FOSYA^2

The Katy Trail App v1.0

Presented by	
Fredo Derazin	Team lead
Oliver Johnson	Lead programmer
Shraddha Belbase	Database Developer
Yanni Patel	Database Developer
Anna George	Developer
Anna Lechner	UI Designer

Table of Contents

Revision History

Description	Author	Date
Initialized document	Fredo Derazin	Monday January 27, 2020
Added/edited project scope	Fredo Derazin	Wednesday January 29, 2020
Added/edited project scope	Fredo Derazin	Wednesday January 29, 2020
Updated section Project Stakeholders	Fredo Derazin	April 14, 2020
Updated template content under Use case diagram section	Fredo Derazin	April 16, 2020
Updated section Project Stakeholders	Fredo Derazin	April 17, 2020
Updated template content under User Story section	Fredo Derazin	April 22, 2020
Updated sections: • Template structure • Team roles • App version • Risks and constraints	Fredo Derazin	April 24, 2020
Updated section Risk and constraints	Fredo Derazin	April 30, 2020
Added usage, revised key features from README file	Anna George	Thursday April 30, 2020
Added installation instruction	Oliver Johnson	Thursday April 30, 2020
Updated Out of Scope	Anna George	Sunday May 3, 2020
Added updated use case diagram	Anna George	Monday May 4, 2020
Added requirements	Anna George	Monday May 4, 2020
Section for retrospective; needs to be updated	Anna George	Tuesday May 5, 2020
Moved project scope to top, revised sections under scope	Anna George	Tuesday May 5, 2020
Updated project features	Anna George	Wednesday May 6, 2020
Added Covid-19 pandemic to risks & constraints, updated info on push notifications	Anna George	Wednesday May 6, 2020
Added Android and iOS build for deployment from README file	Anna George	Thursday May 7, 2020
“Tidied” up the document (fixed spelling/grammar errors, different font sizes, spacing, etc.)	Anna George	Thursday May 7, 2020
Wrote user stories	Anna George	Thursday May 7, 2020
Updated Out of Scope	Oliver Johnson	Thursday May 7, 2020
Updated Retrospective	Oliver Johnson	Thursday May 7, 2020
Updated sections: • Added Project Timeline sections	Fredo Derazin	Thursday May 7, 2020

• Added Burndown chart sections • Updated Features & Requirements section title		
Adding Location Data to the Database	Anna George	Thursday May 7, 2020
Specification in running Android Emulator on Windows	Nirali Patel	Friday May 8, 2020
• Elaborated on items under Risks and Constraints section • Updated Retrospective section • Updated Table of Contents page	Fredo Derazin	Friday May 8, 2020
• Added Technical Guide, added database section • Fixed Table of Contents	Anna George	Friday May 8, 2020
Added Coding Standards	Oliver Johnson	Friday May 8, 2020

Project Statement

Introduction

The Katy Trail's open space is getting smaller as new construction and development projects emerge. Every year, a part of the historical trail is lost. To help preserve the Katy Trail, Dr. Smith and colleagues requested the development of a phone application. The Katy Trail app is intended to be available for Android and iOS phones. See non-functional requirements section for more details.

Project Stakeholders

Full Name	Phone#	Email	Role
Fredo Derazin	636-734-9497	fredoderazin@gmail.com	Team lead
Oliver Johnson	314-369-1887	oa_johnson@outlook.com	Lead programmer
Shraddha Belbase	636-493-5323	Shraddha.belbase1@gmail.com	Database programmer
Nirali Patel	314-201-9115	Nilupatel1996@gmail.com	Database programmer
Anna George	314-714-8396	anna.m.george@outlook.com	Programmer
Anna Rodrigues	281-508-1855	annalech@hotmail.com	UI/UX analyst
Dr. Jeffrey Smith	x4991	jsmith@lindenwood.edu	Product owner
Stephen Blythe		Sblythe@lindenwood.edu	Director

Project Scope

Business Requirements

1. As a user, I want to explore the Katy Trail through the application
2. As a Product Owner, I want to add more locations to the app

Product Scope and Features

1. Home page with map, explore, and about views
2. See Katy Trail locations on map view
3. See Katy Trail locations on explorer view
4. See brief summary of Katy Trail app on about view
5. Bookmark locations / remove locations from bookmark view
6. Geographical map view
7. GPS capability
8. Bookmarking capability
9. Site explorer
10. Alert and push notification when nearing a location
11. Display sites' historical information, including pictures and text
12. See app analytics, such as number of installs and uninstalls, app ratings, and crash reports
13. Compatible with Android
14. Compatible with iOS

Out of Scope

1. Publish the Katy Trail app on Google Play and the Apple Store
2. Adding Katy Trail locations to app after May 4th
3. App maintenance after May 4th (fixing future bugs, adding more features, etc.)

Risks and Constraints

We identified many activities as a risk in the beginning, but the most pervasive items were:

1. Time

   Given the scope of the project, just one school semester was not enough time to get everything done, as described in requirements under Functional Requirements section. We assigned programming-related tasks to team members who were more comfortable with the new technology.

2. Covid-19 pandemic

   The sudden change in the way we met, worked, and collaborated created a considerable amount of slippage time as we attempted to find alternatives. We used a free Zoom account where we met online twice a week.

3. Learning new technology (Flutter/Dart/Firebase)

   We collectively agreed to learn new technology, but the learning curve was steep, and we had to pivot to deliver on all requirements.

4. Database integration (Firebase)

   Initially, we decided to integrate Firebase as database system, but we eventually had to pivot away from database integration altogether because of constraints of cost, time, and effectiveness. The contents are hardcoded in the source code in the app.

5. App notification feature

   Given the local government's order, we were not able to test on the Katy Trail. We set up virtual mobile capability to test notifications when approaching a Katy Trail location that is in the app.

6. System report & analytics

   IOS: Usability metrics will be available when the app is live on public domain in iOS App Store.

   Android: Usability metrics will be available when the app is live on public domain on Google Play Store.

Functional Requirements

1. The system shall allow users to visualize the Katy Trail iconic locations displayed on a geographical map.
2. The system shall have the capability to display location information in pop up view when user taps over a location on the map.
3. The system shall allow users to bookmark a location by tapping bookmark from geographical map view.
4. The system shall allow users to visualize a location from bookmark view by tapping on bookmark menu option.
5. The system shall allow users to read more about a location on map view by tapping on location marker.
6. The system shall have the capability to display in app notification when user is physically approaching a location proximity.

User Manual

Getting Started

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes. See deployment for notes on how to build the project so it can be published onto the Google Play Store or Apple App Store.

Hardware Requirements

If you plan to run this app on a real mobile device, here are the hardware requirements:

- iPhone running iOS 10 or later
- Android 4.0 or later

Prerequisites

An installation of the Flutter framework and Dart SDK is required. Instructions to installing Flutter can be found here.

To test the app on Android, you must also install the Android SDK and Android Studio. Installing Android Studio should also install the Android SDK, which you can find the installer here.

Note: If your computer is using a Ryzen AMD processor and you wish to use an emulator, you must download the Beta version of Android Studio instead which can be found here.

If you wish to test the app on iOS, you'll need to install Xcode from the App Store.

If you plan to use an Android emulator to test the app, ensure that your PC has virtualization enabled.

Installing

Clone or download the repository onto your local hard drive. You may download the repo as a .zip file from the download button above, or use this command onto your bash terminal or git bash:

git clone https://github.com/oliverj96/katy_trail_app

Once the repo is downloaded, launch the terminal (on MacOS or Linux) or command prompt (on Windows) and navigate to that directory.

Next, install the app's dependencies by running:

```
flutter pub get
```

Running the Tests

To run the Flutter app, execute:

```
flutter run
```

In the root directory of the repository.

Build for Deployment

Here are the instructions to build the app for each platform for deployment.

Android

1. Open the "android" folder (under the repo root directory) as a project in Android Studio.

2. On the menu bar, click Build > Generate Signed Bundle / APK
3. Select APK and click Next

If you are creating the app bundle to upload to Google Play store, select Android App Bundle instead

4. Specify a key store path

If you do not have one, generate a new one, and keep note of the credentials you use

5. Make sure credentials are filled out, then click Next
6. Under Build Variants, select "release" and check the box "V2 (Full APK Signature)"then click Finish

The App should be saved under ../katy_trail_app/android/app/release/app-release.apk

iOS

1. Open the "ios" folder (under the repo root directory) as a project in Xcode

Once your project is open, verify that your Apple Developer account is linked to Xcode

2. On the menu bar, select Product > Destination > Generic iOS device. Wait for the build to complete
3. On the menu bar, select Product > Scheme, and make sure the last option is checked
4. On the menu bar, select Product > Archive
5. On the new window select the project you want then distribute app
6. From there, you can select one of the four options that will better suit your needs
7. Follow the on-screen options. The last option will allow you to specify the save location for the build file

Key Features

1. Easy navigation
2. Geolocator
3. Push notifications
4. Maps integration
5. Android and iOS compatibility
6. Simple design and color scheme

Usage

A user must tap the app icon on their phone's home screen to open the app. The app opens to a home page with three navigation options: Map, Explore, and About. There is also a bookmark icon on the blue title bar in the upper-right hand corner that leads to a bookmark page.

Map Page

1. Shows map of St. Charles with red location icons at different points on the Katy Trail
2. Ability to zoom in and out on map by pinching together two fingers or spreading two fingers apart on phone screen
3. Ability to move around on map by placing a finger on phone screen and swiping in any direction

4. Tapping a red location pops up a location card with that specific location's name, a short description, and two side-by-side texts that say "Learn" and "Bookmark"
5. Tapping "Learn" on a location card navigates to a page with that location's name on the top blue title bar, one or more images of the location, and a historical description of the location
6. Tapping "Bookmark" moves the location data (location name, image/s, and description) to the user's bookmarks and changes "Bookmark" to "Remove"
7. Tapping "Remove" removes the location data from the user's bookmarks and changes "Remove" to "Bookmark"
8. Tapping anywhere outside the location card makes the location card disappear
9. Tapping the bookmarks icon on the blue title bar in the upper-right hand corner navigates to the bookmark page

Explore Page

1. Shows all locations on the Katy Trail that have been added to the app; each location appears on a separate card with the location's name, a short description, and two side-by-side texts that say "Learn" and "Bookmark"
2. Ability to scroll up and down by swiping a finger up and down on phone's screen
3. Tapping "Learn" on a location card navigates to a page with that location's name on the top blue title bar, one or more images of the location, and a historical description of the location
4. Tapping "Bookmark" moves the location data (location name, image/s, and description) to the user's bookmarks and changes "Bookmark" to "Remove"
5. Tapping "Remove" removes the location data from the user's bookmarks and changes "Remove" to "Bookmark"
6. Tapping the bookmarks icon on the blue title bar in the upper-right hand corner navigates to the bookmark page

About Page

1. Shows an image of the Katy Trail, the Lindenwood University logo, and the Magnificent Missouri logo
2. Has a brief welcome message and summary of the app
3. Tapping the bookmarks icon on the blue title bar in the upper-right hand corner navigates to the bookmark page

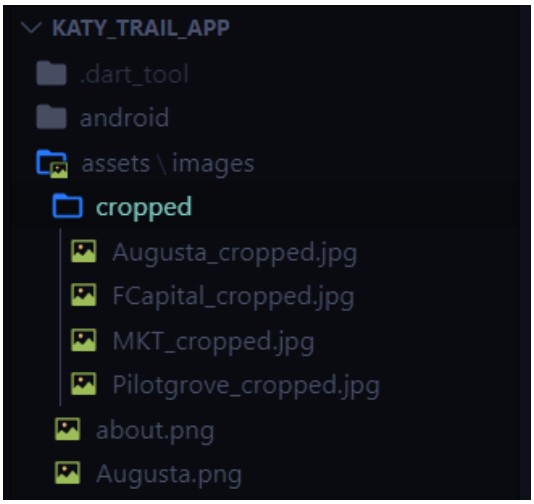
Bookmark Page

1. Shows all locations that user has bookmarked on separate cards with the location's name, a short description, and two side-by-side texts that say "Learn" and "Remove"
2. Ability to scroll up and down by swiping a finger up and down on phone's screen (given that there are enough bookmarked locations to scroll down)
3. Tapping "Learn" on a location card navigates to a page with that location's name on the top blue title bar, one or more images of the location, and a historical description of the location
4. Tapping "Remove" removes the location data from the bookmarks page so the location card disappears

Adding Location Data to Database

Since more locations will be added to the app in the future, we have created a single file that will be used as a database. This file is called “location.dart”, and it is located under folder “lib” in the Katy Trail app folder.

First, this is how images must be added so they can later be added to the app’s database. Under the Katy Trail app folder, find the assets folder and click on it. Then, navigate to the images folder in the assets folder. There will be one last folder in this called “cropped”. This folder will only hold images that are cropped into a square, and there should only be one of these images per location. The rest of the images folder contains all other images in the app.
To add an image, simply drag a downloaded image into the correct folder. You can do this in file explorer, or you may do this in a developer's environment (Android Studio or Xcode).



After the image is added here, there is one final step for adding images.
In your developer’s environment (Android Studio or Xcode), navigate to the pubspec.yaml file. Here, you will scroll down until you see assets:

```
assets:
# Icons and Graphics
  - assets/images/map.png
  - assets/images/explore.png
  - assets/images/about.png
  - assets/images/location.png
  - assets/images/noimage.png
# General Photographs
  - assets/images/katy_trail.png
# Location Images
  - assets/images/FCapital1.png
  - assets/images/FCapital2.png
  - assets/images/Augusta.png
  - assets/images/MKT.png
  - assets/images/PilotGrove.png
  - assets/images/LUlogo.png
  - assets/images/Missouri.png
# Cropped Images
  - assets/images/cropped/Augusta_cropped.jpg
  - assets/images/cropped/FCapital_cropped.jpg
```

Now, add your images to the appropriate location. If it is a square cropped image, add it under the "# Cropped Images" section, and if it is a basic locations image, add it under the "# Location Images" section. Follow the examples of the already-added images by adding in either "- assets/images/" before the image or "- assets/images/cropped/". After this is done, save the code and run "flutter pub get" in your terminal.

Now that images have been added, all location data can now be added to the database.
Below is a template for how data can be added to the app in the location.dart file (as mentioned earlier, this file is under the Katy Trail app folder and under the "lib" folder):

```
/* This is the database.
  New locations will be added from here and old locations will also be managed here.
  If you would like to add a new location please copy the following:
  {
    "name": "LOCATION NAME ",
    "description": "Short Description",
    "LongDes":
        "Long description " +
        "Description" +
        " " +
        " ",
    "citations":  "() ",
    "lat": LATITUDE VALUE,
    "long": LONGITUDE VALUE,
    "images": "Image.png",
    "croppedImages": "Square-cropped-image.jpg"
  },
*/
```

Note that there are line numbers on the side, 1-18. To add data, first copy all of lines 4 through 17. Next, scroll to the bottom of the location.dart file and paste those lines here:

```
      "croppedImages": "MKT_cropped.jpg"
    },
    /*PASTE THE FORMAT HERE TO ADD THE NEW LOCATION*/
  ];
  List<Map<String, Object>> get data {
    return _data;
  }
}
```

Finally, add data as it corresponds with each location. This includes: name (name of location), description (a short description of location), LongDes (a longer description of location), citations (any image citations needed), lat (latitude of location), long (longitude of location), images (an image/s you wish to have for location), and croppedImages (a single, square image of location).
Note that images must be written as either .png or .jpg, depending on their type.

Known Issues

1. Tapping on a notification will push the learn page twice, causing the user having to hit back twice
2. Bookmarks are not saved once the app is terminated

Works Cited

Icons on Home Page:

1. "Maps Icon." FlatIcon, Dinosoft Labs, www.flaticon.com/.
2. "Idea Icon." FlatIcon, Freepik, www.flaticon.com/.
3. "Bookbag Icon." FlatIcon, Freepik, www.flaticon.com/.

Authors

- Oliver Johnson - Lead Programmer
- Anna George - Programmer
- Anna Rodrigues - UI Designer
- Shraddha Belbase - Database Developer
- Nirali Patel - Database Developer
- Fredo Derazin - Project Manager

Developed on MacOS and Windows using Visual Studio Code, Xcode, and Android Studio

Technical Manual

Database Information

All location data is stored in a single class file called “location.dart”, where the format for each location looks like this:

```
class Locations {
  var _data = [
    {
      "name": "Missouri’s First Capital",
      "description": "Missouri's first capital building",
      "LongDes": "The first Missouri State Capital Historic Site holds years of history. " +
          "It was Missouri’s first capital building between statehood in 1821 and the capital moving to Jefferson City in 1826. " +
          "The building was restored in the 1960s. It features 11 rooms to take you back in history. " +
          "Tours are available at the State Capital Historic Site office next door. " +
          "Here’s the site before the state restored it and then how it looks today. ",
      "citations":  "(Images: St. Charles County Historical Society, Missouri State Parks) ",
      "lat": 38.784926,
      "long": -90.478854,
      "images": "FCapital1.png:FCapital2.png",
      "croppedImages": "FCapital_cropped.jpg"
    },
```

Here, data is stored in the form of key/value pairs. These keys can then be used in other files to access data, as shown on lines 65 and 73 of the maps.dart file:

```
62          var temp = new Marker(
63            width: 45.0,
64            height: 45.0,
65            point: new LatLng(location["lat"], location["long"]),
66            builder: (context) => new Container(
67              child: IconButton(
68                icon: Icon(Icons.location_on),
69                color: Colors.red,
70                iconSize: 45.0,
71                onPressed: () {
72                  _showLocationCard(context, location);
73                  print("Location: " + location["name"] + " was tapped.");
74                },
```

See User Manual, in its section “Adding Location Data to Katy Trail App” for details on how to add location data to the database.

Coding Standards

Limited use of global variables

Global variables should be avoided at all cost. If a class has a property that must be accessible from other classes, use the "get" method that dart provides.

Naming conventions for local variables, private variables, constants, classes and functions

- All local variables should use camel case starting with lowercase (e.g. localVariable)
- All private variables must start with an underscore followed by camel case (e.g. _privateVariable)
- Constants should follow the rules of local and private variables
- Class names must use upper camel case (e.g. ClassName)
- Function names should use lower camel case (for public functions, i.g. publicFunction) and underscore lower camel case (for private functions, e.g. _privateFunction)

Indentation

- There must be a space after giving a comma between two function arguments
- Each nested block should be properly indented and spaced
- Specific to Flutter, all named arguments of a widget (including the last argument) must be followed by a comma with no whitespace. Each argument should be on a new line with proper indentation
- All braces should start from the same line that they are used for and the code following the end of braces should start from a new line.

Error handling conventions

In the cases where an error is expected but won't cause a runtime stop error, use a try-catch statement and print the error for debugging purposes.

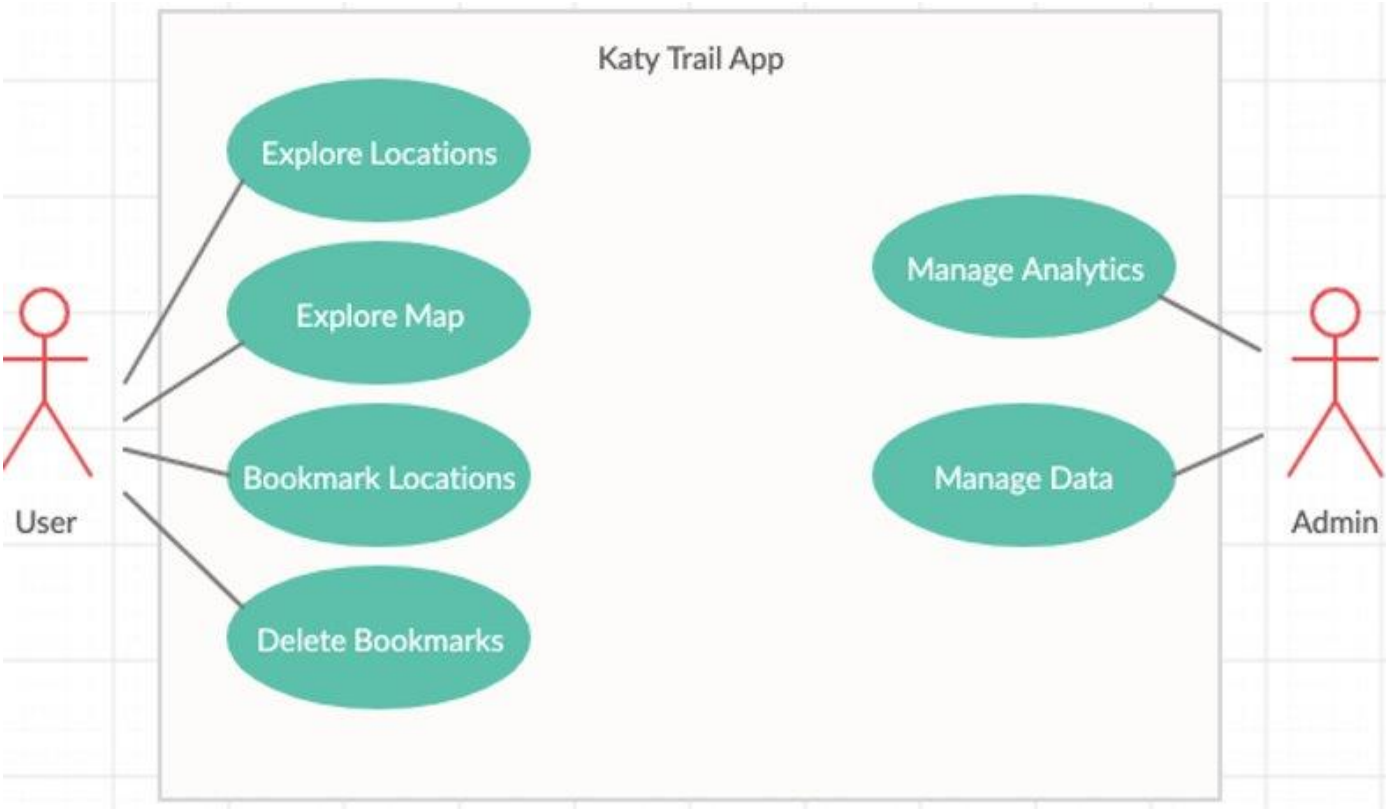
Code should be well documented

The code should be commented for understanding easily. Not every line (or even every function) is required to have comments, but every function header and class header must be commented.

Data types for variables

In most cases, assign a value to a variable as they are declared. When they are declared like this, use "var" or "const" to let dart infer the data type. Only when you need to declare a variable without assigning any value should you specify the data type of that variable.

Use Case Diagram



User Stories

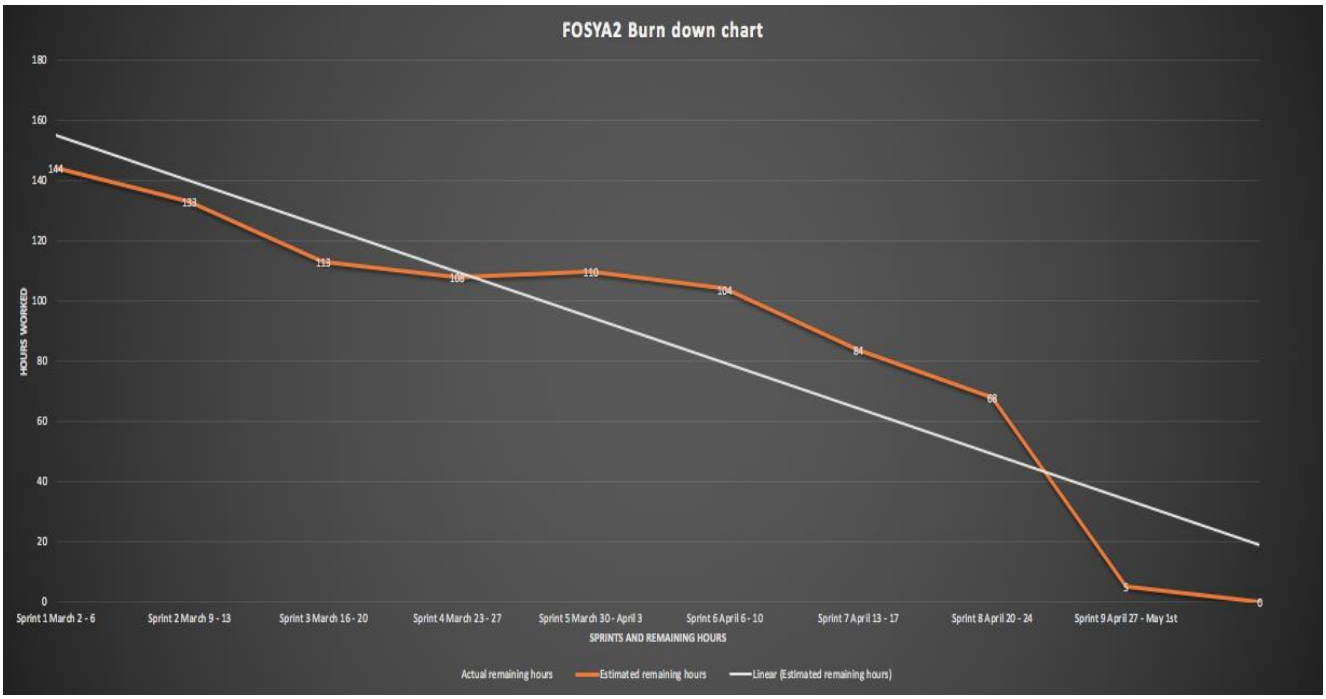
- *The application shall be able to show all locations that have been added to the system*
- *The application must allow the user to bookmark any desired locations they may want to save*
- *The application must allow the user to remove bookmarks of any location*
- *The application must allow the user to explore the map*
- *The admin will be able to view and manage the analytics once the application is published on the Google Play and App stores*
- *The administrator must be able to manage location data by adding and removing as desired*

Retrospective

Project Timeline

Our project development methodology is agile. We set up a backlog with various tasks that were distributed over nine different Sprints. See User Manual section for specific details about GitHub, our backlog management tool. Each Sprint lasted one week. Our team met every Friday at 9pm to test each Sprint release. As shown on the burndown chart below, we started tracking activities from March 2nd and finished off on May 7th.

Burndown Chart



What Went Right

One of the successes we had during the development of this project was the choice of framework. Using Flutter allowed us to work on one source code that is cross-platform so we could test it on Android and iOS. Using Flutter also gave us the ability to quickly build a UI using the built-in widgets that were quick and easy to implement. Another success we had was the use of git. Thanks to git, we've been able to collaborate on the code as a team with a very solid git workflow. In the beginning, we did not have a well-defined workflow, but we eventually decided to utilize an already established workflow that worked well for developing this project. This workflow consisted of branching off a development branch for every feature that was being worked on, sending a pull request once the feature was done, and having someone else review the changes before merging it into the development branch. One last success we had was with testing the push notification feature. With the global pandemic, it was hard to find a way to test push notification since the feature relied on going out to the Katy Trail and going to different locations, but with the finding of the "set location" function of the emulator, we were able to test how the app behaves as the phone moves to different locations, allowing us to discover bugs and fully implement the feature.

What Went Wrong

As far as what went wrong with the project, one of the most prominent problems we had was incompatibility with specific platforms when attempting to add a new feature. There were a few instances where a team member would try to add a feature and the feature ended up working on Android but not iOS or vice versa. While there were not many occasions of this, they were very difficult to solve and since not every member had access to MacOS (which is required to test on iOS), it took much longer to solve these problems. It was even harder to get access to a Mac because of the pandemic. With great effort of certain team members, we were able to sort those issues eventually. Another major issue we had during the project was the implementation of Firebase. We had underestimated the work that was needed to implement Firebase, and our team members were unable to find a way to implement it. By the time the issue was brought up, there was not enough time to consider alternative database methods of retrieving location data, so it was decided that the only way we can have the project done in time was to hard code the location data into the source code. At the very least, we did make sure that the location data only needed to be added in one file of the source code, and all the data would be pass to all widgets as necessary. This was possible due to how modular we built this app from the ground up. Outside programming itself, the largest issue we had was the distribution of tasks and work amongst the team. Based on the amount of code and hours each team member put in, some members' lack of participation put lots of constraint into the project which sometimes compromised the quality of the features that were added.

Lessons Learned

If we were to start all over again, there are a few things we would do differently:

1. We would schedule workshop meetings

   One of the biggest constraints during the project development lifecycle we faced was the ability for team members to deliver on their tasks, on time. That is because team members had a very tough time learning Flutter/DART/Firebase technology. If we had live group training where we learn what we need to learn specific to the project, we believe the development process would be much smoother.

2. We would consider a familiar DBMS (Database Management System)

   The DBMS we selected (Firebase) seemed very promising at first, but we soon discovered many constraints including hosting costs, functional structure, and training materials.