Spring Midterm Progress Report
Capstone Project

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

Instructor:
Kevin McGrath
Kirsten Winters

CS 463, Spring 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.

May 18, 2017

Contents

1 Introduction

1.1 Project purposes and goals

This product 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.

2 Jiawei Liu's Section

2.1 Current Progress

For iOS app development, I finished almost all functions that we need, such as map rendering, display waypoints, iOS UI design, download files from the server, download offline maps, setting page and about page.

For the iOS app, I used the "Tab Bars" style as the default app UI. There are four main pages of the iOS app, which are "My Tours", "Discover", "Setting" and "About". On the "My Tours" page, I use a table view object to contain all waypoints, then the app will display a list of available waypoints that include name, picture and description. Users can go through the waypoint list and click the waypoints to learn more details. On the "Discover" page, I use Mapbox framework as the map provider. When the user looks at the map, the map will displace user's location with a shining blue spot and waypoints with red pin symbol. When the user click the red pin symbol, it will show the waypoint's name, the user can back to Discover page to see more details about this waypoint. In addition, I set the map zoom level at 13, when the user looking at the map, the app will download the offline map with zoom level 16 automatically. When the offline map download complete, the app will have an alert window to tell user offline map download successfully and how many data used. On the "Setting" page, users are able to change the font size by clicking a button. iOS provide a font sizing technique called Dynamic Type. When users click the font size button, it will bring users to the iOS setting app, then users can find Text Size option under the Accessibility and General option. In addition, if our server provides more accessible regions and content packages, they will also appear on the setting page, each region will come with a download button, users can click the download button and download the content package from our server. On the "About" page, it is quite simple, it just provides the app version and our credits information.

2.2 Rest Problem

Currently, the iOS app implements almost all features that we need, the only left thing is synchronization. Since my group member was working on the server development, and the server and tours information were unstable. Thus, I didn't allow the iOS app read the waypoint information from our server directly, I use the local file and data to test and display waypoints. But I already prepared download and parse content package functions. Therefore, when the server works stable, I will implement download and parse content package, then display all available waypoints on the iOS app by end of this term.

2.3 Problems and Solutions

During the iOS app development, the major problem is nobody familiar with iOS development, and OSU doesn't teach iOS development, I went to Rob Hess's office hour, the instructor of CS496 Mobile Software Develop, but he didn't have experience about iOS development. And then, iOS using Swift to develop apps, so I need to learn a brand-new language to develop the iOS app. Our group decides to use Android style slider menu as our app's UI,

but this is not the default iOS UI, I will include extra style file to achieve it. Due to the limitation of Apple, Xcode doesn't like Android Studio, it only works on Mac OS, but the other two group members are using windows laptop, so they cannot run Xcode on their laptop and provide some help. For the map, since most map providers charge for using offline map, our team also need to do some research for the map provider to avoid charging.

For solving above problems, I start to learn how to develop iOS app on the Internet. Since I am the only person who own a Mac device in our group, I decided to work on iOS app development individual. There are some good tutorials on YouTube and Apple developer website to help me begin this project, I also download a book called "Beginning iOS Programming with Swift". I used these books and tutorials to learn iOS development by myself, so it helped me a lot to solved the problem that nobody familiar with iOS development. For learning swift, I found a website called "runoob.com", this website provides a good tutorial for swift, I learn the basic knowledge of swift programming during the winter break. For the UI, since Xcode didn't include this UI template, so I google how to design slide bar in iOS, and I found a great tutorial about how to create slide menu on iOS, which is "ashishkakkad.com". Then I use this tutorial as the template to develop our project. However, I was try to contain a table view and map view in a single page, but it is failed. After some research, I can implement it by using subclass method. But I only found an Object-C example and my project is using Swift. Therefore, I couldn't implement this feature, and I used the regular iOS Tab Bars style instead Android slider bar style. I re-designed the iOS app, then let waypoints and maps have separate pages. This change brings some new challenges to solve and I spent a lot of time to learn that. For the map provide, Since Mapbox can provide the free offline map and the customized map. We choose Mapbox as the map provider and the default map framework. In addition, Mapbox has a detailed tutorial to teach me how to include the map in the iOS app and display waypoints.

2.4 Swift Code of Download Offline Map

```
import UIKit
import Mapbox

class MyCustomPointAnnotation: MGLPointAnnotation {
    var willUseImage: Bool = false
}

class DiscoverVC: UIViewController, MGLMapViewDelegate {
    @IBOutlet weak var mapview: MGLMapView!
    var mapView: MGLMapView!
    var progressView: UIProgressView!

    override func viewDidLoad() {
        super.viewDidLoad()

        mapview.delegate = self

        let pointA = MyCustomPointAnnotation()
        pointA.coordinate = CLLocationCoordinate2D(latitude: 44.564178, longitude: -123.279444)
        pointA.title = "OSU Campus"
        //pointA.subtitle = "Corvallis, OR"

        let pointB = MyCustomPointAnnotation()
        pointB.coordinate = CLLocationCoordinate2D(latitude: 44.5535883, longitude: -123.2726553)
        pointB.title = "Avery Park and Natural Area"
        //pointB.subtitle = "Corvallis, OR"

        let pointC = MyCustomPointAnnotation()
        pointC.coordinate = CLLocationCoordinate2D(latitude: 44.5595762, longitude: -123.2814208)
        pointC.title = "Reser Stadium"
        //pointC.subtitle = "Corvallis, OR"
```

```
    let pointD = MyCustomPointAnnotation()
    pointD.coordinate = CLLocationCoordinate2D(latitude: 44.5652979, longitude: -123.2760134)
    pointD.title = "The Valley Library"
    //pointA.subtitle = "Corvallis, OR"

    let myPlaces = [pointA, pointB, pointC, pointD]
    mapview.addAnnotations(myPlaces)

    mapview.setCenter(CLLocationCoordinate2D(latitude: 44.564174, longitude: -123.279306), zoomLevel: 1
    mapview.userTrackingMode = .follow

    NotificationCenter.default.addObserver(self, selector: #selector(offlinePackProgressDidChange), nam
    NotificationCenter.default.addObserver(self, selector: #selector(offlinePackDidReceiveError), name:
    NotificationCenter.default.addObserver(self, selector: #selector(offlinePackDidReceiveMaximumAllowe

}

// Note: You can remove this method, which lets you customize low-memory behavior.
override func didReceiveMemoryWarning() {
    super.didReceiveMemoryWarning()
    // Dispose of any resources that can be recreated.
}

func mapView(mapView: MGLMapView, annotationCanShowCallout annotation: MGLAnnotation) -> Bool {
    // Always try to show a callout when an annotation is tapped.
    return true
}

// Or, if you're using Swift 3 in Xcode 8.0, be sure to add an underscore before the method parameters:
func mapView(_ mapView: MGLMapView, annotationCanShowCallout annotation: MGLAnnotation) -> Bool {
    // Always try to show a callout when an annotation is tapped.
    return true
}

//*******************
//offline map
//*******************
func mapViewDidFinishLoadingMap(_ mapView: MGLMapView) {
    // Start downloading tiles and resources for z13-16.
    startOfflinePackDownload()
}

deinit {
    // Remove offline pack observers.
    NotificationCenter.default.removeObserver(self)
}

func startOfflinePackDownload() {
    // Create a region that includes the current viewport and any tiles needed to view it when zoomed f
    // Because tile count grows exponentially with the maximum zoom level, you should be conservative u
    let region = MGLTilePyramidOfflineRegion(styleURL: mapview.styleURL, bounds: mapview.visibleCoordin

    // Store some data for identification purposes alongside the downloaded resources.
    let userInfo = ["name": "My Offline Pack"]
    let context = NSKeyedArchiver.archivedData(withRootObject: userInfo)

    // Create and register an offline pack with the shared offline storage object.
```

```
    MGLOfflineStorage.shared().addPack(for: region, withContext: context) { (pack, error) in
        guard error == nil else {
            // The pack couldn't be created for some reason.
            print("Error: \(error?.localizedDescription ?? "unknown error")")
            return
        }

        // Start downloading.
        pack!.resume()
    }

}

// MARK: - MGLOfflinePack notification handlers

func offlinePackProgressDidChange(notification: NSNotification) {
    // Get the offline pack this notification is regarding,
    // and the associated user info for the pack; in this case, `name = My Offline Pack`
    if let pack = notification.object as? MGLOfflinePack,
        let userInfo = NSKeyedUnarchiver.unarchiveObject(with: pack.context) as? [String: String] {
        let progress = pack.progress
        // or notification.userInfo![MGLOfflinePackProgressUserInfoKey]!.MGLOfflinePackProgressValue
        let completedResources = progress.countOfResourcesCompleted
        let expectedResources = progress.countOfResourcesExpected

        // Calculate current progress percentage.
        let progressPercentage = Float(completedResources) / Float(expectedResources)


        // If this pack has finished, print its size and resource count.
        if completedResources == expectedResources {
            let byteCount = ByteCountFormatter.string(fromByteCount: Int64(pack.progress.countOfBytesCo
            print("Offline pack \\(userInfo["name"] ?? "unknown")" completed: \(byteCount), \(completed
            // create the alert
            let alert = UIAlertController(title: "Offline Map", message: "\(byteCount) download sucessf
            // add an action (button)
            alert.addAction(UIAlertAction(title: "OK", style: UIAlertActionStyle.default, handler: nil)
            // show the alert
            self.present(alert, animated: true, completion: nil)

        } else {
            // Otherwise, print download/verification progress.
            print("Offline pack \\(userInfo["name"] ?? "unknown")" has \(completedResources) of \(expec
        }
    }
}

func offlinePackDidReceiveError(notification: NSNotification) {
    if let pack = notification.object as? MGLOfflinePack,
        let userInfo = NSKeyedUnarchiver.unarchiveObject(with: pack.context) as? [String: String],
        let error = notification.userInfo?[MGLOfflinePackUserInfoKey.error] as? NSError {
        print("Offline pack \\(userInfo["name"] ?? "unknown")" received error: \(error.localizedFailure
    }
}

func offlinePackDidReceiveMaximumAllowedMapboxTiles(notification: NSNotification) {
```

```
        if let pack = notification.object as? MGLOfflinePack,
            let userInfo = NSKeyedUnarchiver.unarchiveObject(with: pack.context) as? [String: String],
            let maximumCount = (notification.userInfo?[MGLOfflinePackUserInfoKey.maximumCount] as AnyObject
            print("Offline pack \\(userInfo["name"] ?? "unknown")" reached limit of \(maximumCount) tiles."
        }
    }
}
```

3 Charles Henninger's Section

3.1 Android App Development

Currently the android Trail Companion application has nearly complete functionality. The application can download tour files (in the form of zip files) from our server, unzip the files, use the meta-file to download an offline map area, and add waypoints containing the assets (images and videos) included in the tour file. I've added some basic texturing and style, just because, and the application can display user location, all relevant waypoints for a specific tour including the waypoint title when the user clicks the waypoint. I feel confident that I have build this app to be on par or superior to the design that we decided upon during the first few months of development, and that I have very little functionality left to add to have the application meet and exceed our client's expectations.

3.2 Dynamic Waypoint List

Creating a dynamic list of tour waypoints, which included the waypoint title, description, and any asset files associated with that application, posed a lot of problems for me during this terms development. The main problem was creating such a complex ListView that could be altered and customized based on the dictates of the meta-file. After talking with an OSU professor, Rob Hess, I went with the RecycleView class instead of a normal ListView. RecycleView allows me to add a dynamic list of individual views, instead of just a list of valuesl. Using this, I can create a waypoint view, containing views for any/all possible asset file types that a waypoint could have. Any view that I don't use in this waypoint view can be made invisible by setting the views Visibility parameter to GONE.

```
public class MyToursAdapter extends RecyclerView.Adapter<MyToursAdapter.WaypointItemHolder> {

    private ArrayList<String> titles;
    private ArrayList<String> descriptions;
    private ArrayList<String> ids;
    private MyToursAdapter.OnImageclickedlistener clicklistener;
    private MyToursAdapter.OnVidclickedlistener clicklistener1;
    Context mcontext;

    //just for exp
    private ArrayList<String> assets;

    public MyToursAdapter(Context thiscontext){
        titles = new ArrayList<>();
        descriptions = new ArrayList<>();
        ids = new ArrayList<>();
        mcontext = thiscontext;

        //just for expo
        assets=new ArrayList<>();
```

```
}
//add more params and lists to create more complex recycle views
public void addWaypoint(String id, String title, String description,String expoasset) {
    titles.add(title);
    descriptions.add(description);
    ids.add(id);

    //just for expo
    assets.add(expoasset);
    notifyDataSetChanged();
}

@Override
public int getItemCount() {
    return titles.size();
}

@Override
public WaypointItemHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    LayoutInflater inflater = LayoutInflater.from(parent.getContext());
    View view = inflater.inflate(R.layout.waypoint_list_item, parent, false);
    WaypointItemHolder viewHolder = new WaypointItemHolder(view);
    return viewHolder;
}

class WaypointItemHolder extends RecyclerView.ViewHolder {
    private TextView waypointTitleView;
    private TextView waypointDescView;
    public ImageButton viewPicButton;
    public ImageButton viewVidButton;
    public ImageView imageview;



    public WaypointItemHolder(View itemView) {
        super(itemView);
        waypointTitleView = (TextView)itemView.findViewById(R.id.waypoint_name);
        waypointDescView = (TextView)itemView.findViewById(R.id.waypoint_desc);
        imageview = (ImageView) itemView.findViewById(R.id.waypoint_pic);
        viewPicButton = (ImageButton) itemView.findViewById(R.id.pics_button);
        viewVidButton = (ImageButton) itemView.findViewById(R.id.video_button);
    }

    public void bind(String title, String desc) {
        waypointTitleView.setText(title);
        waypointDescView.setText(desc);
    }
}

public interface OnImageclickedlistener {
    void imageclicked(ImageView picview, Context mcontext);
}

public void setImageclickedlistener(OnImageclickedlistener l) {
    clicklistener = l;
}
```

```
    public interface OnVidclickedlistener {
        void vidclicked(String id, Context mcontext);
    }

    public void setVidclickedlistener(OnVidclickedlistener l) {
        clicklistener1 = l;
    }

    @Override
    public void onBindViewHolder(final WaypointItemHolder holder, final int position) {
        holder.bind(titles.get(titles.size() - position - 1), descriptions.get(descriptions.size() - positi
        holder.buttonbind(ids.get(titles.size() - position - 1));

        holder.viewVidButton.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View view){
                if(clicklistener1 != null){
                    clicklistener1.vidclicked(ids.get(titles.size() - position - 1),mcontext);
                }
            }
        });

        holder.imageview.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View view){
                if(clicklistener != null){
                    clicklistener.imageclicked(holder.imageview,mcontext);
                }
            }
        });
    }

}
```

4 Further Work

All that needs to be done for the Android application is the addition of audio files for the Waypoint assets view, and to iron out some storage issues that I have had with saving and reading large amounts of files from the Android application's internal storage. While the application works in its current state, changing this so that all files relevant to the functioning of the app are contained in the application's internal storage is necessary if we want to eventually put this application on the Google Play store.

5 Duncan Millard's Section

5.1 Web Control Panel Status

At the time of this report, the Web Control Panel is nominally functional. Functionality to create and publish tours has been added and is functional. The tours are added through a dynamic HTML form in the Web Control Panel (WCP). This form is submitted to the "Action" router embedded in our Node server. Once this "Action" router received the submission, it handles splitting the tour's information into the Tour table of our MySQL database, as

well as uploading the assets needed for the tour. These assets are stored into a staging folder that is uniquely keyed off of a hash that is based on the tours name, description, and waypoint information. Once the upload is complete and the MySQL returns without error, it is pulled from the staging folder and loaded into a fileVault directory for permanent storage. Theoretically, in the future, tours in this section will be editable through form access. This would be done to add additional information, or to edit existing information and roll its version number. Currently, editing is not supported. The API calls needed by the Mobile Applications are functional and can provide a JSON listing of all available tours. In addition, a tour's zip archive may be downloaded based off of a tourID value that is found within the prior information request.

5.2 Problems Encountered

This stage of our project ran into two primary issues for me. The first was dealing with Node's use of callbacks for every operation that was not near instantaneous. Given Node is primarily single threaded for requests, having it wait while MySQL runs a potentially time intensive operation would significantly cut into its ability to serve other requests in the meantime. As a result, all function calls that rely on a potentially slow operation are pushed to an event loop running in the background. Designing my code to consider this an advantage rather than a limitation took a great deal of adjusting and experimenting in order to make it readable and functional.

Second, I was unable to attend the Engineering Expo due to illness that resulted in multiple hospital visits. Fortunately, my team was able to still pull together a proper demonstration of our project and represent us at the Expo.

6 Conclusion

At this point we have achieved all goals discussed in our Client Requirements Document, and the only work that needs to be done involves polishing the look and function of the application. Another big step that will let us know whether or not our current application is up to par will be expo, where our client will see the fully functioning application and provide us with feedback.