The Preliminary Design Document
Capstone Project

Developers: Charles Henninger, Duncan Millard, Jiawei Liu
Sponsor: Nancy Hildebrandt

Instructor: D. Kevin McGrath
CS 461, Fall 2016, 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.

Dec 1, 2016

Contents

1 Introduction

1.1 Purpose

The purpose of this design document is to create a roadmap for our project and describe how we will use different technologies to achieve all of our client's requirements.

1.2 Scope

This project will include a mobile application, the Trail Companion, that will work with custom content packages downloaded in the app. A website will be included for administrators to upload content packages to a server where the application then request and download the packages. The mobile application will provide the user with an interactive map of an area, the user's location, and waypoints on within the area containing information about the park. This application will require no Internet access after downloading the content packages and application itself.

1.3 Summary

The Sweet Home Ranger District of the Willamette National Forest hosts a number of interpretive events throughout the year, which are led by staff experts. These events incur a substantial time and labor cost for their setup and operation, which occupy a significant portion of Ranger Station staff resources. Due to a desire to expand public outreach, volunteers at the Sweet Home Ranger District have proposed the development of a mobile application to showcase one of their most well known trails, the Santiam Wagon Road Trail, with a self-guided tour. The mobile application must be developed with the capability to provide the required services without the need to connect to the internet. The application will be able to provide an interactive map that provides the user with their current location on a trail, as well as waypoints that provide information about points of interest along the trail in the form of videos, audio recordings, and text articles. Ranger Station staff must also be able to design and publish new events through a website with little technical knowledge. Finally, this project must have minimal recurring costs, and must be available for both iOS and Android devices. The relevant viewpoints that we will be concerning with for this project are that of the developers, the administrators, and the users. We will take into account these viewpoints when designing our solutions to each of the requirements, and mold our design in accordance with what is best for each point of view.

2 Glossary

Term	Definition
Admin / Administrator	User who has access to create, modify, and publish content packages.
Android	Mobile operating system developed by Google for use in smartphones and tablets.
Application Program Interface (API)	Interface specifying how software components can integrate with various parts of a project or external resources.
Content Package	Compressed and signed archive containing a map of a region, as well as waypoints and other multimedia resources.
Web Control Panel	Administration website used to create, update, publish, or otherwise control content packages.
Global Positioning System (GPS)	Tool used to provide high-accuracy location services to devices without requiring internet service. Available as long as there is a sufficient view of the sky.
iOS	Mobile operating system developed by Apple for use in Apple mobile products, such as the iPhone, iPad, or iTouch.
Interpretive Event	Educational events hosting exhibits or other multimedia resources.
Mobile Application (app)	A piece of software designed to be run on a particular type of mobile operating system.
Open Street Maps (OSM)	Open/Freely licensed map software for map images, GPS locations, and directions. [7]
POI	Point of Interest
Santiam Wagon Road	A historical site in the Willamette National Forest. [8]
Signing	A process used to ensure software is coming from a known, good source and has not been tampered with. [9]
User	Visitors utilizing the application
USFS	United States Forest Service
Waypoint	Used to mark Points of Interest on maps, which will refer to sites with some form of multimedia content.

3 Body

3.1 Identified Stakeholders

Our primary stakeholders will be the Sweet Home District Ranger Station staff, as they will be the first content curators utilizing our application ecosystem. Our secondary stakeholder is our client, Nancy Hildebrandt, who is our primary point of contact and sponsor.

3.2 Design viewpoint: Context

3.2.1 Map rendering

What must be accomplished in this part of the solution is making a map visible and capable of interaction with the user. We will need to display a relevant map of the tour area, capable of zooming, refocusing on the location of the user, and displaying waypoints on the map that contain relevant information about the area, including videos and text files. In order to get quality offline map tiles that we can use in our application, we have decided to use libraries connected to OpenStreetMaps, a project that provides free geographic data for use in rendering maps. OpenStreetMaps has many libraries that may provide us the means to render maps offline. At this point in time, we are planning on going with our second option, Mapbox GL, for implementation on both the IOS and Android platform. Mapbox GL is a free service that offers high quality rendering and mapping features offline, and brings

with it many useful extra features such as custom waypoint and path editing. Charles Henninger will be leading the development of this piece of the solution

This aspect of our solution will almost entirely be under the viewpoint of information. After this piece is finished, Map Rendering will occur automatically once the User downloads a content package. As stated in another section of our design, the content package will contain a metadata file with information on the map needed for the tour. After receiving the completion signal for the download with the relevant download ID, our app will open the metadata file and pull a few pieces of information from it. We will get two sets of latitude and longitude points, the name of the area, and a list of waypoints, each containing their relevant information (meaning location on the map, identification number, and any video or text information). Using the two sets of latitude and longitude points, we will use the Mapbox library to create a download request for an offline map. The downloading itself will be handled by the Mapbox SDK. We will use the name of the area (found in the metadata file) to label the map. This offline map tile file will be organized and saved within the Mapbox directories, and will be edited via native services in the Mapbox Mapview class. We will create waypoints on our newly downloaded map via this Mapview class, using the waypoint data found in the metadata file. Once the map has been completed, our Mapview object can be placed in and statically manipulated by our UI.

3.3 Design viewpoint: Interface

3.3.1 Android UI

This piece of our solution will be the UI of the mobile app on the Android platform. We plan to conform to the Android UI guidelines for the layout of most of the UI. We have decided to use XML for our UI on our Android platform due to XML's simplicity, clarity, and speed compared to other options like Java or SDL. This section of our solution will be developed by Charles Henninger.

This portion of our application will be subject to the structure viewpoint. Once the app is opened, an XML file will display a menu of the tours currently downloaded will be shown to the User. The available tours will be listed as a list from top to bottom, one column and one tour in each row, ordered alphabetically by the name of the tour area. Swiping left on one of the tours will display an icon for deleting the tour, and an icon to display information on the tour. The information icon will bring up a small window detailing the size of the tour, and the number of waypoints located in the tour. The User can click an icon in the top left corner to display the menu for the app, or click on one of the available tours to display the map and waypoints for that tour. Both the menu button and available tour buttons will be Button XML objects that react with an onclick function. The menu will display buttons that, once clicked, will open the related XML file. Currently we plan on having menu buttons navigating to an available tours page (where the user can download new tours), a credits page (containing information on the developers), and a page displaying tours that have already been downloaded. The page for downloading new tours will be similar to the available tours page, with a list of tours organized by the name of the area they relate to. We will get this list via an API call to our server. Next to each tour will be a download icon that the User can click, if the tour has not already been downloaded, to begin the download for the tour files and the rendering of the map tiles for offline use. Navigating in to an available tour will open an XML file that will display the map of the area, complete with waypoints spread along the route. This map can be navigated by swiping across across the map to pan around the area, and pinching the map to zoom in/out. Clicking on the map will cause the application to check the coordinates of the area clicked. If the area matches a waypoint latitude and longitude values, then a menu will be displayed below the map containing that waypoints information, including videos and text files. Clicking back on the map will close this menu. All of what is described here is completed in XML with buttons and onclick events, with some calls to other sections of the app to get relevant information (such as using the Mapbox element's native functions to cause the map to pan and zoom). Lastly, our credits page will be a simple XML file with text introducing the developers and perhaps a short message to the user.

3.3.2 iOS UI

One of the platforms that we will be designing on for this project is the iOS platform. Due to the nature of this project, the usability of the mobile app is an important part of the project. In this iOS user interface design section,

we will discuss the different options for handling problems such as designing and implementing the UI for selecting content packages, downloading and removing them, as well as basics for tour layout. For solving these problems, I choose Xcode as the solution.

Xcode is an integrated development environment (IDE) that developed by Apple. Developers can use Xcode to develop both iOS and macOS applications. The purpose of using Xcode is to design and implement the UI by a tool or a IDE, Due to the high integration of Xcode, this can bring some convenient and save our time to build the development environment.

In order to use Xcode for Swift, it is necessary in order to get acquainted with Xcode. As Figure 1 show, Xcode has the object library on the right side, and allow developers to drag an object from the library to the application. There are not that many techniques in the UI design section, it basically selects the object from the library and drags it to the design area. It is simple and visible. Xcode is the official IDE for developing iOS application, and is offered for free. Therefore, most iOS applications are developed by Xcode. For example, Safari, eBay, Expedia, etc.

Figure 1: Xcode Interface [1]

When I start to design iOS application UI, I will create a new project in Xcode first. Xcode includes many built-app templates for developing iOS apps, such as tabbed application, game and page-based application. For this project, I will use single view application to start our project. Then, Xcode will show a dialog to let me name the app and choose additional options. After these setting steps, the Xcode will show the design interface which similarity with Figure 1. For our project, I will drag the necessary objects from the library to the application. I will add a few (3 or 4) buttons on the bottom of application to separate functions into different sections. In addition, since we are designing a map application, I will show the map on the home page with user's location and viewpoints. Here is a rough iOS UI for our project.

For design the UI, I will drag a Map View from Object Library to the view controller and use MapKit API display the map with user's location on the home screen, and list the viewpoints under the map. On the bottom of app, there are four (or less) menus to classify all function. For example, since our project need to download content packages for using app without network connection, the downloading/removing content package function will in the setting menu. In a word, I will create iOS app UI by dragging the relevant object from the library to the application to make the UI easy to use and understand.

3.3.3 Web Control Panel UI

What has to be accomplished in this part of the solution is designing and implementing the tour creation. We need to design a website to allow administrators such as our client, Nancy to setup for map frame, click to add waypoints, click waypoints to add texts and videos. In order to control resource on the web, we need to design a dynamic website with a clear UI. Therefore, we are going to use Bootstrap and jQuery to create web control panel UI.

Figure 2: Rough iOS UI

Bootstrap is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first web sites [2]. In addition, Bootstrap is offered for free. Everybody can download and use it. The purpose of using Bootstrap is creating a dynamic website as the web control panel for our client to control online texts and videos resources.

In general, syntax of Bootstrap is similar with HTML technique. However, Bootstrap is more powerful than HTML because it provided more functions such as preprocessors and universal framework. In addition, as Figure 3 show, there are abundant themes available for free. We can apply these themes to create the website with an aesthetically pleasing UI. As the most popular framework for developing websites, there are a number of websites are using Bootstrap, such as NBA.com, Walmart, gliffy, etc.

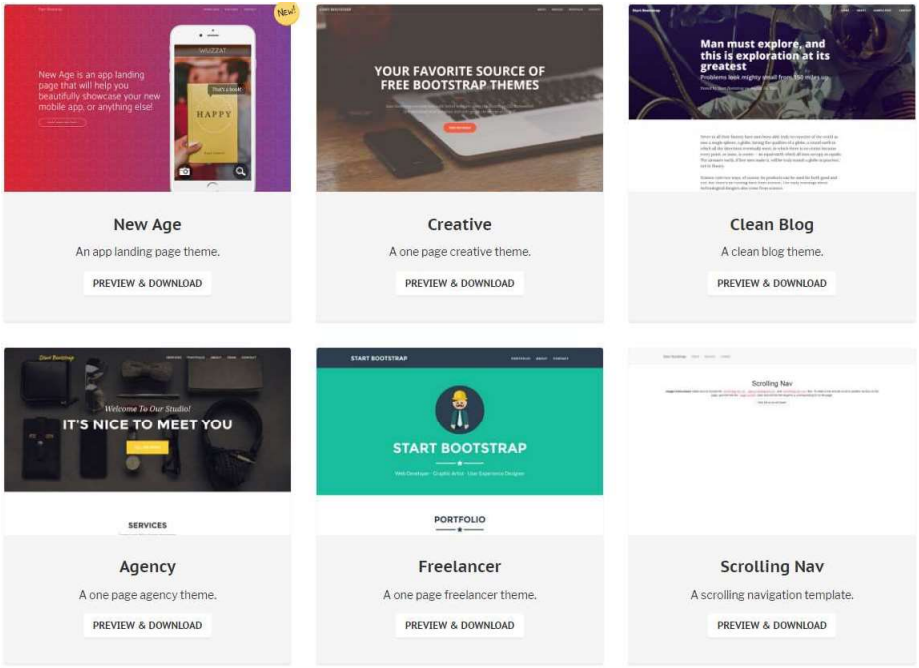

Figure 3: Free Bootstrap Templates [3]

jQuery is a library of JavaScript. The purpose of using jQuery is it easy to learn and apply in the web page.

The jQuery library contains these features, which are HTML/DOM manipulation, CSS manipulation, HTML event methods, Effects and animations, AJAX and Utilities [4]. With jQuery, developers can write less code to achieve the same goal on web pages.

Since jQuery is a JavaScript library and has many features of HTML, it based on JavaScript, HTML and CSS. We can consider jQuery as a highly-integrated development tool to simplify web development process. jQuery is a necessary technology for developing dynamic web pages. Therefore, a lot of famous websites used jQuery, such as Google Maps, Google Doc, Netflix, etc.

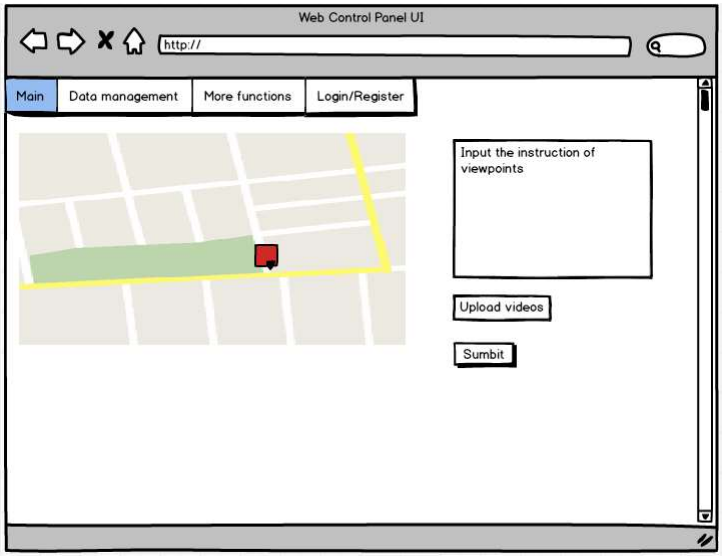

Figure 4: Rough Web Control Panel UI

For design the Web Control Panel UI, I will start with a bare Bootstrap template, which only contains a navigation bar at the top. I provide a rough web control panel UI (Figure 4), the basic operation logic will be similarly with my iOS UI, which is display the map on the main page. After login/register, the administrator will be able to add new viewpoints with text instructions and videos. Once the administrator input all information of the viewpoints, the administrator just click "Submit" button to upload all information into our database. For displaying the map, I will use Bootstrap to leave an area and use jQuery to make the map dynamic and operable. On the navigation bar, I will add a few HTML buttons for providing more functions. For example, a webpage to let administrator modify the viewpoint which already saved in the database. I will use a HTML form to display all saved viewpoints and let the administrator to edit them. In a word, Bootstrap is the main method to design the layout of web control panel UI, and jQuery is the main method to make the map dynamic and operable.

3.3.4 Content Package Server API

In order to successfully communicate between our mobile applications and our Web Control Panel, we will be creating an application program interface (API) to handle both Content Package creation and downloading. These interfaces must be able to fully support both the creation and editing of a Content Package, publishing or hiding packages, as well as any and all functions that the mobile applications will be utilizing to view and download packages. This system will be viewed from an interface viewpoint and will be led by Duncan Millard.

In order to properly manage the database of Content Packages, there will be several functions required. The first of which will be the ability to open a publish request on a particular package. This functionality will hold the content package in a review stage, where it awaits approval from a configurable number of admins. Each reviewer will inspect the content package from within the Web Control Panel UI, and submit their approval or denial. An approval will mark the package and 'ready' in the database, rather than the previous 'pending'. Once a package is marked as 'ready', the original publisher may choose to release it. A single denial will revoke the publish request and force the process to restart. This will take the form of four functions: startPublish, approvePublish, denyPublish, and pushPublish. These functions will be keyed to the specific user performing the action. StartPublish will do the bulk

of the staging. It will wrap up all uploaded files for the Content Package into a staging folder, begin malware scans on the files, and prepare to create a releasable zip file. On top of these it will handle the required database additions to mark the package as pending.

The mobile applications will be heavily reliant on the functionality of the Content Package download server. From an interaction viewpoint, the mobile applications will have several tiers to their requests. The first, and most general, request will be to request a list of all available sites. This will be a list of all parks, districts, or other organizations that are part of the Trail Companion publishing community. From there, when a user asks for a particular site, the applications will request a list of available Content Packages. This request will be expecting a stripped down metadata for each package, consisting of the name, overall description, and filesize. Additional information may be added as it becomes available.

Once the user has narrowed down which content package they want, the API will provide two additional functions. The first will be a complete metadata download, which will provide additional details in a preview form. The second will be a call to initiate a full download. After this download completes, the mobile applications will be able to confirm their download by hitting one last API function and sending up the computed hash of the zip they receive. In the event that this hash passes, a complete download will have taken place and the process can end.

3.4 Design viewpoint: Interaction

3.4.1 Android Remote API Interactions

This piece of our solution involves how we will download our content package files from our server onto the mobile device. Users will view available content packs in app and select which content packs that they want to download. Once a user confirms a download, the mobile app will make an API call to our web server and begin downloading the file. the service that we use for downloading files on the Android platform will be DownloadManager, which is a native Android service that is meant for larger files that require long running downloads and comes with many build in features, such as notifications for completed downloads, that would have to be manually added with other services. This section of our solution will be developed by Charles Henninger.

Our Remote API interactions will on the Android platform (meaning requesting a download for the tour files from our server) will be under an interaction perspective and occur when the User confirms the download of a currently undownloaded tour. Once this happens, we will use the native DownloadManager class to make an HTTP download request to our server. From here, all we have to do is let DownloadManager handle the file download, and indicate an action when the file is completed. This action will initiate the map rendering process to begin, which will use Mapbox and a metadata file located in the newly downloaded tour file to create an offline map.

3.4.2 iOS Remote API Interactions

In this section, we are going to provide a solution that allows users download content packages to their iOS devices. Users also permit to see all available content packages and select the package that they wish to download. After users choose the package, the application start to download the package by making an API call to the web server. To achieve this goal, we have two technologies available, which are WKWebView and NSURLConnection.

WKWebView is a class of iOS to interactive web content, such as for an in-app browser [5]. The purpose of using WKWebView object in our application is interactive with our server and load content package list for users. There is a symbol in WKWebView called loading content. With this symbol, we are able to set the webpage contents and base URL for users, then the user can select and download the package. Here is an example about creating a WKWebView programmatically [5].

```
import UIKit
import WebKit
class ViewController: UIViewController, WKUIDelegate {
```

```
var webView: WKWebView!

override func loadView() {
    let webConfiguration = WKWebViewConfiguration()
    webView = WKWebView(frame: .zero, configuration: webConfiguration)
    webView.uiDelegate = self
    view = webView
}
override func viewDidLoad() {
    super.viewDidLoad()

    let myURL = URL(string: "https://www.apple.com")
    let myRequest = URLRequest(url: myURL!)
    webView.load(myRequest)
}}
```

WKWebView was starting at iOS 8. It is based on iOS operating system. Besides, most iOS web browsers and in-app browsers are using WKWebView. For example, Safari and Twitter's in-app browser. WKWebView is popular and necessary for iOS development.

NSURLConnection is similar with WKWebView. It is also a class iOS. A NSURLConnection object lets you load the contents of a URL by providing a URL request object [6]. Therefore, the purpose of using NSURLConnection is loading content packages from the web server by a URL request.

As a class of iOS, it is built on iOS operating system, and has its own syntax. Most iOS applications are using NSURLConnection to connect to server and load packages. For example, some huge iOS games are required download data packages to the device. These games used NSURLConnection technology. And then, this situation is analogous with downloading content packages to iOS devices.

In our application, we are going to use WKWebView class to display the available package data from our server. First, I will create a new WKWebView object using the init(frame:configuration:) method. And then, I will use the load(_:) method to begin loading web content from our server and display all available content packages. If the user wants to stop loading the page, I will provide a stop button by stopLoading() method to stop loading. As Listing 1 show, I will include my URL in the code and begin to load the content by an API call. In addition, we are going to utilize NSURLConnection technics to download content packages from the web server. I will create a URL connection object with a delegate object that conforms to the NSURLConnectionDelegate and NSURLConnectionDataDelegate protocols. And then, once the user selects the content package, the app will send a request to the server. In detail, the NSURLConnectionDelegate protocol is mainly used for credential handling, but also handles connection completion. connection(_:willSendRequestFor:) is the actual method to send requires. We just replace with our URL in the code, the request will send to our server and start to download the content package.

3.4.3 Operating System Environment

The proposed system has a heavy requirement on a specific software ecosystem that must be maintained. In this part, we have several requirements we must fulfill. From a top level interaction viewpoint, we must satisfy the ability to serve a dynamic web page to our admin users for tour creation, auditing, editing, and publishing. We must also provide a means for the storage of our Content Packages, as well as a gateway system for the mobile applications to call home and request information. All of this must come on top of a single operating system for ease of maintenance and inter-connectivity. Our system configuration tasks will be headed by Duncan Millard.

To satisfy the ability to serve up a dynamic and responsive website, we will be relying on the hosting abilities of Node.JS to serve as our web server [12]. This will handle split duty between serving our Web Control Panel and acting as the primary handler for our Content Package download API. To better suit our needs, we will be taking advantage of the ExpressJS middle-ware that works along with NodeJS to provide robust and modular interactivity with HTTP interfaces [13].

On the same server, we will be hosting a MySQLi database. This database will be accessed through the NodeJS server, and will not have any direct public facing interactions.

As our program will be taking in data from admin users and distributing it to mobile devices, the system must take reasonable steps to ensure that content is not malicious or unknowingly infected with known vulnerabilities. To achieve this, we will be using a NodeJS plug-in that links to ClamAV, which will also need to be installed and configured on our server.

Underlying all of this will be the CentOS operating system. This will provide our application core with a free, well maintained, and stable platform to minimize base layer changes during our application's life cycle. While our product will inevitably go through revisions and our underlying software stack may likely change, CentOS operates on 10 year support cycles, so transitions should be infrequent. As our system is heavily based on other open source projects, we will not be limited by license expiration or upkeep.

To further support migration paths, our software environment will be delivered in the form of an imaged CentOS install. This will provide our team the ability to rapidly deploy our system on a new hardware platform without requiring hand-crafted software environments or tedious manual configuration. While some changes will need to happen, such as host name edits, the primary systems will be configured in an RPM or image format [14].

3.5 Design viewpoint: Structure

3.5.1 Content Package Generation

As the primary method of delivering content to the users, the generation of our system's Content Packages will be critical to the creation of a properly integrated solution. As these Content Packages are, theoretically, a complete record of the information that is necessary for our Mobile Applications to deliver complete guided tour functionality, completeness is key as well as an ability to verify the authenticity of the content. As a result of this, there are several components that must all work together in order to satisfy this feature overall. This feature will be headed by Duncan Millard.

The primary component is deciding on what content should be in a Content Package in the first place. From a structure viewpoint, these Content Packages will contain a manifest file that describes all relevant metadata information for a package. These manifest files will be formatted as JSON files, allowing direct compatibility with the proposed Node.JS web server backend [10]. Each manifest will contain the following attributes: A package name formatted as a string, the name of the park the particular package is part of, a mapName that will be used internally with MapBox in the mobile applications, a pair of latitude/longitude pairs defining the map dimensions for a particular package, a description for the Content Package as a whole, and an array of waypoints. Each waypoint must contain a unique name, an integer for ordering, a coordinate pair for location, an overall description of that particular waypoint, and an assets array. The assets array will contain a listing of any and all video, audio, or text references and files relevant for a particular waypoint.

Along with this metadata manifest, all video/audio/text files that have been uploaded by the content curator and are referenced at a waypoint must be present in the Content Package as well. This content will be used to provide a multimedia experience to a user of the application. This content will be placed into folders sorted by the name of the referencing waypoint. Each waypoint asset field will hold a file path its resources, and these paths must conform to the exact directory that an asset is located within. These assets will be pulled from a temporary staging area on the content server and bundled into a Content Package once the content curator has completed the tour creation screens in the Web Control Panel UI. All assets pulled from this directory should be scanned for known malware to prevent malicious content [11].

To cut down on the required device space and download times, all data inside a Content Package will be compressed using zip compression. Zip compression allows singular files to be decompressed at a later time without decompressing the entire archive, which should function better in the storage constraints of a mobile device.

As verifying the authenticity and completeness of Content Packages is a core requirement, there will be a layer to the Content Package download API that provides verification services. A mobile application will hash the complete zip

archive upon completion of its download and request the expected hash string from the Content Package download server. If there is any discrepancy between the hashes, the download should be marked as a failure and the user will be prompted to redownload the Content Package.

4 Design rationale

This overall system was setup in such a way that any and all content required for a complete guided tour would be preloaded onto a user's device. This was done in order to allow for users to visit locations without access to the Internet and still be able to access the tour content. All systems and libraries in use through this solution are free of charge for the scale that is expected, and do not rely on paid subscription services, as this will be a minimal budget application.

5 Conclusion

The unifying feature of each of our individual design pieces was the need for clarity and ease of use. This app will be used and maintained predominantly by people that have little to no developmental experience. Because of this, we must focus our designs to be simple to understand and use.

6 Gantt Chart

References

[1] "Apple," *Interface Builder Built-In.* [Online]. Available: `https://developer.apple.com/xcode/interface-builder/` [Accessed: 14-Nov-2016].

[2] "w3schools," *Bootstrap 3 Tutorial.* [Online]. Available: `http://www.w3schools.com/bootstrap/` [Accessed: 14-Nov-2016].

[3] "Start Bootstrap," *All Templates.* [Online]. Available: `https://startbootstrap.com/template-categories/all/` [Accessed: 14-Nov-2016].

[4] "w3schools," *jQuery Introduction.* [Online]. Available: `http://www.w3schools.com/jquery/jquery_intro.asp` [Accessed: 14-Nov-2016].

[5] "Apple Developer," *WKWebView.* [Online]. Available: `https://developer.apple.com/reference/webkit/wkwebview` [Accessed: 14-Nov-2016].

[6] "Apple Developer," *NSURLConnection.* [Online]. Available: `https://developer.apple.com/reference/foundation/nsurlconnection` [Accessed: 14-Nov-2016].

[7] "OpenStreetMap," *OpenStreetMap.* [Online]. Available: https://www.openstreetmap.org/about. [Accessed: 04-Nov-2016].

[8] G. Harrison, "Santiam Wagon Road," *The Oregon Encyclopedia.* [Online]. Available: https://oregonencyclopedia.org/articles/santiam_wagon_road/. [Accessed: 04-Nov-2016].

[9] "Introduction to Code Signing," *Microsoft Developer Network.* [Online]. Available: https://msdn.microsoft.com/en-us/library/ms537361.aspx. [Accessed: 04-Nov-2016].

[10] "JSON.parse()," *Mozilla Developer Network.* [Online]. Available: https://developer.mozilla.org/enUS/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse. [Accessed: 01-Dec-2016].

[11] K. Farris, "Clamscan," *Node Package Manager.* [Online]. Available: https://www.npmjs.com/package/clamscan. [Accessed: 01-Dec-2016].

[12] "Node.js" *Node.* [Online]. Available: https://nodejs.org/en/. [Accessed: 01-Dec-2016].

[13] "Express - Node.js web application framework," *Express JS.* [Online]. Available: http://expressjs.com/ [Accessed: 01-Dec-2016].

[14] "RPM Package Manager," *RPM.* [Online]. Available: http://rpm.org/. [Accessed: 01-Dec-2016].

Sponsor

Developer

Developer

Developer