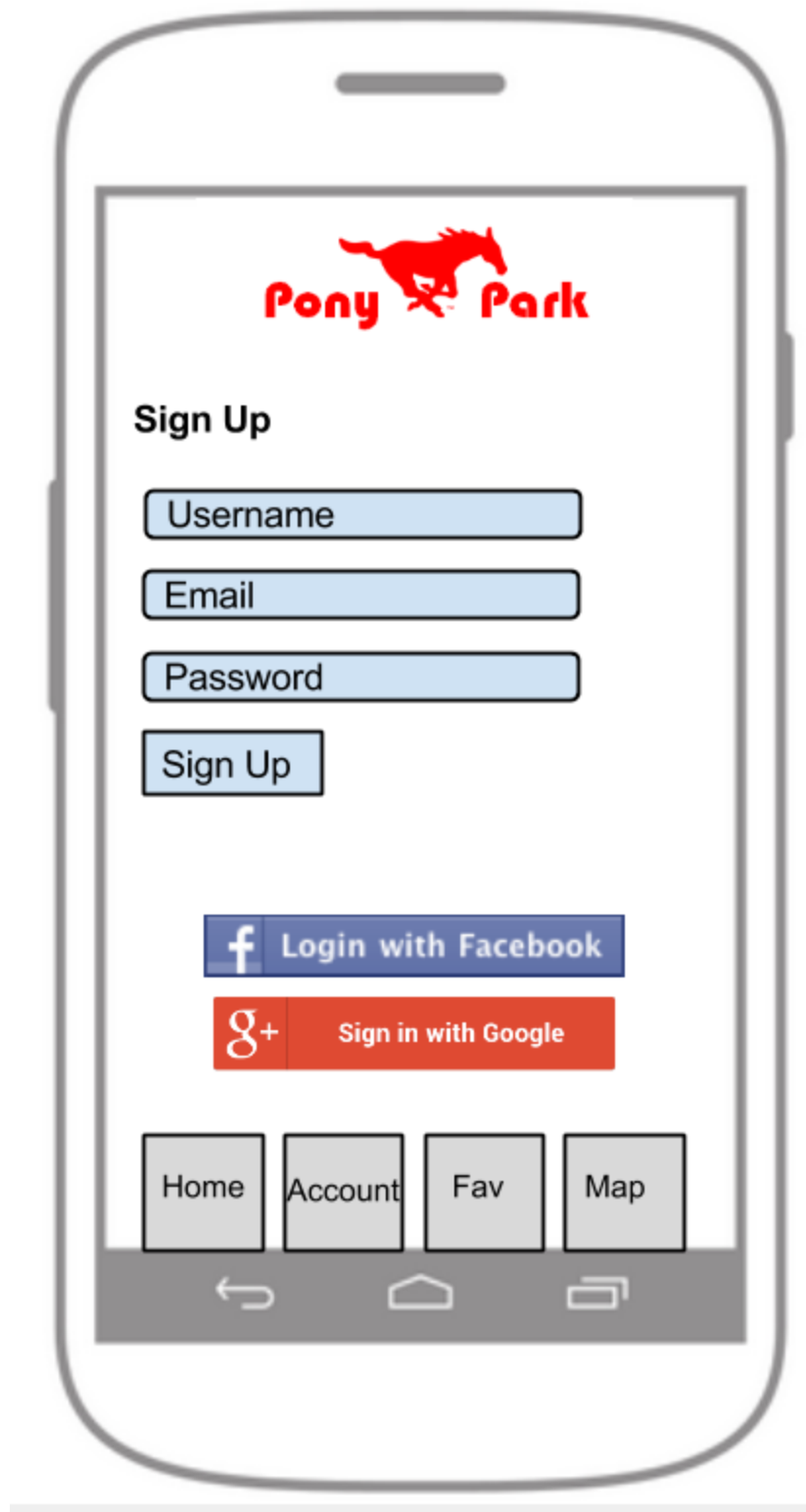


Figure 19

User Profiles

College Commuter (CC):

- Full-time college student
- 19 years old
- Lives 30 minutes away from campus
- First class is at 11 am every day.
- Sometimes ends up late to his first class due to difficulties finding any parking
- Likes to eat off-campus
- Fairly tech-savvy, like most college students
- Likes sharing things on social networks at all times
- Dislikes reading instructions

Working Commuter (WC):

- Full-time office worker (9–5)
- 42 years old
- Lives 2 hours from office
- Always leaves ½ hour early to ensure she finds a parking space
- Wants to minimize her commute time in order to sleep more
- Has a smart-phone but only uses it for texting/email/news
- Has Facebook but rarely uses it

Downtown Traveler (DT):

- Retired
- 70 years old
- Likes going to museums often
- No set schedule
- Doesn't have a smart-phone
- Doesn't use social media
- Dislikes small text

Admin (AD):

- College graduate
- 23 years old
- Maintains PonyPark as a part time job
- Very adept at using the Internet
- Little to no coding experience

Technical Analysis Based on User Profiles

Q: What are some of the challenges you face during your commute to work/school?

A:

[CC] – I always have a hard time finding a parking spot. Sometimes I have to visit several different garages before finding a spot.

[WC] – Besides traffic, finding a parking spot is very challenging. Oftentimes, the garage will need to be circled several times to find a spot.

[DT] – As a retiree, I do not have a specific commute. However, traffic and parking were always a concern for me. I tried to take public transit when possible.

[AD] – I live very close to work, I often walk or ride my bike.

Q: Do you often leave work/school, perhaps for lunch or a break, and then return via motor vehicle?

A:

[CC] – I head back to my apartment whenever I have a break to grab some food. I hate eating and staying on campus when I don't have to be on campus.

[WC] – When I find a parking spot in the morning, that's where my car stays the entire day. I pack my lunch.

[DT] – When I park in a spot, I usually don't leave unless I don't have to come back for a while.

[AD] – I'll usually ride my bike to go and get lunch.

Q: When you need to park, do you have an assigned place or are there several options?

A:

[CC] – We have several options for parking. The type of permit you have on campus determines which options are available to you.

[WC] – We have one multi-story garage, and we have to make the single garage work for all the employees.

[DT] – When visiting museums, there is usually a parking lot and meters around the area. However, depending of the museum, the lots can fill quickly.

[AD] – Each employee has an assigned parking place.

Q: In terms of reducing the stress of a commute, how helpful would it be to know the general location of available parking?

A:

[CC] – My stress level would significantly decrease if I knew a certain garage had some availability.

[WC] – The traffic would likely keep my commute stressful, but knowing a general location of availability, such as a floor, would save me a lot of time.

[DT] – Parking is often difficult when traveling downtown, if I knew where some spaces were available, that would definitely reduce my anxiety.

[AD] – I bike or walk most places.

Q: Would you trust your peers, co-workers, or random strangers to give you advice on where parking is available?

[CC]– Absolutely, I often let my buddies know where I parked so they can go straight there to find a spot.

[WC]– I don't know every single person I work with, but if someone told me there was parking on a certain level, I'd definitely go straight to that floor.

[DT] – I would be more than willing to follow and trust the advice of someone, even a stranger.

[AD] – I'm always skeptical of strangers, but it couldn't hurt to see if what they're saying is true.

Q: If you were to inform a neighbor, friend, or stranger on where there was available parking, how would you do so?

A:

[CC] – When I tell my buddies where to park, I will include the garage and floor.

[WC] – I would include the floor and how close (or far) the nearest empty space was to the elevator.

[DT] – I would inform the person of where the lot was, about how many spaces I saw, and include landmarks to aid them in their search.

[AD] – I guess I would say how close to the destination I parked and qualitatively describe the availability of the lot.

Q: How likely would you be to inform this neighbor, friend, or stranger on the availability of parking?

A:

[CC] – I would be very likely. I am a strong believer in what goes around comes around. I feel if I help them, someone will help me in the future.

[WC] – I guess I wouldn't mind. I just wouldn't want to be the one always-giving advice; I would want to receive some too.

[DT] – Yes, I most certainly would help out a fellow stranger. It doesn't hurt me to help, so why not?

[AD] – I wouldn't mind helping out someone with parking availability.

Q: How likely would you be to give this advice if it required reporting over a computer, smartphone, or tablet?

A:

[CC] – Wow, that seems really easy. I would definitely do that, provided others give advice as well.

[WC] – Yeah, I suppose something like that would be neat. It would have to be user friendly. I hate complicated software.

[DT] – Why would I need to use an electronic device to give advice? Seems like a lot of work. I probably would not do it.

[AD]– I would use my smartphone to report conditions/availability when I drive.

Q: With respect to reporting availability, would you be more likely to report qualitatively or quantitatively?

A:

[CC] – I think it would be difficult to give a quantitative response.

[WC] – A qualitative response would probably be easier.

[DT] – Quantitative would definitely be easier. It would be nice to have an estimated count of spaces.

[AD] – I feel like quantitative would be difficult for the user. Qualitative has my vote.

Q: With respect to viewing availability, would you be more satisfied with a qualitative or quantitative report?

A:

[CC] – It would be nice to a number of spots available.

[WC] – A quantitative report would definitely be more helpful.

[DT] – Quantitative would allow me know how many spots are available.

[AD] – Quantitative would help with knowing how much longer a garage will be “open”.

Q: What other features would you find useful in an application that allowed you to report and view availability?

[CC] – A feature that would tell me the availability of a garage about 10 minutes before I left for class would be nice.

[WC]– If a point system that rewards users for reporting was created, it would help encourage reports. If there is no incentive, the idea will fail.

[DT]– I often find that construction leaves maps and useful documentation outdated. Perhaps a nice function to request the addition or removal of lots and garages.

[AD]– The system should be easily manageable by admins without having a lot of coding experience.

Q: As an admin, what features would you need to be able to control?

[AD] – Admins definitely need to be able to edit information about certain garages. Traits such as addresses, levels, and restrictions should be able to be modified by an administrator. Admins should definitely be able to add/remove garages/lots.

Object/Action Analysis and Matrix

ParkingLocation

Attributes

1. ParkingID
2. Name
3. Address
4. Cost
5. Comments
6. NumberOfLevels

Actions

User Role: Admin

1. Create
2. Delete
3. Edit

User Role: All

1. View

Rating

Attributes

1. RatingID
2. ParkingID
3. UserID
4. Timestamp
5. Rating
6. Level

Actions

User Role: User

1. Submit

User Role: All

1. View

User

Attributes

1. UserID
2. Username
3. Email
4. Password
5. PasswordSalt
6. PhoneNumber
7. UserType
8. ExternalType
9. ExternalID

Actions

User Role: User

1. Create account
2. Delete account
3. View account settings
4. Edit account settings

CommuteTime

Attributes

1. CommuteID
2. UserID
3. Time
4. Day
5. WarningTime
6. TimeOfNotification

Actions

User Role: User

1. Create
2. Delete
3. View
4. Edit

Request

Attributes

1. RequestID
2. UserID
3. Name
4. Address
5. Cost
6. NumberOfLevels
7. Comments
8. Status

Actions

User Role: User

1. Submit

User Role: Admin

1. View all requests
2. Approve
3. Decline

FavoriteGarage

Attributes

1. FavoriteID
2. UserID
3. ParkingID
4. Priority

Actions

User Role: User

1. Create
2. Delete
3. View

		Create/Submit	Delete/Decline	View	Edit	Approve
ParkingLocation	ParkingID	A	A	X	A	
	Name					
	Address					
	Cost					
	Comments					
	NumberOfLevels					
Rating	RatingID	U		X		
	ParkingID					
	UserID					
	Timestamp					
	Rating					
	Level					
User	UserID	U	U	U		
	Username					
	Email				U	
	Password				U	
	PasswordSalt					
	PhoneNumber				U	
	UserType					
	ExternalType					
	ExternalID					
CommuteTime	CommuteID	U	U	U		
	UserID					
	Time				U	
	Day				U	
	WarningTime				U	
	TimeOfNotification				U	
Request	RequestID	U	A	A		A
	UserID					
	Name					
	Address					
	Cost					
	NumberOfLevels					
	Comments					
	Status					
FavoriteGarage	FavoriteID	U	U	U		
	UserID					
	ParkingID					
	Priority					
Color-Type Relationships						
ParkingLocation	User					

Figure 20 – Object/Action Matrix (pictured above)

Software Lexicon

Request: Some might get confused of what this means but simply allow a user to request a garage to be added to the system.

Context: *John wants to **request** Airline parking garage to be added.*

Favorites List: A list created by a user which allows them to keep track of certain garages which they might like get daily updates on.

Context: *Susan said I just added Airline parking garage to my **favorites list**!*

Cost: The cost field on the request form allows a user to input cost if the parking garage has fees associated with it.

Context: *John just requested a garage to be added that **costs** visitors \$4.00 to park!*

Listview: Instead of just showing a map of the parking garages, a listview can be selected which will show available parking garages in a row oriented format.

Context: *Susan just launched her PonyPark app on her phone and went straight to the **listview** screen.*

Report: Allows a user to report the availability of parking last seen at a garage.

Context: *Susan just **reported** Airline had 10+ parking spaces available.*

Commute times: A defined range of times set by the user that represents the times which they would typically be commuting to campus.

Context: *John said, "I want the ability to be notified automatically at certain times of the parking garage status." And then Susan said, "Well just go and add your typical **commute** times to campus and be notified before those times."*

Usability Test Outline

We will use the hallway testing method for our usability test. According to the Nielsen Norman Group, the best results in testing occur when five people are tested¹. Our hallway test will consist of testing four random people (Professor Raley will be our fifth user). Using the hallway test for the testing method will help ensure that users are chosen randomly and are not colleagues or IT experts. In order to not drive users away from testing our prototype, we need to make the testing process quick, but effective. We can identify three key features of our service: report availability, access garage availability, and request the addition of a garage to the system.

Our test will encompass “quizzing” the user on the three main tasks of the service. Before we quiz the user, we will describe the purpose of our service. The user will then be asked how they expect the site to look. The answer to the question does not need to be detailed. We just need to be able to understand the user’s expectations.

In order to successfully carry out this test, each GUI team member will need to have a specific role. The role of facilitator, computer and log keeper will need to be fulfilled. Justin will be the computer. The computer role will be the person who changes the screen upon selection by the tester. Jim will be the facilitator of the test. He will describe our service, ask the expectation question, and then quiz the user on accomplishing the three tasks. Evan will be the log keeper. One of the roles of the log keeper is to evaluate what the user may believe about the system. The log keeper will have the most crucial role, as they will record where the user gets “stuck” or has difficulty. It is the hope that these three roles will allow us to properly evaluate the effectiveness of our design.

Upon ending the testing of the tasks, the user will simply be asked what they liked or disliked about each task. The response to this question will be recorded, and used to measure task efficiency. Finally, we will ask the user if they believe this application would be useful in their everyday life, and what improvements they believe should be made.

Once the user has been dismissed and the process is repeated four more times, the data will be analyzed. We will classify each trouble spot, prioritize the problems, and work to fix them.

While no testing method can grab 100% of the problems, we believe this method will allow us to catch the maximum number of issues, and improve the user’s experience with the PonyPark application.

¹<http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/>

Usability Report

Our usability test consisted of testing the above procedures on five individuals, including Professor Raley. When first opening the service, users expected to be introduced to a simple screen. When users were asked to complete the three tasks, they had very little difficulty reporting the availability of a garage. However, some did comment that the term report would be better replaced with rate. When users were asked to access a garage's availability, some problems arose. Some users found the search bar confusing or unnecessary given the service's restriction to SMU. Another user found the list view should be more explicit in the availability, rather than stars. Since accessing availability is a key feature of our service, it seems we have to make this a little more user friendly. Finally, we asked users to request the addition of a new garage to our system. All users struggled with the addition of an address. Perhaps we could just add a box that offers a short description of location instead of an address. Some users also expressed concern with the verbiage send. Perhaps submit would be more appropriate. When the users were asked what they liked about our system, they all said it was fairly easy to complete each task. However, many expressed concern with the default of map view on the homepage. Perhaps the two can be combined in some nature. All users noted that this service would be useful in their everyday life. Before the user left, we asked what suggestions they have for improvement. We received numerous tips. Some tips included combination of map/list view, push notifications on mobile device, and radio buttons for reporting availability.

To make improvements on our service easier, the biggest problems have been noted and rated below (one is biggest priority, 5 is least):

1. Very hard to understand availability in list view – medium
2. Map view makes easy access difficult – annoying
3. Confusing verbiage (rate/report, floor/levels, send/submit) – annoying
4. Difficult to obtain address when requesting– annoying
5. Confusing search bar – annoying