Design Document

CS 307: Software Engineering

Nomad

Team 9

Kristin Beese

Janka Gal

Rajalakshmy Iyer

Joyce Kao

Table of Contents

Purpose

There are so many people of all ages who love to go on road trips, but documenting such trips is not so easy. Though GPS location and timestamps are automatically stored for pictures on iPhones, there is no way to clearly display the progression or the important highlights of a trip in a visual manner. The goal of this app is to pin pictures at locations with a description in order to have a visual scrapbook at the end of the trip. Even though there currently are some travel blogging and GPS tracking applications that can help track your trips, there is no single application that provides a cohesive overview. We can address this problem by organizing the different locations, media, and comments in a chronological list view for each trip with an additional mapview with pinpoints at each entry.

Non-Functional Requirements

- ❏ As a user, I would like
    - ❏ all of my trip information to be safely stored locally on my device
- ❏ As a developer, I would like
    - ❏ this app to be usable on an iOS device
    - ❏ the interface to be clear and simple but attractive
    - ❏ to be able to integrate with the CoreLocation API
    - ❏ to be able to integrate with the Google Maps API (time permitting)
    - ❏ to be able to integrate with the Yelp API (time permitting)
    - ❏ to be able to integrate with the Instagram API (time permitting)
    - ❏ to be able to integrate with the Facebook API (time permitting)

Functional Requirements

- ❏ As a user, I would like to
    - ❏ create a trip

- ❏ delete a trip
- ❏ add entry to trip
- ❏ return to a previous entry and edit it.
- ❏ view map of each trip with entry pinpoints on the map
- ❏ delete entry from trip
- ❏ track the route I have taken on my trip (if time allows)
- ❏ see basic statistics of my trip (total miles traveled, avg speed, time, avg number of stops per day, etc.) (if time allows)
- ❏ view each entry pin point as an icon of what the entry contains (if time allows)
- ❏ track travelers (if time allows)
- ❏ share my trip through social media (if time allows)
- ❏ view my friends' trips (if time allows)

❏ As a user creating a trip, I would like to

- ❏ add the title of the trip
- ❏ add description of the trip
- ❏ press a start button to start tracking my trip
- ❏ store the starting GPS coordinates
- ❏ store the starting date
- ❏ edit the title of the trip
- ❏ edit description of the trip
- ❏ add a trip icon

❏ As a user adding an entry, I would like to

- ❏ add a title
- ❏ add a description
- ❏ have the current GPS location automatically stored
- ❏ have the current timestamp automatically stored
- ❏ take or upload a picture
- ❏ edit a title

- ❏ edit a description
- ❏ add location names to pictures (if time allows)
- ❏ upload a video (if time allows)
- ❏ select entry type/pin point icon (if time allows)

❏ As a user ending a trip, I would like to

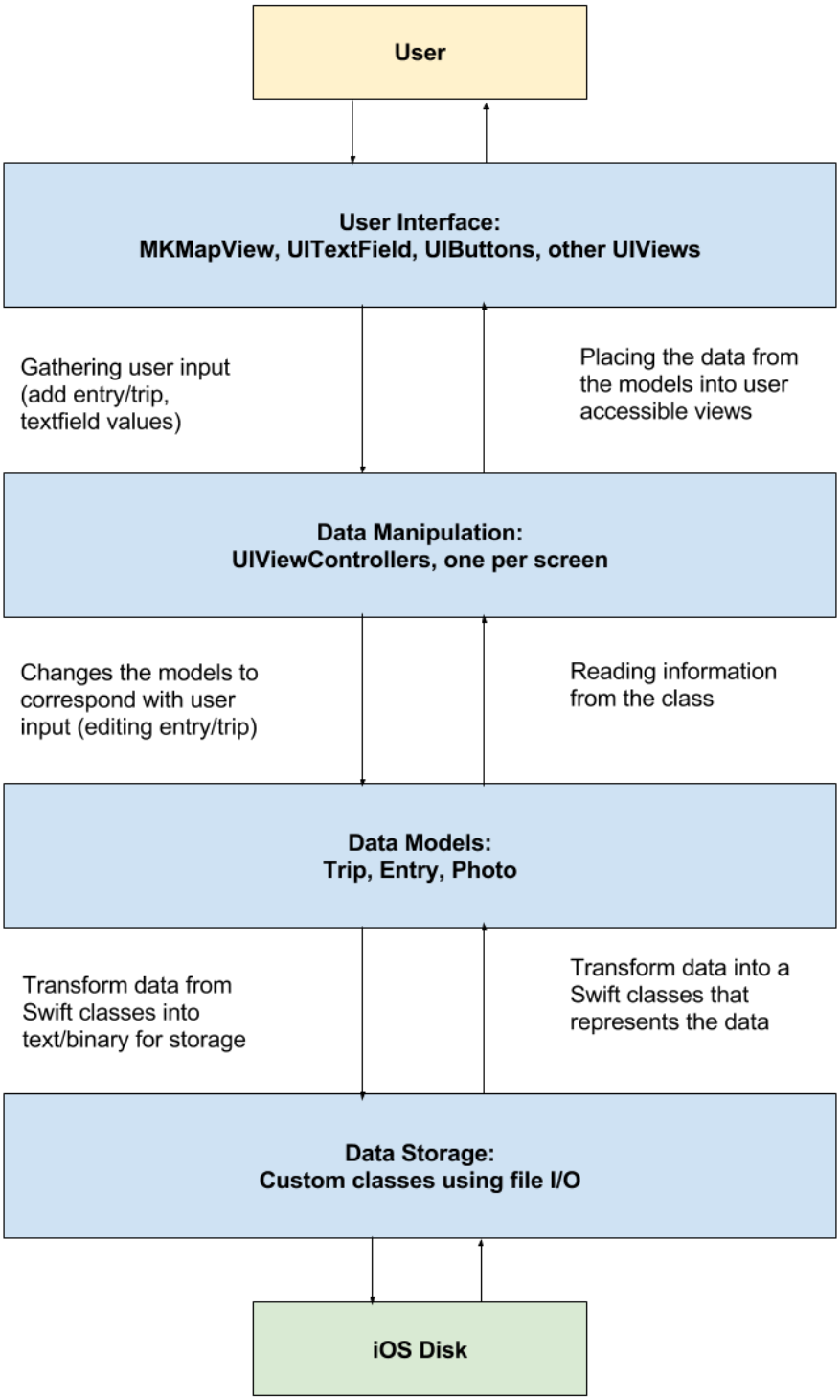
- ❏ press an end trip button to stop tracking the trip
- ❏ store the end point
- ❏ store the end date
- ❏ have a dialog popup to confirm ending the trip

❏ As a user wanting trip recommendations, I would like to

- ❏ see the most instagrammed locations around my current location (if time allows)
- ❏ see the most popular eateries around my current location (if time allows)

Design Outline

High-level Overview of the System



We are going to use the Model-View-Controller (MVC) paradigm because it's built into iOS. Doing it another way would require extra work and be atypical for an iOS application. As you can see from our diagram, we are going to employ a strong separation of concerns so that each component of the application only interacts with two other components. This separation will make the code easier to write and debug because each component can be simpler. We are writing the application in Swift which has excellent support for modern object oriented practices.

User Interface

The User Interface portion will display trip and entry information to the user, while gathering input and actions from the user such as adding a new trip, editing an entry, or uploading a photo. It should be user friendly, easy to use, and look relatively nice. We won't have enough time to make the most perfect User Interface, but we can make good decisions along the way to make one that is very usable. We will use built in classes provided by Apple for buttons, text fields, maps, etc. We will create custom views for certain aspects like the photo picker button.

The views in the User Interface will be laid out and managed by the view controllers in the Data Manipulation section. The User Interface gathers user input by adding a trip/entry, editing information, and uploading a photo and then sends that information to the view controllers in the Data Manipulation section.

Data Manipulation

The Data Manipulation portion will consist of multiple view controllers, one for every unique screen in the application. We will have atleast a HomeScreenViewController, AddEntryViewController, AddTripViewController, TripViewController, EntryViewController, and a MapViewController. The view controllers will receive user input from the User Interface. From there, the view controllers will check if the input is

valid. If the input is not valid, it will manipulate views in the User Interface to display error messages. When the input is valid, the view controllers will mutate the Data Models to fulfill the user's request.

Data Models

The Data Models will be classes that are Swift representations of the trips, entries, and photos. We will have a Trip, an Entry, and a Photo class in order to keep track of the user's travel experiences without having to know the implementation details of every part of the data storage. They provide a nice abstraction for the Data Manipulation so that data can be smoothly passed throughout the system. It interacts with the Data Storage by providing the information about the different types and fields in the class to store the data efficiently on the disk.

Data Storage

The Data Storage consists of custom classes using file I/O that we will create. Its purpose is to store the Data Models onto the disk so that they can be loaded back into Swift classes at a later time. It will read the information that needs to be saved from the Data Models, and then the Data Storage section will control where the data files are written to on the iOS device.

Design Issues

Non-Functional Issues

Issue 1: What iOS development language will we use for this application?

- ❏ Option 1: Swift
- ❏ Option 2: Objective-C
- ❏ Choice: Swift is our choice for this application.
- ❏ Discussion: All of the members in the group do not have any iOS experience, but we all are excited to challenge ourselves by learning a new language. We decided to choose Swift over Objective-C because it has more helpful and up to date resources made by Apple. Swift is made specifically for iOS app development, and since Apple is transitioning from Objective-C to Swift our code will be more maintainable in the future.

Issue 2: How will we store the user's information?

- ❏ Option 1: Using a database online
- ❏ Option 2: Storing the information locally on the user's device
- ❏ Choice: Storing the information locally on the user's device
- ❏ Discussion: We chose to store the information locally on the user's device because we are on a time constraint for this project and we thought it would be better if we focused on Swift instead of learning an online database system as well. Since we are also not focusing on the social media component yet, it should be fine for the user to just look back at trips on their own device.

Issue 3: Should we create a server so that users have to create an account within the app and then would they be able to log in from different devices to view their information?

- ❏ Option 1: Yes, the user has to log in using a custom account for the app
- ❏ Option 2: Yes, the user has to log in using a social media account

- ❏ Option 3: No, the user doesn't have to log in
- ❏ Choice: No, the user doesn't have to log in
- ❏ Discussion: Because the trips will only be stored on the user's phone and not in a database, we felt it is unnecessary to have the user create an account and the app will function similar to the notepad on a user's phone. Having to create an account and log in would create unnecessary work for the user, since there is no website or social media component to the app as of now.

Issue 4: Should we use the Google Maps API or the iOS Map Kit Framework for the map that will display the user's pin points?

- ❏ Option 1: Google Maps API
- ❏ Option 2: iOS Map Kit Framework
- ❏ Choice: iOS Map Kit Framework
- ❏ Discussion: We will go with the iOS Map Kit Framework because it is built in with iOS and uses XCode 6 so it would be simpler and more efficient to implement than Google Maps API allowing us to focus on mastering Swift and improving the app. If time permits, we would consider implementing the Google Maps API for recommendations and the additional features that it provides.

Functional Issues

Issue 1: Should a user create a trip the moment their traveling starts, or be able to set up their trip on the app in advance?

- ❏ Option 1: Start trip automatically after adding the trip
- ❏ Option 2: User creates a trip and the program waits until the start date to allow the user to add entries
- ❏ Choice: Start trip automatically after adding the trip
- ❏ Discussion: Starting the trip automatically after adding the trip will help make our application more user friendly. If we allow the user to wait until the start date, it might be confusing to the user since our home page is focused on having a

"Start Trip" or "End Trip" button. Since our application is not used to plan a vacation, and is instead used to log your vacation as you go, creating the trip before it actually starts would not be useful to the user. We will have to make sure it is clear to the user that the day you start your trip is also the same day that the user creates their trip.

Issue 2: How should the route be tracked?

- ❏ Option 1: Continuously have the app monitor the GPS coordinates and update the route in the app
- ❏ Option 2: Collect GPS location at certain time intervals and display an approximate route
- ❏ Option 3: Store GPS coordinates every time a user submits an entry and then create pinpoints on the map that correspond with that specific entry
- ❏ Choice: Store GPS coordinates every time a user submits an entry and then create pinpoints on the map that correspond with that specific entry
- ❏ Discussion: We are going to start by just recording the GPS coordinates every time an entry is recorded. Then that information will be sent to our MapViewController for that trip's specific map screen and it will send the User Interface the information to create a pinpoint in that specific location. If we find that we have more time at the end of the project, we will look into options 1 and 2, but we are going to start with option 3.

Issue 3: How should the users share their trips with other people?

- ❏ Option 1: Show their friends the app in person
- ❏ Option 2: Get a custom link to their trip to share
- ❏ Choice: Show their friends the app in person
- ❏ Discussion: Due to time constraints, we are going to have users share their trips with other people by showing them in person. In the future, if we have more time we will definitely explore how we could create a shareable link so that you could

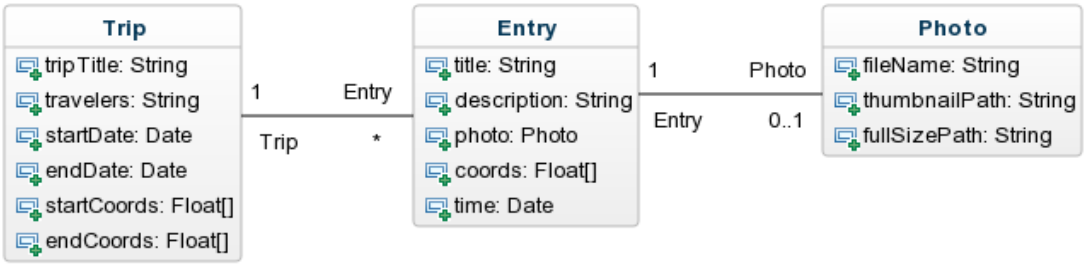
share your travels with friends and family. If we were to continue this app beyond this class, we would want to make it more of a social media app, where users could share or follow each other's trips.

Issue 4: Should the users be able to reopen a trip and add entries to it if the trip has already ended?

- ❏ Option 1: Yes, in case they accidentally ended the trip
- ❏ Option 2: No, there will be a confirmation before ending a trip so accidents will be minimized
- ❏ Choice: No, there will be a confirmation before ending a trip so accidents will be minimized
- ❏ Discussion: We will create a dialog popup to appear every time the user clicks "End Trip" to ask the user if they are sure they want to end their trip. This will help us because we can be sure that the user meant to do that, and then we know that that specific trip is over by collecting the current date and then disabling the "add entry" buttons for that trip. We will also log the user's last GPS coordinates when they end the trip and mark that on the map so the user can see their trip from start to end. We decided to this because not allowing users to edit a trip once it is over will make the code more clear and consistent, or else users may attempt to edit past trips while logging an active trip. Additionally, the idea of the app is to be more spontaneous and in the moment so it captures the trip as it happened.

Design Details

Class Diagram



This is a class diagram of all of the models

We did not create a class diagram of all of the view controllers because the class diagrams would look very similar and the important parts of the code are within the handling of the views. For example, the AddEntryViewController will have a field for the entry. All of the view controllers will utilize these model to properly and safely pass data throughout the application.

Descriptions of Classes and Models

Trip:

The Trip class encapsulates a real trip that the user goes on in real life. It has fields that include the trip destination, travelers, start date, starting coordinates, ending coordinates, and end date. None of the fields of the trip have to be unique, because the user could travel to the same destination multiple times and could also travel with the same people multiple times. The Trip will allow you to access all of the Entry objects associated with it.

Entry:

The Entry class imitates real life experiences the user will have on the trip. It has fields that include a title, description, photo, GPS coordinates, and a date. None of the data fields need to be unique because a user could want to log multiple entries at a single time, in the same place, or with similar descriptions. Every Entry is associated with one single Trip.

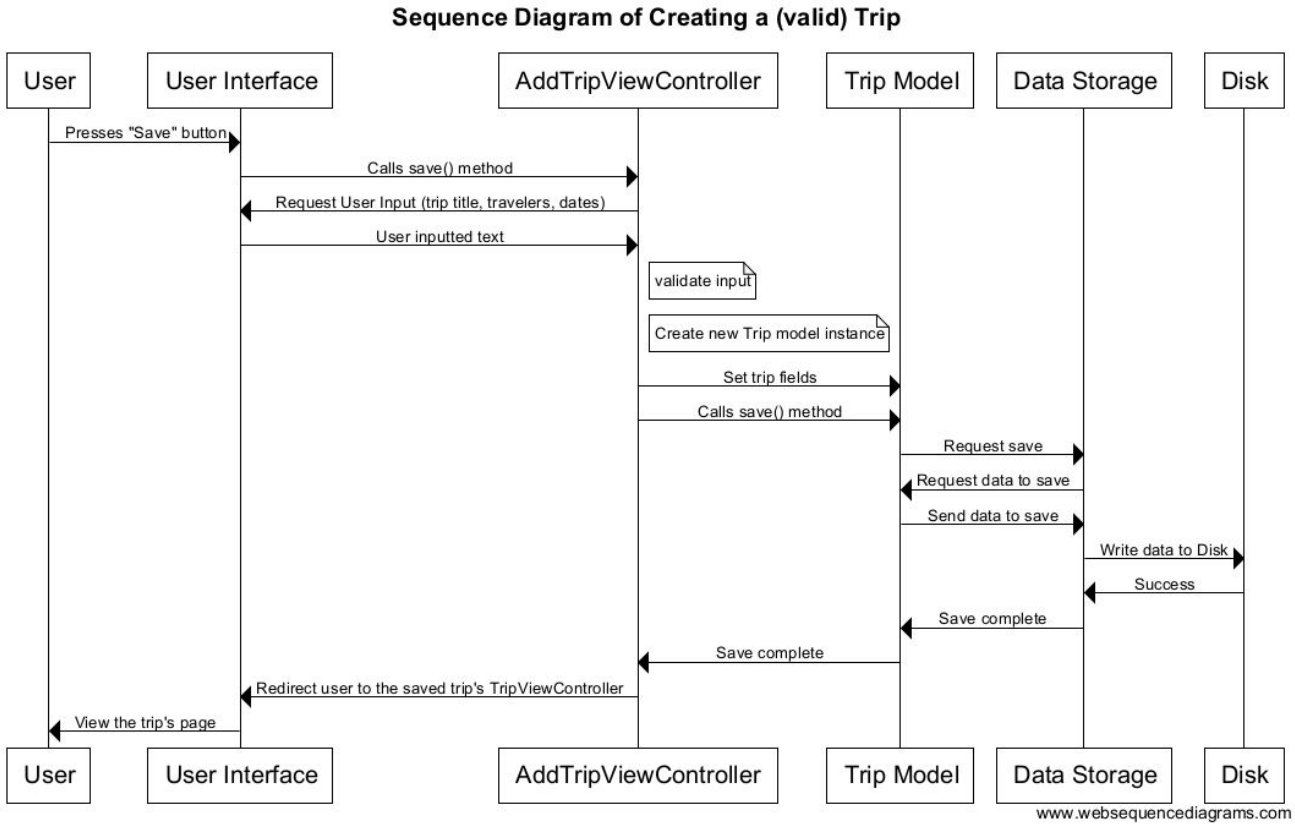
Photo:

The Photo class encapsulates a photo and provides access to it as well as a thumbnail. This class will allow users to visually relive their travel experiences. Every photo object is associated with one single entry. But, every entry does not have to have a photo.

View Controller Classes:

- ❏ HomeViewController: the view controller for the home screen that will give the user options to start a trip, end a trip, and view all of your trips.
- ❏ AddTripViewController: the view controller for the add/edit trip screen will be written in such a way that can be reused for adding a trip and editing an existing trip since the data that needs to be collected remains the same.
- ❏ TripViewController: the view controller for the view trip screen of any trip that will allow the user to select an entry to view.
- ❏ AddEntryViewController: the view controller for the add/edit entry screen will be written in such a way that can be reused for adding an entry and editing an existing entry since the data that needs to be collects remains the same.
- ❏ EntryViewController: the view controller for the view entry screen of any entry that will allow the user to view all of the information about that specific entry
- ❏ MapViewController: the view controller for the map screen of a specific trip that generates a map with pins for each entry by using the GPS coordinates

Sequence Diagram

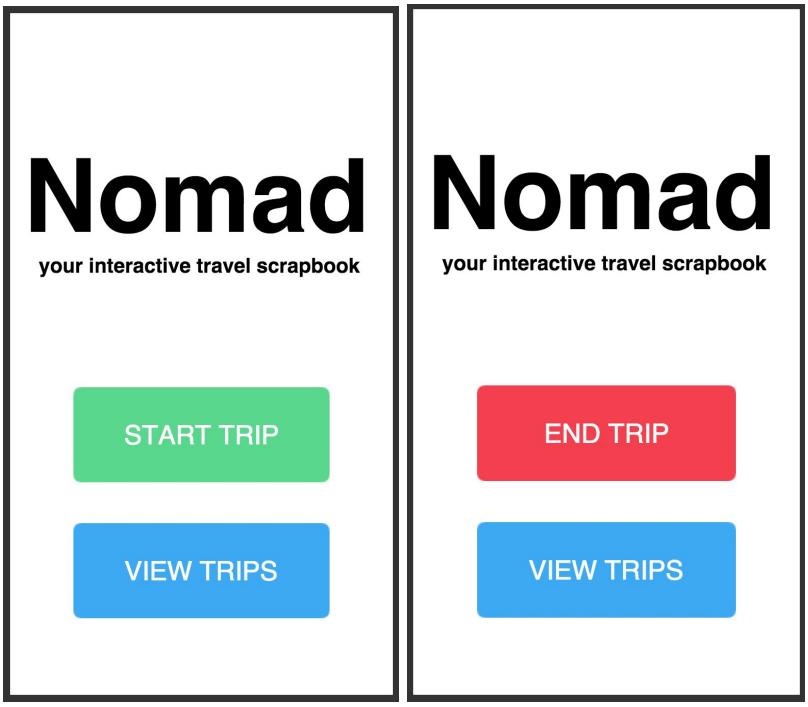


The diagram above is an example of a Sequence Diagram for when a user adds a valid trip. All of the components of the system work together to deliver a smooth experience to accomplish the user's requests.

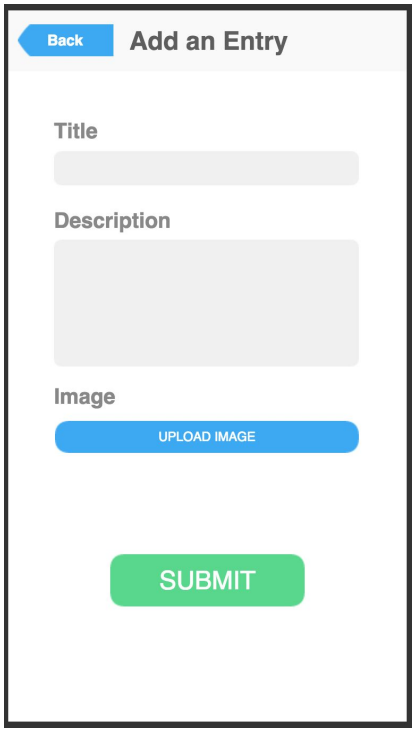
UI Mockups

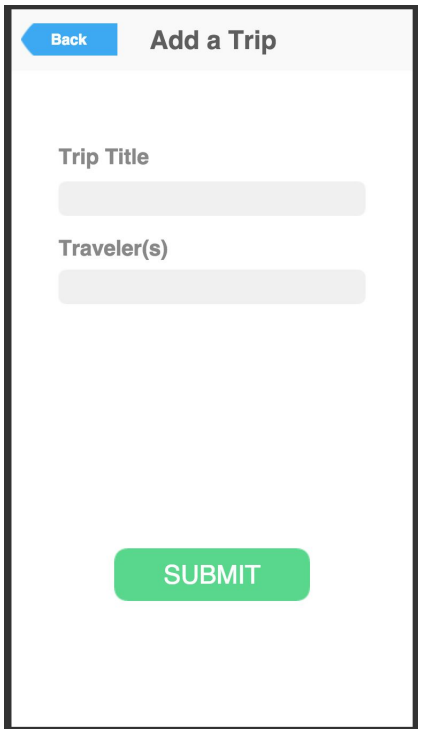
The following mockups are not exact replicas of what our finished product will look like, but are representative of the user flows that will occur while using our application. We hope to achieve a simplistic and clear experience for the user with our straightforward designs.

Home Screens:



When a user opens up the app and is not currently on the trip, they will see the screen to the left. When the user is currently on a trip their home screen will look like the screen to the right. If the user has not logged any trips yet, the “My Trips” screen will display a message notifying them of that.

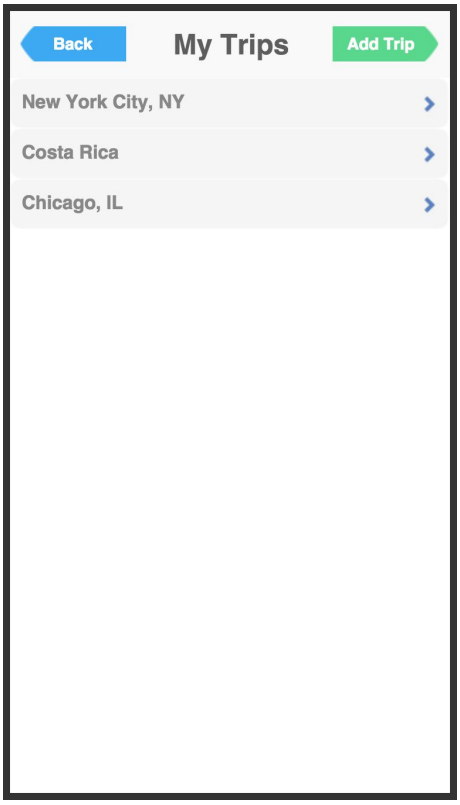


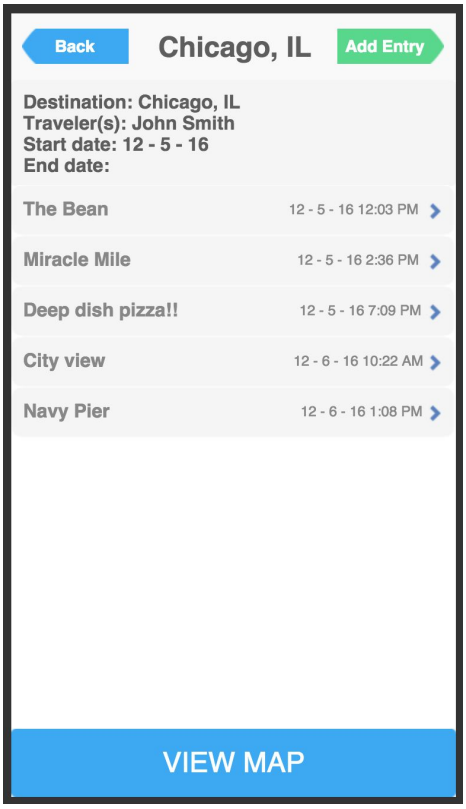


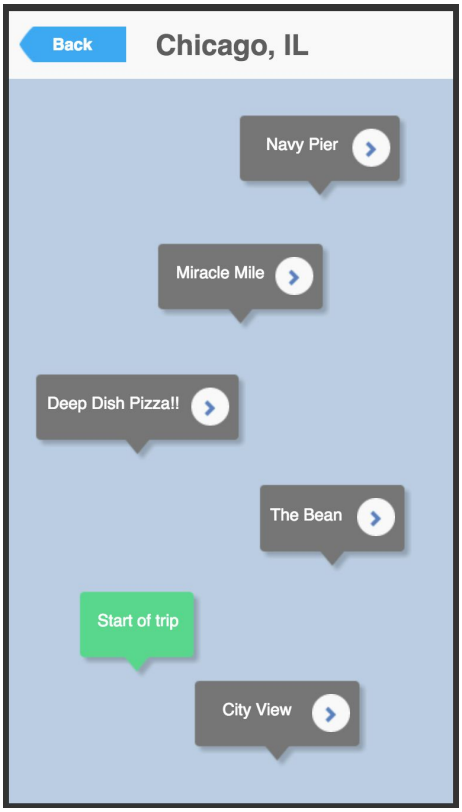
Add Entry/Trip Screens:

These screens allow the user to easily enter in information about the trip or entry.

Information Screens:







The following mockups show various screens throughout the app where information is showed to the user.

The upper left hand screen is seen when a user views their trips.

The upper right hand view is seen when a user clicks a particular active trip.

The lower right hand view is seen when a user clicks on a specific entry within a trip.

The lower right hand view is seen when a user clicks on “View Map” in a trip.