Winter Progress Report
Capstone Project

Author: Duncan Millard
Sponsor: Nancy Hildebrandt

Instructor: D. Kevin McGrath
CS 462, Winter 2017, Oregon State University

Abstract

The Santiam wagon trail is historic trail located in the Willamette National Forest. The local ranger stations wish to have a mobile app that is capable of taking users on a tour of the wagon trail without needing a connection to the internet. The mobile application come in two forms: one developed for the Android mobile platform, and the other developed for the iOS mobile platform. While these two forms of the mobile application will be developed separately, they will be using the same methods of providing a tour to the user. The mobile application will render a map using a pre-downloaded map tile file and place waypoints onto the map that will be related to relevant information, in the form of videos and text files, about that area of the map. In order to achieve this functionality without internet access, we will rely on pre-downloaded content packs that will contain the map tiles, videos, text files, and waypoint information. These content packages will be created by staff of the local ranger stations, and uploaded via a website that will be developed along with the mobile app. Our team will divide these three larger sections of this project (the Android application, iOS application, and the Web Control panel/backend engineering) between the team members as follows: Android application development: Charles Henninger, Web Control panel/backend engineering: Duncan Millard, iOS application development: Jiawei Liu. This document outlines the possible technologies that will be used to address problems in each of these three major development sections, written by the team member heading each of the sections.

March 24, 2017

Contents

1 Team number and project name

Team Number: 43
Project Name: Santiam Wagon Trail Mobile App

2 Web Content and Server Development

3 Progress and Status

Our progress as a whole has been very reassuring, as we have focused heavily on proof of concepts and ensuring our tools and chosen technologies will properly suit our needs. Our project is implementing MapBox and multimedia content into Mobile Applications that can retrieve downloadable content from a central server via archives internally referred to as "Content Packages". These Packages are designed in such a way that the user should be able to download them ahead of visiting a trail or park and, due to being pre-downloaded, be able to view all available content in that pack without requiring data or WiFi connectivity. Our app will only rely on the phone's GPS signal, which is almost always available except when there is heavy foliage or weather above.

My primary task for this project is designing and implementing our Web Control Panel as well as our Mobile Application API. Our Web Control Panel, herein referred to as our WCP, will provide the publishing mechanism for creating and publishing additional tours and Content Packages to be available for download by the Mobile Applications (Apps). Much of the work this term has been additional research and integration efforts to provide our project with a clean architecture for implementing our solution's features and functionality. Fortunately, this has been mostly achieved.

Our system will be primarily based on ExpressJS, a framework that implements the HTTP(s) protocol and exposes GET/POST request functionality to the developer in the form of callback functions. This allows a developer to very easily add additional HTTP targets or queries. As shown in the snippets section, adding requests is a simple matter of registering a new function with a map abstracted behind a "Router". These Routers are used to group and separate segments of a solution, or to serve different routes/sub-urls. In our solution, we have a router for the API actions "ip/api/functionName", WCP pages, and WCP actions "ip/actions/functionName". This allows us to have entirely separate handlers for these three use cases, while managing it out of the same Express instance. A convenient side effect of this is that they are less prone to race conditions, as Express runs a single-threaded event loop. Except in some asynchronous configurations, calls are processed synchronously which should cut down on database and file system interactions. Eventually, we will possibly (stretch goal?) add SSL support to our system and we may end up implementing NGINX to act as a reverse proxy in that situation. That would fall into end server configuration and not strictly part of our solution, however, as it really handles aspects beyond our scope.

We have encountered some slowdowns and roadblocks along the course of this term with this system. While our team received an Intel NUC to use as a development server, it took us three weeks to properly get it onto the OSU network with the requisite firewall ports opened. That was partially due to needing to get the ports approved as an OSU firewall exception, and heavily due to a misconfiguration that I caused in the local firewall configuration causing it to block ports that had been previously opened locally.

In addition to issues with opening ports, there have been general development slowdowns as ExpressJS and NodeJS are both in active and constant development. NodeJS releases new versions every six months (with the 4.4.3 version as a long term service release) and ExpressJS has several major versions in production. This becomes an issue when looking for examples or other templates in that advice or security concerns may or may not be relevant for the version we are using. This came up frequently with a middleware (extension/library) for Express called BodyParser. BodyParser extracts parameters from the body of HTTP POST requests and exposes it to ExpressJS for easy consumption. The mechanisms for including this tool have changed between Express v2, v3, and the current v4. To make matters more interesting, v4's implementation has some security concerns that forces developers to sideload the middleware, rather than include it directly through Express. To top all of that off, there is an Express v5 in alpha development that may have some early releases later this year.

Lastly, we have had great success with a templating tool called Mustache. Mustache is a tool that allows a developer to write HTML markup and embed placeholders into it that are later replaced with real data. See the snippets section below for an example. This allows us to dynamically serve pages to our users without designing our pages as PHP resources or having to hand-code the dynamic segments into our NodeJS server. Instead, we write generic HTML pages and run them through the Mustache parser engine with additional JSON information to populate the pages. This data can be fed from a database call or any other runtime store. This differs from other templating tools such as Jade in that it looks like otherwise complete HTML, and simply adds to the familiar syntax. Jade requires learning their own implementation of HTML, which would significantly increase our development efforts. Beyond the advantage of not requiring PHP or manual page generation, Mustache also allows us to minimize the JQuery / client-side Javascript running on a user's browser. While we will use JQuery for submitting data to our application, we will not need to request a page's dynamic content after loading the barebones page.

4 Relevant Snippets

4.1 Javascript for Invoking Mustache

```
pageRouter.get('/demo', function(req, res){
    var rData = {elementList: demoData}
    var page = fs.readFileSync('demoPage.html', "utf8");

    var html = mustache.to_html(page, rData);
    res.send(html);
});
```

4.2 Example Mustache Template

```
<!DOCTYPE html>
<html>
    <head>
        <title>Basic Mustache Demo</title>
    </head>

    <body>
        <h2>Mustache Test</h2>
        <ul>
            {{#elementList}}
             <li>{{.}}</li>
            {{/elementList}}
        </ul>
    </body>
</html>
```

4.3 Rendered Mustache Template

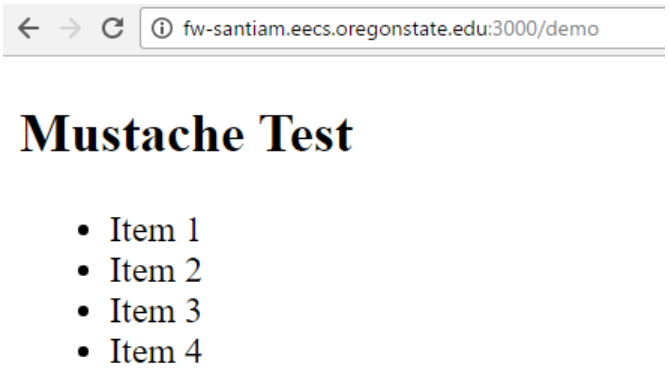

Figure 1: Rendered Mustache

5 Retrospective

Starting in the second week of the term, we designated a three hour time block Thursday to be our long work period each week. This was mostly maintained up until about week 6. At this point, midterms and other projects derailed our weekly meetings and we began working more independently. During this first period, we laid out a graphical overview for our Web Control Panel and Mobile Applications, as well as started development on the Mobile Applications. In week three, I received an Intel NUC and began using this as our server. As part of this, we started the process of getting the appropriate ports opened for use so that our Mobile Applications could properly call home. It took until week six to properly get the ports opened (by OSU and our own firewall). In this time, the Mobile Applications were primarily focused on getting menus and layout features implemented, as well as focusing on getting the first revision of MapBox implemented. By week 6, both Mobile Applications had functional proof of concepts as to using MapBox. From then onwards, server integration and general development were the primary goals. We had a two/three week drop off of productivity, however. This was caused by a combination of midterms/other classes projects, as well as my own distance due to family concerns. As a whole, setting strict goals and deliverables will be paramount to getting to a satisfactory place in our development before Expo. This term has been busy for all of us, and we have also focused on exploratory tasks to ensure we can actually achieve what we were desiring. In that sense, we have been quite successful in proving we can use the technologies we set out to use with minimal disruption.

6 Brief Evaluation

Each team member of our group has a distinct pillar of the project that they are heading. For Jiawei, he is managing the development of the iOS mobile application and is assisting with the design and implementation of the Web Control Panel's (WCP) UI aspects. Charles is the Android stack lead and assisted with paper prototypes and plans for the app and WCP layouts. I have been in charge of configuring our server/software stack as well as developing the WCP backend and app API. Each of these components are mostly standalone and do not have a large amount, if any, of code overlap. In fact, each section has a distinct set of skills.

As a result of each segment being managed individually, if one lags behind, the others feel it quickly. While there were several times when one group member or the other was lacking a feature that we wanted to test, no segment of our solution fell behind enough to prevent progress on other sections. I feel that our group has been very equal in contribution, though it is fair to say Jiawei has been more prompt on documents and feature implementation that either Charles or myself.

With all of that said, our group is highly functional and we get along as a team well. We have active and frequent communication and, come crunch time, all team members are present and active. Seeing several of the other teams fight and complain about their teams makes me appreciate that we get along and interact outside of Capstone as friends.

7 Retrospective Table

Positives	Deltas	Actions
Our team communication is overall very healthy. Communication is rapid and frequent, and is helps that we get along as a social group. It works out very well that each team member is in charge of a separate sub-system. By working on these pieces simultaneously, we are all becoming experts in our particular component, while still having to be familiar with the other components.	Scheduling is probably our biggest issue. Documents are often prepared the day before or the same day, and are almost always finalized the day they are due. While the poster draft was an exception to this, individual components are frequently subject to delays.	We have spoken as a group and plan on aggressively developing during the first weeks of the term, before other classes get too heavy.