Project 2

**Bryce Canyon National Park**

Bryce Canyon National Park is one of several national parks located in the arid desert of Utah. It’s been a national park since June of 1924 and has had the moniker Bryce Canyon National Park since 1928 (NPS). Outside of the nearly 100 year old history of the park, the region itself contains many beautiful geographic features such as slot canyons and rock spires. The park contains several walkable locations above 6,000 feet elevation, and dozens of miles of trails for all visitors. The unique, picturesque views has drawn more than 2 million viewers annually since 2016.

I chose to build a mobile app around Bryce Canyon National Park because of the availability of NPS data, the history of the park, and its unique geographic features. Bryce Canyon National Park remains one of the many national parks I have not visited; the app I’ve designed is intended to help provide greater information for me when I do visit, as well as provide a way to digitally journal my adventure there.

**App Design**

With the idea of chronicling a journey to Bryce Canyon National Park in mind, I decided to build a mobile-app that would enhance a user’s visit. The app was designed to provide the user with a digital trail map, annotations for overlooks, restrooms, and picnic areas. Lastly, the app would allow the user to input their own information in either review format or as a note for a hazard.

First, it was important for the user to be able to orient themselves on the map. I added the button to allow a user to center the map on their location. Additionally, the app tracks the user location, updated every second, and captures high accuracy data. The user also has the ability to zoom in and out using a double tap, pinch gestures, or the zoom control buttons on the lower right corner. Finally, the map has a compass enabled to give the user the best perspective on their location at any given time. I removed a lot of extraneous features on the map’s default styling to add to the user’s back country experience and to remove Google’s points of interest and busy linework.

National parks are notorious for poor cell reception. With this consideration in mind, I decided to use Google Firebase’s Fire Cloudstore as a prominent feature in storing and retrieving data. The default functions on this NoSQL database enable caching. The caching of a table of information gives a user without access to mobile network the chance to have points of interest on the app, even without a connection. The other benefit to using Google’s technology is that writes to the database will update when a user returns to a coverage area. This lets a user create reviews or issue reports without worrying about an error pushing the data to the database. The app will just wait for connectivity and update the data when the phone connects again.

Another feature in the app was the ability to add and remove layers of the map. On a mobile device, several of Bryce Canyon National Park’s features are nearly on top of one another. To allow the user clarity in identifying features, I enabled the ability for the user to turn on and off layers. This does not interfere with caching data, as the data is stored in the app until the user reloads the app. For the user’s benefit, I also stored the trails overlay in local storage to give the user access to the trails all the time. That way, even if the user boots the app without mobile connectivity, they can at least view the trail features on the map.

There are two different methods for a user to update information regarding the park. The user can add a review of a place or the user can add an issue report. An issue report is a hazard, such as a downed power line or damage, like a pothole or broken sign. The review gives the user the opportunity to make free form comments on one of the park’s features. Additionally, I enabled the ability for the user to view reviews based on park feature.

**Backend Design**

The mobile app itself was built using React-Native. React-Native is a javascript library that works to render javascript into native code. The benefit of using React-Native is to be able to write one program and have the program work on both iOS and Android. I only have an android phone for testing, and I don’t have a Mac, so I can’t test that claim yet. Another stated benefit of React-Native is the ability to produce reusable components. Components are just different parts of the user interface. React-Native’s focus on components, state management, and context were difficult to grasp; however, putting in the work to learn as I worked let me reap a lot of benefits. The project taught me a great deal about javascript, using libraries, state management, and writing reusable code. This should make subsequent projects more enjoyable. The other benefit to React-Native was the live testing of the app. I only had to save a javascript file and the code was compiled into native script and executed.

All of the data, with the exception of the park’s trails, was stored in Google Firebase’s Cloud Firestore. Cloud Firestore is the newest database solution from Google. It is a NoSQL database where collections are analogous to tables, documents are analogous to rows, and the fields would be column headers. Documents could be stored by an auto-generated ID, which is the equivalent of a primary key. Cloud Firestore also allows simple and compound queries. For this project, I only had time and patience to use simple queries. A simple query is written similar to an SQL query. For example, you would query on a field’s value using a ‘where’ statement. Adding entries to the tables was much like adding data in PostgreSQL or MySQL. Unlike MongoDB or Google’s Realtime Database, the information isn’t stored in a JSON-tree. The drawback to this is the developer has to make significant edits to a GeoJSON file to add spatial data. This is most problematic with polylines or polygons.

**ER Diagram**

