6-13-2014

Location Based Recommendation Application

Nicholas Dario
Santa Clara University

Steven Goetter
Santa Clara University

Christopher Polson
Santa Clara University

SANTA CLARA UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

Date: June 12, 2014

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Nicholas Dario
Steven Goetter
Christopher Polson

ENTITLED

Location Based Recommendation Application

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

[illegible]
Thesis Advisor

[illegible]
Department Chair

Department Chair

Location Based Recommendation Application

by

Nicholas Dario
Steven Goetter
Christopher Polson

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 13, 2014

Location Based Recommendation Application

Nicholas Dario
Steven Goetter
Christopher Polson

Department of Computer Engineering
Santa Clara University
June 13, 2014

ABSTRACT

For our senior design project we decided to make an iOS application that could generate a list of nearby locations for the user to visit. We wanted the recommendation list to be unique for each user instead of a list of the most popular locations in the area. To accomplish this we developed our own recommendation algorithm from scratch. The algorithm uses a tagging system in which users and system administrators are able to add and modify the tags associated with locations. By using the tags associated with each location and with each user, our algorithm is able to generate a recommendation list tailored to the interests of each user. We made an intuitive user interface by creating a simple, clean layout for each screen the user see. We wanted the users to be able to glance at the screen and immediately know what is possible to do while at that screen and what is the purpose of said screen. We have clearly labeled buttons for each of the possible actions, and use color coordination to draw the user's attention to important information.By the end of the project we had an application with an intuitive user interface and an algorithm that could achieve our goal. We still need to link the application, server, and algorithm together, so that they act as a single system. One this is done the core project will be functional, and we can begin adding in additional features.

Table of Contents

List of Figures

Chapter 1

Acknowledgments

Through the entirety of our project we have reached out to many individuals and received a tremendous amount of help and feedback. Ranging from the support from our advisor, assistance from multiple sources regarding the technologies we employed, and the members of our test group, we had many resources at our disposal that contributed to the overall success of our project.

Among many individual who influenced our project we would like to thank the following:

Dr. Yi Fang

Our Computer Science and Engineering advisor.

Dr. Melissa Donegan

Our Thesis advisor.

SCU IT Department

For giving us a static IP address.

Dr. Ruth Davis

Assisting with communication and follow through from IT department.

Various Friends

Help setting up server and forming test group.

Chapter 2

Introduction

2.1 Problem Statement

From restaurants to hair salons, bookstores to parks, people often settle for the most convenient location to fulfill their needs. The most convenient location is often the most obvious one. Searching for a good place can be tedious, and taking the time to investigate new locations can be more work than people are willing to invest. Because of this, people may be unaware of a location that is best suited to them simply because it is not primely situated. A system that directs users to locations that match their interests and needs would benefit businesses and users alike.

2.2 Background

When users search for places, applications like Yelp and Foursquare are currently the common vehicles of choice. These programs give users the chance to review a location and rate it on a scale of zero to five stars. These star rating systems are poor metrics for describing a location. When people have different sets of values than the norm, popular locations with high star ratings are not always the best choice for them. Sifting through reviews is the only way for such users to identify whether they might like a particular place. Reviews are not always available, and it may take more time than users are willing to invest to read more than a few. As it stands, people have no easy way of discerning which location best suits them. Searching for places in person favors those situated in obvious locations, and applications like Yelp favor popular opinion.

2.3 Proposed Solution

Our solution is a mobile application that recommends places to people based on a person's interest and the attributes of a location. This method forgoes a five-star rating and builds a description of locations based on short attribute tags. Users can tag specific features of a location. The application will use these tags to build a description of locations. It will also build a profile of our users. Profiles will be based on tags that users can apply to themselves. By matching the attributes of a location with the interests of a user, the application will direct people to locations they enjoy and feel comfortable in. This is more effective than systems that evaluate businesses based on general popularity. Our system takes into account the possibility that a popular place is not the best option for everyone, and that often times people have tastes that differ from the norm. For businesses this means that they will not be evaluated as simply better or worse than their competitors, but rather evaluated based on the unique qualities of their business.

2.4 Motivation

People are often content to settle for the most convenient location that fulfills their needs. Because of the difficulty in finding worthwhile places to spend time, the most convenient place is often the most obvious one. Be it a park, grocery store, coffee shop or somewhere else the user is searching for, the places that a user will enjoy the most or fulfills their needs best are not always situated in obvious locations. We want to make it convenient for people to discover places of interest that are well suited for them despite being less obviously located than other places. People will be directed to locations that have attributes that match their interests. More than that, people with similar interests will end up being directed to the same places. What results is many people who have the same values in a comfortable location. This situation is conducive to fostering strong relationships and connections.

Chapter 3

Project Requirements

Given the nature of our project, the requirements that we worked with were mainly defined by what our team wished to accomplish. As such we defined our requirements base on the criteria for an intuitive, aesthetically pleasing mobile application and a useful recommendation system. The requirements we gathered are separated into four categories: critical functional, critical non-functional, recommended, and suggested.

3.1 Critical Functional Requirements

Critical functional requirements define what our application must do and are necessary for our project to be considered complete and successful. Functional requirements define a function of a system. These requirements are critical, meaning they are crucial to the operation of the system.

Critical Functional Requirements:

- Recommend locations to the user
- Learn about the user
- Allow users to clear their own interest tags

These requirements mainly refer to the functionality of our application. The system will learn about the user. It will gather data about the user's interests, which will provide more input for the algorithm. The more data that is put into the algorithm, the more accurate it will be; thus, the more someone uses the application the more accurate the recommendations will be.

3.2 Critical Non-functional Requirements

Critical non-functional requirements define what our application must be like and are necessary for our project to have in order to be user friendly and complete. Non-functional requirements describe what the system should do, rather than what it is supposed to be. As with the above list of requirements, the following list is crucial in the functionality of the system.

Critical Non-functional Requirements:

- User friendly
- Efficient recommendation algorithm
- Helpful
- Secure

The above list of requirements deals with how our system will behave. An important factor when considering if the application is successful or not is usability. We want it to be very user friendly so people will enjoy using it and recommend it to their friends. The more users like the application the more popular it will become. The main goal of our system is to help people, so ideally we want it to be as easy and efficient as possible for our users to get what they're looking for. Finally, above all else is security. It is crucial that our database be secure so that no user data is leaked and used for unintended purposes.

3.3 Recommended Requirements

Recommended requirements are elements our team will be implement after the critical requirements are completed, allowing for continuation of the project. These are requirements that are not necessarily critical for the system to function. These can be thought of as added features that will be implemented time permitting.

Recommended Requirements:

- Be able to poll local gas prices

- Incorporate time into the algorithm
- Incorporate social media accounts

The items on this list would be beneficial to implement because they would benefit users. For instance, it would be useful for them to be able to check gas prices near them, so they do not have to have an extra application just for that one function. Additionally, it would be nice to have time implemented into the algorithm; that is, it will only recommend points that are currently open. Social network integration is another feature that users would enjoy but that does not directly affect functionality.

3.4 Suggested Requirements

Suggested requirements are additional features for us to consider researching and adding after the recommended requirements. These are generally of low priority and will be looked into once the entire system is done and verified as working. The suggested requirements will bring extended functionality to other devices.

Suggested Requirements:

- Bring over to Android devices
- Bring over to browsers
- Adapt the algorithm to support other types of opportunities (e.g. volunteer opportunities)

These requirements will extend the scope of our system. Building an Android application would allow us to reach a broader audience, and in turn generate more users. Allowing users to access a browser based version of our system would increase this audience even further, allowing virtually anyone with a personal computer to use our system. Volunteer opportunities would be another aspect we could implement into the algorithm, allowing users to search for places to volunteer.

Ensuring that our crucial requirements are fulfilled will give us confidence in our system's success. Structuring our requirements gives us the opportunity to focus on core functionality first, then add extra features later. This saved us both time and effort throughout the course of the project.

Chapter 4

Use Cases

In order to ensure user satisfaction, throughout the development process we analyzed use cases to show how the user will interact with our system. We summarized these cases into our Use Case Diagram, Figure 4.1. This diagram shows the major actors and the functions they will perform. The six cases outlined below represent the use cases shown in the Use Case Diagram.

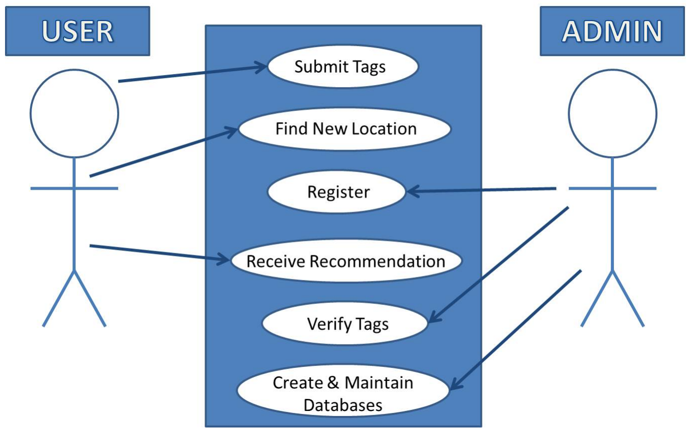


Figure 4.1: Diagram showing the actors' roles in our system.

Each actor plays an important role in the success of our system. As shown in Figure 4.1, the user of our system will be able to submit tags, but only the administrator will be able to verify these tags. We do this to ensure that no explicit content or false representation is expressed through the tag system. We recognize that any time users hold the power to publicly edit content there are going to be some users that add fake/explicit content for the sake of messing with the system and its functions. We put the tag approval process in the hands of the administrator to ensure that malicious users will not negatively affect the usability and functionality of our system. The user can also use our system to find a new location and receive a recommendation of a point of interest that he or she might be interested in. As expected, the users will be the actors who will be using our application, so they will be the ones receiving the recommendations. In addition to verifying tags, the administrator will also be responsible for creating and maintaining databases, and registration. The database will be used to hold all the user data, and it is essential that is be maintained by the administrator. Our server ideally needs to be up and running at all times because any time it is down, our application will not be able to function correctly.

4.1 Use Case 1: Submit Tags

Tag submission is arguably the most important aspect of our system, apart from our algorithm itself. Without tags the algorithm would have nothing to compare, thus making the system unable to make a recommendation.

Actor: User

Goal: Submit a tag that applies to a particular business.

Preconditions: User has application downloaded and is registered and logged in.

Postconditions: Tag successfully linked to business.

Scenario:

1. Open application
2. Login, if not already
3. Select a business to tag
4. Type out tag
5. Save tag

Exceptions:

1. User forgets to save
    (a) Ask the user if he or she would like to save
2. Tag fails to save
    (a) Ask the user to retry
3. User tries to submit blank tag
    (a) Ask the user to type a tag

4.2 Use Case 2: Find Location

To be able to make a recommendation, we must first pinpoint the user's location. Once the location is obtained, we will be able to search a redefined radius around the user to find new points of interest for him or her.

Actor: User

Goal: Obtain user location in order to narrow search radius.

Preconditions: Application is downloaded and user is logged in.

Postconditions: Location used to search for a new point of interest.

Scenario:

1. Open application
2. Login, if not already
3. Allow location services
4. Search for location
5. Find location

Exceptions:

1. User mistypes a location name.
    (a) Return "no results found" or suggestions.
2. No results found.
    (a) Return no results found.

4.3 Use Case 3: Registration

Each user will have have his or her own unique login credentials, allowing us to associate user data with a particular account.

Actor: User

Goal: Successfully register

Preconditions: Have the application installed

Postconditions: Register for the application successfully

Scenario:

1. Open application
2. Click Register
3. Enter all necessary information
4. Press OK to register

Exceptions:

1. User leaves out required information.
   (a) Send back to register screen to fill in forgotten fields.

4.4 Use Case 4: Receive Recommendation

Our application will use a unique algorithm to compare user preferences with tags from locations around him or her in order to return recommendations to the user.

Actor: Administrator

Goal: Verify tags are accurate and appropriate.

Preconditions: Application is installed and user is logged in.

Postconditions: Recommendation is given.

Scenario:

1. Open application.
2. Click button to search for recommendation.
3. Application runs algorithm with user and business data to find appropriate places of interest.
4. Point of interest is returned.

Exceptions:

1. Connection interrupted.
    (a) Prompt to try again.
2. No matches found.
    (a) Prompt user to recommend more places he or she likes.

4.5 Use Case 5: Verify Tags

As previously stated, it is crucial to the performance of our system that the tags remain accurate. The administrator is the sole actor who is able to verify newly added tags as appropriate.

Actor: Administrator

Goal: Verify tags are accurate and appropriate.

Preconditions: Logged into server and able to see submitted tags.

Postconditions: Tag verified and added to database.

Scenario:

1. Administrator logs into server and database.
2. Administrator receives list of user submitted tags to verify.
3. Administrator reviews tags and verifies they meet criteria.
4. Tags are accepted or denied.

Exceptions:

1. Server login failed.
    (a) Check Internet connection and login credentials.

4.6 Use Case 6: Create and Maintain Database

The database that we will use to hold all the user and business data will reside on the server, allowing for the computations to also be performed server-side. This will save both processing power and battery life for the users' smart phones. This database will be created and maintained by the administrator.

Actor: Administrator

Goal: Build and maintain a database for user and business information.

Preconditions: Have a server and appropriate database software.

Postconditions: Create a simple and maintainable database.

Scenario:

1. Login to Server.
2. Create database.
3. Perform appropriate maintenance to keep database working and up to date.

Exceptions:

1. Server login failed.
   (a) Check Internet connection and login credentials.

Chapter 5

System Design

5.1 Technologies Used

For our project the core technologies we used revolve around iOS and OS X server functionality. To work with iOS we used xCode as the development platform to create the mobile application that the users will interact with to receive recommendations. In addition to using the interfaces provided by xCode, we used Objective C to program some of functionality of our application. Objective C is the programming language used by all iOS and OS X devices. The application will communicate with our server to generate the recommendations for the users. Our server is an Apple Mac Mini running OS X server. It will hold our databases and run our algorithm, to make the mobile application less process intensive. For our databases we used the database management system MAMP and used MySQL to store location and user information. For the algorithm we coded it in C++.

5.2 Design Rationale

When this project topic was proposed to our team it had no software or hardware requirements associated with it, so we had the freedom to chooses what platform we wanted to create this for. The two platforms we considered, given the scope of the project, were iOS and Android. Both platforms try and make it easy for programmers to create applications for devices running these operating systems. Programming for Android had the added benefit of not needing to buy a developer's license to install our application on a physical device. However none of us own an Android device nor had any programming for Android, whereas some of us had done some iOS programming before. In addition between the three of us we had three different generations of iPhones and an iPad 2 to test on, and OS X server has built-in functionality to work with iOS applications. Therefore we chose to create an iOS application, despite the fact that this meant that we would need to buy the developer's license and a device running OS X server.

Our System involves the junction of several different technologies. We distinguish the design between the front-end application code, and the back-end server code. Different designs are needed due to their differing functions. The front-end application code is designed to interact with users, and display information relevant to an individual user. The back-end server code must handle large amounts of data, and receive and respond to requests from individual iOS devices.

5.3 Application Software Design

5.3.1 Object-Oriented Programming Language

iOS applications are written in the programming language Objective-C. Objective-C is an object-oriented language. Object-Oriented languages require more overhead causing less efficiency. Because user interaction is driven by user input, this lack of efficiency is not relevant. The time it takes to create and manage the many objects in our application code is still much faster than the time it takes for a user to lower and raise their finger from the screen of their device. Using an object-oriented language allows us to reuse large portions of code and divide segments of code easily based on their functionality.

5.3.2 Model-View-Controller Structure

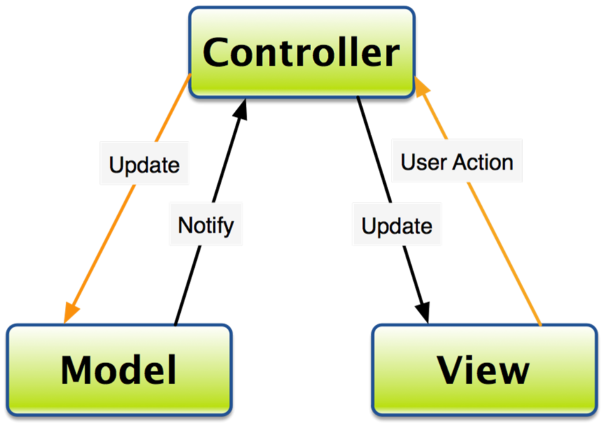
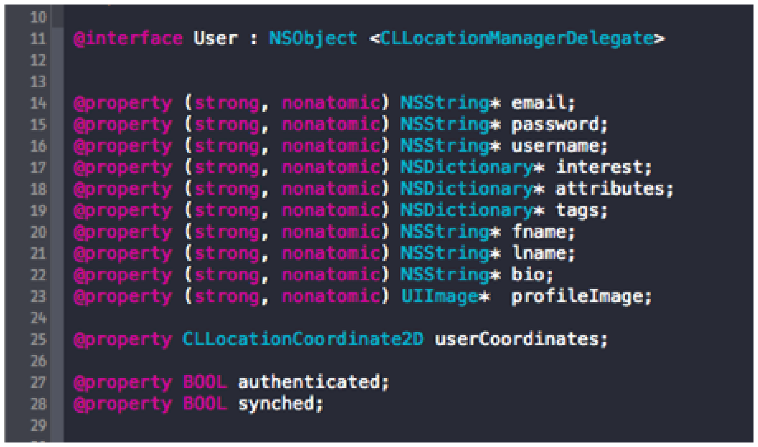


Figure 5.1: High Level description of MVC paradigm.


```objc
@interface User : NSObject <CLLocationManagerDelegate>


@property (strong, nonatomic) NSString* email;
@property (strong, nonatomic) NSString* password;
@property (strong, nonatomic) NSString* username;
@property (strong, nonatomic) NSDictionary* interest;
@property (strong, nonatomic) NSDictionary* attributes;
@property (strong, nonatomic) NSDictionary* tags;
@property (strong, nonatomic) NSString* fname;
@property (strong, nonatomic) NSString* lname;
@property (strong, nonatomic) NSString* bio;
@property (strong, nonatomic) UIImage*  profileImage;

@property CLLocationCoordinate2D userCoordinates;

@property BOOL authenticated;
@property BOOL synched;
```

Figure 5.2: Sample model of a User from the application code.

We employed a Model-View-Controller design paradigm in our application implementation as shown in fig5.1. This method splits our application into 3 distinct parts, a Model, a View and a Controller. In this paradigm the model contains data that the user will be accessing and viewing. The view provides the mechanism by which users can view and change data within the models. The Controller uses the view to display the data from the model, and determines what to do with the user inputs the view informs it of.

We use several models for data. The user is simulated as one, and is populated with information from our database at login as show in fig 5.2. The collection of locations is another, with tags and information about nearby locations being pulled from our database when the list of nearby locations is updated. The controllers that manage each view of our application refer to these models and display the information they contain in a way that helps the user identify which data they would like to view more of.

5.3.3 Navigation Stack

Our application code has been split based on each distinct navigate stack shown. Each view and its controller is grouped with the other views and view controllers that are within their navigation stack. This organization groups relevant code, allowing changes and bugs to be tackled in a more focused scope than if they were grouped with every other view and view controller.

Above we see an example navigation stack shown in fig 5.3. Each navigation stack contains a starting view called the root view. The root view is shown on the left, and contains a broad overview of

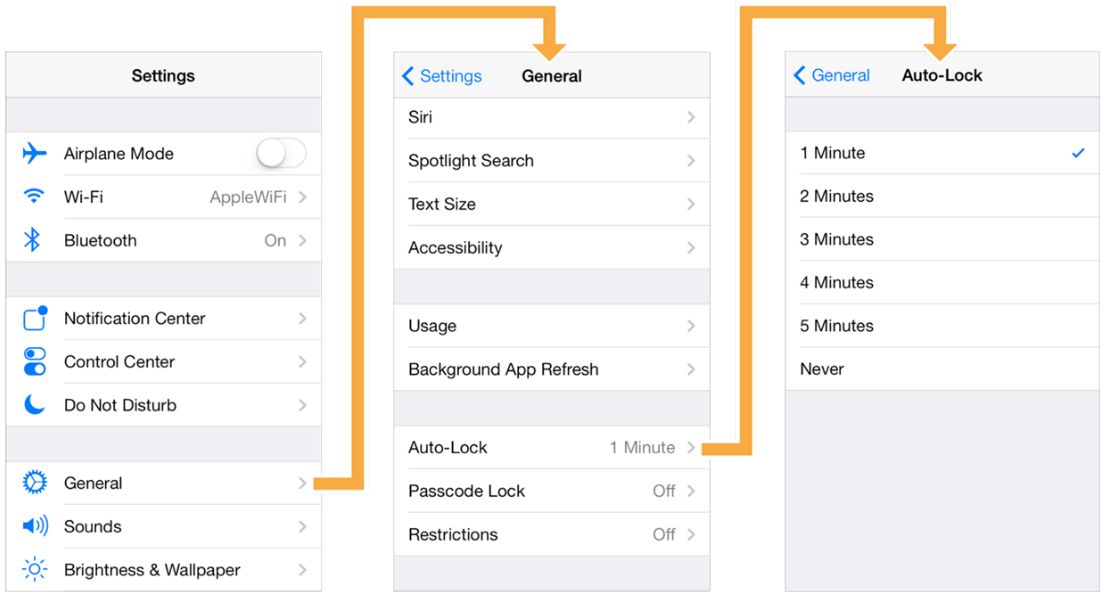


Figure 5.3: A sample navigation stack.

information to be displayed. More specific information is displayed when users select which general category they want. A distinctive back button at the top left will lead back to the root of the navigation stack. This method of organizing content provides users a way to find specific and relevant information within a large amount of data.

5.3.4 Tab Bar Layout

We have four main navigation stacks: Recommendations, Map, Profile, and Settings. Users can select which navigation stack to view using a tab bar that persists between views.

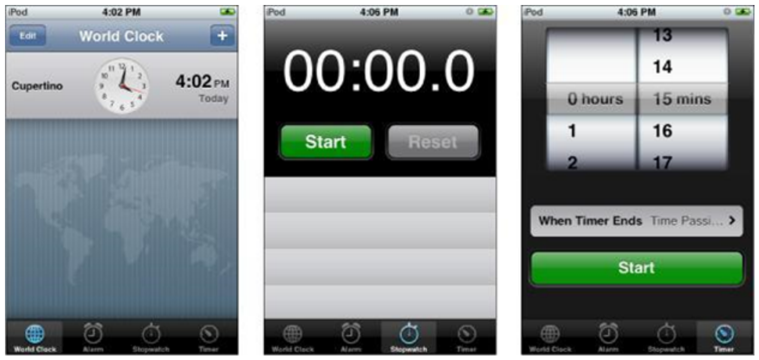


Figure 5.4: A sample of tabs in the clock application.

This method groups related content together and has two effects. First, each navigation stack may be shallower as the tab bar acts as a persistent root view for all for navigation hierarchies. Second, it provides a quick way to jump between different navigation stacks without the need to ascend and descend the information hierarchy.

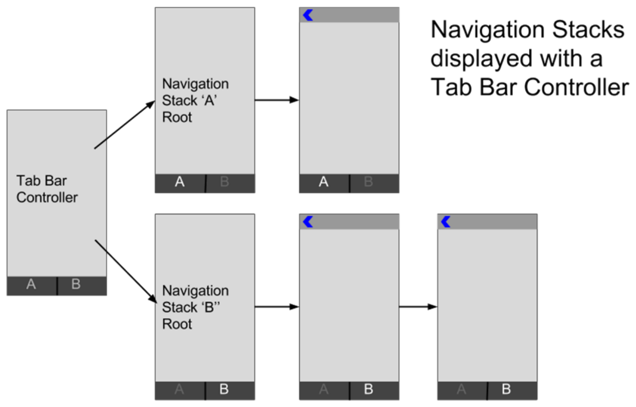

Navigation Stacks displayed with a Tab Bar Controller
Tab Bar Controller
A
B
Navigation Stack 'A' Root
Navigation Stack 'B" Root

Figure 5.5: A sample of tabs within the stack.

5.4 Server Software Design

5.4.1 PHP

PHP is an interpreted language. This means it will be slower than compiled languages. We use PHP for handling network interaction between server side functionality and the application. It was built to work within web pages and can handle network interaction well.

5.4.2 C++

Because C++ is a compiled language it is much quicker than PHP. The PHP code passes large collections of data to the C++ so they can be operated on more efficiently.

5.4.3 MySQL

MySQL is used to manage the databases. It organizes all the data we collect into a relational database. The PHP code sends MySQL messages to the database requesting specific information about a location or user.

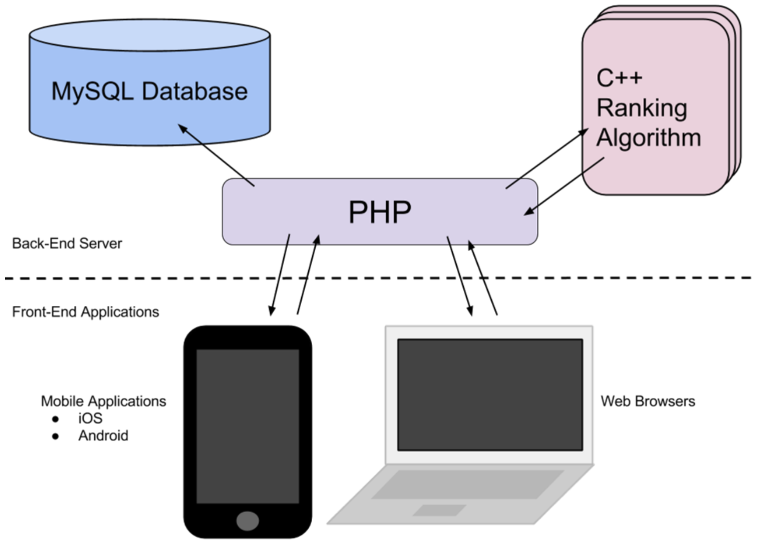


Figure 5.6: How PHP is involved in our system.

5.4.4 System Structure

PHP code acts as the mediator between our database, our data processing algorithm, and the iOS device. When an iOS device messages our server, it communicates with the PHP files. The PHP code examines the request sent from the iOS device and acts accordingly. This action involves querying the database for requested information or adding to the database with user submitted information, and if need be communicating with our sorting algorithm.

The PHP code interacts with the database through MySQL statements. It constructs these statements based on the information passed to it by the iOS application. To prevent SQL injections, we use prepared statements. This technique stops unauthorized queries from being sent to the database and compromising the security of our system.

With the PHP as the gateway to our server side functionality, the system is nicely segmented between front and back end. Because of this, the back-end code can be reused between various front ends, such as a web page or android application.

5.4.5 Database Structure

We use a relational database for our application. A relational database is better than a hierarchical database for our system because it allows us to add information to our database easily. A hierarchical database is not able to easily accept new fields after it has been established. Some structure is lost from a hierarchical database, but relational databases offer the flexibility to add new values simply and can be altered as the system develops. The database is divided into two tables. One is for user information, and the other contains location information. These tables contain information about each location and user as well as the tags associated with them. Having two tables allows queries to be more focused.

5.5 System Diagrams

To aid the reader in visualizing how we plan to layout our system we have prepared the following high-level system diagram. In figure 5.7, an end-product user of our system would interact with our server through our application that the user installed on his or her iOS device. The server will receive the user's login information, location, and what the user is looking for. Based on this information, the server will use our recommendation algorithm to generate a list of recommended points of interest tailored to the needs and interests of the user. The database storing the user information will be updated based on what the user chooses. Another thing that user can do from within the application is to clear our database of all their interests.

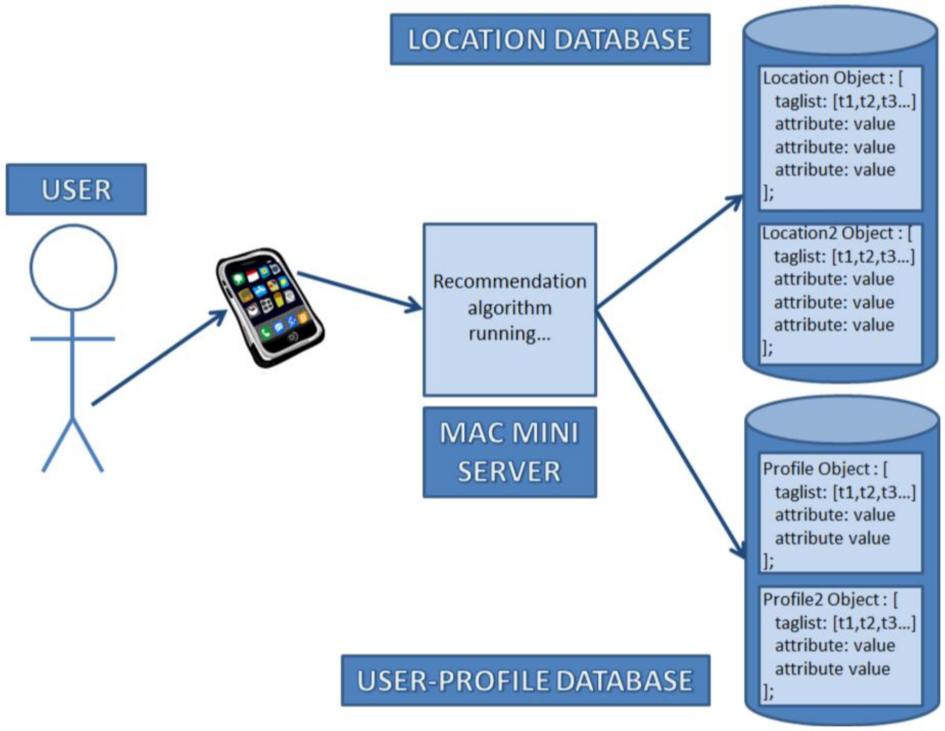


Figure 5.7: A high-level system diagram of our project.

Chapter 6

Algorithm

6.1 How does it work?

The core idea behind design of the algorithm was to create a formula that could generate unique lists of recommendations for each user. Before we could start our own algorithm, we needed to take a look at existing recommendation algorithms and determine the key factors as to why they work. These factors include: how to deal with new users and how to generate recommendations for similar minded users. The issue with most methods is that they generate similar recommendation lists for users who are categorized as the same or similar. While these algorithms are able to deal with large amounts of data quickly, they sacrifice each user's unique interests in favor of the majority's.

The way we went about solving this issue is through the use of tags. Tags are our way of linking interests to users and attributes to locations. Current tags associated with each user are manually by the user inputed upon account creation and over time as the user uses the application. As of right now, tags for each location have to be manually added by system administrators and are later augmented by user tagging. Within the current framework of the system, every time a user adds a new, unique tag it is added to our database for both users and locations to use. To implement the tags in our algorithm we assign weights to each one based on the strength of association between the tag and the user or location. For users, this means that the weighting of the tags associated with them represents how much they like or dislike a certain attribute. Whereas for locations, the weighting represents how prevalent the attribute is for that location as determined by the users who frequent that location.

The math behind the algorithm is as follows, the rank of each location is the summation of the

$$Rank = [\sum_{0}^{N}(UsrTagWt_N * LocTagWt_N)] + DistMod$$

Figure 6.1: A formulaic representation of our algorithm.

product of the tag pairs between the user and each location plus the distance modifier. The distance modifier is simply a boost we give to each ranking based on how close the location is to the user. This algorithm works as intended because of the weightings of each tag. If a user strongly dislikes a certain attribute, the rank of that location will be different than for a user who strongly likes the same attribute. Also if a user strongly likes an attribute that is not popular amongst the majority the algorithm will recommend locations with that attribute even if there is not a strong association with that tag. This will allow users to find new locations that they may enjoy.

6.2 How is it different

As mentioned before, most ranking/recommendation algorithms only recommend the most popular locations to users of a certain category. We discarded the idea of grouping users into categories and treat each user individually. We let the users naturally form their own groups/communities at each location by only recommending people who are genuinely similar. Also our algorithm will recommend locations that will typically not appear on other recommendation services, such as small businesses. In the future, we would like to add social integration that will allow people to create their own impromptu group events and our algorithm will be able to recommend nearby people to it.

6.3 Challenges

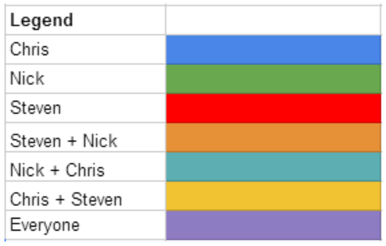
There were a couple challenges we faced while designing the algorithm. One such challenge is how to create recommendations to users who do not provide any interest data. Most recommendation services would simply generate a list of the most popular locations in the area, however this would defeat the point of our algorithm. As such the algorithm will recommend locations based on how close they are to the user using the distance modifier. Another issue we had to deal with is whether or not to limit the tags the users can add. For now we chose to allow users to add any tag they want; however, later on we will switch over to providing a large pool of tags for users to choose from that will allow them some degree of uniqueness but will prevent redundant or inappropriate tags.

Chapter 7

Development Timeline

After compiling a list of requirements, we determined a schedule of tasks for the project. The Gantt charts below lists out the tasks necessary to complete the project. A Gantt chart is a chart that visualizes the project timeline by breaking it into phases and displaying start and end dates for each task.

7.1 Gantt Charts



Legend	
Chris	🟦
Nick	🟩
Steven	🟥
Steven + Nick	🟧
Nick + Chris	🟦
Chris + Steven	🟨
Everyone	🟪

Figure 7.1: The Gantt chart

We divided our work up by having one person focus on the server side components and another working on the application side. A third member had a hand in both components to ensure that the two halves come together cleanly.

The member working on the application side focused on the user interface, as well as the integrity of the application. He was tasked with building a system capable of gathering what information is needed from the user. In later stages he was tasked with creating the graphics and making the application visually appealing.

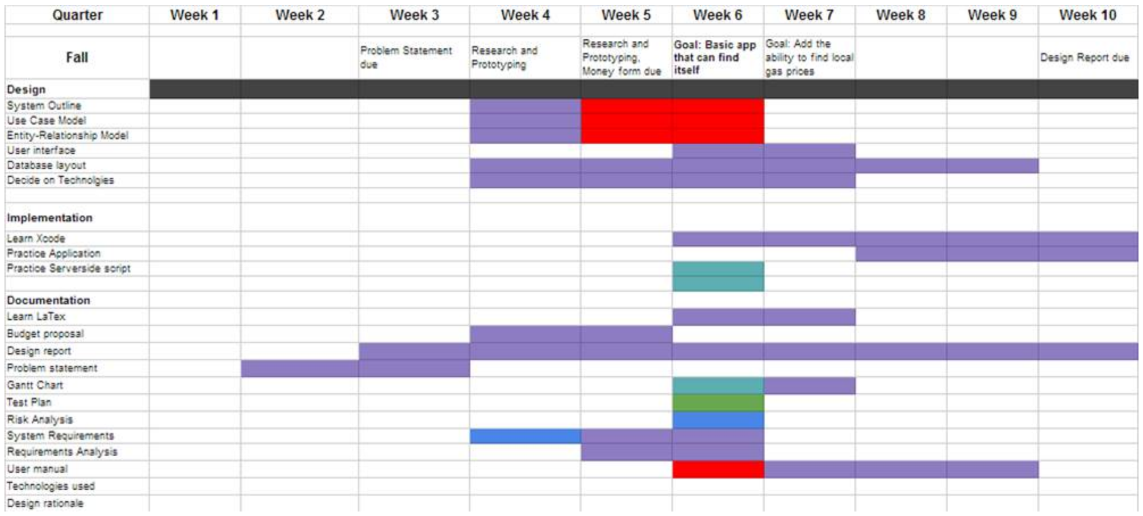


Figure 7.2: Team members' tasks throughout Fall Quarter.

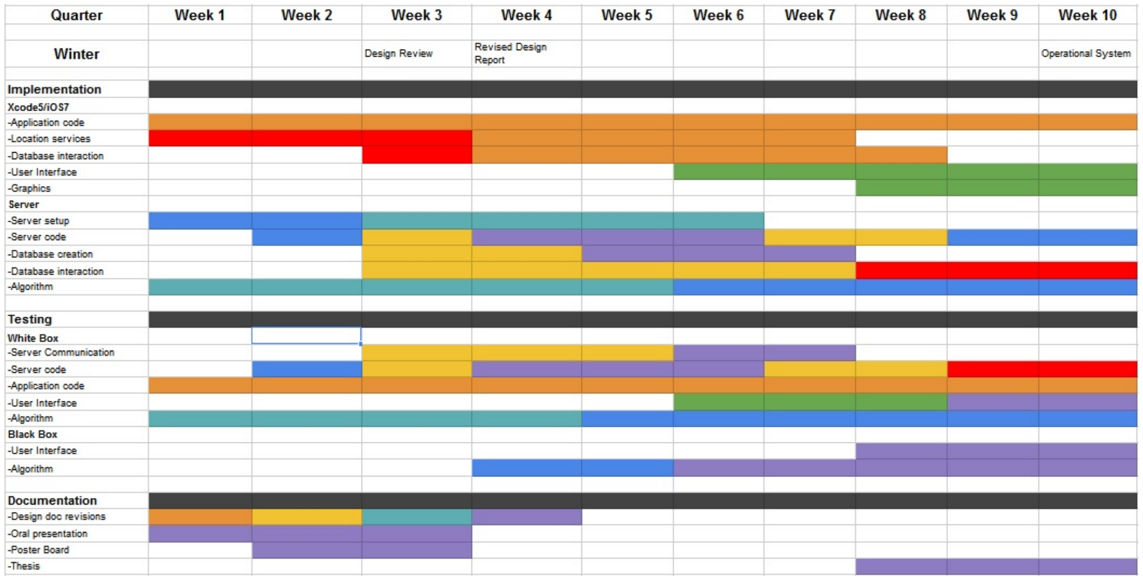
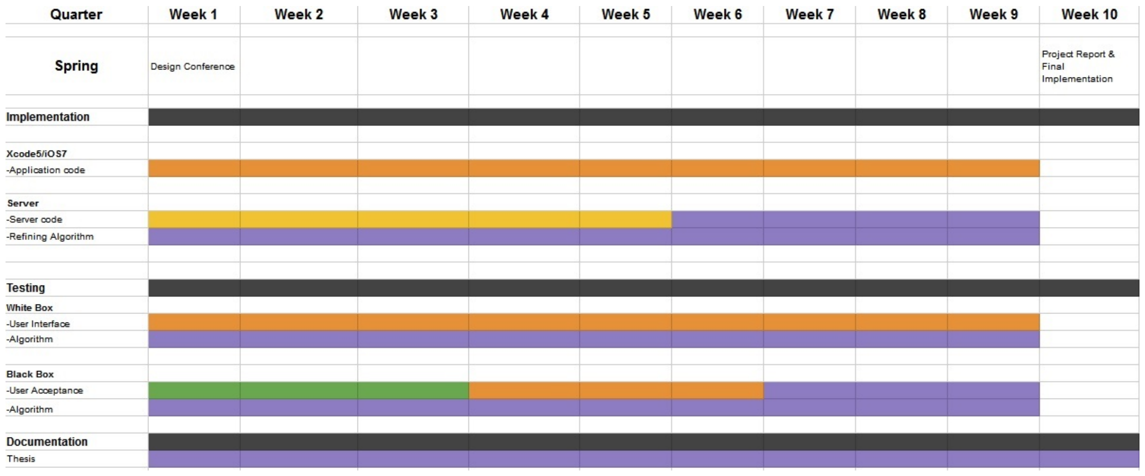


Figure 7.3: Team members' tasks throughout Winter Quarter.

The member working with the server spent the most time with our algorithm. It was his job to make sure the algorithm functions smoothly on our server. He also did the scripting necessary for the server to function as it should.

The third member assisted the others in creating a communication system that sends and receives the appropriate information. He made sure that throughout the process both the application and



Quarter	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10
Spring	Design Conference									Project Report & Final Implementation
Implementation	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛
Xcode5/iOS7										
-Application code	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	
Server										
-Server code	🟨	🟨	🟨	🟨	🟨	🟪	🟪	🟪	🟪	
-Refining Algorithm	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	
Testing	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛
White Box										
-User Interface	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	
-Algorithm	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	
Black Box										
-User Acceptance	🟩	🟩	🟩	🟧	🟧	🟧	🟪	🟪	🟪	
-Algorithm	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	
Documentation	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛
Thesis	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪

Figure 7.4: Team members' tasks throughout Spring Quarter.

the server were developed in parallel and were able to easily and quickly communicate. This means he had a hand in both the application coding and the server coding and scripting.

Chapter 8

Project Risks

8.1 Overview

We assessed the potential risks we could encounter over the course of our project and have summarized them into the following table. This risk analysis table describes the possible risks, as well as the corresponding consequences, likelihood of each happening, and the impact on our project. The table also includes the mitigation strategies we employed to avoid each risk.

We have organized these risks in descending order based on the possible impact to the project. Risks at the top of the table have much higher impact of consequences to the project than the ones lower in the table. The impact of each risk is calculated by multiplying its probability of happening and its associated severity(scale of 1-10, 10 being most severe).

8.2 Risk Table

Impact = Probability * Severity

Risk	Consequence	Probability	Severity	Impact	Mitigation Strategy
Unfamiliar Technology	Unable to implement intended functionality	1.00	4	4.00	Independent research. Study and prototype functionality before committing to an idea.
Insufficient funding	Unable to finish project	0.50	5	2.50	Trying to find multiple sources of funding.
Time	Unfinished Project	0.50	5	2.50	Prioritize functionality and cut unnecessary requirements
Code Loss	Having to rewrite portions of the project	0.65	3	1.95	Version control and regular backups
Database corruption	Having to recreate the databases	0.55	3	1.65	Regular backups
User information leaked	Users will need to change their passwords and loss of user trust	0.001	10	0.01	Store hashes of user passwords instead of plain text.

8.3 Analysis

The greatest risk we face is unfamiliar technology. If we must commit too much time to basic challenges simply because they are foreign to us it will detract from the features and functionality of our application. The best thing we can do is put as much time as possible into learning the technologies we are unfamiliar with. Becoming familiar with the basics of these technologies is vital, and the earlier we begin to become acquainted with them the easier it will be for us to incorporate them. We can anticipate which technology each team member should invest time into by using our Gantt chart to anticipate which technologies they will be working with the most.

Chapter 9

Test Plan

Our system was tested throughout development. During the design phase we went through several prototypes and ideas.

9.1 Design Phase

Early on in the design phase we worked with simple mockups. This allowed us to quickly move toward a desirable design. We tested each mockup with consideration to extendibility and our requirements. The ideal design would fulfill our highest priority requirements, and also allow us to easily implement features as well. Usability was a high priority for us. Creating an experience that people would be amicable towards was necessary for our application. If our application is difficult to use and no one uses it, then our hard work is moot.

We tested these mockups rapidly, often moving through several in a couple hours. When we came to one that seemed concrete, we worked through use cases to expose its strengths and weaknesses. Moving onto the next design, we attempted to mitigate the weaknesses, and work with the strengths we had developed. Maintaining extendibility throughout these designs was vital. Any mockups that relied on technologies unnecessary for our application to function were discarded. In the later part of the design phase, we were close to a design we were happy with. We began to move our design to a more concrete implementation to better test its nuances. This involved moving the user interface to Xcode, and creating code samples to test the recommendation algorithm we planned to use. We simulated inputs to test that our algorithm could fulfill its part of the requirements. We also moved through use cases in Xcode mockups to test that our initial mockups would hold up in practice.

We wanted to test some of the mockups we had for server side functionally, such as gathering gas prices from gas stations local to the user. The site Gas Buddy was a promising source of such information as its site provided this functionally. Initially we believed that mining this site for information we not be too difficult; however, upon closer inspection this proved not to be true. The site created an event validation string that it sent to their servers in addition to the information provided by the user. This prevented us from sending our own requests to its servers as we were unable to recreate its validation function.

9.2 Implementation Phase

The application was tested throughout development, both in the simulator and on an iOS device. By testing on the simulator, we were able to provide coordinates for any location. This helped us make sure the basic functionality of our application was working, but there were more minute details that became apparent when we began field tests.

9.2.1 Unit Testing

Testing on a device revealed several hidden issues, one of which was caused by the device moving from place to place. We had to tune the application to provide up to date information, but initially this interfered with the display of our maps. This issue was not apparent in the simulator.

9.2.2 Acceptance Testing

To ensure unbiased testing of our application we performed some user testing by allowing individuals not associated with our project to interact with our system. To this we gave a phone with our application pre-downloaded to a group of individuals and watched to see how they would use the application. We wanted to see if they would use it as we expected. By this time we already worked out the majority of the software bugs and functionality issues. In this stage of testing we were testing if our system was as intuitive and easy to use as we thought it would be.

9.3 Test Results

The device testing made clear several large errors that were fairly simple, but the fact that tests had to be conducted away from a workstation impeded the process. Thankfully the larger errors were few enough that we were able to perform thorough field-testing before too long, including tagging locations, and receiving a location list.

Testing the algorithm proved to be more challenging than anticipated. We were unable to test the server-algorithm interaction as the libraries that we were depending on to use mySQL within C++ were not working for us. We tried switching over to a PHP wrapper for the algorithm instead; however, in the process of trying the get the additional libraries working the links for the core libraries broke. As such we were only able to test the functionality of the algorithm by itself. To test the algorithm we passed in sample requests that we expect the algorithm would receive, and check to see if the algorithm produced the expected results. If it did then it meant the math and logic were correct; otherwise, it meant there was an issue in the code. The typical issues were mainly spelling mistakes and the occasional logic error.

In the acceptance testing we were happy to receive an overall positive reaction to our project. The users who tested the application were very enthusiastic about the idea and gave positive feedback about the design and intuitiveness of the system itself. This testing gave us confidence in the design of the application itself, allowing us to focus more on minute details and fine tuning of the ascetics and functionality of the application in the weeks to follow.

Chapter 10

Lessons Learned

Along the course of our project we ran into many unforeseen problems, encountered unforeseen obstacles and overcome several challenges. From these issues and hurdles we learned many lessons ranging from team dynamic issues, how to work with outside personnel, and how to deal with the learning curve of new technologies. Some of the main challenges we faced include lack of communication, poor division of work, implementing too many new technologies.

10.1 Importance of Communication

Communication is possibly the most important factor in any team based project. It can be essential to be in contact with each other for various reasons, and if for whatever reason this contact is broken it can create a division in the team and cause problems. We experienced some issues in this department which sometimes affected our ability to meet up in a timely fashion or meet certain deadlines. Because of this it was also hard at times to keep track of what other members were working on. We learned from experience that communication is essential in order to have a successful project and working dynamic.

10.2 Clear Division of Work

By equally distributing the workload, more progress is able to be achieved simultaneously and will prevent internal conflict amongst group members. If one person is doing the majority of the work, progress will be slowed down if the remaining works is dependant on the functionality that the other person is working on. There is only so much one can do without the core aspects that he or she is supposed to be building upon. Also when the group member doing the majority of work becomes sick or is rendered unable to continue working, the project comes to a near complete stall. If the work was evenly distributed, then the other group member would either be able to work around the issue or be able to take other until the incapacitated member is able to continue working. When other group members fail to deliver their parts on time it delays the project and causes unrest amongst the other group members. If one member repeated fails to do their part, then they are not contributing to the project and are taking advantage of the other group members.

10.3 Challenges of New Technology

We took on the substantial task of having to learn many new technologies in order to accomplish our project. We did this because we felt these were the best technologies suited to accomplish the functionality we wanted our system to have. While these technologies made it easier to implement our project once we learned them, the actual act of learning took longer than we anticipated. Each of us had at least one new form of technology we learn in order to complete our section of the product. Because we had to learn something entirely new to us, we were set back at times as much as a few weeks. With multiple technologies being implemented, this time quickly added up and caused us to have to prioritize our requirements and edit our project deadlines. Although new implementing new technologies can be expected in most projects, we made the mistake of not account for adequate time to overcome this challenge. From this we have taken away that it is important to account for this learning curve time when planning out the development timeline of a project.

Chapter 11

Ethical Issues

11.1 Security

Ethically, our project holds very sound: we have created this application for the betterment of our society and chose not to allow any outside factors to influence our recommendations. With these two main thoughts in mind we set out to make our application. In addition to the overarching goal to connect people with locations they will enjoy, our project also has an added benefit that within these places people are able to find like-minded people they will be able to easily connect with. Another ethical factor we dealt with was the secure storage of user specific data. We are representing companies on our application, and while we want the companies not to be able to influence their own profiles too much, we also must ensure that these profiles arent negatively misrepresenting businesses and causing them hardship or loss of business. Keeping these key factors in mind throughout the term of our project, we were able to build an application which successfully conformed to our code of ethics.

11.2 Social Issues

Our application works because we store and use information provided by the users, as such we have a set of responsibilities towards our users that we must uphold. By storing the users data in our database, we must maintain a certain level of security and protect the users data from unauthorized third parties from accessing that data. We must prevent all intrusions to the best of our abilities by using hashes to store the users login credentials, and should a breach of security occur we must quickly resolve the issue, notify our users, and research how to prevent a second occurrence. Our application sends relevant user information to third parties to learn about locations nearby the user and display our recommendations on a map, along with directions.

11.3 Economic Issues

Our application describes locations through the use of tags, as such we have a responsibility to each location that we recommend. If we allow negative tags to be associated with a location; then we would be harming that locations reputation and cause that location to lose customers. Also if we fail to check if the tags associated with a location truly represent the traits of that location, then we will hurt the reputation of the location and of ourselves. Users will go to that location expecting a certain kind of experience, only to receive a completely different one. Users will think that either the location was horrible or that our application is unreliable, neither of which we want to happen. Some locations may try to pay us to change the weightings of their tags or add tags to their list in order attract a wider range of users, however this would be lying to our users and will not be helping them find a location that they will genuinely enjoy.

Chapter 12

Usability

12.1 Usability Goals

Creating a usable design will encourage users to use the application as often as possible. Our application relies on crowdsourcing to identify the attributes of locations. It is vital users face no trouble contributing to our database. To achieve the level of user input needed, there are two goals that must be met.

1. Ease of use For both new and experienced users, the interface should not offer confusion. Possible actions should be made clear as well as the results of such actions. The user should be able to receive location recommendations, investigate the details about a location, and add an attribute to a location without trouble. The user should be able to view their interests and add to their interests just as easily.

2. Speed of use The application must take as little time to use as possible. Users should be able to receive recommendations immediately and be able to see what they would enjoy about a location that was recommended. Users should also be able to tag a location with an attribute in as little time as possible.

To ensure an effective user interface, we have taken into account six primary design principles:

Visibility - The current state of the application is clear and possible actions are obvious.

Feedback - Inform the user what the system is doing or has just done, e.g. a loading bar.

Affordance - The layout of the U.I. help indicate the correct way the to use application.

Mapping - Action buttons and controls have distinct purposes.

Constraints - The user cannot make invalid actions.

Consistency - The interface is similar to others and uses standards when applicable.

These principles are to help identify how comfortably and effectively a user will be able to use our application.

12.2 Usability Design

We began focusing on these principles in our tagging use case. It is vital we have user inputs via tags so that we can build accurate attribute models of places. To achieve this accuracy we need users to tag attributes as much as possible. We make it clear when a location is available to be tagged, and within two touches the user can be typing their tag into our database. We have clearly labeled buttons, and a short sentence that explains the purpose of tags.

Receiving recommendations is the service our application provides to users and it is made as simple a process as possible. Upon logging in the user is shown the list of nearby locations that we have found match their interests. Relevant tags are presented in the table to help the user find a location they are interested in at that moment without having to look at the details of each location in the table. Some constraints we provide prevent users from tagging locations they are not present at. If a user tries to tag an invalid location, we prompt them with an alert that informs them that they must be present at a location to tag it, to make it clear how the system works.

Xcode offers tools that help give all iOS applications a consistent feel. The blocks they provide to build applications with are effective at what they do, and shared between all applications. This will help users who have never used our application before, understand how the functionality works. As well as consistency, the iOS objects that we use to design the user interface offer great examples of mapping and affordance that carry into our applications user interface as we implement them.

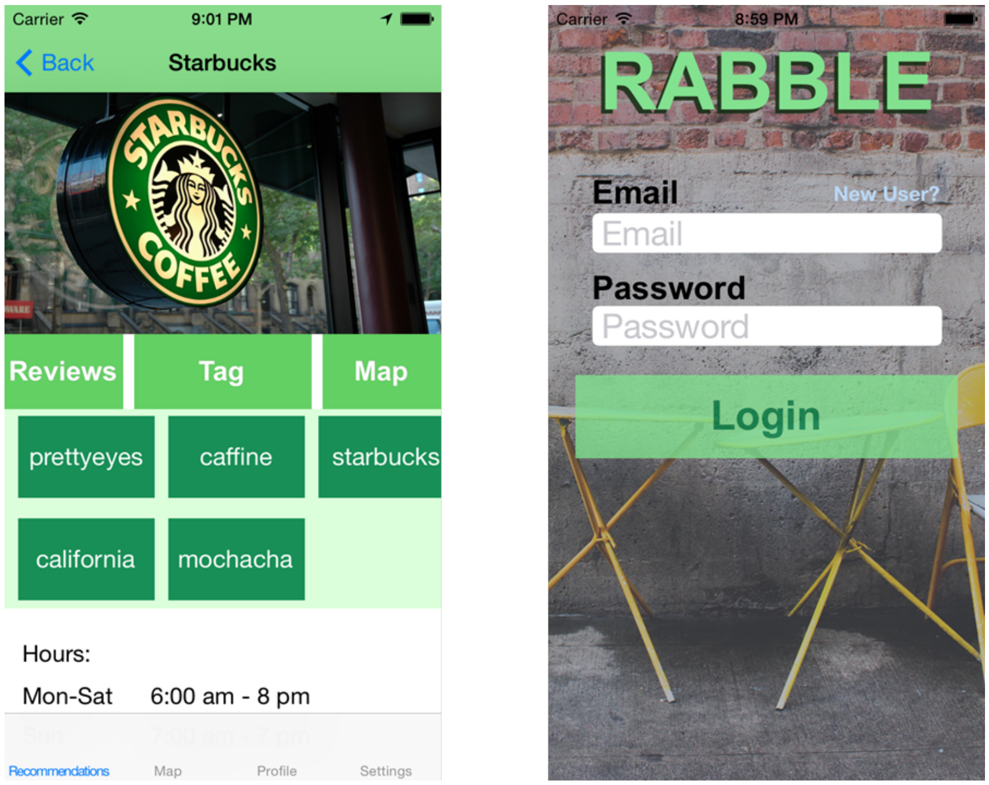


Figure 12.1: Screen shots of a place page and our login page within the application.

12.3 Design Ascetics

We began focusing on these principles in our tagging use case. It is vital we have user inputs via tags so that we can build accurate attribute models of places. To achieve this accuracy we need users to tag attributes as much as possible. We make it clear when a location is available to be tagged, and within two touches the user can be typing their tag into our database. We have clearly labeled buttons, and a short sentence that explains the purpose of tags.

The aesthetics of our application may be approached in two distinct ways. From a user perspective, the aesthetics of the system lie in the application and the user interface. From a programmers perspective the aesthetics lie in the code of our project.

12.4 Graphical Aesthetics

An attractive user interface will inspire trust in users, and encourage new users to continue using the application. The graphical libraries in Xcode gave us a strong starting point. We were able to have our application appear consistent with other iOS applications with little effort.

The application has a green color scheme. Items of interest in the user interface are given a slightly bluer hue, and are much darker. Having a lighter green for the tab bars and navigation bars makes darker green objects stand out, while being maintaining a consistent color scheme.

Pictures are used to give the application more vibrancy and draw in the user. The pictures used are photographs of real places, matching the spirit of the application. The photographs were chosen to be generic but look appealing, a place that could be found anywhere someone may look.

12.5 Code Aesthetics

The aesthetics of the system code is closely linked to the rationale behind its design. Our code is divided neatly into modules that are used and reused throughout the application. The modules have been designed to be functional independent of other modules. This decoupling allows for our code to be flexible and extendable. Because modules are independent of each other, when we improve or change on module we need not seek out those modules that depended on it to fix them. Changes are easier to make and the code is easier to understand as all related data and functionality is grouped together.

12.6 Design Simplicity and Functionality

Recommendations are even simpler. Upon logging in the user is shown the list of nearby locations that we have found match their interests. Relevant tags are presented in the table to help the user find a location they are interested in at that moment without having to look at the details of each location in the table.

Some constraints we provide prevent users from tagging locations they are not present at. If a user tries to tag an invalid location, we prompt them with an alert that informs them that they must be present at a location to tag it, to make it clear how the system works.

12.7 Intuitive Design

Xcode has given us an incredible ability to make our application consistent with other applications available on iOS. The blocks they provide to build applications with are effective at what they do, and shared between all applications. This will help users who have never used our application before, understand how the functionality works. As well as consistency, the iOS objects that we use to design the user interface offer great examples of mapping and affordance that carry into our applications user interface as we implement them.

Chapter 13

User Manual

The user manual section strives to create a concise set of steps that will allow the user to download and use our application successfully. Through this manual we seek to convey the simplest set of instructions for navigating and using our application.

13.1 Obtaining The Application

Currently, our application is not released on the Apple Application Store, but once polished we expect the application to be available to the public. This manual will assume the fact that the application is available on the App Store.

In order to download our application the user just has to follow a few simple steps:

1. Search for our app on the app store and navigate to the Rabble page.
2. Click the download button.
3. Enter your apple ID password when prompted.
4. Wait for your application to download to your device

13.2 Using The Application

Once the application is successfully downloaded to the phone you are ready to enjoy and find recommendations for places! Be sure to create an account when the application is initially loaded and populate your profile with tags that apply to you.

In order to add tags to your profile:

1. Navigate to your profile page
2. Type in a tag and submit it to add it to your profile

Now that some tags are associated with your profile, you are ready to find some places to enjoy.

1. Navigate back to the home screen and pull down to refresh your list
2. The algorithm will run and return to you your custom list of places ranked in order of applicability to your personal interests.

By following these simple steps you will now be able to confidently use our app and be able to find places of interest you will enjoy.