Project Design Document - QuBar App

Stakeholders: Orlando Beavers, Andrew Boudousquie, Thomas Clarke, Dave Murray

Document Modification History

Version	Date	Author	Description
0.1	2/28/2017	Team Q	Initial Draft
0.2	2/28/2017	Orlando	Added Introduction
0.3	2/28/2017	Andrew	Added Functional Requirements
0.4	2/28/2017	Dave	Added Non-Functional Requirements
0.5	3/1/2017	Thomas	Added User Case Scenarios
0.6	3/1/2017	Orlando	Added definitions
0.7	3/1/2017	Andrew	Added Class Diagram
0.8	3/2/2017	Dave	Added Activity Diagrams
0.9	3/2/2017	Thomas	Figure Notes Appendix
0.10	3/2/2017	Dave	Prioritized features
1.0	3/2/2017	Team Q	Final Draft
1.1	3/29/2017	Thomas	Moved Figure Notes
1.2	4/4/2017	Andrew	Moved Figure Notes/Added Descriptions to Seq Diagrams

Table of Contents

Introduction

Service Need

This solution is needed due to the aspects that lines can be long for specific bars or restaurants. Also you would not want to be blindsided by admission fees or you arrive to the location of your desire just to find out that the establishment is closed.

Project Purpose & Scope

The purpose of this app is to provide more accuracy with ease to specific bars or restaurants that are in the surrounding area in the categories of admission fees, popularity, wait times, gender ratio, open/close times, and possible specials that are being given by each establishment.

Definitions

- Map - a diagrammatic representation of an area of land or sea showing physical features, cities, roads, etc.
- Push Notifications- a message that pops up on a mobile device. App publishers can send them at any time; users don't have to be in the app or using their devices to receive them.
- API(Application Program Interface) - a set of routines, protocols, and tools for building software applications that specifies how software components should interact.
- GPS(Global Positioning System) - a radio navigation system that allows land, sea, and airborne users to determine their exact location, velocity, and time 24 hours a day, in all weather conditions, anywhere in the world.
- UML Diagram(Unified Modeling Language) - a standardized modeling language enabling developers to specify, visualize, construct and document artifacts of a software system.
- Class Diagram - a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.
- Activity Diagram - a graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency.

Project Description

Product Perspective

The app is directed to the viewpoint of the outgoers that like to go out on days where bars or restaurants are giving specials on that day, or for those that like to sit down and enjoy their meal with other associates but don’t want to have to wait very long to be able to sit. It also gives the

perspective of those that want to go and have a drink or go to a night event that may have a big crowd and admission fee.

Product Features

Integrated Google Map that shows surrounding bars and restaurant in the area, customer accounts as well as business accounts, drop down menu features, estimated wait times, admission fees, accurate specials, push notifications, and demographics for business use.

Required Features

User Accounts

A user can set up either a customer or business account. A customer account will request information such as Gender, Age Range, etc to help with demographic data. A business account will need to be a verified business owner/manager, currently a solution to verifying identity hasn't been achieved. A business owner may wish to update their establishment's information (i.e. Hours), also a owner to business relationship is 1:N.

Location

Once a user arrives at an establishment, verified by location data, they are able to check in and update their wait time and any other information useful to fellow patrons.

Map Feature

User will be able to filter bars and/or restaurants near them. If both are selected, unique icons will decipher between bar or restaurant. If filtered to just bars or restaurants, the pins will be color coded by wait time (short, moderate, long) to allow the user to quickly browse.

Possible Features

Owner Verification

Need a way to prove an owner is who they claim to be. Google sends postcards with verification codes to the business address, hopefully we find a more automated solution.

Push Notifications

One benefit to a business account is the ability to get in contact with users who have been to your establishment. A business user would have the ability to push notifications to users who have "Checked In" to their business in the past. Potentially here they could target customers who haven't been in for a while.

Customer Analytics

Demographics, historical wait times, all of this can help a business owner make sound choices for their business.

Reviews

Allow users to write reviews and push to a popular review site, but only show relatively recent reviews. The whole goal of the application is to show up to date wait times, admission, etc. Why taint that with five year old reviews?

Requirements

Functional Requirements

Primary

1. Users can see wait times by a filtered google map.
2. Users can check-in to a restaurant or bar.
3. Users can update wait times and cover if applicable.
4. Users check out by leaving the area of the restaurant or bar.
5. Owners can push notifications, change their entity details, and see analytics about their entity.

Secondary

1. Users can create a customer or business profile, or sign in as a guest for customer.
2. Users can opt in to receive updates from entities of their choice.

Non-Functional Requirements

Primary

1. GPS Services are required to access location information from the user's phone accessibility.
2. Google Maps API is required to locate bars and restaurants near the user's location or within a given city.

Secondary

1. Documentation, Design, and Implementation should be executed with the maintainability of the project going forward.
2. Design should maintain code modularity in order to ensure portability and expandability.
3. Design should be an enjoyable and beneficial for both customers and businesses.

UML Diagrams

Use Case Scenarios

Use Case (USER)	Description
View Map	A new USER does not need to register on the application in order to view the map. Actor(s): USER Pre-condition: none 1) The USER selects the VIEW MAP option on the main screen. 2) The application retrieves and displays a Google Maps landscape with venues. 3) The USER may search venues by name or filter and sort by criterion. 4) The USER may also select back to return to the main menu. Post Condition: USER may view information about selected venue
View Events/Wait Time/Cost of Admission	A USER may view additional information about selected venues. Actor(s): USER Pre-condition: USER must have selected at least one venue from the choices on the Google Maps landscape. 1) The USER selects “View Additional Information” 2) The system retrieves and displays the respective specials/events, average wait time, and cost of admission for each venue. 3) The USER may refine this list or select a return button to return to the Google Maps landscape screen. Post Condition: The list is refined. Alternatively, USER may return to Google Maps screen or to main menu.
Register	A new USER must register in order to check in at any venue or redeem special offers. Actor(s): USER Pre-condition: An unregistered USER. 1) The USER will select the REGISTER button on the main menu of the application screen. 2) The system will display the registration form. 3) The USER will input all of the required information. 4) The USER will select a complete registration button. 5) The system ensures that all required information was provided. a. If YES, system will update USER record in USER table in database. b. If NO, system prompts USER to fill out required. 6) The system will log the USER in and return the USER to the main screen of the application. Post Condition: The new USER has registered, the USER database is updated.

Use Case (USER)	Description
Log In	A USER must log in before checking into a venue. Actor(s): USER Pre-condition: A registered user. 1) The USER selects the log in option on the screen. 2) The USER inputs his/her uID and password. 3) The USER selects ENTER. 4) The system compares and validates USER input against USER database. 5) USER is authorized; system prompts user with an option to stay logged in indefinitely and returns him/her to main screen of application. a. NOT authorized, system displays error message. Post Condition: The USER has been authorized and their information is sent to a database for venue usage.
Input Wait Time	A USER must input the wait time for aggregation while checking into a venue. Actor(s): USER Pre-condition: A registered user. 1) The USER inputs the time that they have waited before checking in or expect to wait before they receive service. 2) The system adds this time to the venue database for averaging with other user/owner input times; returns USER to main screen of application. Post Condition: The USER input has been added to the database for venues in order to receive an average wait time.
Check In	A USER may choose to Check In at a venue from the main screen of the application or the Google Maps landscape. Actor(s): USER Pre-Condition: User have logged in. 1) The USER selects CHECK IN via a button in the application. 2) The system prompts the USER to input the estimated wait time. 3) The USER can then select ENTER to send their input and user information to the venue database. a. The USER can choose to return to the main screen of the application via a provided button. Post Condition: The USER input and user information will be added to the venue information.

Use Case (OWNER)	Description
Register	A new OWNER must register in order to check in at any venue or redeem special offers. Actor(s): OWNER Pre-condition: An unregistered OWNER. 1) The USER will select the REGISTER button on the OWNER's menu of the application screen. 2) The system will display the registration form. 3) The OWNER will input all of the required information. 4) The OWNER will select a complete registration button. 5) The system ensures that all required information was provided. a. If YES, system will update OWNER record in OWNER and VENUE table in database. b. If NO, system prompts OWNER to fill out required. 6) The system will log the OWNER in and return the OWNER to the owner's screen of the application. Post Condition: The new OWNER has registered, the venue and its geolocation have been added and validated, the OWNER and venue databases are updated.
Log In	An OWNER must log in before checking into a venue. Actor(s): OWNER Pre-condition: A registered user. 1) The OWNER selects the log in option on the screen. 2) The OWNER inputs his/her oID and password. 3) The OWNER selects ENTER. 4) The system compares and validates OWNER input against OWNER and VENUE database. 5) OWNER is authorized; system prompts user with an option to stay logged in indefinitely and returns him/her to main screen of application. a. NOT authorized, system displays error message. Post Condition: The OWNER has been authorized and they are granted permissions to update the VENUE database.

Use Case (OWNER)	Description
Edit Venue Information	An OWNER can edit venue database containing information such as wait time, events, specials, admission cost, etc. Actor(s): OWNER Pre-condition: A logged in owner. 1) The OWNER selects to add event, wait time, or admission cost to their respective venue in VENUE database 2) The OWNER inputs desired information. 3) The OWNER selects ENTER. 4) The system updates the VENUE database. 5) The OWNER may also return to the previous screen via a back button. Post Condition: The VENUE database is updated. The owner is returned to the owner's menu.
Add Event/Wait/Admission	An OWNER can edit venue database containing information such as wait time, events, specials, admission cost, etc. Actor(s): OWNER Pre-condition: A logged in owner that selected to edit venue information from the owner's screen. 1) The system displays current venue information for logged in OWNER. 2) The OWNER selects information to edit or add to the VENUE database. 3) The OWNER selects ENTER. 4) The system updates and displays the VENUE database. 5) The OWNER may return to the previous screen via a back button. Post Condition: The VENUE database will be updated. The owner will be returned to the owner's menu.
View Logs	An OWNER that has logged in and been verified by the system can view logs for their venue such as attendance throughout day, peak times, effectiveness of marketing strategy etc. Actor(s): Owner Pre-condition: A logged in owner that selected to view logs from the owner's screen. 1) The system displays a menu of options for individual logs, ranging from population over time to gender-ratios. 2) The OWNER selects an option from one of these on menu. 3) The system displays the log in visual form and text form for owner. 4) The system prompts the OWNER to export log as document. 5) The system prompts the OWNER to reset log data. 6) The OWNER can navigate through menu by selecting BACK button or selecting new option. Post Condition: The OWNER is returned to the owners menu

Use Case Diagrams

Figures 1.1 & 1.2

1.1

This figure depicts the use case diagram for the normal user of the application. Upon starting the application, the user will be presented with three options: view map, register, or check in.

1.2

This figure depicts the use case diagram for the owner users of the application. Upon starting up the owner menu of the application, they will be able to perform actions such as view venue metrics, edit venue information or post events and specials.

Class Diagram

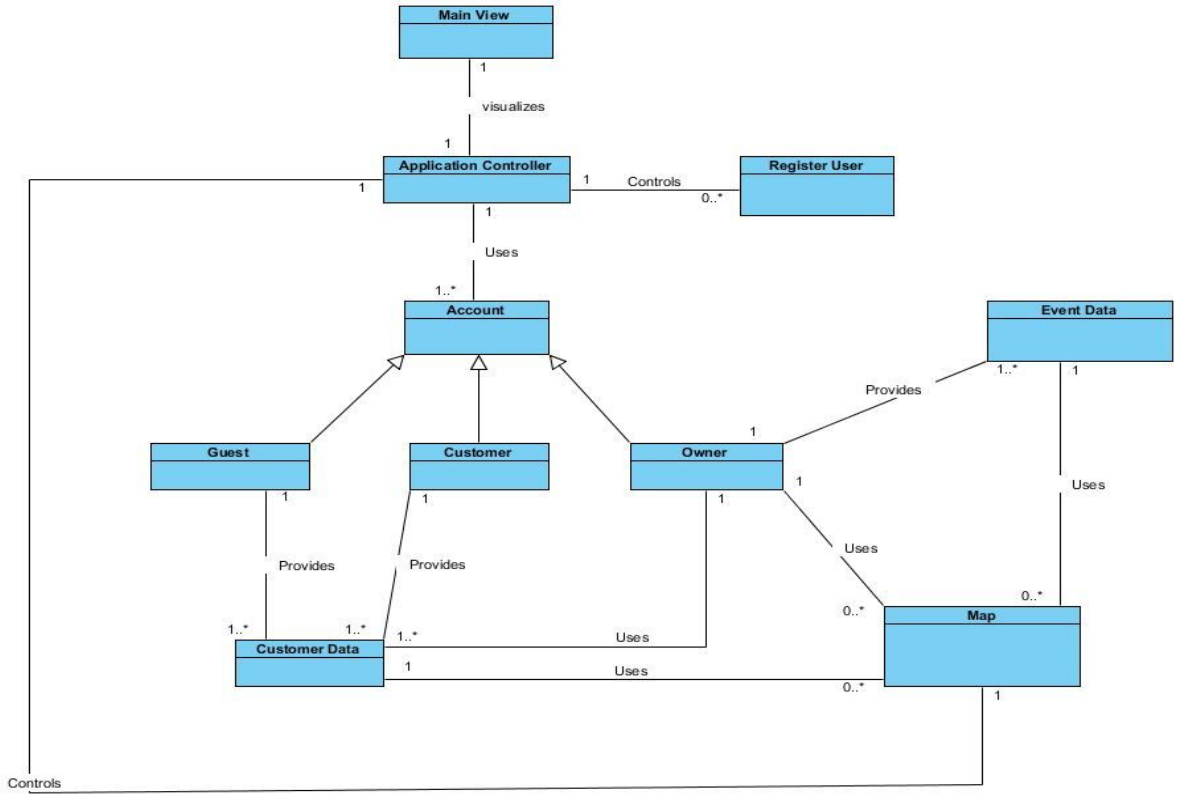

Figure 2.1

This figure depicts the high level class diagram for users and owners of the application. It has a main driver controller called application controller that controls the way in which the application is run. It uses the account class to register users who have not yet signed up with the application. If a user has signed up it will guide the user to their custom map. The main view is a visualizer for the application controller. Guest, customer and owner all inherit from account. When a customer and guest uses the application they provide data, this will be stored in the back-end server. The customer data class will help in this process. Owners will be able to provide event data. Event data and customer data will be used in creating the custom map.

Activity Diagrams

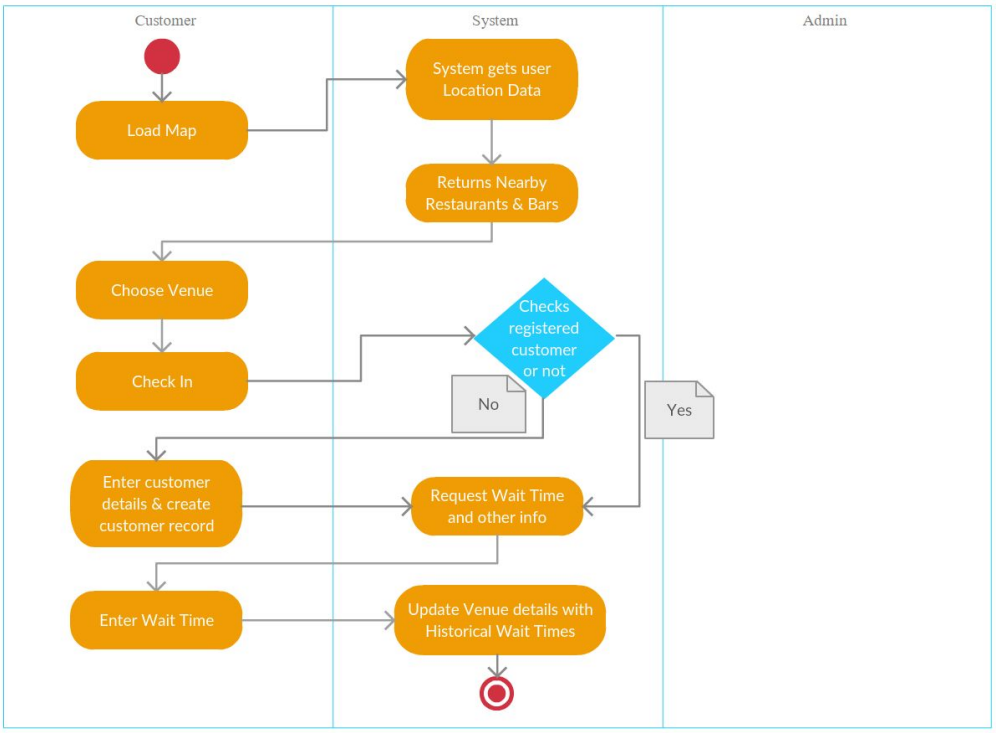

Figure 3.1

A customer will load into the map, behind the scenes the system will pull user location data and return a map to the user with nearby restaurants and bars. The customer will select their venue of choice and proceed to it's page, where data from Google Maps API and any Owner added information will be displayed. If their location confirms they are located at the venue, they will have the option to check-in. Once they choose to check-in we will require a login, if not previously registered the system will prompt the user to create an account. Once logged in they will be checked in and the user will be asked wait times, admission fees, etc. where applicable. The system will update the database with the information provided.

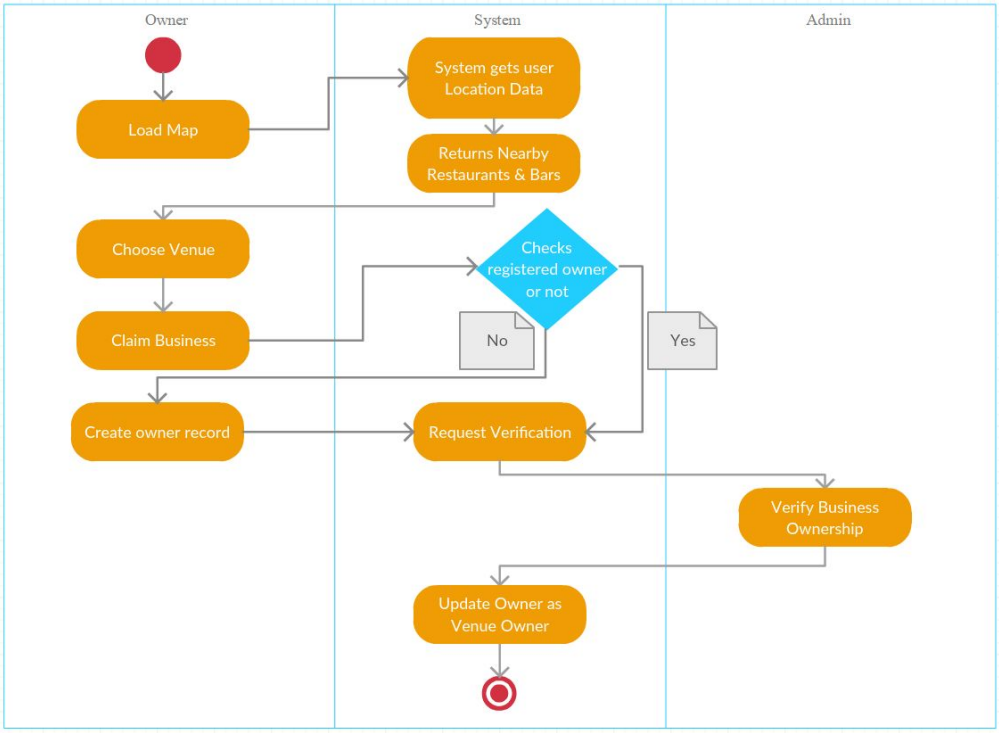

Figure 3.2

A owner's experience will start out in the same way as the customer. They will load a map, produced by the system via Google Maps API. Once they select a venue and proceed to it's page they will see an option to claim business. At this point we will require a login, if not previously registered the system will prompt the user to create an account. Once logged in they will be asked to verify some information about their business and behind the scenes we will need to implement an owner verification process. After the owner is verified the system will update the database and the owner will be notified. At this point, the owner can update specials, events, hours, etc.

Detailed Class Diagrams

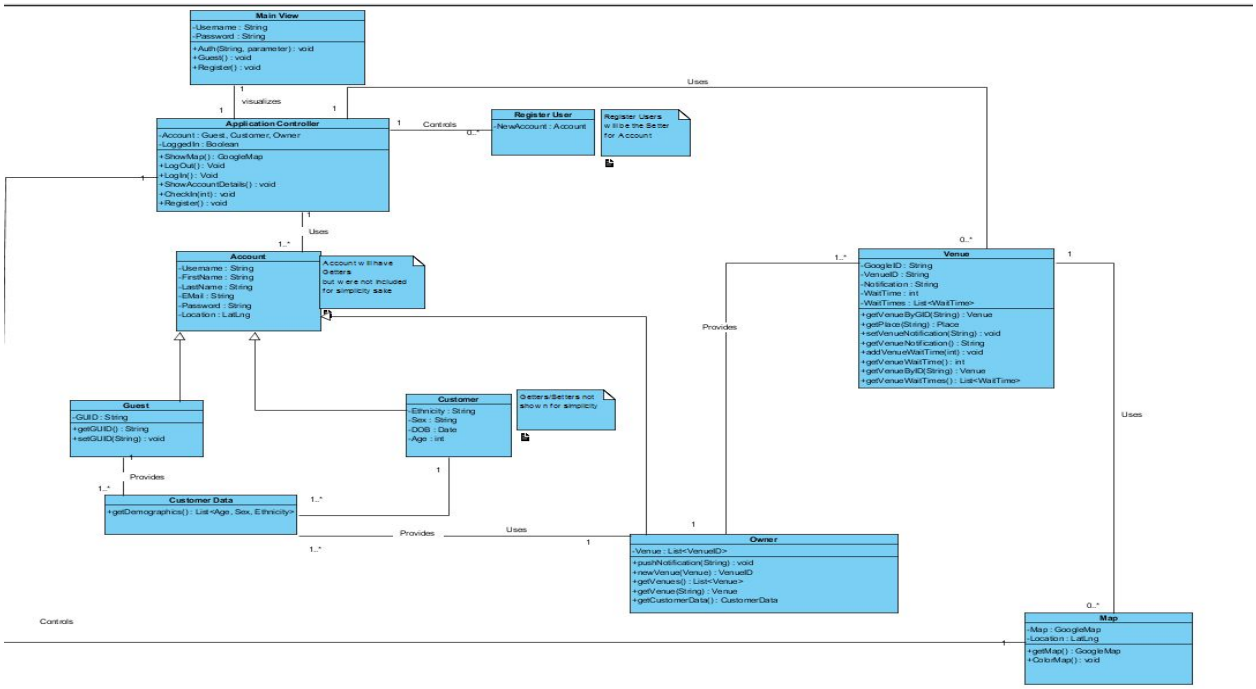

Figure 4.0

A more complete class diagram breakdown. Showing methods and attributes of the high level class diagram

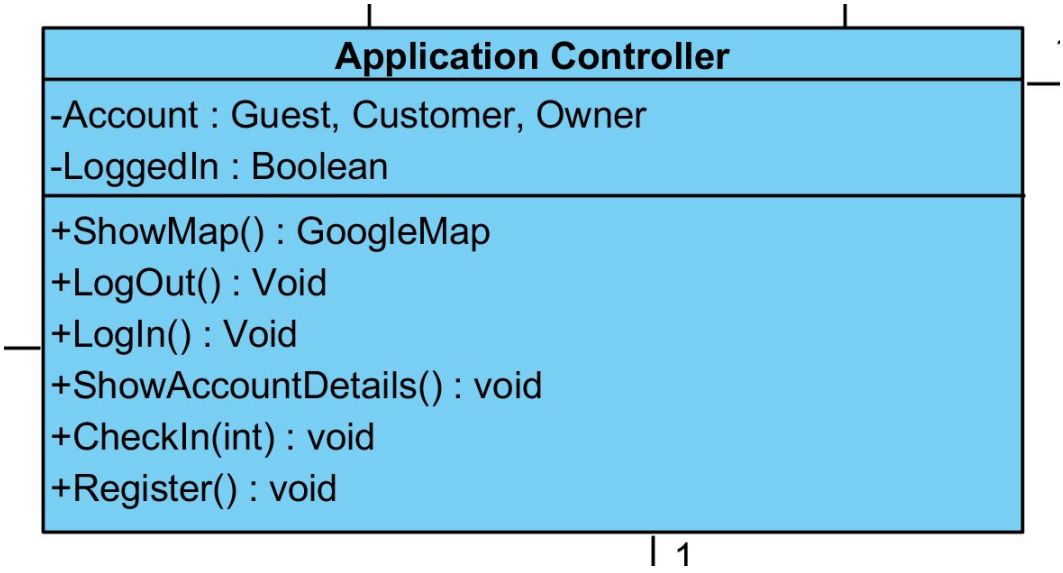

Figure 4.1

The Application Controller is the backbone of our application. It controls what happens when our application is in use. It keeps track of the account that is logged in, whether that be a customer, owner or guest account. It has a method to log the user out. The method *ShowAccountDetails* will provide the user with all their information about their account. Once the account is logged in, into any of our three accounts, there will be a couple of options for the user: *ShowMap*, *CheckIn*, *ShowAccountDetails*, *LogOut*. If the user clicks on *CheckIn* the application will check to make sure the user is at least close to the venue they have chosen and then offer the user to put in a WaitTime. The *Register* method will pass along the user intent to create an account to the RegisterUser class.

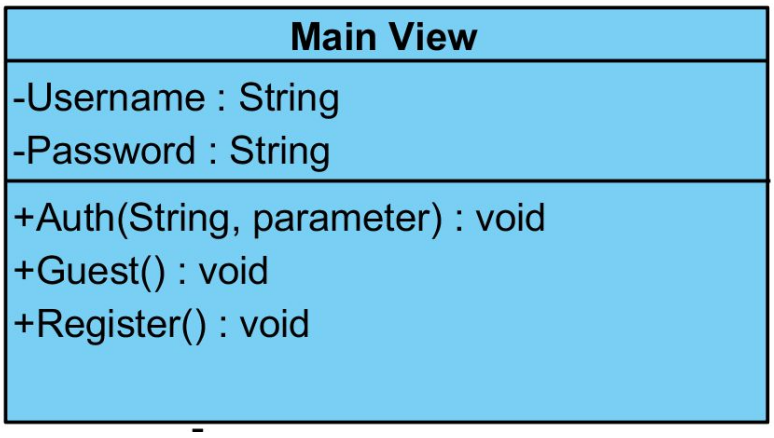

FIgure 4.2

Provides an interface for first starting up the application. It provides a way for the user to enter a username and password, or to use a guest account, or to register.

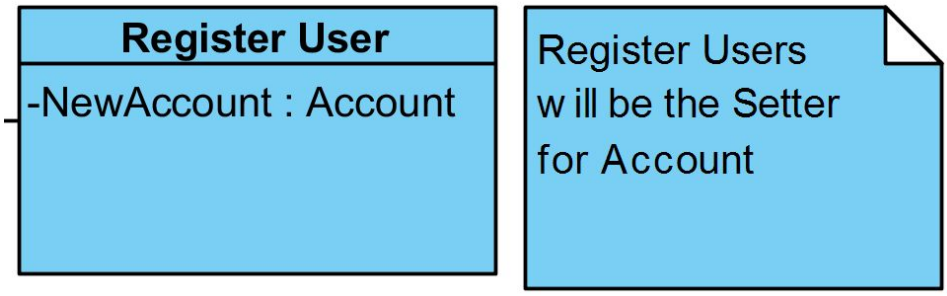

Figure 4.3

The Register Account class will have an attribute called NewAccount, that will hold an Account object. In this class the setters for the account class will be used.

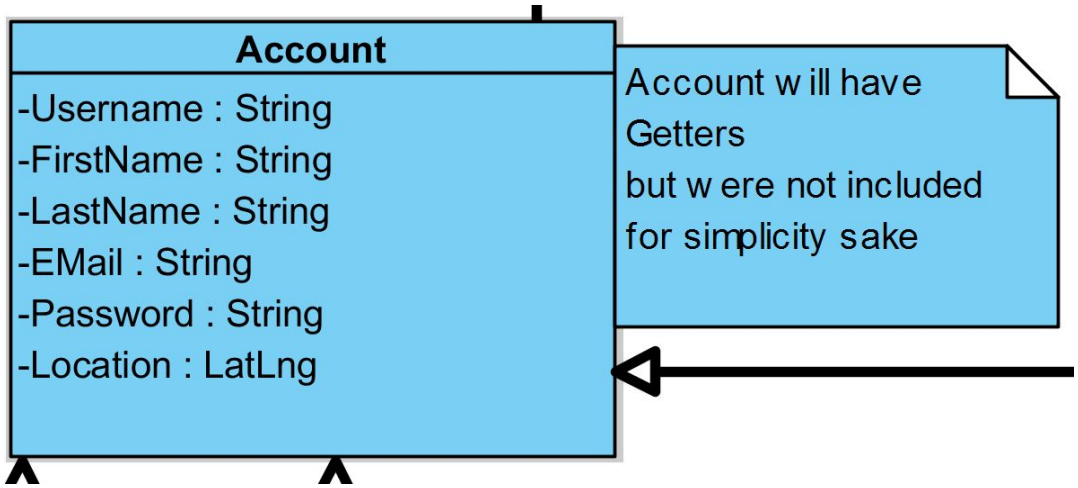

Figure 4.4

The Account class holds information on accounts. It contains a Username, FirstName, LastName, Email, Password, and Location attribute. It will have for each attribute a getter, the setters are in the Register Account class. The Account class will be inherited by the following classes: Guest, Customer, Owner.

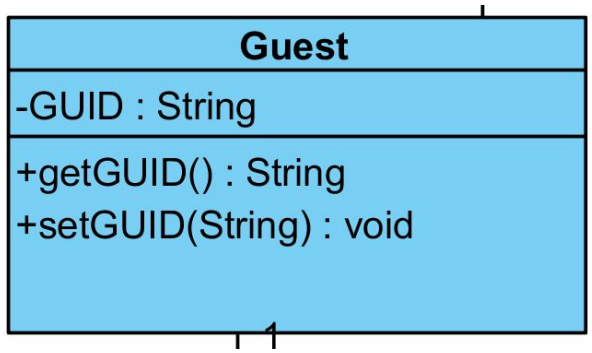

Figure 4.5

For users who do not wish to create an account, a guest account can be used. This guest account inherits from Account but leaves the Account attributes blank. The Guest class automatically creates a Guest User ID from Google's User Data ID. Source for Google GUID: https://developer.android.com/training/articles/user-data-ids.html

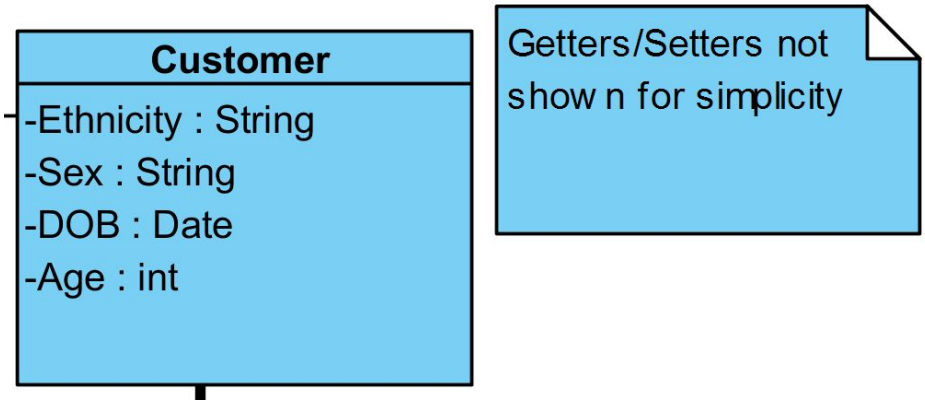

Figure 4.6

The customer class are for users who wish to create a profile with our application. It contains a bit more information from the Account class, such as the following: Ethnicity, Sex, DOB, Age. There are getters and setters for each of the new attributes. These attributes will provide data to the Customer Data class.

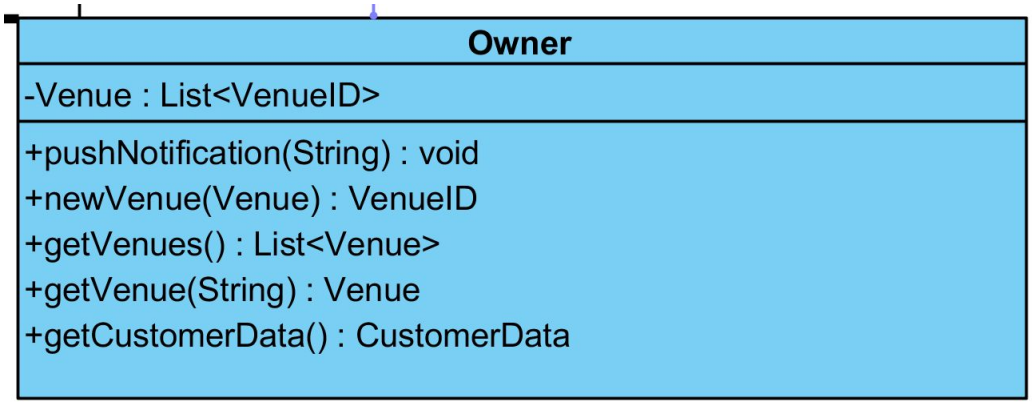

Figure 4.7

The Owner class has an extra attribute Venue alongside the Account information. The Venue attribute will keep track of the venues the owner owns. The Venue attribute is a list in case the Owner owns more than one venue. The venue attribute accepts VenueID objects. The owner may want to provide notifications to users about deals for their venue, this is done through pushNotification. Owners can also access customer data about their venue by using the method getCustomerData. If the owner wants to see their venues or a specific venue they can use the getVenues or getVenue respectively.

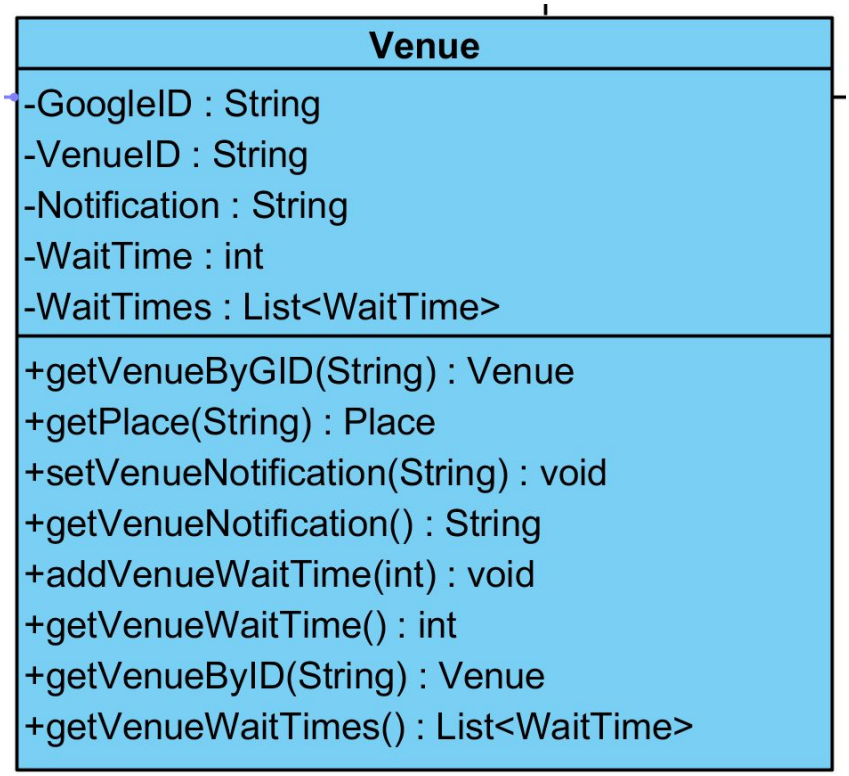

Figure 4.8

The venue class are the markers on our map. They contain the following attributes: GoogleID, VenueID, Notification, WaitTime, WaitTimes. The GoogleID will be a Place from google. Google Place source: https://developers.google.com/places/android-api/place-details The VenueID will be used to track individual venues in our database. The Notification will be used to set or get a notification. WaitTime, and WaitTimes keep track of the WaitTimes throughout the day.

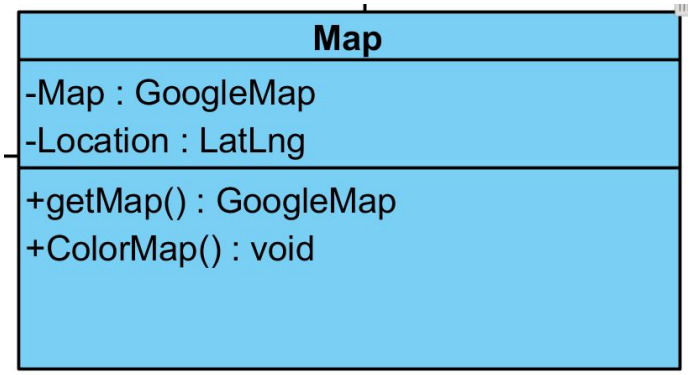

Figure 4.9

The Map class will use Google Map as the backbone. And it will use Google's location in the form of LatLng. The method getMap will be called to show the map with the markers or venues shown on the map. The method ColorMap will be used to show the different wait times on the markers.

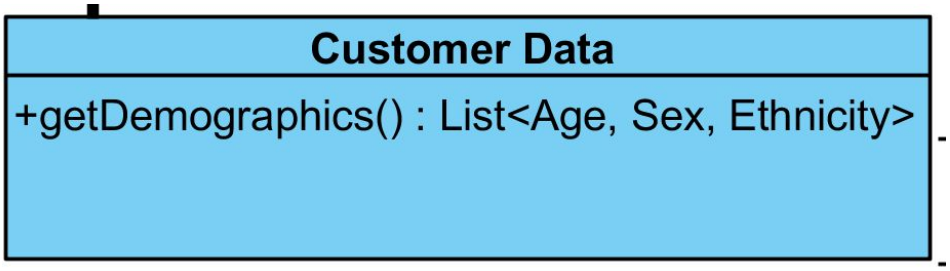

Figure 4.10

Get demographics to be used when the Owner wants it. It also provides customer data to the Map class.

Sequence Diagrams

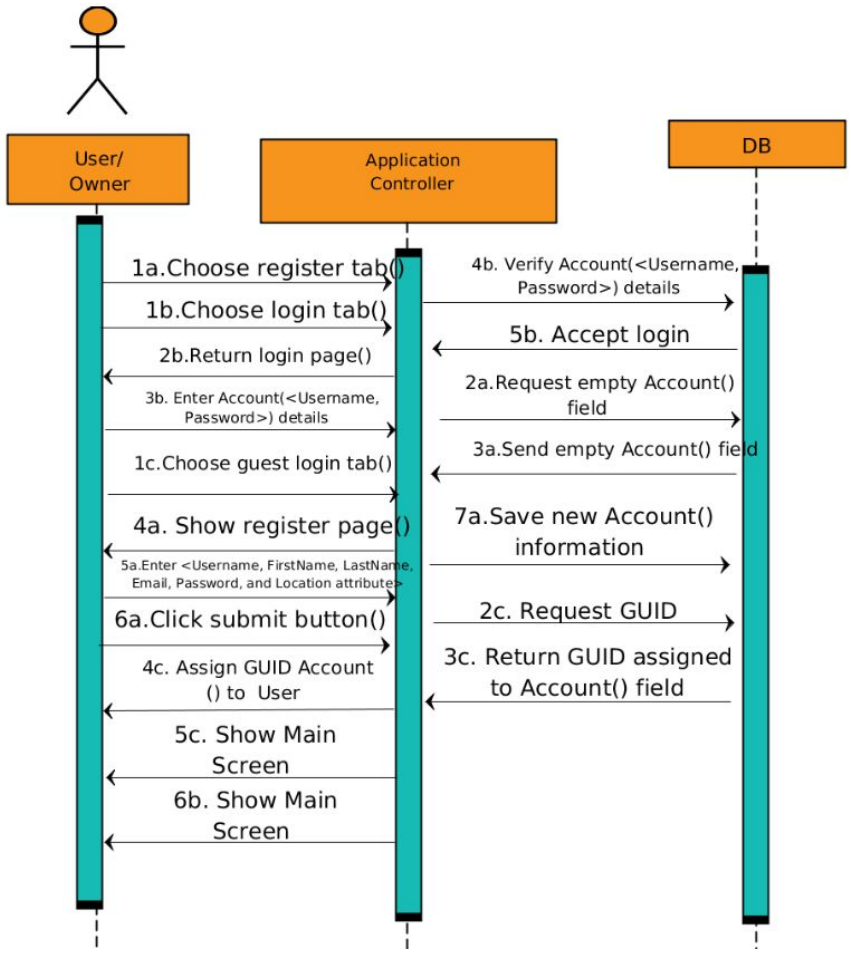

Figure 5.0

The Register Sequence Diagram has three possible courses of action depending on the input of the user:

1a. The User chooses to register an account. The (a) course of action is then followed to completion.

1b. The User chooses to login using an existing account. The (b) course of action is then followed to completion.

1c. The User chooses to login as a guest, using a GUID provided by Google. The (c) course of action is then followed to completion.

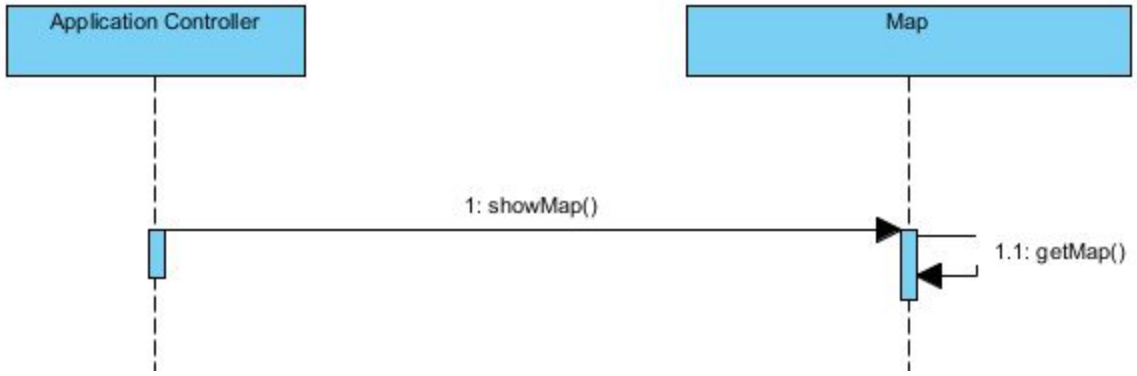

Figure 5.1

This sequence diagram shows how to show the map. Application Controller calls its showMap method, which calls the object Map. Map calls its getMap method which uses a google API to show the map on the screen.

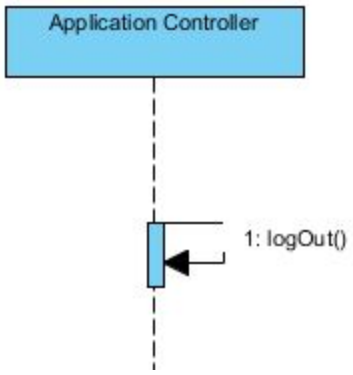

Figure 5.2

This sequence diagram describes the logout procedure. Application controller will call its logOut method which will log the user out of their session.

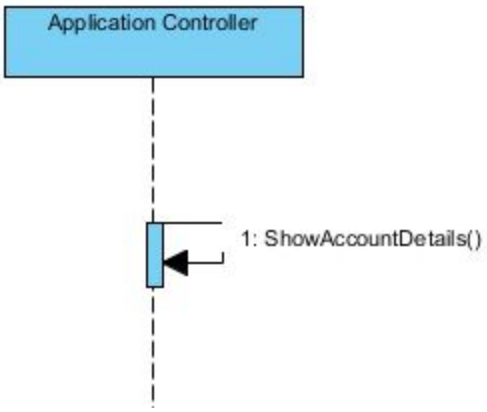

Figure 5.3

This sequence diagram describes showing account details, such as e-mail, and username. The application controller handles this by calling its method ShowAccountDetails.

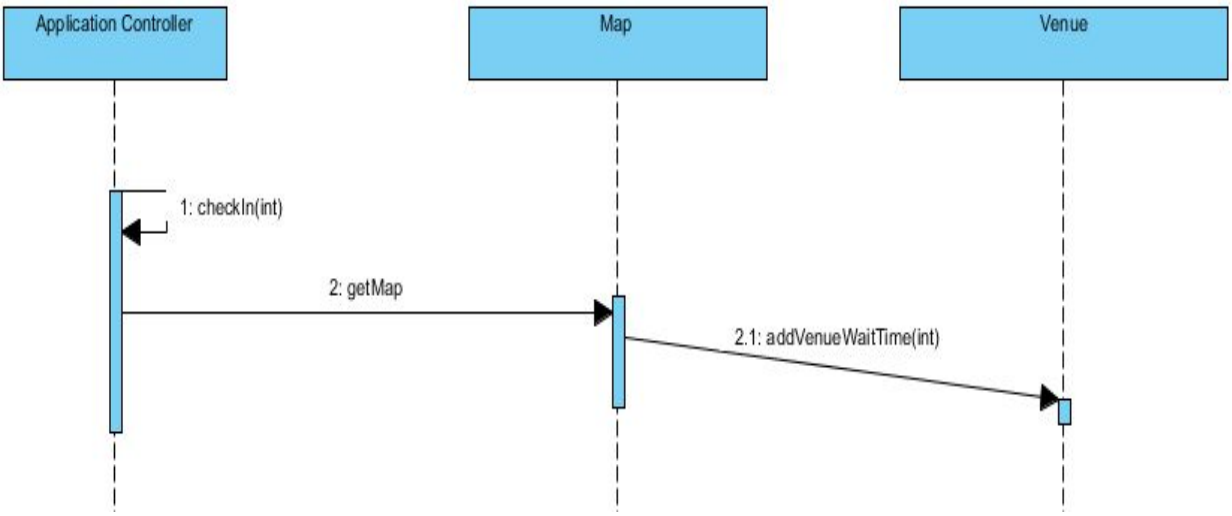

Figure 5.4

This sequence diagram describes adding a wait time to a venue. A person check ins and then is asked to input a wait time. This is done by the application controller calling its method checkIn which brings up a time to put in, and then Application Controller calls the Map. Venue is then found on the map and a wait time is added.

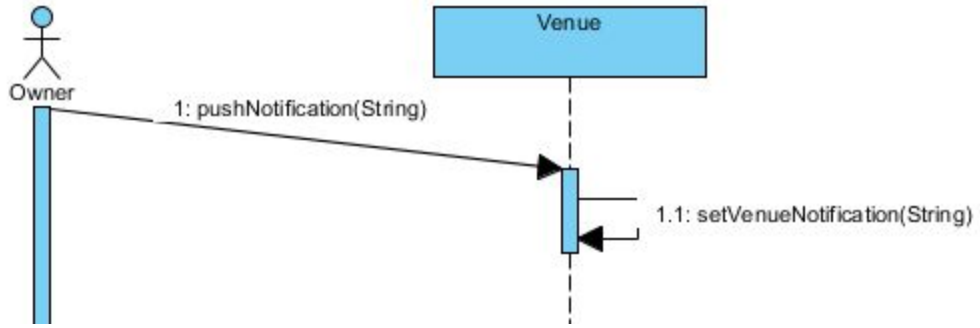

Figure 5.5

This sequence diagram describes an owner pushing a notification to its venue(s). The owner calls pushNotification and inputs a notification. Venue is then updated with a new notification by way of setVenueNotification.

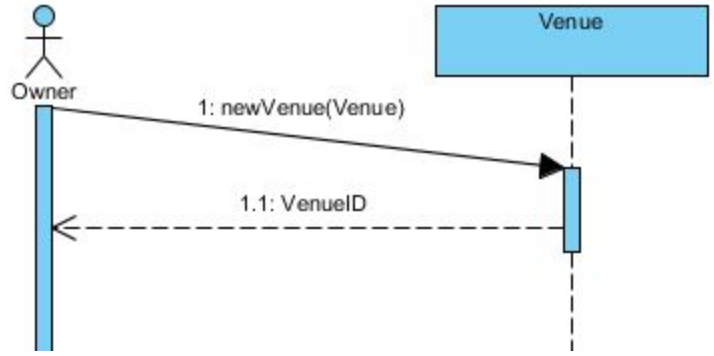

Figure 5.6

This sequence diagram describes an Owner making a new venue on our application.

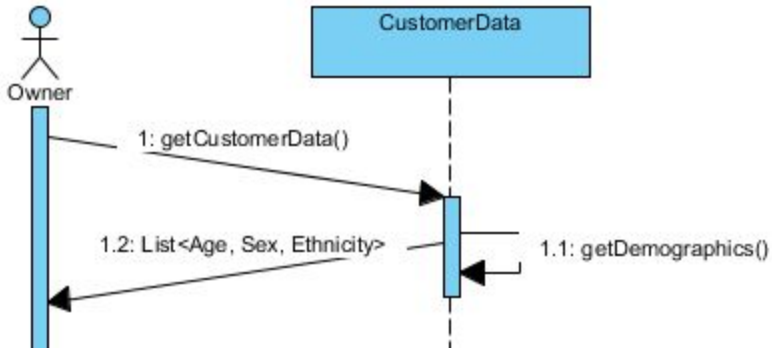

Figure 5.7

The above sequence diagram describes an Owner wanting to gather analytics about their customers that went to their venue.

What has changed?

More heavily focused towards Bars and other small venues.

Competing with apps such as NoWait, BuzzTable, Table's Ready is outside of our scope.

Have updated models to reflect a 1 to N Owner to Venue Relationship.

Known Issues

Need Notification and WaitTime classes to have CreatedDate, CreatedBy, etc. instead of just string/int respectively.

Additional Future Work

Merge past Guest activity with User Account upon creation by storing Guest activity via GUID.