Software Architecture Specification

March 21, 2018

Team 3 (Jack Taylor, Mitchel Smith, Elan Kainen, Jason Wen)

1. System and Architectural Context

1.1. Rationale

The overall architecture will be using the individual pages of the app as building blocks for the entire app. This is useful for two reasons: 1) It allows us to build up a base and utilize the prototyping model by building a little bit on each page and then testing that everything works together, and 2) It allows us to work on multiple pages concurrently without fear of breaking something else at the same level (i.e. changing something on the profile page won't affect the trip selection page since they do not interact).

The primary communications will be between the pages of the app. There are three primary user-visible pages(Profile, Trip Creation, Trip Selection) and one hidden-page (Existing Trips) with the possibility of others being added later. The Trip Creation and Trip Selection pages need to be able to access information on the profile page. This will allow auto-filling of information during trip creation and basic filtering during Trip Selection. The Trip Selection page additionally needs to be able to access the Existing Trips page to display available trips. Finally, the Trip Creation page needs to be able to interact with the Existing Trips page so that duplicate trips cannot be created.

For our server needs, Amazon Web Services (AWS) mobile services provides what seems to be a straightforward and well-supported developer kit for mobile application developers (for both Android and iOS). This will be valuable both for allowing us to implement a cloud-supported backend for our application's server needs as well as giving us experience using AWS services.

1.2. Scope

The system that will be our travel buddy application will be separable by the different pages and functions of the software itself. It will be key for our group to plan out the architecture (and code as a whole) of our project so that each part, however small or large, can be designed and developed by different people. This will allow each portion of the application to fit seamlessly into the greater project and interface with one another. Our project will essentially be a collection of objects that make up our pages written in Swift and managed by Xcode. In this sense, the architecture of our application will be similar to its design. It will be built upon the divide and conquer rule, where larger portions and the project as a whole are formed by partitioning into smaller, more manageable parts.

As part of this divide and conquer style, we will split our team into two teams (this was done based on interest with Jason and Elan on one team and Jack and Mitchel on the other). Team E&J will handle development of the app. Team J&M will handle development and integration of the AWS based server. There will, of course, be some overlap between the teams when helpful or necessary.

1.3. Definitions, Acronyms, and Abbreviations

Swift - Apple's programming language which we will be using.
XCode - An apple produced IDE which is optimized for iOS app production
Page - Our app will consist of multiple screens that accomplish different tasks, these screens will be referred to as the pages of our app.
Travel Buddies - The name of our app. Abbreviated as TB
Trip Page - The Trip Selection and Trip Creation pages.
Trip - A trip will be a custom object which will include information such as date range, type of trip (camping, skiing, vacation, etc.), expected cost, users committed, and number of users desired.
User - A User will be a custom object representing an account that each user creates and can update through their profile.
AWS Mobile Services - Amazon Web Services Mobile. Amazon provides a SDK specifically for iOS development to configure and integrate cloud-based backend servers for mobile applications.

1.4. Behavior

In regards to architecture and as alluded to previously in this document, this project will behave similarly (or as similarly as we can make it) to other apps like it. We plan to implement a simple multi-page structure that can easily be swiped or "pressed" through by the user. The major pages in our application (user account page, trip creation page, and etc.) will be navigated between by the user and will be connected in that regard. The contents of these page will be produced by distinct objects and portions of Swift code, and certain user actions will connect them to one another.

The AWS will handle the storage and access of accounts and trips, most likely as two JSON files. It will be read/writable by the trip pages as well as the user account page. Rather than constantly accessing the server, we will require some form of request from the app (such as opening a new page, hitting a refresh button on a current page, or submitting a creation request).

2. Architecture Views

 2.1. Views

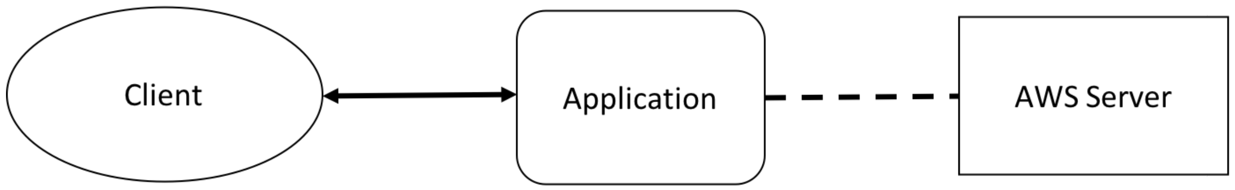
Module View

In the module view, we consider the project as a collection of code units. In a modular view of the architecture of our project, we will most likely consider the different pages of our application to be distinct modules. This provides a fairly straightforward way to consider the different aspects of our project and divide it into code units. The relations or “connections” (not to be confused with those in the component and connector view) between each unit are also code-based, but manifest themselves as buttons or sliders that allow the user to navigate from page to page of the application.

Component and Connector View

A C&C view describes the runtime structure of a system. In this view, components are computational elements and connectors describe how these components interact.

For the purposes of our application, the components would be the client or user, the application itself, and the server if we are able to implement it in the time allotted for the project. As shown in the diagram below, the client will continually interact with the application, and the application would make requests to the server when necessary to keep the user and trip proposal data updated. The client and application will interact on a continuous, 2-way basis, where the client observes changes based on their actions in the application. Interaction with the server will be a request-reply basis where user sends a request through the application to the server. The user will never directly access the server nor will the server request anything of the application.



Allocation View

For our purposes, an allocation view is not overly important. We do not have a very firm grasp of the hardware components and allocation principles of the iPhone. Also, most importantly, Xcode ensures that the code we are writing will run optimally and utilize the device’s memory strategically.

2.2. Element Catalog

Module View

These will be the pages of our application. Each of the four pages mentioned in the rationale will interact in some way with at least one of the others.

Page 1: User Profile:

The User Profile page will interact with the Trip Creation and Selection pages by allowing them to access it. The user profile page will house a custom object, called a User, which holds all of a users public and private information. The trip pages will have access to both public and private information.

Page 2: Trip Creation:

The Trip Creation page will create a custom object, called a Trip, which will then be stored in the Existing Trips page. This means that the Trip Creation page needs access to the Existing Trips page at the time of the actual trip creation.

Page 3: Trip Selection:

The Trip Selection page will allow the user to sign up for a trip. This means that this page must have access to the Existing Trips page. For ease of use, rather than allowing constant access, access will only be allowed upon request. Requests come at three times: 1) When the page is first loaded. 2) When a trip is selected. Or 3) When the page is refreshed.

Page 4: Existing Trips:

This page exists to store the Trips. It will be accessible by the Trip pages but will not have access to any of the other pages. It will be invisible to the User so that they can only make edits to it through the Trip pages.

Component and Connector View

In terms of the component and connector view, there will be three components (one being the server to be implemented if time permits) and two connectors.

Component 1: Client - The user of the travel buddy application. This individual will have downloaded the application to their iphone and most likely created a user account. They will interact with the application by making trip proposals and selecting trips in which to partake.

Component 2: Application - The application will be the actual iOS app written in Swift and developed in Xcode. It's structure will follow that outlined in the modular view in section 2.1.

Component 3: Server - The small-scale server will act as the primary data store of the the app. It will hold user and trip data and will only be accessible through the app, not directly by the user.

Connector 1: Continuous, 2-way interaction between client and application.

Connector 2: Request-reply based relationship between application and server when necessary. Examples of this would be account creation or submittal of trip proposals**.**

3. Across Views Description (needed only if multiple views are included in the previous section.)

   3.1. Views

   Module View - Elements in the module view relate to the components in the "Component and Connector View" by the fact that such elements are the building blocks for some of the components. The application component is built through coding. The architecture of the code is laid out by the module view. Hence, by putting a magnifying glass up to the application component, one will be able to get a view of the "Module View." The "Module View" will also dictate the functional nature of a connector. The 2-way relationship between client and application is established by the functions that the underlying code performs. Any interactions with the server, be it client-side or application-side will be driven by elements established in the module view.

   Component and Connector View- Both the component and connector elements relate to the "Module View," because a zoomed out view of the "Module View" forms the application, which is a component, and all the interrelations that occur as connectors are laid out in the module view. Aspects of the module view sharpen and define connector functions between components.

   Allocation View (not relevant): The client component utilizes the iPhone hardware in order to perform functions that would be defined as connectors. The Allocation View is not relevant to us, despite such, here is a brief description of its relation with the code: The module view seamlessly fits into the allocation view due to the fact that the application will be built using Xcode. Applications written in Xcode are molded by the IDE itself in order to fit the specifications of what is required of an iOS app. This means that the code in the module view is inherently influenced by the hardware.