Acadia Visitor Study: A Mobile Tracking Application

An Interactive Qualifying Project submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfilment of the requirements for the
degree of Bachelor of Science

by:
James Plante
Joseph Hogan

Date:
August 2, 2019

Report Submitted to: Professor Frederick Bianchi

Abstract

Visitor congestion is an issue in Acadia National Park, as it is the most densely visited national park in the United States. We addressed this problem with a visitor tracking cell phone application. The data collected from the application aims to aid the park in efficiently using their limited resources. We tracked WPI students and administered a survey to park visitors. We then visualized all of the data into intuitive formats. Finally, we generated three ways for the park to deploy the project in various capacities.

Acknowledgements

The group would like to thank both Professors Frederick Bianchi and Thomas J. Balistrieri for the guidance and editorial oversight throughout this project, as it was essential for the creation of this paper, and Abraham Miller-Rushing who was the project's liaison to the National Park Service. Additionally, the group would like to thank Abdulrahman Alatiq for assistance in creating the icon and branding for Acadia Visitor Study, as it greatly influenced the survey's success. Also, the group would like to thank Ermal Toto for server administration help over the WPI network, as well as providing an informative interview. The group appreciated the WPI students who volunteered to use Acadia Visitor Study whenever they went to Acadia National Park. The group also thanks Paige Neumann for help with initial research and citation assistance. Finally, the group would like to thank Stephanie Clement and Paige Steele for information regarding park recommendations.

Authorship

Abstract	**Joseph Hogan, edited by James Plante**
Executive Summary	**Joseph Hogan and James Plante**
1. Introduction	**James Plante and Joseph Hogan**
2. Background	
2.1 The National Park Service	James Plante and Joseph Hogan
2.2 Acadia National Park	James Plante and Joseph Hogan
2.3 Overcrowding	James Plante and Joseph Hogan
2.4 Interview with Expert	James Plante, edited by Joseph Hogan
2.5 Android Studio	James Plante, edited by Joseph Hogan
2.6 Servers	Joseph Hogan, edited by James Plante
2.7 Data Analysis Software	James Plante and Joseph Hogan
3. Methodology	
3.1 Benchmarking	Joseph Hogan, edited by James Plante
3.2 Development of the Application	James Plante, edited by Joseph Hogan
3.3 Surveys	James Plante, edited by Joseph Hogan
3.4 Assessment of Data Analysis Software	Joseph Hogan, edited by James Plante
4. Results and Analysis	
4.1 Survey Results and Analysis	James Plante, edited by Joseph Hogan
4.2 Data Visualization	James Plante and Joseph Hogan
4.3 Possible Improvements	Joseph Hogan, edited by James Plante
5. Recommendations and Conclusion	
5.1 Future WPI Involvement in Development and Hosting	James Plante and Joseph Hogan
5.2 Incorporation of Functionality into Existing Application	James Plante, edited by Joseph Hogan
5.3 OnCell	Joseph Hogan, edited by James Plante
5.4 Conclusion	Joseph Hogan, edited by James Plante

Table of Contents

Executive Summary

Figure 1: Icon for Acadia Visitor Study. Reprinted from “Icon for Acadia Visitor Study” by Alatiq, A, 2019. Used with permission.

Overcrowding is an increasingly serious problem facing the National Park Service. Although visitation is encouraged, this influx of visitation is causing a myriad of problems. The most important goal of the National Park Service is preservation. With limited space and resources, the NPS is having difficulty accommodating all of their visitors and properly preserving the lands. This issue is most prominent at Acadia National Park where the visitors per acre is higher than any other national park (National Park Service, 2019).

Our project was designed to allow the park to do more with the same amount of resources by providing essential GPS data to the park staff. It is a proof of concept that paves the way for a park-backed tracking application that will reach the majority of park visitors. By using the data that is collected, the NPS will be able to more accurately deal with the crowds of visitors each year.

The park has already addressed overcrowding with the new Transportation Plan, which encourages visitors to take public transportation around the island, and aims to better manage

parking locations, among other goals (National Park Service, 2019). The data our app collects can also be used to assess the success of this transportation plan.

Our main objectives are to:

1. Build a GPS tracking mobile application targeting the Android operating system
2. Distribute the application to the public and collect data about visitor mobility
3. Determine the efficacy of the application by portraying the data using different analysis methods

We developed the application for Android, and distributed it to fellow WPI students. We used a server hosted by WPI to receive and save the time-stamped GPS data using an HTTPS connection and later exported it for analysis. We were unable to get it into the hands of the public because Google Play Protect flagged the application.

We administered an entrance survey at the Visitor Center to gauge the applications favorability among its user base. We had initially planned to distribute a version of the survey as both an entrance and exit survey on the application. However, the limited rollout of our application prevented us from doing so.

We first asked the public what mobile platform they used and found that 76.7 percent of respondents owned iPhones. This is significant, as it indicates that an iOS platform should be included in future application development. We also determined that many visitors already use their phone in the park, as 67.1 percent of visitors to Acadia National Park stated that they either use their phone alone or a phone and a map to navigate. Additionally, we found that visitors were generally apathetic towards tracking features in other mobile applications, as 66.7 percent of respondents were comfortable with companies like Google collecting user data. Finally, when visitors were told our application would similarly track them, but use the data to help the park, 91.7 percent of people were willing to download the application.

After collecting nearly twenty-five thousand data points from six WPI students, we used ArcGIS to analyze our GPS data, and generate meaningful data visualizations demonstrating the vast amount of useful information that could be obtained from our app. This information

includes: facilities used, trails hiked, and transportation methods utilized. We determined that this method of data collection is a viable way to monitor visitation and could potentially be used for predictive analytics in the future given a large enough data pool.

Finally, we developed three main recommendations over the course of the project that the park can follow to advance the project. Our three main recommendations are as follows: WPI continues the development of Acadia Visitor Study, incorporate the tracking functionality into an existing phone application, or develop an application from scratch using the app development platform OnCell.

Table of Tables and Figures

1. Introduction

Overcrowding and climate change both pose threats to the National Park System. Overcrowding has led to an increase in carbon emissions and has decreased visitor experience in the National Parks over time. It also has had a deleterious effect on the ecology of the parks, which has increased many parks maintenance costs. Climate change threatens the local ecology, biodiversity, and geology that the National Park system is dedicated to protecting. These features are the main attractions for visitors, and all of these issues are expected to worsen in coming years unless action is taken to accommodate overcrowding.

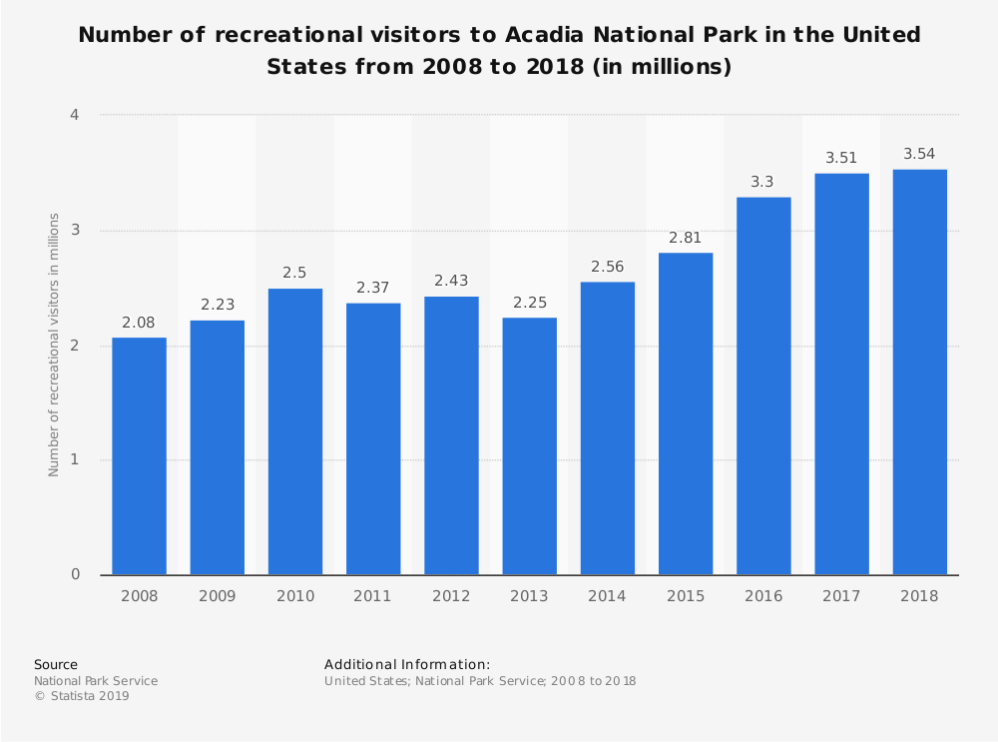


Figure 1.1. Number of recreational visitors to Acadia National Park in the United States from 2008 to 2018 (in millions). Reprinted from *Statista.com*, by National Park Service, 2019, https://www.statista.com/statistics/253808/number-of-visitors-of-acadia-national-park-in-the-us/.

The aforementioned issues the National Park System faces are magnified in Acadia National Park because of visitor density. The 3.5 million annual visitors make Acadia National Park the most densely visited national park in the United States, which presents acute challenges

(National Park Service, 2019). First, increased transportation in the parks has led to the degradation of many man-made landmarks along Park Loop Road, including the road ways which can become dangerous for drivers (National Park Service, 2019). The destruction of land in Acadia is simultaneously the destruction of the park's biggest tourist attraction. Additionally, as the park becomes more crowded, it becomes more susceptible to pollution, littering and carbon emissions.

As such, the ecological, socioeconomic, and psychological impacts of congestion have been the focus of various IQPs, journals, and studies in the past. In 2017, a research team of WPI students started to study possible ways of limiting transportation throughout Acadia National Park. Their recommendations included a reservation system and a two-way gate system to decrease traffic (Cosmopulos, E., Gaulin, J., Jauris, H., Morisseau, M., & Quevillon, E., 2017). Another study revisited the idea by analyzing the possibility of tracking vehicles in the park using a GPS phone tracking system. This project was in response to the limited accuracy of car counters in the parking lots (Nadeau, G. T., Charbonneau, J. L., Caltabiano, J. P., & Fischler, M. A., 2018). Other studies have been conducted to analyze the impact of climate change on the visitor experience while in Acadia. According to a study by the *Journal of Outdoor Recreation and Tourism*, park visitors indicated that inclement weather may impact their experience when surveyed on the impacts of climate change (De Urioste-Stone, Scaccia, & Howe-Poteet, 2015).

In response to visitor congestion, on March 25, 2019, Acadia National Park released a detailed transportation plan, which includes the relocation and construction of various parking lots in order to streamline visitor experience. They also plan to implement a reservation system to limit congestion in high traffic areas. In the study, the park considered various solutions to congestion while also considering how it would affect both the environment and the visitor experience. The study concluded that a reservation system for popular locations around the park would be the best option and would be implemented over the next decade (National Park Service, 2019). In theory, this solution would help eliminate traffic congestion, but they also noted that it would negatively affect the historic experience of the park (National Park Service, 2019).

As detailed as this plan is, we hope to add valuable information to the plan by supplying visitor GPS data. With a mobile application we have bridged the information gap and can provide accurate, GPS data on individuals in the park. Our purpose is to develop and distribute a full stack, GPS mobile tracking application to gather data, and analyze the application's efficacy by suggesting alternate analytical solutions.

2. Background

2.1 The National Park Service

The National Park Service has played a role in conservation for over a century. Land conservation in the United States originated on March 1, 1872 when President Ulysses S. Grant signed the Yellowstone National Park Protection Act making Yellowstone Park the first national park in the world ("Yellowstone National Park Protection Act (1872) - Yellowstone National Park (U.S. National Park Service", 2018). During that time the Department of the Interior was responsible for protecting and maintaining all of the lands. Additionally, the United States established monuments and historical areas, which would be overseen by the Department of War and the Forest Service of the Department of Agriculture. Eventually, President Woodrow Wilson created the National Park Service (NPS). The NPS was created on August 25, 1916 by the National Park Service Organic Act. The NPS was now responsible for the preservation of all 35 sites, consolidating the work of three different organizations into one dedicated entity, whose mission is to preserve the integrity and natural beauty of the United States' geography for future generations ("Quick History of the National Park Service (U.S. National Park Service)", 2018).

The National Park System has gradually expanded. In 1956, supported by President Eisenhower, Mission 66 was passed, which was a decade long effort to expand the current park facilities. The bill created thousands of trails, roads, paths, and park facilities, and revolutionized the architecture and schematic of the modern national park (Walser, 2017).

Now, the National Park System protects more than 400 parks, making up roughly 84 million acres, or over twice the size of Georgia. The NPS supports over 306,000 jobs, and directly benefits communities surrounding the parks ("Zinke Announces $35.8 Billion Added to U.S. Economy in 2017 due to National Park Visitation - Office of Communications (U.S. National Park Service)", 2018). Additions to the park service are made by Congress, or by the President proclaiming national monuments on federal land ("Quick History of the National Park Service (U.S. National Park Service)", 2018).

2.2 Acadia National Park

Figure 2.1. Main Acadia National Park map, showing the majority of the park located on Mount Desert Island near Bar Harbor. Reprinted from *Wikimedia Commons*, by National Park Service, 2013, http://npmaps.com/wp-content/uploads/acadia-map.jpg.

Maine was first settled by Paleo-Indian hunters around 11,000 years ago. Large game such as mammoths and mastodons sustained inland hunters, while coastal natives primarily subsisted on seals and walrus. Increasing temperatures following the end of the Ice Age led to a mass northward migration of arctic and sub-arctic species. These species were soon replaced by a variety of smaller mammals, similar to the ecology of Maine today. The Wabanaki people, who populated much of eastern Maine at this time, are associated with the early settling of Mount Desert Island (Kaiser, 2018). The island became known to these tribes as the Pemetic, or "the Sloping Land". ("History of Acadia - Acadia National Park")

Europeans brought diseases such as smallpox to coastal Maine that ravaged the native population. As a result, the population of Wabanaki decreased from ~32,000 to ~1,200 between the early 1600's and late 1700's. In 1603, Samuel de Champlain led a voyage to North America. After landing in Canada, he traveled along the coast going south past Maine. During this voyage, he sighted Mouth Desert Island, naming it "the Island of Barren Mountains". Fifteen years later,

in 1618, 75 percent of Maine's native and coastal population were killed by European diseases (Kaiser, 2018).

Settlement was delayed for many decades due to conflicts between the English and the French. However, in the late 1700s, settlers in search of fertile land expanded up the coast of Maine. Eden, soon to be Bar Harbor was founded during this expansion. The town grew quickly because of industries such as fishing, shipbuilding, and logging. Mount Desert Island, now part of Acadia National Park, was initially owned by Francis Bernard, an Englishman serving as royal governor of Massachusetts (Kaiser, 2018). However, Francis Bernard fled following the Revolutionary War and his son followed suit, losing their claim on the land in the process ("History of Acadia - Acadia National Park").

National attention focused on Mount Desert Island after painters from the Hudson River School of Art visited there in the mid-1800s ("History of Acadia - Acadia National Park"). The Hudson School of Art was primarily an artistic movement famed for their beautiful landscapes. Notably, one of the more famous painters of this movement was Thomas Cole, where he painted numerous landscapes featuring the geographic features of Mount Desert Island (Kaiser, 2018). Cole's work was exhibited in New York and drew enormous attention from other artists, among them was one of Cole's students, Frederic Church. Inspired by Cole's work, he returned to the island to create additional works of Acadia's landscape. Additionally, other artists and tourists began to journey to the island, inspired by what they had seen from Cole and Church ("History of Acadia - Acadia National Park").

During the 1870's and 1880's, Mount Desert Island established an electrical infrastructure, as well as a public transportation system, convincing tourists to travel to Bar Harbor. As tourism increased, money flowed into the area leading to the development of luxury hotels, mansions, and boutiques ("History of Acadia - Acadia National Park"). Growth continued to accelerate over the coming decades. However, locals were concerned that the growth of Bar Harbor was affecting the area's natural beauty. Visitor and former Harvard President Charles Eliot, as well as resident George Dorr founded the Hancock County Trustees of Public Reservations, and began purchasing the land under a tax exempt charter. When the tax

exemption was revoked, Dorr traveled south to Washington and convinced President Woodrow Wilson that the government should invest its resources into the land. Two years later, a National Monument was created by Presidential Proclamation, and was recreated as Lafayette National Park on February 26th, 1919. A decade later, the name was changed to Acadia National Park (Kaiser, 2018).

In the summer of 1947, Maine received only half of its expected rainfall, allowing a small fire to grow into an inferno that ravaged the dry landscape. It burned 17,188 acres of land. The fire burned 10,000 acres in Acadia, and caused 23 million dollars in damages to the park. Spruce and fir trees, which initially dominated the forest, burned down and have since given way to deciduous species. Additionally, many of the mansions on Millionaires Row that burned down were never repaired. Instead, the seasonal population of Bar Harbor left, ending the opulent "cottage era" lifestyle that had dominated the island that was beginning to wane during the Great Depression ("Fire of 1947 - Acadia National Park").

Mount Desert Island sits squarely within the Gulf of Maine. The Gulf of Maine is dense with wildlife, containing ~3,000 species, and frequent, massive phytoplankton blooms caused by high gulf temperatures. Because the Gulf of Maine is a shallow outcrop from the Atlantic Ocean, the Atlantic tides are slightly amplified. This increases the tidal range in Acadia National Park to 8-12 feet. Acadia includes about 40 miles of public shoreline. This offers visitors exposure to the intertidal zone, the ecosystem that exists on the coastline between high and low tide. Organisms at different depths of the intertidal zone spend vastly different amounts of time underwater. Therefore, they have diverged evolutionarily. This biodiversity there makes studying the intertidal zone valuable to scientists (Kaiser, 2018).

Today, Acadia contains roughly 320 species of birds, as well as around 50 species of mammals. There is also a significant elevation change in the park, ranging from sea level to roughly 1500 feet above. The tops of the higher peaks on Mount Desert Island are essentially barren. The elevation supports alpine vegetation among other species that could not survive in the lower lying areas. Humans have brought several non-native plant species to the island. Many of these, such as the Purple Loosestrife from Europe, are now considered harmful to the native

ecology of Mount Desert Island. More threatened than the island is the Gulf of Maine, where heavy fishing caused many fish populations in the region to collapse by the 1980's (Kaiser, 2018).

2.3 Overcrowding

Overcrowding is a global pandemic, as it is such a large problem that it has affected the lives of almost everyone. Overcrowding largely has environmental and economic effects. Symptoms of overcrowding can be felt in various locations such as cities, highways, and even parks. When the occupancy limit of a given area is surpassed, the surrounding land and infrastructure may degrade over time. For instance, highways in the United States are so abused by the amount of traffic that general maintenance of the highway system costs the United States government 22.81 billion dollars annually ("23rd Annual Highway Report", 2018). Also, cities where overcrowding is a common problem face high rates of poverty and pollution (Pencheon, 2012). This is a problem, especially in developing nations such as China, where it is difficult to find apartments due to housing shortages in the region, which has been found to negatively affect the psychology of residents in the area due to reduced living space (Tao, 2017).

Overcrowding also negatively affects the tourism industry. According to tourist demand theory, it is shown that the closer a destination is to its maximum occupancy, the less desirable a place it is to travel. This quantitative theory takes into account many aspects such as environmental and socioeconomic factors. Overcrowding is considered a non-constant supply side attracting factor. A non-constant supply side attracting factor is a factor that changes based on the location itself, like the density of crowds at a given time. For instance, as overcrowding increases, visitor demand goes down, especially in destinations not already known to be crowded such as forests or beaches (Santana-Jiménez, Y., & Hernández, J. M., 2011).

As stated previously, overcrowding negatively impacts the individual experience and the environment surrounding the overcrowded location, especially when in more dense areas. For example, in National Parks, the very land that is being overused is the main attraction for visitors, and the main object of preservation efforts. This compounds the issue of overcrowding

in National Parks therefore putting a greater onus on the National Park Service to mitigate overcrowding.

Turkewitz and Fremson have given insight into this issue in an article exposing how Zion National Park in southern Utah is struggling to deal with the increasing visitor population. Frustrated visitors at the park, many of whom have made plans months in advance, are being turned away at the entrance of the park due to overcrowding. Additionally, park rangers have to spend more time and resources to clean up damages that park visitors cause to geographic features in the park such as littering and carvings on rock faces. Moreover, visitor-made paths and trails have been an issue for the topsoil and wildlife in the region (2017). Additionally, a study conducted at Acadia National Park concerning what factors had the most impact when visiting Ocean Drive found that traffic and visitor congestion was one of the most disliked aspects of visiting the area (Hallo & Manning, 2009).

Acadia National Park has cited similar conclusions about visitor overcrowding, and is currently taking steps to mitigate visitor impact. Previously, Acadia National Park has addressed this issue by expanding infrastructure in the park to account for more people. Acadia National Park also attempted to manage visitor behavior by making the primary road for travel, Park Loop Road, a one-way road and created more transportation options to visitors such as the Island Explorer. However, while these efforts were appropriate for the time, it became apparent that additional work was needed to address the increasing density of visitors in the park. Moreover, as previously stated, on March 25, 2019, they have made plans to expand the infrastructure of the park further to address this issue (National Park Service, 2019). Overall, while the issue of overcrowding can be addressed by developing infrastructure, the larger issue of overcrowding needs to be addressed on a more global scale.

2.4 Interview with Expert

Interviewing is an ethnographic technique that involves a one-to-one interaction with a person in order to gain information about a subject. Although both interviews and surveys concern getting information from individuals, interviews are typically different than surveys as

they can address greater complexities because the interviewer is able to give more open-ended questions. Interviews can be done in three ways: fully structured, semi-structured, and unstructured. Fully structured interviews typically involve the interviewer asking questions in a pre-defined order, and as such are easier to compare when doing multiple interviews. In contrast, unstructured interviews involve the interviewer asking a question about a specific topic and letting the person speak freely about that topic, following the interviewee's lead. Semi-structured interviews are a combination of both fully structured and unstructured approaches. The interviewer asks a predetermined set of questions, and the interviewee has the freedom to make additional observations. The interviewer can ask follow up questions or continue with the predetermined questions (Lazar, Feng, & Hochheiser, 2010). We will be using semi-structured interviewing for this project as it provides additional flexibility for interview questions and we will be interviewing only one person, making fully structured interviewing inappropriate for our purposes.

Ermal Toto was interviewed during our research. Toto is a research data scientist and former senior software engineer at Worcester Polytechnic Institute, where he is currently pursuing his PhD. He mainly specializes in data mining, machine learning, and researching the use of these methods to detect the mood of individuals. Toto also currently works for the Information Technology department at WPI (Toto, 2019). He was questioned primarily regarding techniques and practices of smartphone app development and current server integration as referred to in Appendix H. In addition to helping with the initial planning of the application through the interview, Toto was able to set up the required server backend, database, and web domain (which is described in Section 2.6) on the WPI network as well as secure it by enabling SSL encryption on the website. He was also able to provide us access to WPI's GitLab instance, which easily allowed us to work on the project in remote locations and create a backup for any existing code.

2.5 Android Studio

The project is a mobile tracking application and was developed using Android Studio. Android Studio is an integrated development environment (or IDE) that is developed by Google for creating Android applications. It is based on the IntelliJ IDE which is mainly optimized for developing Java applications. As such, Android Studio has many features that are industry standard across many IDEs such as autocomplete, support for refactoring variable names within an entire project, and integrated unit testing. Moreover, Android Studio has many additional features that make Android development much easier. For instance, it features a layout editor so developers can change the location of buttons, search bars, and other menu elements without having to resort to a text editor ("Download Android Studio and SDK tools", 2019). Another advantage is the Android Debug Bridge (ADB), so when an Android phone is connected to a PC, it can download the current build of the application and run it. Finally, it provides a virtualized Android environment if a phone is not available, allowing for users who either do not own an Android phone or do not have one on-hand to test their code ("Download Android Studio and SDK tools", 2019).

2.6 Servers

A server is a computer than runs a persistent piece of software that provides a service for other programs. Devices that utilize a server are called clients and can connect to a server to use that server's resources. The network formed is called the client-server model. A frequently implemented protocol to facilitate communication between the client and server is called the request-response model. This model involves the client sending a request to the server for information. Afterwards, the server responds with the information requested. The main purpose of a server is to execute work for and share resources among other computers. For example, if there is a task that thousands of applications or devices need to accomplish, exporting that task to a server is highly efficient ("Server - Definition and Details", 2019). Servers can have a wide range of specialized purposes from file storage, hosting websites, and computing large amounts of data using protocols such as SSH, FTP, DNS, and HTTP.

Most servers are relied upon by a large network of clients. Thus, they need to be reliable and efficient. In order to fulfill these unique requirements, some servers run on specialized hardware. To increase reliability and speed, servers can be clustered to increase server uptime as much as possible. Clustering is the process of pooling together multiple servers to handle requests from the client. This allows for individual servers to fail without the entire service going down for the user. This can be done through transparent clustering, where one "dispatcher" server handles all of the requests from the client. Afterwards, the dispatcher can uniformly distribute the requests to the server pool and send the response to the user. Clustering also increases the performance of the service because it can handle more requests (Schroeder, Goddard, & Ramamurthy, 2000).

Our application utilizes a LAMP (Linux, Apache Web Server, MariaDB database, and PHP backend) server to store the data points collected. Additionally, it allows us to search through a large amount of data points in a very short time. The server is currently hosted on <acadiatrails.wpi.edu> and whenever accessed, generates a random unique ID that the application can use, then outputs it to an HTML document. The application then parses this document for its own use. It can also accept data in the form of JSON formatted strings that are then parsed to store the data in a MariaDB database. The server also allows the easy exporting of data to an ASCII text file, which can then be read by various data analysis software.

2.7 Data Analysis Software

We researched the efficacy of various platforms to analyze the GPS data we will obtain. Our research led us to two choices: ArcGIS and the IBM SPSS Statistics pack. Both applications have the ability to analyze the GPS data for trends. ArcGIS is capable of viewing the GPS data on a map overlay and plotting the routes of each person. It can also compare the routes and show heat maps of areas with higher concentrations of visitors (ESRI, 2019). The SPSS Statistics pack can create models of the data to show trends between different people's paths and fill in missing data gaps. It can be used to provide a predictive analysis of future trends assuming more people

will visit the park (IBM, 2019). We will be determining the efficacy of these applications while in the park.

3. Methodology

Our mission was to develop and distribute a full stack, GPS mobile tracking application to gather data and analyze the application's efficacy by suggesting alternate analytical solutions. This will hopefully aid park staff in mitigating tourist congestion in Acadia National Park by providing a way to collect data on visitor behavior. Our main objectives will be to:

1. Build a GPS tracking mobile application targeting the Android operating system
2. Distribute the application to the public and collect data about visitor mobility
3. Determine the efficacy of the application by portraying the data using different analysis methods

3.1 Benchmarking

Before arriving in Acadia National Park, we used benchmarking to assess the various ways that visitor tracking has been addressed in parks of a similar size. Although benchmarking is typically a business tool, we have adapted the technique to fit our needs. Rather than iterating over a physical product, we iterate over our proposed solution. With each pass, we modified our statement and then analyzed the revision to gauge improvement. During our initial research, we underwent a brief benchmarking session.

During our session, we looked for the most effective example of visitor tracking, Disney World. Through the use of several, highly accurate methods, Disney World tracks most of the movement of its 53 million annual visitors ("Walt Disney World Statistics", 2017). They then use this data to improve visitor experience. They can move resources dynamically around the park to cater to visitor needs, as well as alert visitors during times, and in areas where congestion is going to be high. Unfortunately, the park's methods are not transferable to our situation because the nature of their tracking was far too invasive to be employed in a National Park (Geissler & Rucks, 2011).

3.2 Development of the Application

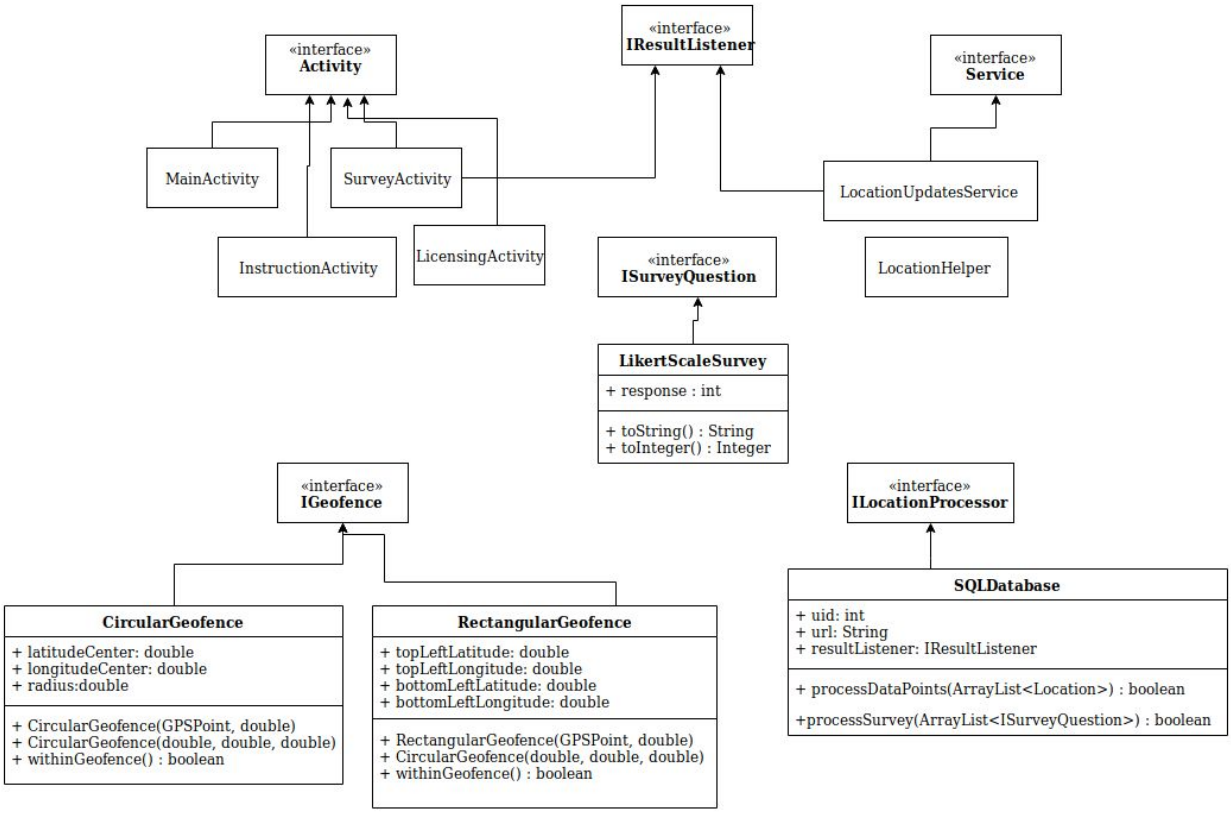


Figure 3.1: Basic UML Diagram of the Acadia Visitor Study application

The application that was developed for this study, Acadia Visitor Study, is an Android application written in Java. In general, the application was originally designed to both track users' GPS data while in the park and as a way to deploy exit surveys. Android was chosen as the platform to target since it has native Java support and it would not require submission to the Google Play Store, since that would slow development due to the limited amount of time allotted for the project. However, due to difficulties installing the application with Google Play Protect, an antivirus that was recently incorporated into the Google Play Store, a full rollout of the application was not possible during the time of the project to visitors in the park and was instead tested with WPI students.

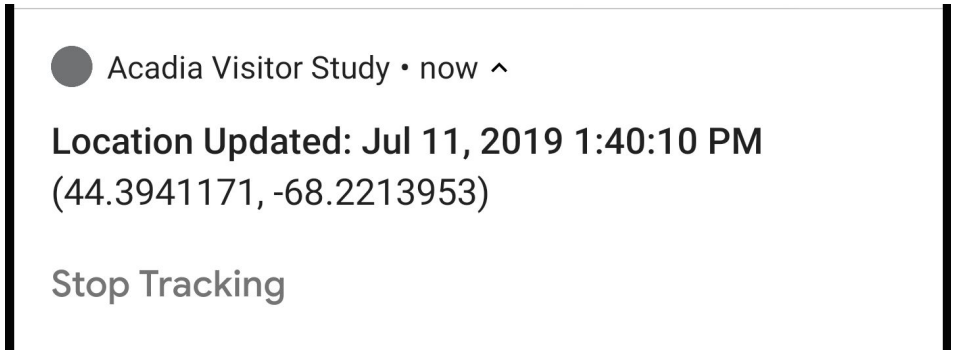


Figure 3.2: Screenshot of the notification shown to users when tracking is enabled

To give a complete overview of the application, it is appropriate to get an idea of the different tasks that the application has to handle. As such, Acadia Visitor Study has a number of threads that work simultaneously to accomplish these tasks. First, the UI thread is the standard method that Android uses to handle any on-screen events that may happen in any of the 'screens' (also known as Activities) that are visible to the user. As such, many of the 'intensive' tasks of the application (i.e. storing GPS information, sending GPS points to a server, etc.) are relegated to other threads.

Next, there is another thread that runs when the application starts and will start collecting GPS points as soon as the user clicks on the 'Start Tracking' button and will keep track of a queue of GPS points that will be uploaded to a WPI server after the queue gets to a certain size and when there is an Internet connection. This system of collecting the GPS points in "batches" is a simple way to account for situations where Internet is not available, which is a common occurrence especially in popular destinations such as Jordan Pond. This service is run in the foreground as a notification so users will be able to see what their current location is in real time and be able to disable the tracking from outside of the application. Additionally, this gives the user transparency as to what the application is doing at any given time. Finally, there are worker threads that will be created on-demand to do any network operations that may be needed by either the UI thread or the GPS thread when uploading to the server.

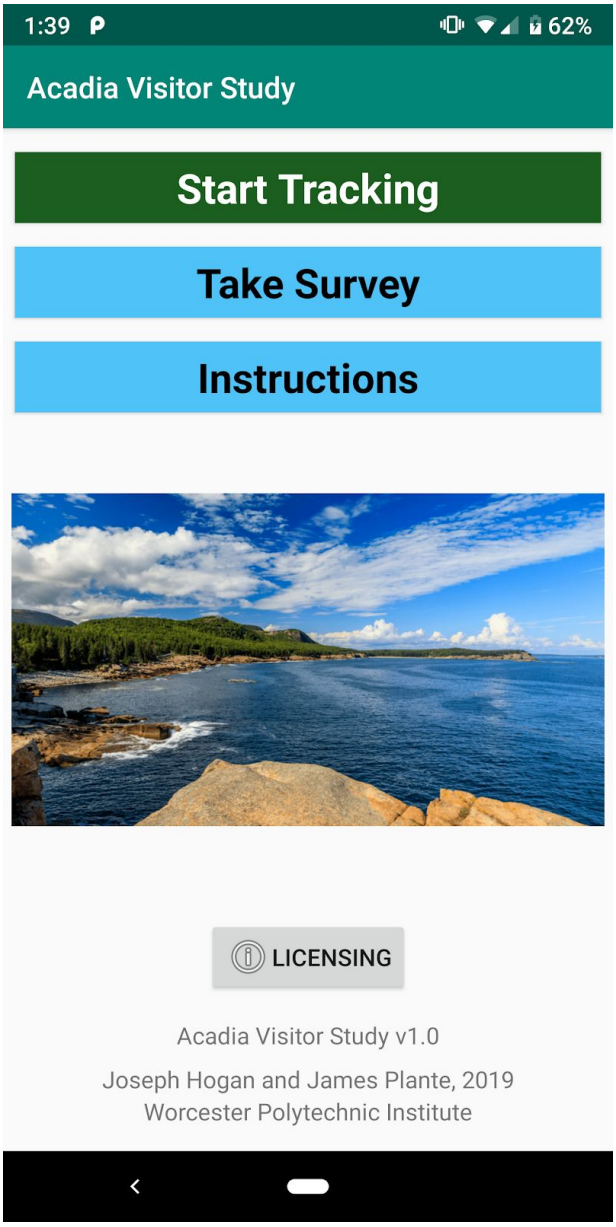


Figure 3.3: Screenshot of Acadia Visitor Study's home screen

The user interface (or UI) is an important aspect of Acadia Visitor Study since our group wanted to prioritize transparency to users above all else. As such, we addressed this in a number of ways. First, as previously stated, we make it clear what data is being collected by having a persistent notification when the application is tracking users. Next, we also made sure that data would not be collected unless the user consented to the data collection. To implement this, we created a toggle button to start and stop the data collection thread and created an instructions

screen detailing how to use the application and what data is collected. Finally, to help gain user trust in our application as to our own intentions, we also gave a brief description on who we are in the instructions page and our contact information so if there were any issues with the application, we could be easily contacted to fix any issues if they did arise.

The security of transmitting the location data for storage and the anonymity of the data collected is another aspect that we considered in creating Acadia Visitor Study. First, the data that is sent to the server in the form of a JSON file is encrypted using the HTTPS protocol, which the server then reads and stores in a secure MariaDB database. As such, the data that is collected by the application will not be vulnerable to man-in-the middle attacks, so the data cannot be easily interpreted by malicious parties. Additionally, the server code was written in a way to prevent any unauthorized access to the database by using parameterized SQL statements, a built-in feature of the PHP programming language. Next, all of the data that is collected from the user is completely anonymous and does not contain any demographic information, although the application could be easily augmented to add this functionality. As suggested by our interview with Ermal Toto, the only unique information that can identify a user is a randomly generated user ID when the application is first launched. The determination of the user ID is done by the server itself to prevent any collisions in the database. Then, the ID is sent to the phone via an HTML page, which the app then parses and stores for later use. Finally, in order to avoid collecting data from the user when outside of Mount Desert Island and the Schoodic Peninsula, geofences are used to check the boundaries in which the application can collect data. If the user is outside of either of these regions no data points are collected, although any points that have been collected will be uploaded to the server to prevent loss of any data points going to or from Mount Desert Island and the Schoodic Peninsula. The user is also notified that no data points will be collected.

The final aspect of the application that our group wanted to address is its extensibility and modularity for other groups to use in the future. First, we wanted the application to be able to accommodate different geographical areas easily. Therefore, we created an interface to add custom geofences of any arbitrary shape that is needed, although we only implemented simple

circular and rectangular geofences due to the limited timeframe. Additionally, most of the code relating to communicating with the server is also abstracted out to the point that a completely new implementation of the server code (as seen in SQLDatabase.java) could be written to support other network protocols. Lastly, we made an interface for survey questions so if any additional types of questions are needed, all that will need to be written would be the classes to support the new type and modifying the UI to accommodate the questions.

3.3 Surveys

Figure 3.4: The stand where the surveys were conducted

Surveys are an essential component of our work in Acadia National Park. During our time in the park, we conducted an entrance survey to aggregate visitor experience. It was graded on a Likert scale ranging from one to seven, with one indicating they strongly disagree with the statement and seven indicating they strongly agree. Primarily, we surveyed visitors because we hoped to get a fresh reflection on their time at the park. Their feedback was essential when analyzing the effectiveness of our application. Knowing how the visitors feel about being tracked shows how practical gathering data will be in the future. Finally, their feedback provided insight into the effectiveness of the parks new traffic management plan. The questions for this survey may be found in Appendix B.

Initially, when developing the survey, we planned to have both an entrance and exit survey: one before users would download the application and the other intended to be filled out

within the application after leaving the park. However during our time at the park, we ran into issues with rolling out the application to users, since Google Play would flag our application as malware. This was problematic due to the fact that in order to get around this issue, certain security features of the Android operating system would have to be disabled and re-enabled during the installation of our app. Therefore, wary of the ethical and moral implications of compromising the security of users, we decided to combine both the entrance and exit surveys into one four question entrance survey.

We chose to set up our booth outside of the Visitor Center to get as many people as possible to answer the survey. Additionally, we collaborated with the WPI Carbon Footprint Team, that was doing a consecutive study on Acadia National Park's environmental impacts to create a poster to attract visitors:

Figure 3.5: Poster that was shown to visitors. Adapted from "Poster for Acadia Visitor Study" by Alatiq, A. 2019. Adapted with permission.

After researching what should be included in the poster, we decided that a QR code that will link to a website to download the application will be best for our needs, since it would attract visitors to the poster regardless if the application was available to download or not. According to an experimental study conducted by the *Journal of Business Research*, while the increased complexity in an advertisement containing a QR code may attract curious customers, if a brand has a perceived fit with the QR code itself, less curious customers will also be attracted to it (Okazaki, Navarro, Mukherji, & Plangger, 2017). As such the choice of using a QR code

alongside WPI branding to advertise the application will attract more people to the booth, as WPI is well known in the northeastern United States as a more technical university.

Finally, while we were on premises for peak times during the hours of 10:30 a.m. to 1:30 p.m. in order to get data from as many people as possible, we also wanted to broaden our sample and prevent selection bias (Nadeau et al., 2018). As such, we decided to survey visitors at off-peak hours such as 9 a.m., when the Visitor Center opens.

In Section 4.1 we describe the results of the survey in more detail.

3.4 Assessment of Data Analysis Software

Data visualization is a valuable tool when collecting large amounts of complex data. In fact, when deploying Acadia Visitor Study to just a few students, over twenty-five thousand data points were collected over two weeks. For that reason an important part of our project is to find a suitable software suite that can visualize the GPS data effectively. We chose two different programs to bring with us to the project site. The first program we used is called ArcGIS. The program was developed by ESRI, and is designed to compile geographic data.

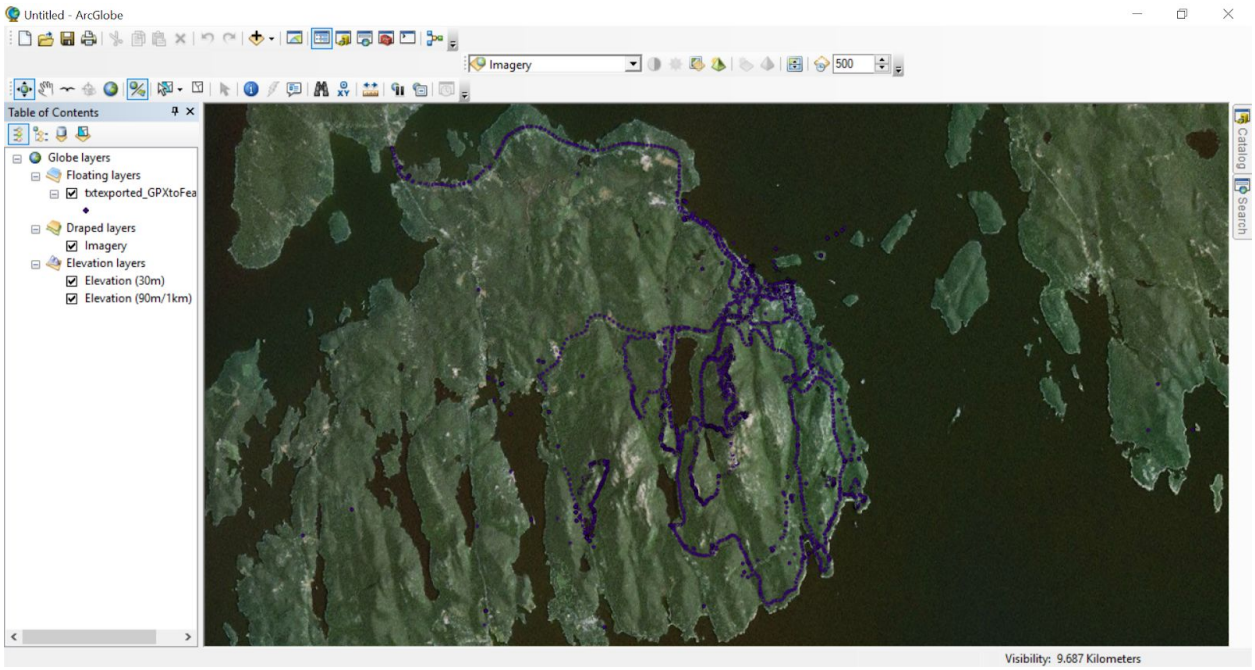


Figure 3.6: Screenshot of the imported data into Google Earth

The second program we used is a more general data analysis package developed by IBM called the SPSS statistics pack.

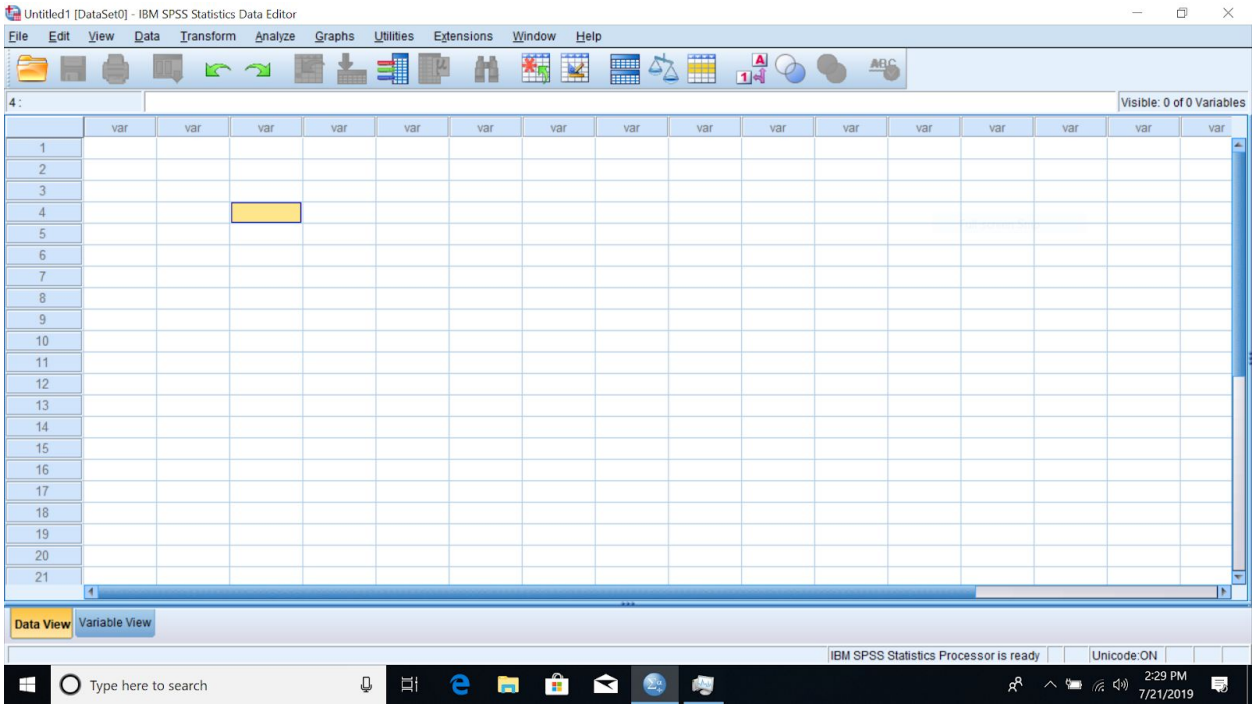


Figure 3.7: Screenshot of an empty spreadsheet in IBM SPSS

Although visualization is effective, looking at the data we directly collect only accounts for a small fraction of the possible value. By cross referencing our data with other available data, like weather, we can begin to extract meaningful correlations. Intuitively, predicting visitor traffic can be straightforward, for example, if a cruise ship is in town, Bar Harbor is going to be more crowded. However, with more experience, comes greater predictive accuracy. So, park rangers may be able to more accurately predict the crowds given a more abstract correlation, like gas prices.

We initially loaded in our GPS data into Google Earth to visualize the paths of our users. During this process, the data had to be converted into the GPX format so the program could read the paths of users correctly. To accomplish this, we created a Python script to convert the data into separate tracks per UID using the GPXPY and SRTM.py modules. Additionally, using the

SRTM.py module, we were able to generate elevation data using only the latitude and longitude values of our data. Unfortunately, when graphing the data, Google Earth does not discriminate between individual GPS track segments, so the data for some users was unreadable for our purposes. The source code for the script is located in Appendix G.

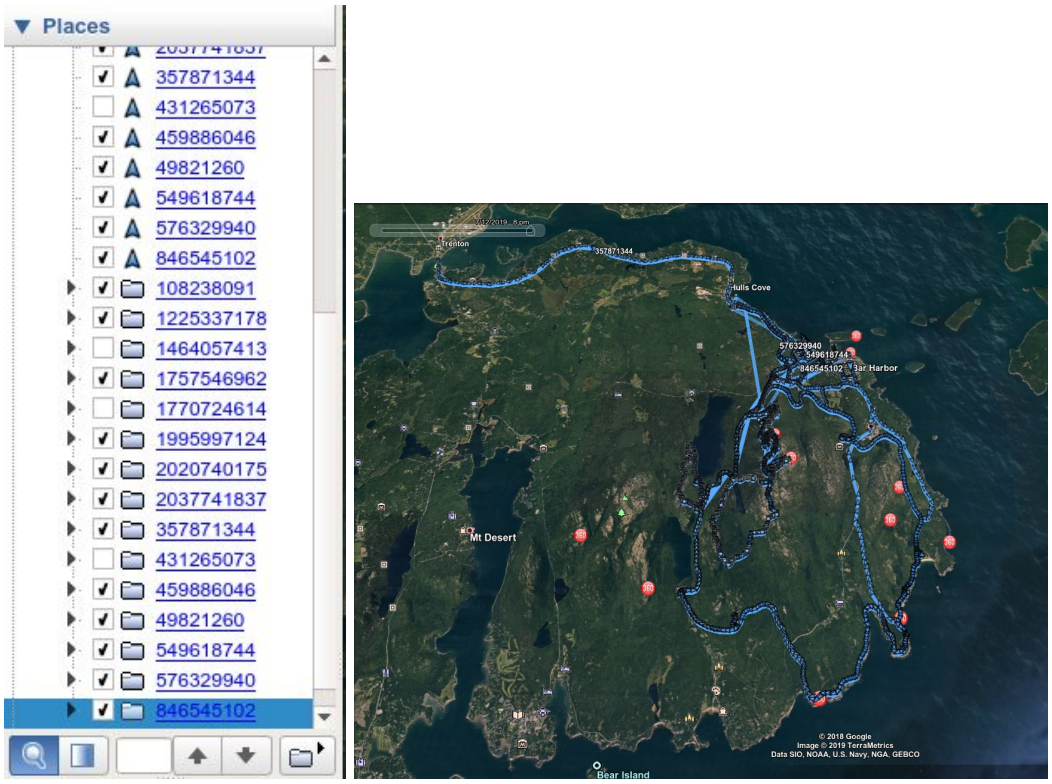


Figure 3.8: Screenshot of Google Earth with the available paths of each user and a visualization

This gave us our preliminary visualization technique. Then we transitioned to using ArcGIS for the visualization of GPS data, which are detailed in Section 4.2. Afterwards, we attempted to use SPSS Statistics to analyze the data collected, but it was not able to create usable data in time for the conclusion of the project.

4. Results and Analysis

4.1 Survey Results and Analysis

The survey as described in Appendix D was distributed to seventy-three participants over the course of two days at the Visitor Center. It paints a very interesting picture of the average first-time visitor of Acadia National Park and their receptiveness to a mobile tracking application. This is also a continuation and affirmation of survey data that was collected by last year's WPI Visitor Tracking team as it was collected in the same location and concerns a similar subject matter.

The first question had visitors indicate which smartphone platform they used. The main purpose of this question to assess the viability of the target platform for Acadia Visitor Study and whether it would be advantageous to port the application to other platforms. The results are shown as seen below:

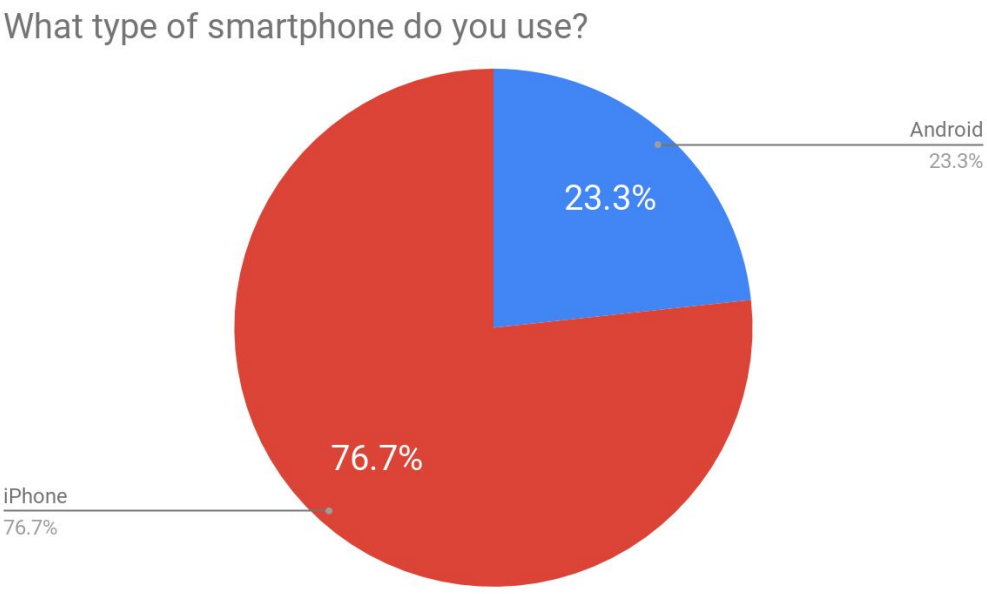


Figure 4.1: Results of Question 1

The response to this question was surprising because it was not consistent with the current worldwide mobile operating system market share, where Android represents 71.62 percent of the market as of June 2019 (“Operating system market share,”, 2019.). In comparison, our results showed that 23.3 percent of respondents indicated that they were Android users whereas 76.7 percent indicated that they were iOS users. A potential explanation for these results may be that most people that visit Acadia National Park’s Visitor Center are first-time guests, indicating that most of them are tourists (Nadeau et al., 2018). Since most people who will go on vacation have some disposable income, it would make sense that many people would also be able to afford an iPhone, since they are more expensive, on average, when compared to many Android phones. Regardless, this may have drastic implications for the deployment of Acadia Visitor Study, since most people surveyed would be unable to download and use it. Based on this result, it would be beneficial for future groups to pursue development of an iOS application in addition to an Android application due to the potential iOS user base.

The second question posed to the visitors was whether they used a phone or a map to navigate through the park. The main purpose of this question was to assess the viability of incorporating the existing tracking functionality in Acadia Visitor Study into another application used for navigation. That way the user would have outside incentive to download and use the application. The results are shown below:

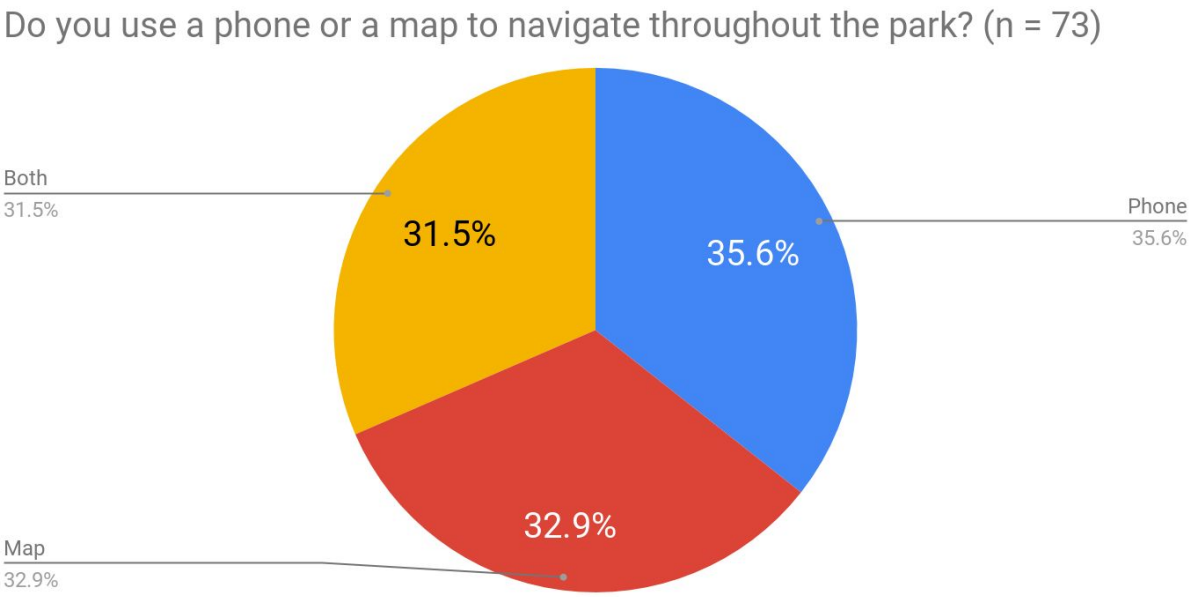
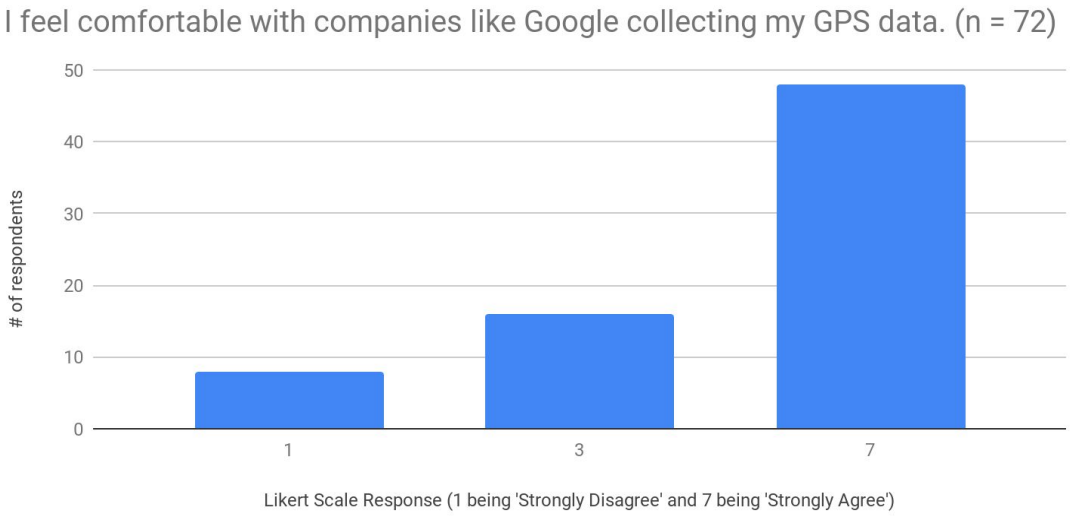


Figure 4.2: Results of Question 2

It appears that there is an almost even split between respondents that would either use a map, phone, or both for navigation, with 32.9 percent of respondents indicating that they would use a map, 35.6 percent of respondents indicating they would use a phone, and 31.5 percent of respondents indicating they would use both while navigating through Acadia National Park. Additionally, with this data, it can be stated that 67.1 percent of respondents would use their phone for navigation at some point during their stay at the park. Therefore, there is a pre existing majority of visitors who already use a mapping application, therefore also using the GPS functionality on their phones.

The third question posed to the respondents was whether they were comfortable with companies like Google collecting GPS data from users. The main purpose of this question was to get the general opinion of tracking software in existing applications. The results are as shown below:



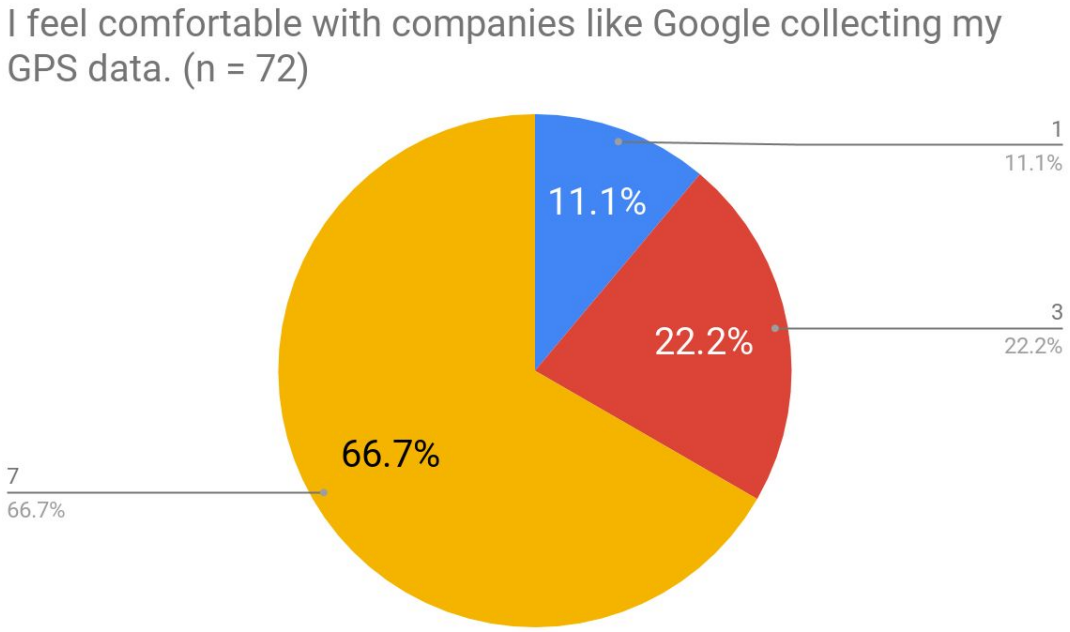
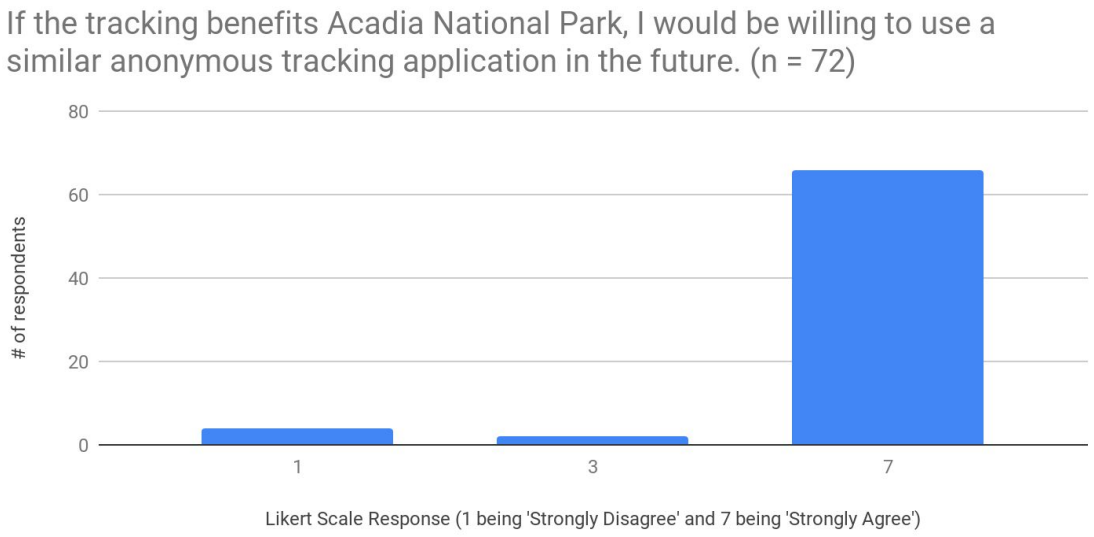


Figure 4.3: Results of Question 3

As seen by the previous graphs, visitors of Acadia National Park generally accept various tracking technologies employed by companies like Google, with around two-thirds of respondents feeling comfortable with companies tracking users in general. However, there is a notable minority of approximately 11 percent of respondents that are absolutely not comfortable

with this, and 22.2 percent who would slightly disagree with being comfortable with the technology. Additionally, many of the respondents that were comfortable with the tracking did so apathetically, with many stating that Google already had most of their data anyway through its other products. However, these results changed drastically when an incentive to help the park was introduced. To assess this, the respondents were asked if they would be willing to download an application that would have this tracking functionality similar to Google, but it would be anonymized and would benefit Acadia National Park. The results of this question are graphed below:



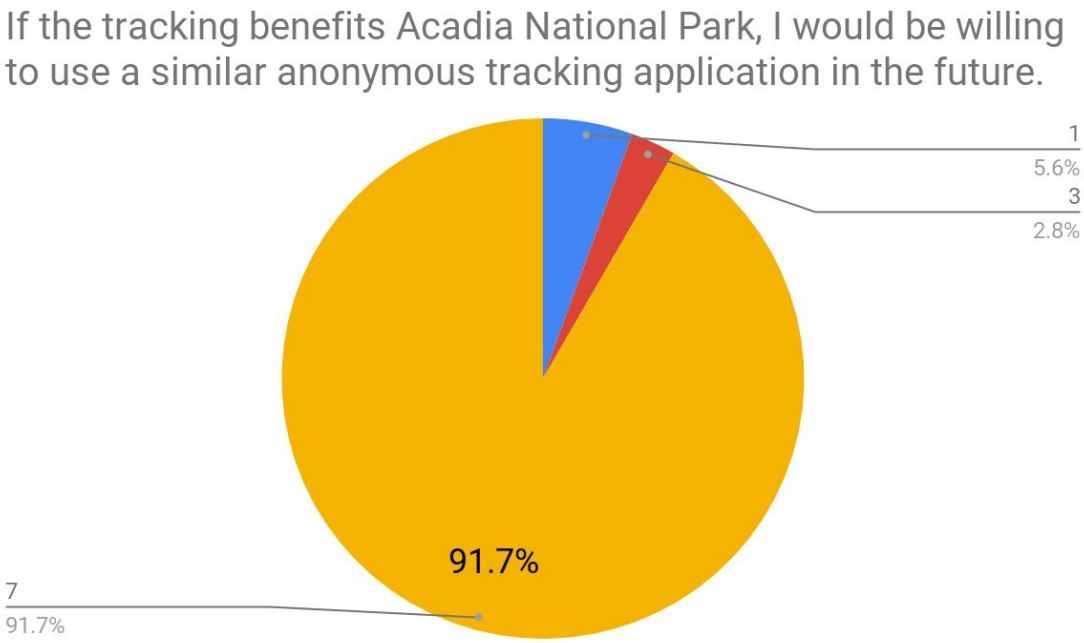


Figure 4.4: Results of Question 4

As both graphs clearly show, there was a vastly positive response to downloading a phone tracking application if it helps the park with visitor congestion and if the data was collected anonymously. 91.7 percent of respondents indicated that they would be download the application. However, while the minority of respondents who refused to download the application is smaller than the previous question, they were more vocal about their refusal citing reasons such as decreased battery life and low incentive to download the application. Additionally, these results are not very surprising due to similar studies in the past. For example, a study that was conducted in Acadia National Park in 2004 to assess the role of GPS trackers as a way to monitor traffic in the park indicated that 4.6 percent of respondents refused to place GPS trackers in their cars for the purpose of data collection (Hallo, Manning, Valliere, & Budruk, 2005).

Although it appears that respondents would be willing to download a mobile tracking application such as Acadia Visitor Study, many would be unable to download the application

because they own an iPhone. However, there is a majority of willing users that a park-sponsored tracking application could take advantage of.

4.2 Data Visualization

As stated in Section 3.4, we primarily used ArcGIS as a tool to visualize the data collected through our application. It is a particularly powerful program that converts data into a visual and intuitive format, portraying the data using multiple parameters. Although ArcGIS can import both tab-delimited text files and GPX files, the time data collected by Acadia Visitor Study had to be converted to a string format that ArcGIS could convert easily to a time data type. So, we created a Python script that takes in a tab-delimited text file and converts it to a useable time format, which is conveniently also human-readable. The source code of this can be found in Appendix H.

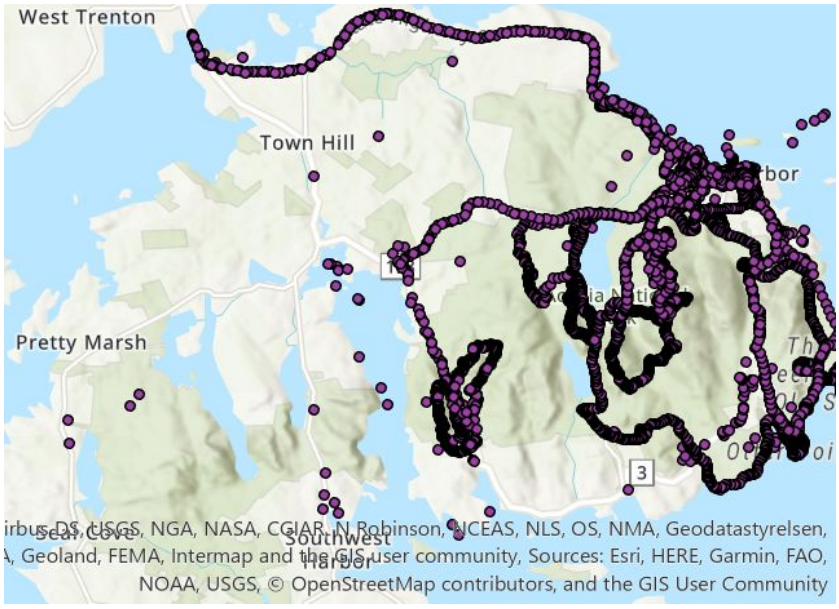


Figure 4.5: Initial Visualization of the Data Collected using ArcGIS

When initially imported into ArcGIS, it was clear that the location is very accurate and sensitive to small changes, even in areas with a low data connection. Even though this is the case, we found that in rare circumstances the data will shift erratically if there is limited connection to GPS. However, these inaccuracies can easily be edited to make the data more

usable. From the data collected, it is observable that the geofencing functionality of our application is working, as there are no points from outside of Mount Desert Island or the Schoodic Peninsula after geofencing was implemented. Once the data was cleaned up from application debugging sessions, we started to create some visualizations.

The first method of visualization was done by using only a single variable. This method used UID to distinguish the users. This was a good test case because it proves that the program can be used to distinguish points by many other use characteristics in the future like demographics for example. In order to accomplish this, we used the Symbology function to change the color of the data generated by each user to create this graph:

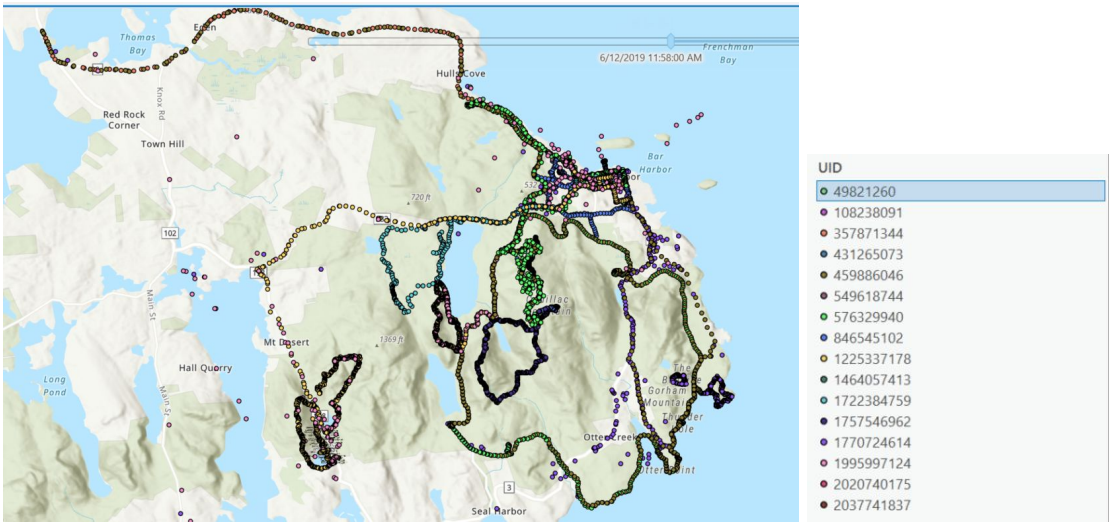


Figure 4.6: Visualization of the Data categorized by UID

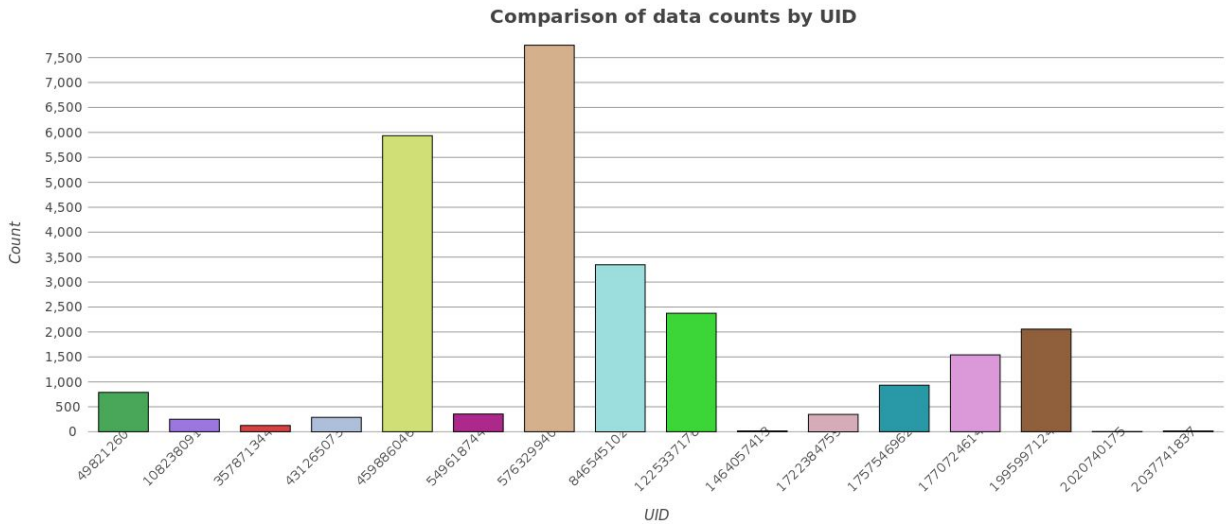


Figure 4.7: Comparison of data Counts by UID

As seen by both the graph and the chart, it is relatively simple to create a categorical visualization based on location and on a single variable. Through this process, we also discovered that ArcGIS can keep track of the frequency of each category used, which is important when creating heat maps. We then converted the existing data to the GPX format to get elevation values then graphed the elevation of each user:

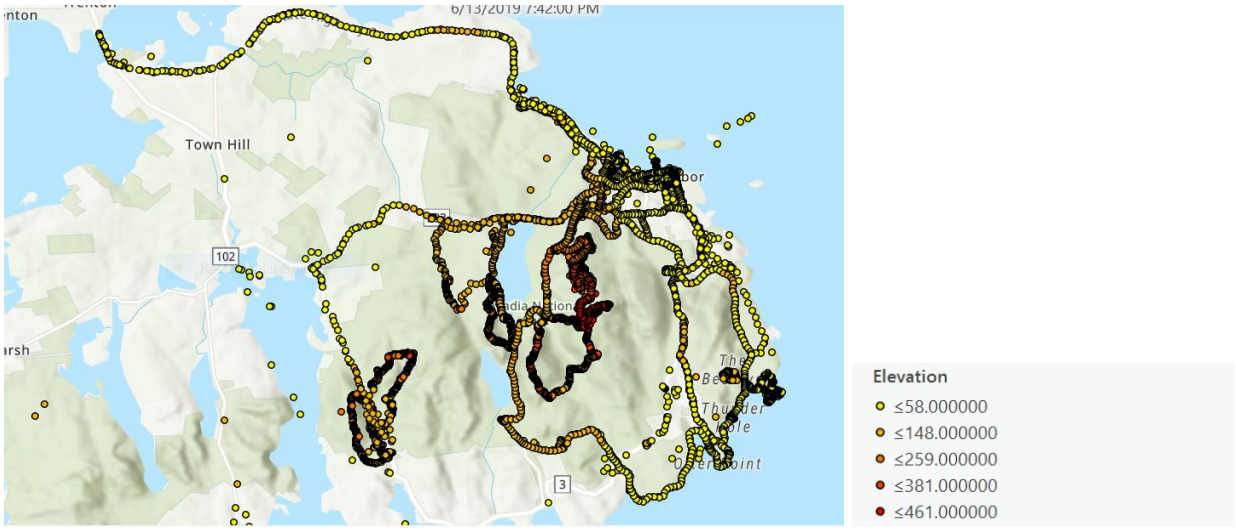


Figure 4.8: Visualization of Data Collected with each data point representing a different elevation

We discovered that the program has the capability to run basic statistics to generate charts. Additionally, ArcGIS has native support for analyzing data in a given time range. This is a very useful feature when looking at historical data, as well as looking at very particular portions of the data. It also has native animation support, meaning that is has the ability to animate the visualizations over time. We were able to use these features to give an animated visualization of a single user that went throughout Mount Desert Island. This animation was useful to extrapolate much more data than is strictly collected. We were able to understand the users method of transportation as well as the attractions they visited in the park that day.

We then attempted to produce multivariable visualization. We used the Symbology function to create graphs that incorporated multiple data fields. Our first attempt was visualizing both User ID and velocity fields represented by both the color and size of each data point respectively:

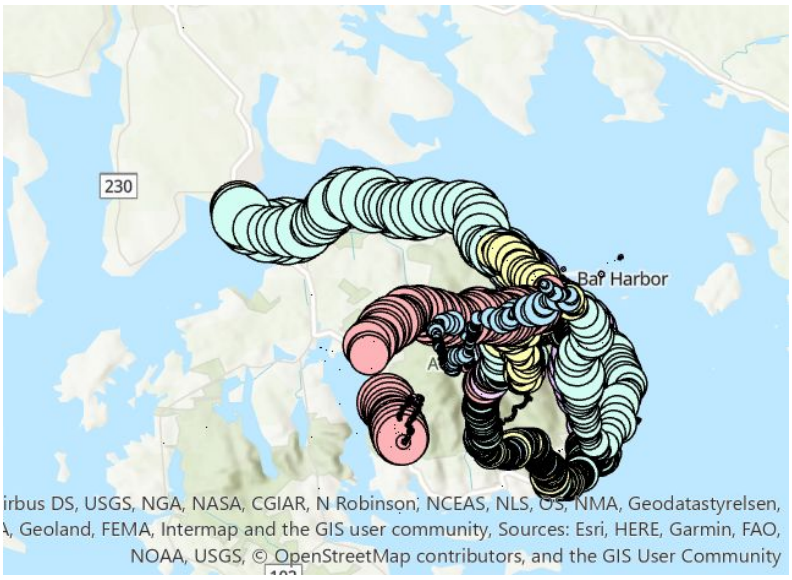


Figure 4.9: Visualization of Data using both velocity and UID.

As seen by our initial visualization, the WPI students made use of cars along the main roads, which are demarcated by the larger data points. We then focused on a period of time when we knew that a student was hiking on one of the trails, and verified that there was a noticeable difference between the size of the data points when walking and when driving:

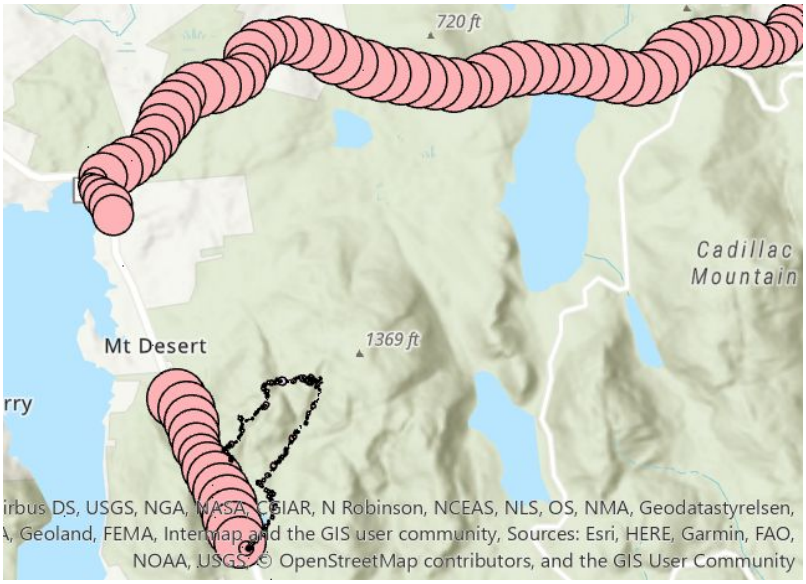


Figure 4.10: Visualization of a single user driving then hiking

ArcGIS's heatmapping feature is another way to visualize the data collected. In short, ArcGIS takes in a 'weight' value such as velocity and is able to plot it out on a map based on the density of the weights of all local data points. As such, we began to have difficulties when using this visualization, as most of the students tended to stay on the College of the Atlantic campus because they were staying there. So the density of the data at that location washed out the density of the students that went out into the park. The heat map below illustrates the weighting issue:

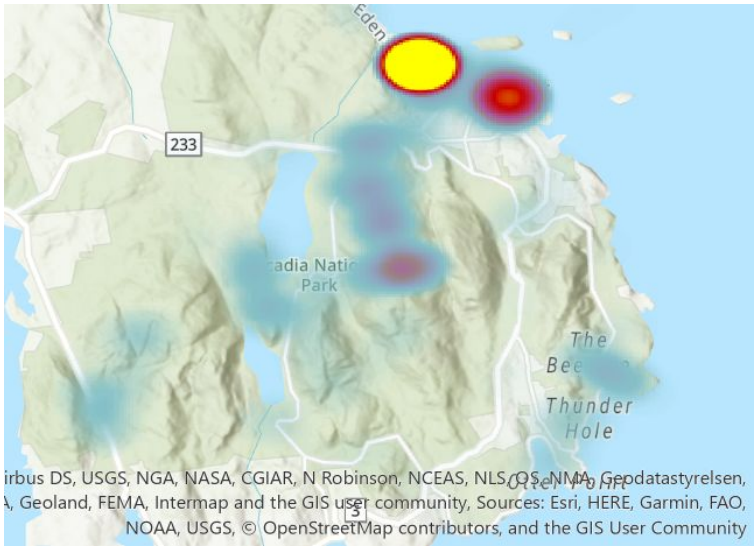


Figure 4.11: Initial density visualization

In order to get a more accurate graph showing the most visited areas of the park, we reduced the area that the heatmap covered to not include the College of the Atlantic. ArcGIS then dynamically adjusts the weight of the heatmap to only include the weight of the currently viewed points. Since many of the students either went to Cadillac Mountain to check on sensors, and to hike, most of the points obtained are from those areas. This can be observed from the visualization below:

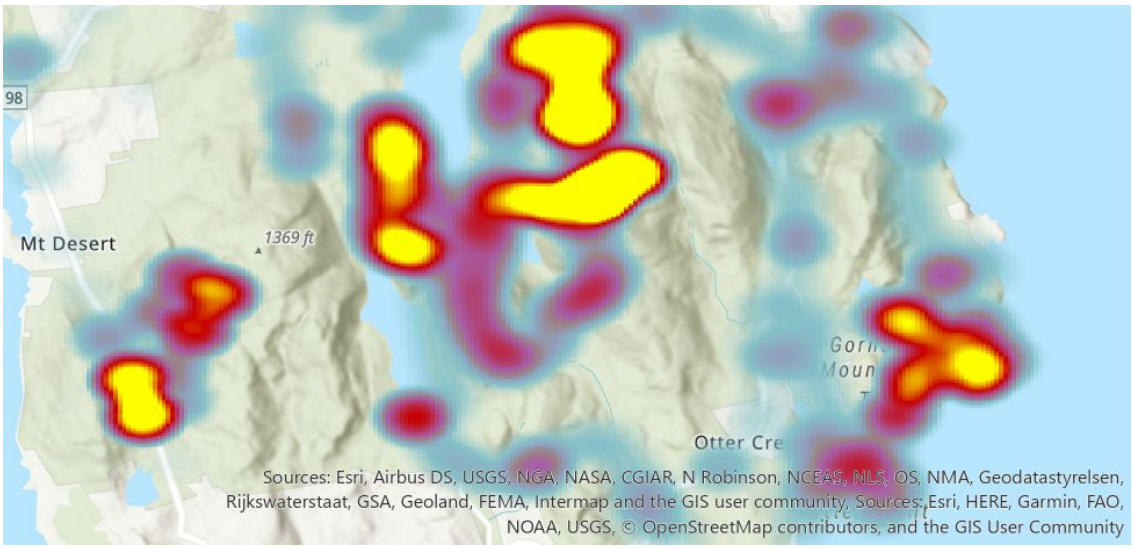


Figure 4.12: Final density visualization

The final visualization that we attempted was to weigh each point based on its velocity, which unsurprisingly highlights Park Loop Road, roads taken to get to any hiking trails, the road up to Cadillac Mountain, and Bar Harbor Road, which is used to get on and off of the island. However, this cannot be used to accurately indicate the travel frequency on specific roads unless the velocity of all cars on all roads observed are the same. This visualization shows where cars traveled the fastest.

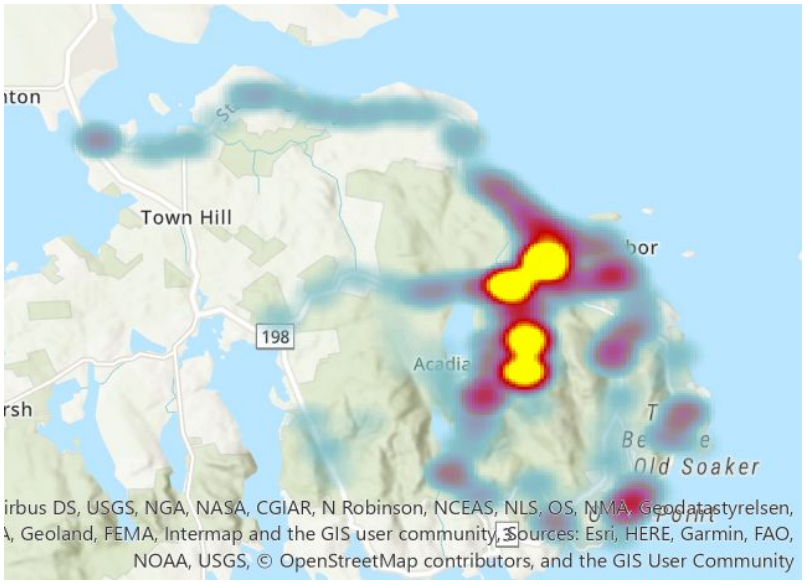


Figure 4.13: Heatmap with velocity as the weight of the data

While the information that was collected as a part of this study has limited value to the park due to the extremely limited sample set, a few conclusions can be made. First, the data collected by the application is accurate and can accommodate the low-reception areas of the park well, as there was little data loss when users were traveling in those areas. Second, ArcGIS is a feature-rich software package that can make various charts and visualizations using existing data sets and can easily handle situations with single or multiple variables, making it ideal for importing large data sets for visualization. Finally, while ArcGIS does not have the ability to do predictive analytics on its own (although there are extensions to do so), these visualizations show that Acadia Visitor Study is an effective way to collect data on visitors that will aid the park staff in handling congestion.

Looking to the future, the data that we collected has useful applications for many of the problems that the park faces. However, there is a potential for much deeper analysis. Assuming that the park can get an app to many visitors, they will be able to collect millions of data points a year. These data points will plot similarly to the data that we have already visualized. However, the trends that emerge will be much more meaningful. The concept of predictive analytics is nothing more that enhanced observation: when enough data is collected over time, patterns will begin to appear. Direct causality for a trend may not be immediately obvious, but given enough

observations, the certainty with which you can predict outcomes increases (Abbasi, Lau, & Brown, 2015). Therefore, when the park has millions of data points over several years, they will be able to accurately predict the behavior of visitors before they ever arrive.

To demonstrate this, we created mockups as to how this data could be visualized and interpreted:

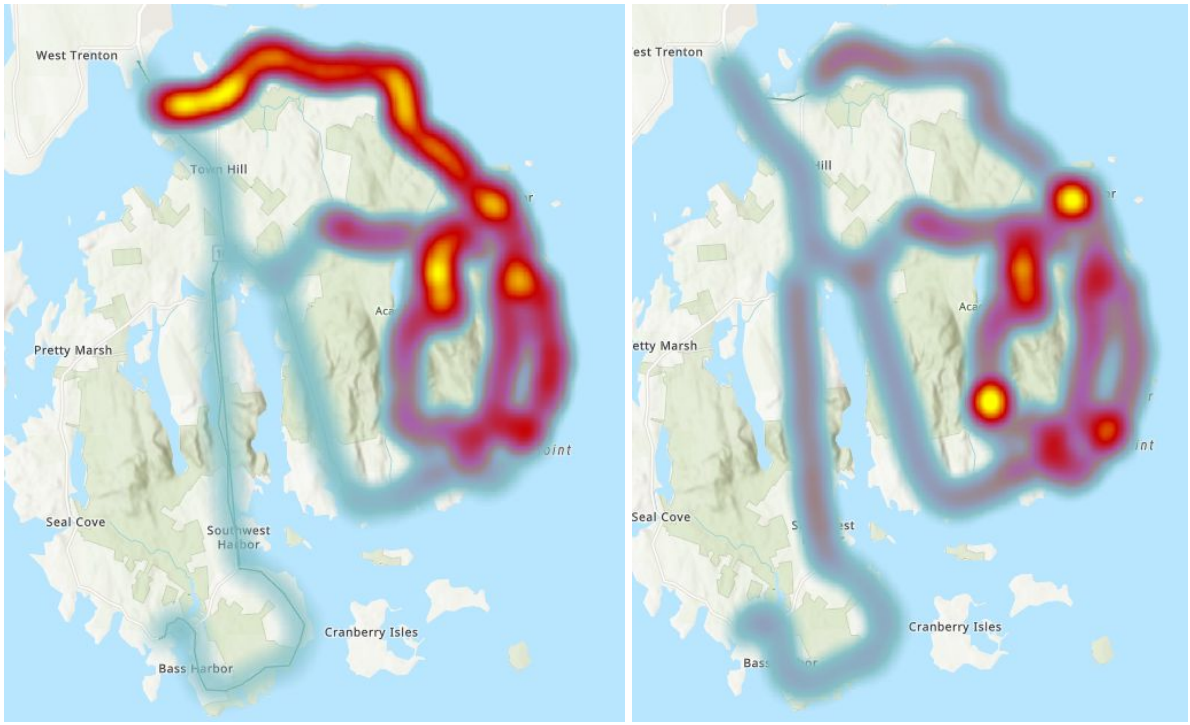


Figure 4.14: Mockup visualizations of a two days in Mount Desert Island where gas prices are extremely low (left) and where they are higher (right).

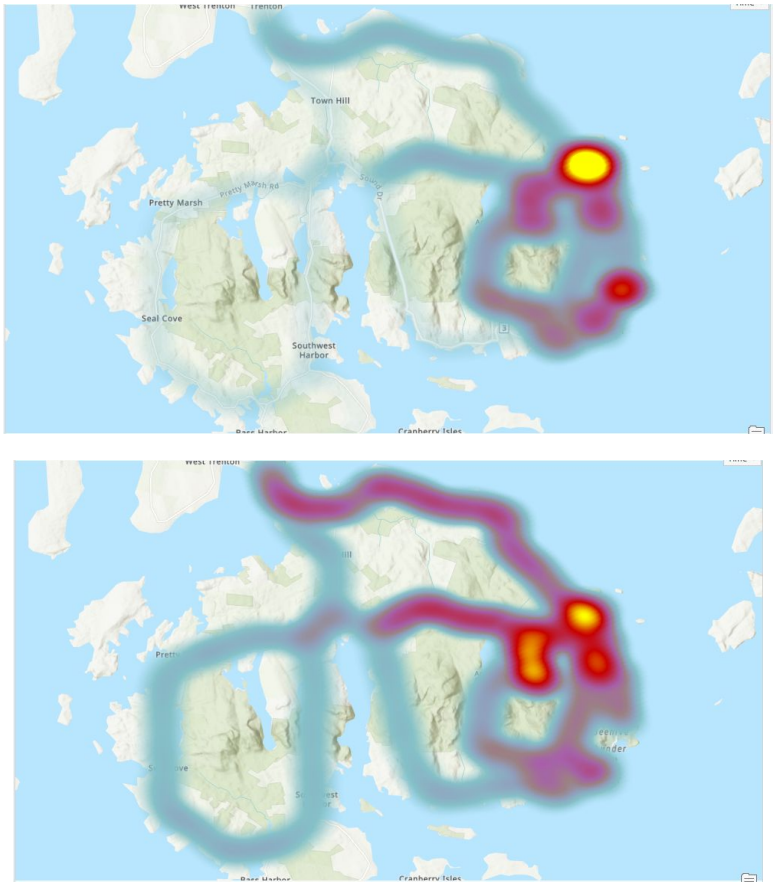


Figure 4.15: Mockup visualizations of a two days in Mount Desert Island where a cruise ship is docked (top) and where a cruise ship is not docked (bottom).

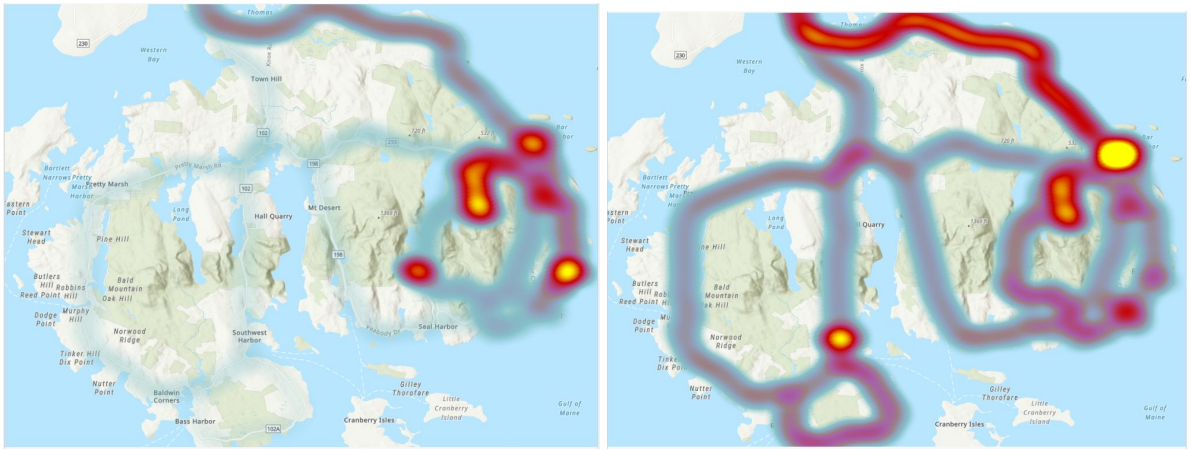


Figure 4.16: Mockup visualizations of a two days in Mount Desert Island before the Final Transportation Plan goes into effect (left) and afterwards (right).

As seen by these mockup visualizations, predictive analysis could be used to analyze a wide variety of variables such as fluctuating gas prices, cruise ship schedules or even the roll out of the Transportation Plan itself. For instance, the first mockup visualization (Figure 4.14), when gas prices are low, Route 3 and Park Loop Road are the most visited destinations in the park due to the fact that more visitors could travel to Acadia National Park by car from outside of the island. In contrast, when gas prices are higher, as seen by the right figure, less people could potentially arrive in the park by car and therefore rely on the Island Explorer to get around the park. The various stops of the Island Explorer are highlighted in that mockup highlighted. In the second mockup (Figure 4.15), whenever a cruise ship is docked, the most densely traveled area could be Bar Harbor, whereas whenever it is not docked, the total density of visitors can be distributed throughout the park more evenly and the data would become less washed out. Finally, in the last mockup (Figure 4.16), it shows that before the Final Transportation Plan is in effect, the most densely traveled areas could be at the main attractions such as the Jordan Pond House and Cadillac Mountain, whereas after it takes effect, the visitation could be much more spread out to other areas of Mount Desert Island.

Predictive analysis can help the park in many ways. First, they can transition from being reactive to traffic patterns, to being proactive in dealing with excessive traffic days. They are working with a limited amount of resources so efficiency is the key. When more visitors come to the park many of the services the park supplies are spread thin. For instance, over the July 4th weekend of this year, park officials and emergency responders at Acadia National Park were struggling to keep up with the demand of visitors. There were over seven-hundred radio calls to park officials, and many common roads such as Ocean Drive were temporarily shut just to respond to these calls because traffic had become such a barrier to emergency vehicles (Broom, 2019). Therefore, predictive analytics can enable the park to preemptively place their resources in the most useful locations, hopefully circumventing, or preventing such traffic issues all together. Overall, Acadia Visitor Study has the potential to collect the raw data to make this possible.

4.3 Possible Improvements

There were several aspects of the project that could have been improved to enhance the overall outcome of the research. The most notable issue was with getting the app onto the phones of users. This issue arose late in the project, and there was not enough time to respond effectively. Google Play Protect, as previously mentioned, labeled the app as malicious software. Since the application was being installed via .APK file, not through the Google Play Store, and because the previous group did not have the same problem, the assumption was made that google would not block the application from getting onto all devices. However, Google had updated their security policies since the last project, meaning that Google Play Protect now scans all .APK files downloaded onto the device. Had we looked into the security regulations much earlier in the development process, accommodating Google's security requirements would have been possible.

Although the application was not successfully roll out to the public, the visitor survey uncovered another piece of data that shows room for improvement: 76.7 percent of all the visitors polled use iOS. The Android development platform was chosen because our team had more experience in Java, and thought Google had less strict app regulations. However, given the survey findings, it is clear; the application should have been developed for iOS first in order to get the app into the hands of as many users as possible.

Many visitors were apathetic towards large companies taking their data, or they saw it as a worthwhile tradeoff for the services those companies provide. The initial design of the exit survey was conscious of user privacy and refrained from asking them any identifying questions. This sacrificed useful data, like demographics and state residence for the users' peace of mind. However, the survey data indicates that the exit survey could have had more detailed demographics questions without unsettling the majority of users. Therefore the exit survey should have included questions regarding detailed user information.

When developing the recommendations for the park to take this project to a larger scale, the team reached out to another popular park application: Chimani. The goal was to begin

facilitating a relationship between the park and Chimani so that at some point in the future, they could incorporate the app's functionality into their preexisting code. However, it takes much more time than what was available to generate a strong enough relationship to move forward with such a project, the two companies would have needed to begin to generate that relationship much earlier. Working so closely together would necessitate preliminary legal work that is beyond the scope of the project, and would take too long to complete. In the future, a project should either be more focused on generating such relationships from the start, or not get involved.

5. Recommendations and Conclusion

The main goal of the application is to show the park that there is value in taking on this project themselves, and rolling it out in a much larger scale. With this in mind, we came up with three possible solutions:

5.1 Future WPI Involvement in Development and Hosting

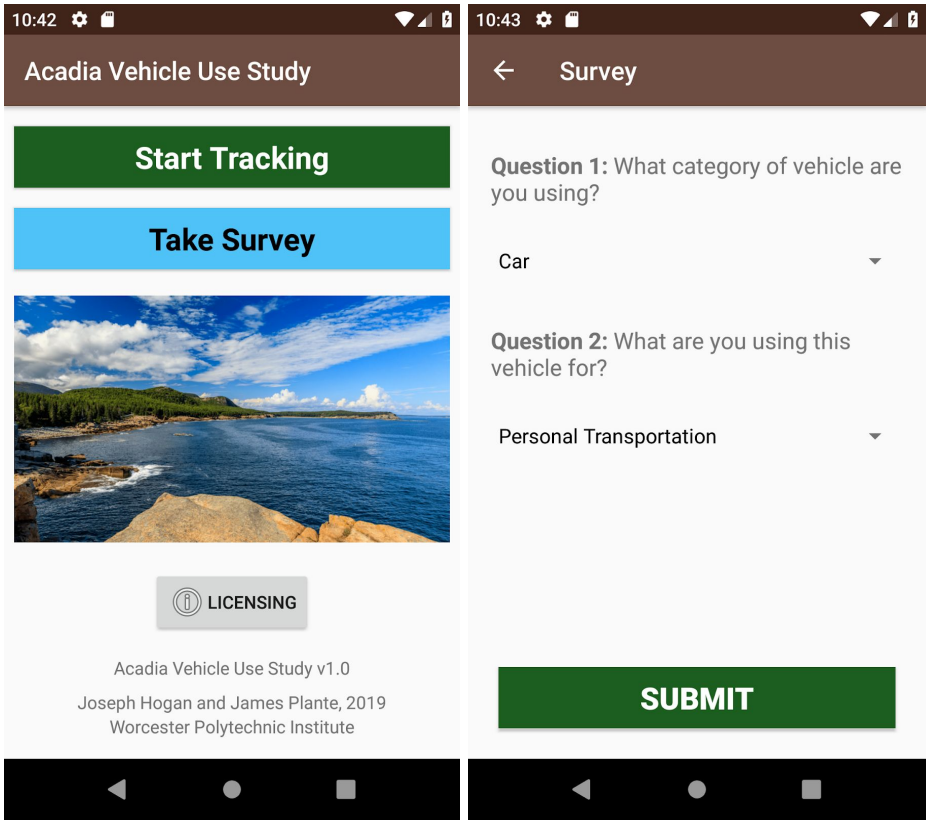


Figure 5.1: Acadia Vehicle Use Study Application

First, we suggest that WPI continues to support the development of Acadia Visitor Study for future use. This solution is beneficial because both the application and server are already written, making deployment easy for future groups. Additionally, the modularity of the application makes it possible for developers unfamiliar with Android Studio to expand the functionality of the application to suit their needs. For instance, there is a fork of Acadia Visitor Study called Acadia Ranger Study for use with rangers in Acadia National Park. In all, with the existing source code, it was easy to modify the existing code and re-brand the application for

another research groups use, with the total development time of the fork only taking a few days to implement. Also, there is another research group from WPI that is going to use our application in Glacier National Park to assess traffic congestion. Finally, the application's source code is licensed under the Apache v2 license, making it available for other groups outside of WPI to use.

However, there are a few drawbacks for continuing this solution. First, because of the nature of the IQP project, there will be many developers working on the project across several different years, which is not conducive to progress. Additionally, there is no iOS version developed yet, which could make visitor data collection difficult for future groups since according to the survey results from Section 4.1, most visitors have iPhones. However, the server will not have to be re-written to accommodate the new application as long as it sends a POST request with the same information and format as Acadia Visitor Study. Finally, as previously mentioned, distributing the application as an APK file was proven to be impractical due to restrictions from Google Play Protect, making publishing on the Google Play Store to be the only realistic option to get the application to users. It would take a considerable amount of time to publish this application on the Google Play Store due to the standards required.

5.2 Incorporation of App Functionality into Existing Applications

Our second recommendation is to insert the functionality of our app into an existing application. This has several benefits. First, we can benefit from the existing user base, and many people already use navigational applications in the park so there is already an incentive to use the application. With this in mind, we reached out to a developer who specializes in hiking applications for mobile devices, Chimani, to see if there was an interest. In response, they expressed an interest in expanding their application to incorporate tracking functionalities to benefit the park. This is a large opportunity for the park due to the potential user base, as observed by our survey results in Section 4.1. Chimani focuses on both iOS and Android development. So their application would be able to reach nearly all of the people that go into the park. Finally, since both iOS and Android applications are already written, it would not take much time to develop this functionality.

However, this approach has one major drawback: if the application ultimately used for incorporation is not what the average visitor would use while in the park, then it would not be effective at collecting data, since users do not have an incentive to use it. This can be mitigated by conducting studies that would assess what users at Acadia National Park would want to use. Then, a final decision could be made as to what application would be best to incorporate this tracking functionality. The park would also be unable to expand or add any features since they are not the main developer of the application.

5.3 OnCell

Finally, our strongest recommendation is that the park develop an application using OnCell's platform. OnCell is a development platform specially designed for informational GPS tours. It requires no coding on the consumer side. OnCell has worked with over seventy different national parks and monuments building GPS based tour and informational applications (Gilbert, 2019). Not only have they worked with the National Park Service in the past, they have just finished developing an application for the Friends of Acadia called Acadia Quest. This application provides interactive travel suggestions, blending information about the park with information about Friends of Acadia. The app also supports a GPS-based badge system, where registered teams can hike around the park competing to get the most badges. Friends of Acadia plans to expand their application to include things like online donation, volunteer registration, and park alerts (Steele, 2019). This preliminary application has paved the way for a smooth transition to the development of an official Acadia National Park app.

Drawing from the application developed for Friends of Acadia, the cost to build the application will be around $2500, and there will be a subsequent subscription-based charge for support, which will vary depending on the application's maintenance needs. In order to get the application operational with just the visitor tracking functionality, it will take the park an estimated 100 work hours to complete (Steele, 2019). Although the analytics still would not be done for the park, this is still the least disruptive way for them to get the GPS data they need.

OnCell provides a drag-and-drop interface for easy app design, and will take care of all the technical needs of the project. This lowers the technical barrier to entry for the park. As mentioned above they will take care of all application support such as system updates. However, the platform that OnCell provides does not support continuous GPS data collection. But, OnCell indicated that they would be willing to work with their development team to generate a slightly modified platform that will accommodate the needs of the project.

OnCell is in the process of becoming the official application vendor of the National Park Service, and Acadia National Park is in dire need of analytical solutions to their congestion issues. The barrier for Acadia National Park to develop this application is very low, and there are experienced individuals, individuals who have worked with OnCell for over two years, that are close to the park. They will act as a valuable resource when the park decides to develop their own OnCell application.

5.4 Conclusion

In all, the project sought to accomplish three objectives. The first objective was to develop an application as a proof-of-concept, designed only to track the GPS location of visitors to Acadia National Park as a way to monitor visitor congestion. While the application was operational in time to distribute to visitors, it was rejected by Google Play Protect, preventing its distribution. This was crucial because asking members of the public to disable this feature safety feature was not an option due to ethical concerns.

The rejection of the application caused a pivot to the second part of our project. We decided we could get sufficient data for assessment using ArcGIS from just WPI students. However, the survey that was in the application could not accurately represent the visitors if it was only filled out by WPI students. Therefore, we administered an edited version of the survey that was in the application to visitors at the Visitor Center, thus concluding there is a willing user base for Acadia Visitor Study, even though we were not able to distribute it to them.

Once the data was collected, we transitioned to the third and final phase of the project. We analyzed the data, and generated several options for moving forward with a large scale version of our application. We generated data visualizations using ArcGIS. From these visualizations we were able to show the wealth of data that can be collected from just the GPS data of the visitor. We then explored the future possibilities of big data analysis on a larger hypothetical data pool. We generated three recommendations: that WPI continue the development of the project, Acadia National Park incorporates the data collection functionality of our application into a preexisting application, or that they develop their own application from scratch. Regardless of what recommendations Acadia National Park pursues, they could aid in monitoring visitor congestion in the coming years.

Bibliography

Abbasi, A., Lau, R. Y. K., & Brown, D. E. (2015). Predicting behavior. *IEEE Intelligent Systems*, *30*(3), 35–43. https://doi.org/10.1109/MIS.2015.19

23rd Annual Highway Report. (2018, February 8). Retrieved April 20, 2019, from Reason Foundation website: https://reason.org/policy-study/23rd-annual-highway-report/maintenance-disbursements-per-mile/.

Beck, K., Beedle,M., Bennekum,A., Cockburn,A., Cunningham,W., Fowler,M.,...Thomas,D. (2001). *Manifesto For Agile Software Development.* Retrieved April 9, 2019, from http://agilemanifesto.org/.

Bell, D. (2004, September 15). *The class diagram*. Retrieved April 3, 2019, from IBM Developer website: http://www.ibm.com/developerworks/rational/library/content/RationalEdge/se04/bell/index.html.

Broom, Dick. (2019, July 11). Acadia was 'swamped' with hundreds of calls for help over July Fourth holiday. Retrieved July 25, 2019, from Press Herald website: https://www.pressherald.com/2019/07/11/acadia-national-park-swamped-with-calls-for-assistance-over-july-4th-holiday/

Budget Prioritizes Improvements to Critical Park Infrastructure while Saving Tax Dollars - Office of Communications (U.S. National Park Service). (2019, March 11). Retrieved April 14, 2019, from National Park Service website: https://www.nps.gov/orgs/1207/03-11-2019-budget-proposal.htm.

Cosmopulos, E., Gaulin, J., Jauris, H., Morisseau, M., & Quevillon, E. (2017). *Preparing Acadia National Park for Modern Tourist Congestion.* (Undergraduate Interactive Qualifying Project No. E-project-072717-222628). Retrieved from Digital WPI: https://digitalcommons.wpi.edu/iqp-all/2746/.

De Urioste-Stone, S. M., Scaccia, M. D., & Howe-Poteet, D. (2015). Exploring visitor perceptions of the influence of climate change on tourism at Acadia National Park, Maine. Journal of Outdoor Recreation and Tourism, 11, 34–43. https://doi.org/10.1016/j.jort.2015.07.001.

Download Android Studio and SDK tools. (2019). Retrieved April 17, 2019, from Android Developers website: https://developer.android.com/studio.

ESRI (2019). About ArcGis. Retrieved from https://www.esri.com/en-us/arcgis/about-arcgis/overview.

Fields, S., Gao, W., Goodale. L., Kirch, R., & Lin, J.. (2016). *The Carbon Footprint of Acadia National Park* (Undergraduate Interactive Qualifying Project No. E-project-072716-191454). Retrieved from Digital WPI: https://web.wpi.edu/Pubs/E-project/Available/E-project-072716-191454/unrestricted/Bar_Harbor_2016_Carbon_Footprint_IQP.pdf.

Fire of 1947 - Acadia National Park (U.S. National Park Service). Retrieved March 28, 2019, from https://www.nps.gov/acad/learn/historyculture/fireof1947.htm

Geissler, G. L., & Rucks, C. T. (2011). *The overall theme park experience: A visitor satisfaction tracking study*. *Journal of Vacation Marketing*, *17*(2), 127–138. https://doi.org/10.1177/1356766710392480

Gilbert, J. (2019, July 29). Personal Interview.

Hallo, J. C., & Manning, R. E. (2009). Transportation and recreation: A case study of visitors driving for pleasure at Acadia National Park. *Journal of Transport Geography*, *17*(6), 491–499. https://doi.org/10.1016/j.jtrangeo.2008.10.001

Hallo, J. C., Manning, R. E., Valliere, W., & Budruk, M. (2005). A Case Study Comparison of Visitor Self-reported Travel Routes and GPS Recorded Travel Routes. *In: Bricker, Kelly, Comp., Ed. 2005. Proceedings of the 2004 Northeastern Recreation Research*

Symposium. Gen. Tech. Rep. NE-326. Newtown Square, PA: U.S. Department of Agriculture, Forest Service, Northeastern Research Station: 172-177. Retrieved from https://www.fs.usda.gov/treesearch/pubs/9588

Hamill, P. (February, 2009). *Unit test frameworks*. Sebastopol (CA): O'Reilly.

History of Acadia - Acadia National Park (U.S. National Park Service). (n.d.). Retrieved March 28, 2019, from https://www.nps.gov/acad/learn/historyculture/history-of-acadia.htm

IBM (2019). Features and modules. Retrieved from https://www.ibm.com/products/spss-statistics/details

Kaiser, J. (2018). *Acadia: The Complete Guide* (5th ed.). Destination Press & ITS Licensors.

Lazar, J., Feng, J. H., & Hochheiser, Harry. (2010). *Research Methods in Human-Computer Interaction 2nd Edition*. Wiley.

National Park Service. (2019). *Acadia National Park Final Transportation Plan Environmental Impact Statement*. 282.

Nadeau, G. T., Charbonneau, J. L., Caltabiano, J. P., & Fischler, M. A. (2018). *Visitor Cell Phone Application: An Innovative Design to Monitor Visitor Mobility in Acadia National Park*. Worcester Polytechnic Institute.(Undergraduate Interactive Qualifying E-project-080218-145202). Retrieved from Digital WPI http://www.wpi.edu/Pubs/E-project/Available/E-project-080218-145202/unrestricted/trafficIQP.pdf.

Pencheon, D. (2012). People and planet: From vicious cycle to virtuous circle: Overpopulation, poverty, and environmental degradation share common solutions. *BMJ: British Medical Journal, 345*(7864), 9-9. Retrieved from http://www.jstor.org/stable/43512334

Okazaki, S., Navarro, A., Mukherji, P., & Plangger, K. (2017). The curious versus the overwhelmed: Factors influencing QR codes scan intention. *Journal of Business Research*. https://doi.org/10.1016/j.jbusres.2017.09.034

Operating system market share. (2019). Retrieved July 16, 2019, from NetMarketShare website: https://netmarketshare.com/operating-system-market-share.aspx?options=%7B%22filter

%22%3A%7B%22%24and%22%3A%5B%7B%22deviceType%22%3A%7B%22%24in%22%3A%5B%22Mobile%22%5D%7D%7D%5D%7D%2C%22dateLabel%22%3A%22Trend%22%2C%22attributes%22%3A%22share%22%2C%22group%22%3A%22platform%22%2C%22sort%22%3A%7B%22share%22%3A-1%7D%2C%22id%22%3A%22platformsDesktop%22%2C%22dateInterval%22%3A%22Monthly%22%2C%22dateStart%22%3A%222018-07%22%2C%22dateEnd%22%3A%222019-06%22%2C%22segments%22%3A%22-1000%22%7D

Quick History of the National Park Service (U.S. National Park Service). (2018, May 14). Retrieved April 14, 2019, from National Park Service website: https://www.nps.gov/articles/quick-nps-history.htm

Schroeder, T., Goddard, S., & Ramamurthy, B. (2000). Scalable Web server clustering technologies. *IEEE Network*, *14*(3), 38–45. https://doi.org/10.1109/65.844499.

Santana-Jiménez, Y., & Hernández, J. M. (2011). Estimating the effect of overcrowding on tourist attraction: The case of Canary Islands. *Tourism Management*. https://doi.org/10.1016/j.tourman.2010.03.013.

Server - Definition and Details. (2019). Retrieved April 20, 2019, from https://www.paessler.com/it-explained/server.

Steele, P. (2019, July 29). Personal Interview.

The Acadia Experience. (2019). Retrieved March 28, 2019, from Friends of Acadia website: https://friendsofacadia.org/programs/sustainable-visitation/.

Tao, L. W. (2017). *The drawbacks of housing overcrowding characteristic to rural migrants' life in Beijing*. *HBRC Journal*, *13*(3), 315–320. https://doi.org/10.1016/j.hbrcj.2015.11.002

Turkewitz, J., & Fremson, R. (2017, December 22). National Parks Struggle With a Mounting Crisis: Too Many Visitors. *The New York Times*. Retrieved from https://www.nytimes.com/2017/09/27/us/national-parks-overcrowding.html

Toto, E. (2019). Ermal Toto - Home Page. Retrieved April 23, 2019, from http://users.wpi.edu/~toto/

Walser, L. (2017, February 8). *How Mission 66 Shaped the Visitor Experience at National Parks | National Trust for Historic Preservation*. Retrieved April 14, 2019, from National Trust for Historic Preservation website: https://savingplaces.org/stories/how-mission-66-shaped-the-visitor-experience-at-national-parks

Walt Disney World Statistics. (2017). Disney World Statistics - The Truly Fascinating Numbers Behind Disney. Retrieved April 19, 2019, from Magic Guides website: https://magicguides.com/disney-world-statistics/

Wells, D.(October 8th, 2013). *Extreme Programming: A Gentle Introduction*. Retrieved April 9, 2019, from http://www.extremeprogramming.org

Yellowstone National Park Protection Act (1872) - Yellowstone National Park (U.S. National Park Service). (2018, July 13). Retrieved April 14, 2019, from National Park Service website: https://www.nps.gov/yell/learn/management/yellowstoneprotectionact1872.htm

Zinke Announces $35.8 Billion Added to U.S. Economy in 2017 due to National Park Visitation - Office of Communications (U.S. National Park Service). (2018, April 25). Retrieved April 14, 2019, from National Park Service website: https://www.nps.gov/orgs/1207/04-25-2018-economic-impact.htm

Figures:

Figure 1. Icon for Acadia Visitor Study. Reprinted from "Icon for Acadia Visitor Study" by Alatiq, A, 2019. Used with permission.

Figure 1.1. Number of recreational visitors to Acadia National Park in the United States from 2008 to 2018 (in millions). Reprinted [or adapted] from *Statista.com*, by National Park Service, 2019, https://www.statista.com/statistics/253808/number-of-visitors-of-acadia-national-park-in-the-us/.

Figure 2.1. Main Acadia National Park map, showing the majority of the park located on Mount Desert Island near Bar Harbor. Reprinted [or adapted] from *Wikimedia Commons*, by National Park Service, 2013, http://npmaps.com/wp-content/uploads/acadia-map.jpg.

Figure 3.5. Poster that was shown to visitors. Adapted from "Poster for Acadia Visitor Study" by Alatiq, A. 2019. Adapted with permission.

Appendix A: Unit Testing

Unit testing is a process that happens during app development. A program is made up of many small tasks, that work together in intricate ways. When a program crashes any number of these interactions could be the cause. Unit testing isolates each of these tasks and tests their functionality to quickly find the problem and is important for project management and integration (Hamill, 2009). In our project, we will be using the white box method for unit testing our application. White box testing is when the person testing the code, typically the software developer, knows how it was written so that changes can be made real time (Wells, 2013). Additionally unit tests can be reused as the project increases in complexity each iteration, which allows testing for any feature regressions. We will be using JUnit testing suite to test the application . This is the best solution for our team because JUnit interfaces very easily with Android Studio and we have had experience working with JUnit in the past for undergraduate work. Overall, test-driven design is essential to the agile development process.

Appendix B: Survey Questions

- What type of phone do you have?
- Do you use a phone or a map to navigate throughout the park?
- I feel comfortable with companies like Google collecting my GPS data.
- If the tracking benefits Acadia National Park, I would be willing to use a similar anonymous tracking application in the future.

Appendix C: Interview Questions

The answers provided during this interview with Mr. Toto will be paraphrased for brevity.

Question 1: How will we be able to handle spotty coverage in the park, since the application needs internet access in order to send data? How will we be able to store data locally?

Answer: You will need to have a SQLite database on the phone itself, and you can record the data as JSON objects or just plaintext. The best way would be to record them as JSON objects. Then, you can sync that SQLite database with another SQLite database on the server. They are identical copies, except with the app you can just purge it, since you don't want it to grow too much. Make sure to sync with the server once you have the data you need.

Question 2: What steps do we need to take to ensure the security of the individuals' location data when using the application?
Answer: First, you should have an SSL certificate on the server to ensure that the data is encrypted and make sure the application can communicate with the encrypted port. Second, do not store any personally identifiable data on the server itself. The network traces can technically identify the person, but do not store their name or anything that identifies them. Come up with a unique identifier per person and base the data on that.

Follow up: Is there a universal common standard for doing this?
Answer: It depends. Of course, you could get the phone ID that clearly identifies them. If you want to go a step further, you generate a random number and you make sure the number does not already exist on the database itself. Then you would store everything based on that number. It is important that you do not use the same number for two people. You should generate the number on the server, which is how the session will start, and then they submit the data with that number.

Question 3: We want to be able to disable the application at certain times, especially when the visitor would exit the park. How would we be able to handle that situation?
Answer: You can create a geofence inside of the application around the GPS locations of the parks. This way, if a user were to step outside of the bounds of the park, the background process running the data collection can be disabled. Thus, the tracking component of the application will be disabled until the user turns it back on.

Question 4: Since we do not need to display the person's location while using the app, we have thought to strip down the Google Maps API to only track user movements throughout the park. Is this the correct approach?
Answer: Most likely, you would not need the Google Maps API at all. You could map Acadia National Park with a set of points instead of using the API to do that for you and you could create your own geofence. You could figure out a polygon of where the park is and detect if you are in that polygon to detect if you are in the park or not. You should also put a buffer so there is a margin of error in visitor detection.

Follow up: So, is accessing the GPS data on the phone itself similar to a camera or other sensors on the phone itself?

Answer: Yes, it is built-in. The GPS is available to apps to use and you do not need an API to gain access to it. While Android Studio may recommend the use of the Google Maps API to track user location, the location sensors on the phone itself could give the current latitude, longitude, and likely altitude depending on the phone. If the altitude is not provided from the phone, you will likely need the Google Maps API to do so.

Appendix D: Apache 2.0 License

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by

combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Appendix E: Application Source Code

MainActivity.java

```
package com.example.acadiavisitorstudy;

import android.Manifest;
```

```
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.ServiceConnection;
import android.content.SharedPreferences;
import android.content.pm.PackageManager;
import android.location.Location;
import android.os.Bundle;
import android.os.IBinder;
import android.preference.PreferenceManager;
import android.support.annotation.NonNull;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v4.content.LocalBroadcastManager;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

import com.google.android.gms.location.FusedLocationProviderClient;

public class MainActivity extends AppCompatActivity {

        // Middle button
        static private Button trackingButton;

        // Used to check button state (should be ported to a preference
        static private boolean ifNotTracking;

        private FusedLocationProviderClient fusedLocationClient; // Fused Location Client

        // Class tag
        private static final String TAG = "MainActivity";

        // Permission code number (doesn't matter what it is)
        private static final int LOCATION_PERMISSION_CODE = 0;

        // ID for the notification channel
        private static final String CHANNEL_ID = "AcadiaVisitorStudy";

        // A reference to the service used to get location updates.
        private LocationUpdatesService mService = null;

        // The BroadcastReceiver used to listen from broadcasts from the service.
        private MyReceiver myReceiver;

        // Tracks the bound state of the service.
        private boolean mBound = false;

        private final ServiceConnection mServiceConnection = new ServiceConnection() {
```

```
@Override
public void onServiceConnected(ComponentName name, IBinder service) {
Log.d(TAG, "onServiceConnected: This method is called");
LocationUpdatesService.LocalBinder binder = (LocationUpdatesService.LocalBinder) service;
mService = binder.getService();
mBound = true;
}

@Override
public void onServiceDisconnected(ComponentName name) {
mService = null;
mBound = false;
}
};

/**
* Receiver for broadcasts sent by {@link LocationUpdatesService}.
*/
private class MyReceiver extends BroadcastReceiver {
@Override
public void onReceive(Context context, Intent intent) {
Location location = intent.getParcelableExtra(LocationUpdatesService.EXTRA_LOCATION);
if (location != null) {
        Toast.makeText(MainActivity.this, LocationHelper.getLocationText(location),
                Toast.LENGTH_SHORT).show();
}
}
}


@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

myReceiver = new MyReceiver();

LocationHelper.requestingLocationUpdates(this);

setContentView(R.layout.activity_main);
trackingButton = (Button) findViewById(R.id.tracking_button);

// If there is no user ID, we want to create it

// If the application is not launched for the first time
// reset the buttons.
ifNotTracking = true;
}

@Override
protected void onStart() {
super.onStart();

PreferenceManager.getDefaultSharedPreferences(this);


// Bind to the service. If the service is in foreground mode, this signals to the service
```

```
        // that since this activity is in the foreground, the service can exit foreground mode.
        bindService(new Intent(this, LocationUpdatesService.class), mServiceConnection,
                Context.BIND_AUTO_CREATE);
        }

        @Override
        protected void onResume() {
        super.onResume();
        LocalBroadcastManager.getInstance(this).registerReceiver(myReceiver,
                new IntentFilter(LocationUpdatesService.ACTION_BROADCAST));
        SharedPreferences settings = getSharedPreferences(getString(R.string.pref_file),
Context.MODE_PRIVATE);
        ifNotTracking = settings.getBoolean("ifNotTracking", true);
        changeButtonState(!ifNotTracking);
        }

        @Override
        protected void onPause() {
        LocalBroadcastManager.getInstance(this).unregisterReceiver(myReceiver);
        super.onPause();
        }

        @Override
        protected void onStop() {
        if (mBound) {
        // Unbind from the service. This signals to the service that this activity is no longer
        // in the foreground, and the service can respond by promoting itself to a foreground
        // service.
        unbindService(mServiceConnection);
        mBound = false;
        }
        SharedPreferences s = getSharedPreferences(getString(R.string.pref_file),
Context.MODE_PRIVATE);
        s.edit().putBoolean("ifNotTracking", ifNotTracking).apply();
        super.onStop();
        }


        /***
        * Method to start collecting data
        * @param view
        */
        public void startCollection(View view) {
        if (ifNotTracking)
        {
        // Check location permissions
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION)
                != PackageManager.PERMISSION_GRANTED) {
                // Need to request permissions.
                requestLocationPermission();

        } else {
                // Notify the user that the tracking has started
                Toast.makeText(this, R.string.on_track_start, Toast.LENGTH_SHORT).show();

                changeButtonState(ifNotTracking);
```

```
                // Start up the location service
                mService.requestLocationUpdates();

                ifNotTracking = false;
        }

        } else {
        // Stop the foreground service
        mService.removeLocationUpdates();

        changeButtonState(ifNotTracking);

        // Notify the user that we are not tracking anymore (we promise)
        Toast.makeText(this, R.string.on_track_stop, Toast.LENGTH_SHORT).show();

        ifNotTracking = true;
        }
        }

        /***
        * Changes the state of the tracking button depending on when it is being tracked.
        * @param ifNotTracking
        */
        private void changeButtonState(boolean ifNotTracking) {
        if (ifNotTracking) {
        // Set to red and stop
        trackingButton.setText(R.string.tracking_button_text_stop);
        trackingButton.setBackgroundResource(R.color.stop_t);
        } else {
        // Set to green and start
        trackingButton.setText(R.string.tracking_button_text_start);
        trackingButton.setBackgroundResource(R.color.start_t);
        }
        }
//comment to check

        /***
        * Launches the SurveyActivity activity
        * @param view
                startActivity(intent);
        }
        /***
        * Launches the LicenseActivity activity
        * @param view
        */
        public void viewLicense(View view){
        Intent intent = new Intent(this, LicensingActivity.class);
        startActivity(intent);
        }

        /***
        * Launches the InstructionActivity activity
        * @param view
        */
        public void viewInstructions(View view){
        Intent intent = new Intent(this, InstructionActivity.class);
```

```
        startActivity(intent);
        }

        /**
        * Method to get user permissions about location. Displays a message box if permission is
        * initially denied.
        */
        private void requestLocationPermission() {
        if (ActivityCompat.shouldShowRequestPermissionRationale(this,
Manifest.permission.ACCESS_FINE_LOCATION)){

        new AlertDialog.Builder(this)
                .setTitle("Permission needed")
                .setMessage("Permission is needed to use the GPS of your phone.")
                .setPositiveButton("ok", new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                        ActivityCompat.requestPermissions(MainActivity.this, new String[]
{Manifest.permission.ACCESS_FINE_LOCATION}, LOCATION_PERMISSION_CODE);
                        }
                })
                .setNegativeButton("cancel", new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                        dialog.dismiss();
                        }
                }).create().show();

        } else {
        ActivityCompat.requestPermissions(this, new String[]
{Manifest.permission.ACCESS_FINE_LOCATION}, LOCATION_PERMISSION_CODE);
        }
        }

        /***
        * Required method to handle the resulting permissions.
        * @param requestCode
        * @param permissions
        * @param grantResults
        */
        @Override
        public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull
int[] grantResults) {
        if (requestCode == LOCATION_PERMISSION_CODE) {
        if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                Toast.makeText(this, "Permission Granted", Toast.LENGTH_SHORT).show();
        } else {
                Toast.makeText(this, "Permission Denied", Toast.LENGTH_SHORT).show();
        }
        }
        }

        @Override
        public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putBoolean("ifNotTracking", ifNotTracking);
```

```
        }

        @Override
        protected void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);
        boolean currentState = savedInstanceState.getBoolean("ifNotTracking");

        SharedPreferences settings = getSharedPreferences(getString(R.string.pref_file),
Context.MODE_PRIVATE);
        ifNotTracking = settings.getBoolean("ifNotTracking", currentState);

        if (!ifNotTracking) {
        // Set to red and stop
        trackingButton.setText(R.string.tracking_button_text_stop);
        trackingButton.setBackgroundResource(R.color.stop_t);
        } else {
        trackingButton.setText(R.string.tracking_button_text_start);
        trackingButton.setBackgroundResource(R.color.start_t);
        }

        }

}*/
        public void launchSurvey(View view) {
        Intent intent = new Intent(this, SurveyActivity.class);
    }
```

LocationHelper.java (modified from Google's Location Code Sample)

```
package com.example.acadiavisitorstudy;

import android.app.Application;
import android.content.Context;
import android.location.Location;
import android.preference.PreferenceManager;

import java.text.DateFormat;
import java.util.ArrayList;
import java.util.Date;

public class LocationHelper extends Application {

/**
 * Copyright 2017 Google Inc. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
```

```
* See the License for the specific language governing permissions and
* limitations under the License.
*/

        // Blank constructor
        LocationHelper(){}

        static final String KEY_REQUESTING_LOCATION_UPDATES = "requesting_locaction_updates";
        static final float NUMBER_OF_METERS_IN_MILE = 1609.34f;

        /**
        * Returns true if requesting location updates, otherwise returns false.
        *
        * @param context The {@link Context}.
        */
        static boolean requestingLocationUpdates(Context context) {
        return PreferenceManager.getDefaultSharedPreferences(context)
                .getBoolean(KEY_REQUESTING_LOCATION_UPDATES, false);
        }

        /**
        * Stores the location updates state in SharedPreferences.
        * @param requestingLocationUpdates The location updates state.
        */
        static void setRequestingLocationUpdates(Context context, boolean requestingLocationUpdates) {
        PreferenceManager.getDefaultSharedPreferences(context)
                .edit()
                .putBoolean(KEY_REQUESTING_LOCATION_UPDATES, requestingLocationUpdates)
                .apply();
        }

        /**
        * Returns the {@code location} object as a human readable string.
        * @param location  The {@link Location}.
        */
        static String getLocationText(Location location) {
        return location == null ? "Unknown location" :
                "(" + location.getLatitude() + ", " + location.getLongitude() + ")";
        }

        static String getLocationTitle(Context context) {
        return context.getString(R.string.location_updated,
                DateFormat.getDateTimeInstance().format(new Date()));
        }

        /***
        * Checks if a Location is within an array of Geofences. Returns true if yes, false if no.
        * @param location
        * @param geofences
        * @return
        */
        static boolean ifWithinGeofences(Location location, ArrayList<IGeofence> geofences) {
        boolean withinGeofence = false;

        /*
        * Check all of the geofences
```

```
        */
        for (IGeofence geofence : geofences) {
        if (geofence.withinGeofence(location)){
                withinGeofence = true;
        }
        }

        return withinGeofence;
        }

}
```

LocationUpdateService.java (modified from Google's Location Code Sample)

```
package com.example.acadiavisitorstudy;

/**
 * Copyright 2017 Google Inc. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */


import android.app.ActivityManager;
import android.app.Notification;
import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.res.Configuration;
import android.location.Location;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.Binder;
import android.os.Build;
import android.os.Handler;
import android.os.HandlerThread;
import android.os.IBinder;
import android.os.Looper;
import android.support.annotation.NonNull;
import android.support.v4.app.NotificationCompat;
```

```
import android.support.v4.content.LocalBroadcastManager;
import android.util.Log;

import com.google.android.gms.location.FusedLocationProviderClient;
import com.google.android.gms.location.LocationCallback;
import com.google.android.gms.location.LocationRequest;
import com.google.android.gms.location.LocationResult;
import com.google.android.gms.location.LocationServices;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;

import java.util.ArrayList;

/**
 * A bound and started service that is promoted to a foreground service when location updates have
 * been requested and all clients unbind.
 *
 * For apps running in the background on "O" devices, location is computed only once every 10
 * minutes and delivered batched every 30 minutes. This restriction applies even to apps
 * targeting "N" or lower which are run on "O" devices.
 *
 * This sample show how to use a long-running service for location updates. When an activity is
 * bound to this service, frequent location updates are permitted. When the activity is removed
 * from the foreground, the service promotes itself to a foreground service, and location updates
 * continue. When the activity comes back to the foreground, the foreground service stops, and the
 * notification assocaited with that service is removed.
 */

public class LocationUpdatesService extends Service implements IResultListener{

        private static final String PACKAGE_NAME =
        "com.google.android.gms.location.sample.locationupdatesforegroundservice";

        private static final String TAG = LocationUpdatesService.class.getSimpleName();

        /**
        * The name of the channel for notifications.
        */
        private static final String CHANNEL_ID = "channel_01";

        static final String ACTION_BROADCAST = PACKAGE_NAME + ".broadcast";

        static final String EXTRA_LOCATION = PACKAGE_NAME + ".location";

        private static final String EXTRA_STARTED_FROM_NOTIFICATION = PACKAGE_NAME +
        ".started_from_notification";

        private final IBinder mBinder = new LocalBinder();

        /**
        * The desired interval for location updates. Inexact. Updates may be more or less frequent.
        */
        private static final long UPDATE_INTERVAL_IN_MILLISECONDS = 15000;

        /**
        * The fastest rate for active location updates. Updates will never be more frequent
```

```
* than this value.
*/
private static final long FASTEST_UPDATE_INTERVAL_IN_MILLISECONDS =
UPDATE_INTERVAL_IN_MILLISECONDS / 2;

/**
* The identifier for the notification displayed for the foreground service.
*/
private static final int NOTIFICATION_ID = 12345678;


/**
* Used to check whether the bound activity has really gone away and not unbound as part of an
* orientation change. We create a foreground service notification only if the former takes
* place.
*/
private boolean mChangingConfiguration = false;

private NotificationManager mNotificationManager;

/**
* Contains parameters used by {@link com.google.android.gms.location.FusedLocationProviderApi}.
*/
private LocationRequest mLocationRequest;

/**
* Provides access to the Fused Location Provider API.
*/
private FusedLocationProviderClient mFusedLocationClient;

private static final int REQUEST_PERMISSIONS_REQUEST_CODE = 34;

private LocationCallback mLocationCallback;

private Handler mServiceHandler;

private ArrayList<Location> locationList; // Temporary List of all Batched location points
private static final int MIN_BATCH_SIZE = 5; // Minimum batch size in order to upload the data

private ConnectivityManager cm;

private ILocationProcessor server;

/**
* The current location.
*/
private Location mLocation;

private ArrayList<IGeofence> geofences;

public LocationUpdatesService() {
}

@Override
public void onCreate() {
// Add a geofence for Mound Desert Island and Schoodic Peninsula
```

```
        geofences = new ArrayList<IGeofence>();
        geofences.add(new CircularGeofence(new GPSPoint(44.323684, -68.288247), 13384.08f));
        geofences.add(new CircularGeofence(new GPSPoint(44.346791, -68.052729), 4212.80f));

        mFusedLocationClient = LocationServices.getFusedLocationProviderClient(this);

        // This is called whenever a new location update is made by the API
        mLocationCallback = new LocationCallback() {
        @Override
        public void onLocationResult(LocationResult locationResult) {
                super.onLocationResult(locationResult);
                onNewLocation(locationResult.getLastLocation());
        }
        };

        createLocationRequest();
        getLastLocation();

        HandlerThread handlerThread = new HandlerThread(TAG);
        handlerThread.start();
        mServiceHandler = new Handler(handlerThread.getLooper());
        mNotificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);

        locationList = new ArrayList<Location>();

        // Android O requires a Notification Channel.
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        CharSequence name = getString(R.string.app_name);
        // Create the channel for the notification
        NotificationChannel mChannel =
                new NotificationChannel(CHANNEL_ID, name, NotificationManager.IMPORTANCE_LOW);

        // Set the Notification Channel for the Notification Manager.
        mNotificationManager.createNotificationChannel(mChannel);
        }

        // Create ConnectivityManager
        cm = (ConnectivityManager)
getApplicationContext().getSystemService(Context.CONNECTIVITY_SERVICE);
        server = new SQLDatabase(getApplicationContext(), this);
        }

        @Override
        public int onStartCommand(Intent intent, int flags, int startId) {
        Log.i(TAG, "Service started");
        boolean startedFromNotification = intent.getBooleanExtra(EXTRA_STARTED_FROM_NOTIFICATION,
                false);

        // We got here because the user decided to remove location updates from the notification.
        if (startedFromNotification) {
        removeLocationUpdates();
        stopSelf();
        }

        // Tells the system to not try to recreate the service after it has been killed.
        return START_NOT_STICKY;
```

```
        }

        @Override
        public void onConfigurationChanged(Configuration newConfig) {
        super.onConfigurationChanged(newConfig);
        mChangingConfiguration = true;
        }

        @Override
        public IBinder onBind(Intent intent) {
        // Called when a client (MainActivity in case of this sample) comes to the foreground
        // and binds with this service. The service should cease to be a foreground service
        // when that happens.
        Log.i(TAG, "in onBind()");
        stopForeground(true);
        mChangingConfiguration = false;
        return mBinder;
        }

        @Override
        public void onRebind(Intent intent) {
        // Called when a client (MainActivity in case of this sample) returns to the foreground
        // and binds once again with this service. The service should cease to be a foreground
        // service when that happens.
        Log.i(TAG, "in onRebind()");
        stopForeground(true);
        mChangingConfiguration = false;
        super.onRebind(intent);
        }

        @Override
        public boolean onUnbind(Intent intent) {
        Log.i(TAG, "Last client unbound from service");

        // Called when the last client (MainActivity in case of this sample) unbinds from this
        // service. If this method is called due to a configuration change in MainActivity, we
        // do nothing. Otherwise, we make this service a foreground service.
        if (!mChangingConfiguration && LocationHelper.requestingLocationUpdates(this)) {
        Log.i(TAG, "Starting foreground service");

        startForeground(NOTIFICATION_ID, getNotification(!LocationHelper.ifWithinGeofences(mLocation,
geofences)));
        }
        return true; // Ensures onRebind() is called when a client re-binds.
        }

        @Override
        public void onDestroy() {
        mServiceHandler.removeCallbacksAndMessages(null);
        }

        /**
        * Makes a request for location updates. Note that in this sample we merely log the
        * {@link SecurityException}.
        */
        public void requestLocationUpdates() {
```

```
        Log.i(TAG, "Requesting location updates");
        LocationHelper.setRequestingLocationUpdates(this, true);
        startService(new Intent(getApplicationContext(), LocationUpdatesService.class));
        try {
        mFusedLocationClient.requestLocationUpdates(mLocationRequest,
                mLocationCallback, Looper.myLooper());
        } catch (SecurityException unlikely) {
        LocationHelper.setRequestingLocationUpdates(this, false);
        Log.e(TAG, "Lost location permission. Could not request updates. " + unlikely);
        }
        }

        /**
        * Removes location updates. Note that in this sample we merely log the
        * {@link SecurityException}.
        */
        public void removeLocationUpdates() {
        Log.i(TAG, "Removing location updates");
        try {
        mFusedLocationClient.removeLocationUpdates(mLocationCallback);
        LocationHelper.setRequestingLocationUpdates(this, false);

        // Update button state
        SharedPreferences s = getSharedPreferences(getString(R.string.pref_file),
Context.MODE_PRIVATE);
        s.edit().putBoolean("ifNotTracking", true).apply();

        // Remove cached points
        locationList.clear();

        stopSelf();
        } catch (SecurityException unlikely) {
        LocationHelper.setRequestingLocationUpdates(this, true);
        Log.e(TAG, "Lost location permission. Could not remove updates. " + unlikely);
        }
        }

        /**
        * Returns the {@link NotificationCompat} used as part of the foreground service.
        */
        private Notification getNotification(boolean outOfRange) {
        Intent intent = new Intent(this, LocationUpdatesService.class);

        CharSequence text = LocationHelper.getLocationText(mLocation);
        CharSequence oorString = getString(R.string.out_of_range);

        // Extra to help us figure out if we arrived in onStartCommand via the notification or not.
        intent.putExtra(EXTRA_STARTED_FROM_NOTIFICATION, true);

        // The PendingIntent that leads to a call to onStartCommand() in this service.
        PendingIntent servicePendingIntent = PendingIntent.getService(this, 0, intent,
                PendingIntent.FLAG_UPDATE_CURRENT);

        NotificationCompat.Builder builder;
        if (outOfRange)
        {
```

```
builder = new NotificationCompat.Builder(this)
        .addAction(R.drawable.ic_cancel, getString(R.string.remove_location_updates),
                servicePendingIntent)
        .setContentText(oorString)
        .setContentTitle(LocationHelper.getLocationTitle(this))
        .setOngoing(true)
        .setPriority(Notification.PRIORITY_LOW)
        .setSmallIcon(R.mipmap.ic_launcher)
        .setTicker(text)
        .setWhen(System.currentTimeMillis());
} else {
builder = new NotificationCompat.Builder(this)
        .addAction(R.drawable.ic_cancel, getString(R.string.remove_location_updates),
                servicePendingIntent)
        .setContentText(text)
        .setContentTitle(LocationHelper.getLocationTitle(this))
        .setOngoing(true)
        .setPriority(Notification.PRIORITY_LOW)
        .setSmallIcon(R.mipmap.ic_launcher)
        .setTicker(text)
        .setWhen(System.currentTimeMillis());
}

// Set the Channel ID for Android O.
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
builder.setChannelId(CHANNEL_ID); // Channel ID
}

return builder.build();
}

private void getLastLocation() {
try {
mFusedLocationClient.getLastLocation()
        .addOnCompleteListener(new OnCompleteListener<Location>() {
                @Override
                public void onComplete(@NonNull Task<Location> task) {
                if (task.isSuccessful() && task.getResult() != null) {
                        mLocation = task.getResult();
                } else {
                        Log.w(TAG, "Failed to get location.");
                }
                }
        });
} catch (SecurityException unlikely) {
Log.e(TAG, "Lost location permission." + unlikely);
}
}

private void onNewLocation(Location location) {
mLocation = location;

// Check if within geofences, then record the data.
if(LocationHelper.ifWithinGeofences(location, geofences)){

Log.i(TAG, "New location: " + location);
```

```
// Notify anyone listening for broadcasts about the new location.
Intent intent = new Intent(ACTION_BROADCAST);
intent.putExtra(EXTRA_LOCATION, location);
LocalBroadcastManager.getInstance(getApplicationContext()).sendBroadcast(intent);

// Update notification content if rufloatnning as a foreground service.
if (serviceIsRunningInForeground(this)) {
        mNotificationManager.notify(NOTIFICATION_ID, getNotification(false));
}

// Add location to exisitng list
locationList.add(mLocation);
Log.i(TAG, "onNewLocation: Location added!");
} else {
Log.i(TAG, "onNewLocation: Location outside of range. Will not record location.");
// Update notification content if running as a foreground service.
if (serviceIsRunningInForeground(this)) {
        mNotificationManager.notify(NOTIFICATION_ID, getNotification(true));
}
}

if(canAccessNetwork(cm) && locationList.size() >= MIN_BATCH_SIZE) {
/*
        * Once there is a network connection, take the bunch of network data and
        * send it to the server
        */
server.processDataPoints(locationList);

}
}


/**
* Sets the location request parameters.
*/
private void createLocationRequest() {
mLocationRequest = new LocationRequest();
mLocationRequest.setInterval(UPDATE_INTERVAL_IN_MILLISECONDS);
mLocationRequest.setFastestInterval(FASTEST_UPDATE_INTERVAL_IN_MILLISECONDS);
mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
}

/**
* Class used for the client Binder.  Since this service runs in the same process as its
* clients, we don't need to deal with IPC.
*/
public class LocalBinder extends Binder {
LocationUpdatesService getService() {
return LocationUpdatesService.this;
}
}

/**
* Returns true if this is a foreground service.
*
* @param context The {@link Context}.
```

```
        */
        public boolean serviceIsRunningInForeground(Context context) {
        ActivityManager manager = (ActivityManager) context.getSystemService(
                Context.ACTIVITY_SERVICE);
        for (ActivityManager.RunningServiceInfo service : manager.getRunningServices(
                Integer.MAX_VALUE)) {
        if (getClass().getName().equals(service.service.getClassName())) {
                if (service.foreground) {
                return true;
                }
        }
        }
        return false;
        }


        /***
        * Checks if the device has an active Internet connection
        */
        public boolean canAccessNetwork(ConnectivityManager cm) {
        NetworkInfo activeNetwork = cm.getActiveNetworkInfo();
        return (activeNetwork != null && activeNetwork.isConnected());
        }

        /***
        * When the data is sumbitted to the database
        * @param result
        */
        @Override
        public void onSubmit(boolean result) {
        // If the server succeeded, clear the list.
        if (result) {
        locationList.clear();
        }
        }
}
```

SurveyActivity.java

```
package com.example.acadiavisitorstudy;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.SeekBar;
import android.widget.TextView;
import android.widget.Toast;

import java.util.ArrayList;

public class SurveyActivity extends AppCompatActivity implements IResultListener {

        // Array to store the questions
        private ArrayList<Integer> questionArray = new ArrayList<Integer>();
        private static final int NUMBER_OF_QUESTIONS = 3;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_survey);

        // Getting the values for the seekbars
        SeekBar seekQuestionOne = (SeekBar) findViewById(R.id.question3_seekbar);
        SeekBar seekQuestionTwo = (SeekBar) findViewById(R.id.question4_seekbar);
        SeekBar seekQuestionThree = (SeekBar) findViewById(R.id.question5_seekbar);

        // For all questions set an initial quantity for the default value
        for(int i = 0; i < NUMBER_OF_QUESTIONS; i++) {
        questionArray.add(4);
        }

        // TextViews for displaying current choice.
        final TextView questionOneDisplay = (TextView) findViewById(R.id.question_3_value);
        final TextView questionTwoDisplay = (TextView) findViewById(R.id.question_4_value);
        final TextView questionThreeDisplay = (TextView) findViewById(R.id.question_5_value);


        seekQuestionOne.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
        int progressValue = 4;
        @Override
        public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
                progressValue = progress;
                questionArray.set(0, progressValue + 1);
                questionOneDisplay.setText(Integer.toString(progressValue + 1));
        }

        @Override
        public void onStartTrackingTouch(SeekBar seekBar) {

        }

        @Override
```

```
public void onStopTrackingTouch(SeekBar seekBar) {
}
});
seekQuestionTwo.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
int progressValue = 4;
@Override
public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
        progressValue = progress;
        questionArray.set(1, progressValue + 1);
        questionTwoDisplay.setText(Integer.toString(progressValue + 1));
}

@Override
public void onStartTrackingTouch(SeekBar seekBar) {

}

@Override
public void onStopTrackingTouch(SeekBar seekBar) {
}
});
seekQuestionThree.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
int progressValue = 4;
@Override
public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
        progressValue = progress;
        questionArray.set(2, progressValue + 1);
        questionThreeDisplay.setText(Integer.toString(progressValue + 1));
}

@Override
public void onStartTrackingTouch(SeekBar seekBar) {

}

@Override
public void onStopTrackingTouch(SeekBar seekBar) {
}
});
}

/***
* Submits survey data and closes the Survey Activity
* @param view
*/
public void onSubmit(View view) {

// This is where we will submit the user's survey answers
ArrayList<ISurveyQuestion> questions = new ArrayList<ISurveyQuestion>();

for (int response : questionArray) {
ISurveyQuestion question = new LikertScaleQuestion(response);
questions.add(question);
}

// Submit the questions to the server
```

```
        ILocationProcessor server = new SQLDatabase(getApplicationContext(), this);
        server.processSurvey(questions);
        finish();
        }

        /***
        * Handles when the survey is submitted (or not)
        * @param result
        */
        @Override
        public void onSubmit(boolean result) {
        if (result) {
        Toast.makeText(this, getString(R.string.survey_submit_success), Toast.LENGTH_LONG).show();
        } else {
        Toast.makeText(this, getString(R.string.survey_submit_failed), Toast.LENGTH_LONG).show();
        }
        }
}
```

LicensingActivity.java

```
package com.example.acadiavisitorstudy;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;

public class LicensingActivity extends AppCompatActivity {

        @Override
        protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_license);

        }

        public void onDone(View view) {
        finish();
        }
}
```

InstructionActivity.java

```
package com.example.acadiavisitorstudy;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;

public class InstructionActivity extends AppCompatActivity {

        @Override
        protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_instructions);

        }

        public void onDone(View view) {
        finish();
        }
}
```

ILocationProcessor.java

```
package com.example.acadiavisitorstudy;

import android.location.Location;

import java.util.ArrayList;

/*
 * This provides the interface for how the locations will be processed externally.
 * regardless of how it is modified in the future.
 */
public interface ILocationProcessor {

        /***
        * Should return true if the GPS points have been processed.
        * False if not.
        * @return
        */
        boolean processDataPoints(ArrayList<Location> locations);

        /***
        * Should return true of the survey has been submitted.
        * False if not.
        */
        boolean processSurvey(ArrayList<ISurveyQuestion> questions);

}
```

SQLDatabase.java

```
package com.example.acadiavisitorstudy;

import android.content.Context;
import android.content.SharedPreferences;
import android.location.Location;
import android.os.AsyncTask;
import android.util.Log;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.io.BufferedReader;
```

```
import java.io.DataOutputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.ArrayList;

import javax.net.ssl.HttpsURLConnection;

public class SQLDatabase implements ILocationProcessor{

        private final String url = "https://acadiatrails.wpi.edu//index.php"; // URL to send the data
to.

        private int uid;
        private static final String TAG = "SQLDatabase";
        final String PREFS_NAME = "Preferences";
        private Context context;
        IResultListener resultListener;

        SQLDatabase(Context context, IResultListener resultListener) {

        this.context = context;
        SharedPreferences settings = context.getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
        this.resultListener = resultListener; // Declare a listener to get the results

        if (settings.contains("uid") && (settings.getInt("uid", 0) != 0)){
        //User already has an ID
        uid = settings.getInt("uid", 0);
        }
        else{
        Log.d(TAG, "SQLDatabase: Generate ID");
        AsyncTaskRunnerGET getUID = new AsyncTaskRunnerGET();
        getUID.execute(url);
        }

        }


        /***
        * Uploads all of the cached data points to a SQL Server.
        * Should return true if the GPS points have been processed.
        * False if not.
        * @return
        */
        @Override
        public boolean processDataPoints(ArrayList<Location> locations) {
        // Currently a stub right now to test the logic.
        Log.i(TAG, "Processing bundle of size " + locations.size());
        JSONObject jobj = new JSONObject();
        try {
        jobj.put("user", uid);
        jobj.put("operation","location");
        JSONArray jarray = new JSONArray();
        // Put all location objects in the JSON array
        for (Location l : locations) {
                JSONObject blob = new JSONObject();
```

```
                blob.put("latitude", l.getLatitude());
                blob.put("longitude", l.getLongitude());
                blob.put("velocity", l.getSpeed());
                blob.put("time", l.getTime());
                jarray.put(blob);
                Log.i(TAG, "Location point: " + LocationHelper.getLocationText(l) + "has been uploaded
to the server.");
        }
        jobj.put("data",jarray);
        AsyncTaskRunnerPOST serverUpload = new AsyncTaskRunnerPOST(resultListener);
        serverUpload.execute(url, jobj.toString());

        return true;
        } catch (JSONException e) {
        e.printStackTrace();
        return false;
        }
        }

        /***
        * Uploads all of the survey responses to a server
        * @param questions
        * @return
        */
        @Override
        public boolean processSurvey (ArrayList < ISurveyQuestion > questions) {
        // Currently a stub right now to test the logic.
        Log.d(TAG, "processSurvey: Processing " + Integer.toString(questions.size()));
        JSONObject jobj = new JSONObject();
        try {
        jobj.put("user", uid);
        jobj.put("operation","survey");

        // Creating the data field
        JSONObject jsurv = new JSONObject();

        // For each question, add to JSON
        for (int i = 0; i < questions.size(); i++) {
                jsurv.put("q" + Integer.toString(i + 1), questions.get(i).toString());
        }

        jobj.put("data",jsurv);
        AsyncTaskRunnerPOST serverUpload = new AsyncTaskRunnerPOST(resultListener);
        serverUpload.execute(url, jobj.toString());
        return true;
        } catch (JSONException e) {
        e.printStackTrace();
        return false;
        }
        }


        private class AsyncTaskRunnerPOST extends AsyncTask<String, String, Boolean> {

        private IResultListener resultListener;
```

```
        AsyncTaskRunnerPOST(IResultListener resultListener) {
        this.resultListener = resultListener;
        }

        @Override
        protected Boolean doInBackground(String... params) {
        HttpsURLConnection httpURLConnection = null;
        try {

                URL url = new URL(params[0]);
                httpURLConnection = (HttpsURLConnection) url.openConnection();
                httpURLConnection.setRequestMethod("POST");
                httpURLConnection.setRequestProperty("Content-Type", "application/json;
charset=UTF-8");
                httpURLConnection.setRequestProperty("Accept", "application/json");
                httpURLConnection. setDoOutput(true);
                httpURLConnection.connect();

                DataOutputStream os = new DataOutputStream(httpURLConnection.getOutputStream());
                os.writeBytes(params[1]);
                os.flush();
                os.close();

                // Just keep this here. The server needs someone to talk to
                BufferedReader br = new BufferedReader (new
InputStreamReader(httpURLConnection.getInputStream(), "utf-8"));
                StringBuilder response = new StringBuilder();
                String responseLine = null;
                while ((responseLine = br.readLine()) != null) {
                response.append(responseLine.trim());
                }

                Log.d(TAG, "doInBackground: " + response.toString());

                if (httpURLConnection != null) {
                httpURLConnection.disconnect();
                }

                return true;

        } catch (Exception e) {
                e.printStackTrace();
                if (httpURLConnection != null) {
                httpURLConnection.disconnect();
                }

                return false;
        }
        }

        @Override
        protected void onPostExecute(Boolean result) {
        // Take the result and give it to the listener class to process
        resultListener.onSubmit(result);
        }
        }
```

```
        private class AsyncTaskRunnerGET extends AsyncTask<String, String, Boolean> {

        private String resp;
        private final String USER_AGENT = "Mozilla/5.0";

        @Override
        protected Boolean doInBackground(String... params) {
        HttpsURLConnection httpURLConnection = null;
        try {

                URL url = new URL(params[0]);
                httpURLConnection = (HttpsURLConnection) url.openConnection();
                httpURLConnection.setRequestMethod("GET");
                httpURLConnection.setRequestProperty("USER_AGENT", USER_AGENT);

                int responseCode = httpURLConnection.getResponseCode();
                Log.d(TAG, "doInBackground: response Code " + responseCode);

                // Just keep this here. The server needs someone to talk to
                BufferedReader br = new BufferedReader (new
InputStreamReader(httpURLConnection.getInputStream(), "utf-8"));
                StringBuilder rsp = new StringBuilder();
                String responseLine = null;

                while ((responseLine = br.readLine()) != null) {
                rsp.append(responseLine.trim());
                }

                Log.d(TAG, "doInBackground: " + rsp.toString());
                resp = rsp.toString();
                uid = Integer.parseInt(resp);

                SharedPreferences settings = context.getSharedPreferences(PREFS_NAME,
Context.MODE_PRIVATE);
                settings.edit().putInt("uid", uid).apply();

                if (httpURLConnection != null) {
                httpURLConnection.disconnect();
                }

                return true;

        } catch (Exception e) {
                e.printStackTrace();
                resp = e.getMessage();

                if (httpURLConnection != null) {
                httpURLConnection.disconnect();
                }

                return false;
        }

        }
```

```
        @Override
        protected void onPostExecute(Boolean result) {
        }
        }
}
```

GPSPoint.java

```
package com.example.acadiavisitorstudy;

public class GPSPoint {

        /***
        * Custom class to operate with the geofence classes
        */
        private double latitude;
        private double longitude;

        // Constructor
        public GPSPoint(double latitude, double longitude) {
        this.latitude = latitude;
        this.longitude = longitude;
        }

        // Getters
        /***
        * Gets the latitude value of the GPS point
        * @return (float) - latitude value
        */
        public double getLatitude() {
        return this.latitude;
        }

        /***
        * Gets the longitude value of the GPS point
        * @return (float) - longitude value
        */
        public double getLongitude() {
        return this.longitude;
        }
}
```

IGeofence.java

```
package com.example.acadiavisitorstudy;

import android.location.Location;

public interface IGeofence {
   /*
   * Common interface for different types of Geofences
        */

   /***
```

```
        * Returns true when the location l is within a geofence.
        * @param l
        * @return
        */
    boolean withinGeofence(Location l);
}
```

CircularGeofence.java

```
package com.example.acadiavisitorstudy;

import android.location.Location;

public class CircularGeofence implements IGeofence{

        private double latitudeCenter; // Latitude of the center of the circle
        private double longitudeCenter; // Longitude of the center of the circle
        private double radius; // Radius of geofence (in meters)

        // Constructor
        public CircularGeofence(double latitudeCenter, double longitudeCenter, float radius) {
        this.latitudeCenter = latitudeCenter;
        this.longitudeCenter = longitudeCenter;
        this.radius = radius;
        }

        public CircularGeofence(GPSPoint point, float radius) {
        this.latitudeCenter = point.getLatitude();
        this.longitudeCenter = point.getLongitude();
        this.radius = radius;
        }


        /***
        * Determine if a given location l is within a geofence.
        * @param l
        * @return
        */
        @Override
        public boolean withinGeofence(Location l) {
        // Create a new Location object
        Location dest = new Location("null");
        dest.setLatitude(this.latitudeCenter);
        dest.setLongitude(this.longitudeCenter);

        float distance  = l.distanceTo(dest);

        return distance <= radius;
        }

}
```

RectangularGeofence.java

```
package com.example.acadiavisitorstudy;

import android.location.Location;

public class RectangularGeofence implements IGeofence {

        private double topLeftLatitude; // Latitude of the first GPS point
        private double topLeftLongitude; // Longitude of the second GPS point
        private double bottomRightLatitude; // Latitude of the second GPS point
        private double bottomRightLongitude; // Longitude of the second GPS point

        // Constructor
        public RectangularGeofence(GPSPoint topLeftPoint, GPSPoint bottomRightPoint) {
        this.topLeftLatitude = topLeftPoint.getLatitude();
        this.topLeftLongitude = topLeftPoint.getLongitude();
        this.bottomRightLatitude = bottomRightPoint.getLatitude();
        this.bottomRightLongitude = bottomRightPoint.getLongitude();
        }

        @Override
        public boolean withinGeofence(Location l) {

        // Get the point's latitude and longitude
        double compareLatitude = l.getLatitude();
        double compareLongitude = l.getLongitude();

        return ((compareLatitude <= this.topLeftLatitude) && (compareLatitude >=
this.bottomRightLatitude)
                && (compareLongitude <= this.bottomRightLongitude) && (compareLongitude >=
this.topLeftLongitude));
        }

}
```

IResultListener.java

```
package com.example.acadiavisitorstudy;

public interface IResultListener {
        /*
        * Interface to listen to the result of POST requests. Implement this method and
        * you can use the ILocationProcessor interface
        * @param - result should be true if operation is successful or false if not
        */
        void onSubmit(boolean result);
}
```

ISurveyQuestion.java

```
public interface ISurveyQuestion {

        /*
        * This interface is for all survey questions to get processed into.
        */
```

```
        /***
        * Converts the survey question into a parsable format as an Integer
        * @return
        */
        int toInteger();

        /***
        * Converts the survey question into a parsable format as a String.
        * @return
        */
        String toString();

}
```

LikertScaleQuestion.java

```
package com.example.acadiavisitorstudy;

public class LikertScaleQuestion implements ISurveyQuestion {

        int response; // value of likert scale question (1-7)

        LikertScaleQuestion(int response) {
        this.response = response;
        }

        @Override
        public int toInteger() {
        return response;
        }

        @Override
        public String toString() {
        return String.valueOf(response);
        }

}
```

Appendix F: Server Source Code

```
<?php

/*
 * Acadia Trails Sample PHP Server
 * James Plante
 * Jack Hogan
 */

require_once 'login.php';
$conn = new mysqli($hn, $un, $pw, $db);
```

```
if ($conn->connect_error) die("Fatal Error");

if ($_SERVER['REQUEST_METHOD'] === 'POST')
{
    $json = file_get_contents('php://input');
    $json_array = json_decode($json, TRUE);
    $user = $json_array['user'];
    echo("<h1>User: ". $user."</h1>");
    if ($json_array[operation] == 'location')
    {
         echo("<h2>Location Mode</h2>");

         /*
          * For every element in the json array, put the point in the server.
          */
         $arr_size = count($json_array['data']);

         for ($i = 0; $i < $arr_size; $i++)
         {
                 // Assigning values
                 $latitude = $json_array['data'][$i]['latitude'];
                 $longitude = $json_array['data'][$i]['longitude'];
                 $velocity = $json_array['data'][$i]['velocity'];
                 $time = $json_array['data'][$i]['time'];

                 echo("Latitude: ".$latitude);
                 echo("Longitude: ".$longitude);
                 echo("Velocity: ".$velocity);
                 echo("Time: ".$time);

                 // Execute prepared statemnt
                 $stmt = $conn->prepare('INSERT INTO location_data VALUES(?,?,?,?,?)');
                 echo($stmt->bind_param('iiddd', $user, $time, $latitude, $longitude, $velocity));
                 $stmt->execute();
                 echo('Statement executed');
         }
    }
    elseif($json_array[operation] == 'survey')
    {
         // Parse the JSON as if a survey then insert into db
         $q1 = $json_array['data']['q1'];
         $q2 = $json_array['data']['q2'];
         $q3 = $json_array['data']['q3'];

         // Execute prepared statement to insert into server
         $stmt = $conn->prepare('INSERT INTO survey VALUES(?,?,?,?)');
         $stmt->bind_param('iiii',$user, $q1, $q2, $q3);
         $stmt->execute();
    }
    else
    {
         echo("ERROR!");
    }
}
elseif ($_SERVER['REQUEST_METHOD'] === 'GET')
{
```

```
    // If it is a GET request, just print this out
    echo(get_random_number($conn));
}
else
{
    echo("ERROR!");
}


/***
 * Gets a random number from 0 to the 32 bit integer limit
 */
function get_random_number($conn)
{
    $can_exit = FALSE;
    while(!$can_exit)
    {
         $value = random_int(0, 2147483647);

         if (if_can_store($value, $conn))
         {
                 $can_exit = TRUE;
                 return $value;
         }
    }
}

/***
 * Checks if a user id with the given number n can
 * be stored.
 * Returns true if yes, false if no.
 * @param (int) $n - The id being searched
 * @param (connection) $conn - The database
 */
function if_can_store($n, $conn)
{
    $query = "SELECT * FROM location_data where UID = ".$n;
    $result = $conn->query($query);
    return ($result->num_rows == 0);
}

?>
```

Appendix G: Text file to GPX Converter Code

```
"""
TXT to GPX Utility to export MySQL plaintext output into GPX data
James Plante (jplante@wpi.edu)
"""

from datetime import datetime
```

```
import gpxpy
import gpxpy.gpx
import sys
import srtm

"""
Takes in a date in UTC and returns a string with the month, day, and year
@param n - (int) UTC date in milliseconds
"""
def utc_to_local(utc):
        return datetime.fromtimestamp(utc/1e3)

"""
Generates the GPX file in the form of a string to be outputted to a file
@param tup_list - list of tuples to parse
"""
def create_gpx_xml(tup_list):
        gpx = gpxpy.gpx.GPX()

        # Create first track in our GPX:
        gpx_track = gpxpy.gpx.GPXTrack(name=str(tup_list[0][0]))
        gpx.tracks.append(gpx_track)

        # Create first segment in our GPX track:
        gpx_segment = gpxpy.gpx.GPXTrackSegment()
        gpx_track.segments.append(gpx_segment)

        #Temp variables
        last_point = None
        current_track = gpx_track
        current_segment = gpx_segment
        last_uid = tup_list[0][0]

        # Process all of the tuples to create points (assume list is sorted)
        for (uid, time, latitude, longitude, velocity, utctime) in tup_list:
        # if it reaches the end of the current UID make another track
        if (uid != last_uid):
        last_uid = uid

        current_track = gpxpy.gpx.GPXTrack(name=str(uid))
        gpx.tracks.append(current_track)

        # Create first segment in our GPX track:
        current_segment = gpxpy.gpx.GPXTrackSegment()
        current_track.segments.append(current_segment)
        last_point = None

        current_point = gpxpy.gpx.GPXTrackPoint(latitude, longitude, speed=velocity, time=time)

        # If the time differs by more than 2 minutes, then create a new track segment
        if (last_point != None and (time - last_point).total_seconds() >= 120):
        current_segment = gpxpy.gpx.GPXTrackSegment()
        current_track.segments.append(current_segment)

        current_segment.points.append(current_point)
        last_point = time
```

```
        # Adding elevation data using SRTM.py
        elevation_data = srtm.get_data()
        elevation_data.add_elevations(gpx)

        return gpx.to_xml(version="1.0")

"""
Takes in a line of the graph and parses it. Returns a tuple of all
of the data
"""
def parse_line(st):
        st_array = st.split();
        uid = st_array[0]
        time = utc_to_local(int(st_array[1]))
        latitude = st_array[2]
        longitude = st_array[3]
        velocity = st_array[4]
        utctime = int(st_array[1])
        tup = (uid, time, latitude, longitude, velocity, utctime)
        return tup

"""
Parses file to get all of the data from the input file
Returns an list of tuples sorted by UID.
"""
def read_file(filename):
        f = open(filename, 'r')
        tup_list = []
        for line in f:
        tup = parse_line(line)
        tup_list.append(tup)

        f.close()

        # Sort the list by UID then return
        tup_list.sort(key=lambda tup:(tup[0], tup[5]))
        return tup_list

# Parsing arguments
if len(sys.argv) != 2:
        print("Usage: python3 txt_to_gpx <filename> where:\nfilename - File to parse.")
        exit(0)

filename = sys.argv[1]

points_tup = read_file(filename) # GPS points in the form of tuples

raw_xml = create_gpx_xml(points_tup)

# Write to the output file
with open(filename + "-exported.gpx", "w") as f:
        f.write(raw_xml)

print("Data exported to " + filename + "-exported.gpx")
```

Appendix H: Time Parser Code

```
"""
UTC to date utility
James Plante (jplante@wpi.edu)
"""
from datetime import datetime
import sys

"""
Takes in a date in UTC and returns a string with the month, day, and year
@param n - (int) UTC date in milliseconds
"""
def utc_to_local(utc):
        return datetime.fromtimestamp(utc / 1e3).strftime("%Y/%m/%d %H:%M:%S")

"""
Takes in a line of the graph and parses it. Returns a tuple of all
of the data
"""
def parse_line(st):
        st_array = st.split()
        uid = st_array[0]
        time = utc_to_local(int(st_array[1]))
        latitude = st_array[2]
        longitude = st_array[3]
        velocity = st_array[4]
        tup = (uid, time, latitude, longitude, velocity)
        return tup

"""
Parses file to get all of the data.
"""
def parse_file(filename):
        f = open(filename, 'r')
        output_file = open(filename + "-parsed.txt", 'w')
        output_file.write("UID\tTime\tLatitude\tLongitude\tVelocity\n")
        for line in f:
        tup = parse_line(line)
        output_file.write("{0}\t{1}\t{2}\t{3}\t{4}\n".format(*tup))
        f.close()
        output_file.close()

# Parsing arguments
if len(sys.argv) != 2:
        print("Usage: python3 utc_to_date <filename> where:\nfilename - File to parse.")
        exit(0)
filename = sys.argv[1]
parse_file(filename)
```