University of Arkansas – CSCE Department
Capstone II Final Report – Spring 2020

Weather Ways: Tracking Weather Along a Route

William Henness, Zachary Cantrell, Audrey Timmerman,
Nicholas Brinkley, Madison Galloway, David Turnbough

Abstract

It is often difficult to consistently find updated information regarding weather conditions while traveling as the common routing applications such as Google Maps, Apple Maps, etc. One can easily find weather related information for specific locations, however planning routes for travel and updating the routes' information is a task which cannot be done in one encompassing application. This Android application provides a simple user interface for the users to plan a trip to and from any destination for which weather information is available. Currently an application which provides routing and weather information along the route does not exist, or is not fully functional. Thus, this service could be provided to potential users with the creation of this application. The user base itself is potentially quite massive due to trips and vacations being quite common, and with almost every individual owning a phone.

The main objective of the application is to provide constant weather information for the user's current location, along the user's route, and at the user's final destination. Following this general idea, the application created may utilize information provided by available software. The purpose of the app will be providing the user with a route planning tool using Google Maps software and its information. An in-app map will aid the user in viewing the layout of the potential route which is being created and guiding them along it. Following a chosen route, information will be provided to the user regarding the weather conditions of the areas which will be traveled to by the user.

1.0 Problem

According to a AAA travel survey in 2019, approximately 100 million U.S. adults were planning on taking a family vacation that year. Of these 100 million around 53% plan on taking a road trip (Hall). This means around 50 million U.S. adults in 2019 plan on taking a road trip within the year. Business trips also make up a large portion of yearly travelers, making the total number of road trips a year even larger than Hall's estimate.

Organizing a road trip can be quite difficult because vast areas will be covered with many differing environmental conditions, and coordinating events to take place along these routes requires specific timing. Once the road trip is in motion, unexpected weather changes may occur which then dampen the enjoyment of the vacation, as well as potentially ruining the plans of the trip which could harm the goers' finances and further delay the remainder of the trip. The desire is to alleviate the surprise of weather changes for the user to enhance their experience.

Weather conditions can change and turn hazardous very quickly, and these updates can be difficult for people to be aware of while driving. Certain conditions, such as a tornado, pose a very high risk of danger throughout a large area, and without consistent updates drivers may be unaware. Lighter weather conditions may also prove to be dangerous for travelers. Unexpected rain or snow can be stressful and downright dangerous. Using this application, users would be able to mitigate the danger and make informed decisions based on the information provided. Safety is a major concern when travelling, therefore this application will aid in that regard.

2.0 Objective

The objective of this project is to create an application which will provide information regarding weather and route tracking for the user while traveling. This application aims to provide relevant weather information in a concise and helpful way. The goal is to allow users to make informed decisions about weather conditions so that long road trips are safer and less hectic.

3.0 Background

3.1 Key Concepts

The technologies necessary for completing this problem are the Google Maps API and the Open Weather API.

Google API's:

Google Maps API allows you to "Build customized, agile experiences that bring the real world to your users with static and dynamic maps, Street View imagery, and 360° views" ("Google Maps Platform Documentation"). We used several other Google API's to aid in the development of our Map functionality. The Directions API, which was used for getting a route between markers, was a particularly important one. Additionally, we used the Geolocation API to convert coordinates to addresses and the Distance Matrix API for getting Arrival Times.

Volley Networking API:

This API was used to make synchronous calls to the different API's we used. It was added in due to some issues we were having where we were attempting to access information received from a request prior to actually having received it. This was not used for all of our http calls, but very helpful in the later stages of the project, as it greatly reduced the effort to make a call to an API and allowed for information to be saved quickly and efficiently.

Open Weather API:

Open weather's API allows access to current weather data for any location and the current weather is frequently updated based on global models and data from over 40,000 weather stations. This API also provides hourly forecasts and is geographically accurate. For our project we utilized the current weather data and applied over time forecast data along a route.

3.2 Related Work

Some developers who have done work in this area are Morecast, who have created a browser and phone application that allows the user to see the weather on their designated route. However, their mobile application is not optimized and does not allow you to zoom out as much as on the web application. The application does not offer any other alternative routes; it only offers one route from each location. Additionally, it provides no routing assistance, so it cannot be used while getting directions to the destination. This makes it either useless or very dangerous for people traveling alone. Our full implementation would fix this problem and allow users to type in their specific home addresses and get unique directions and weather advisories for their specific route. These advisories would layer on top of the directions with no input needed from the user ("Route Weather"). While live advisories are not a current feature of the application, the user can still track current weather conditions in their relative area, giving them some indication on if it is safe to proceed.

4.0 Design

4.1 Requirements and Design Goals

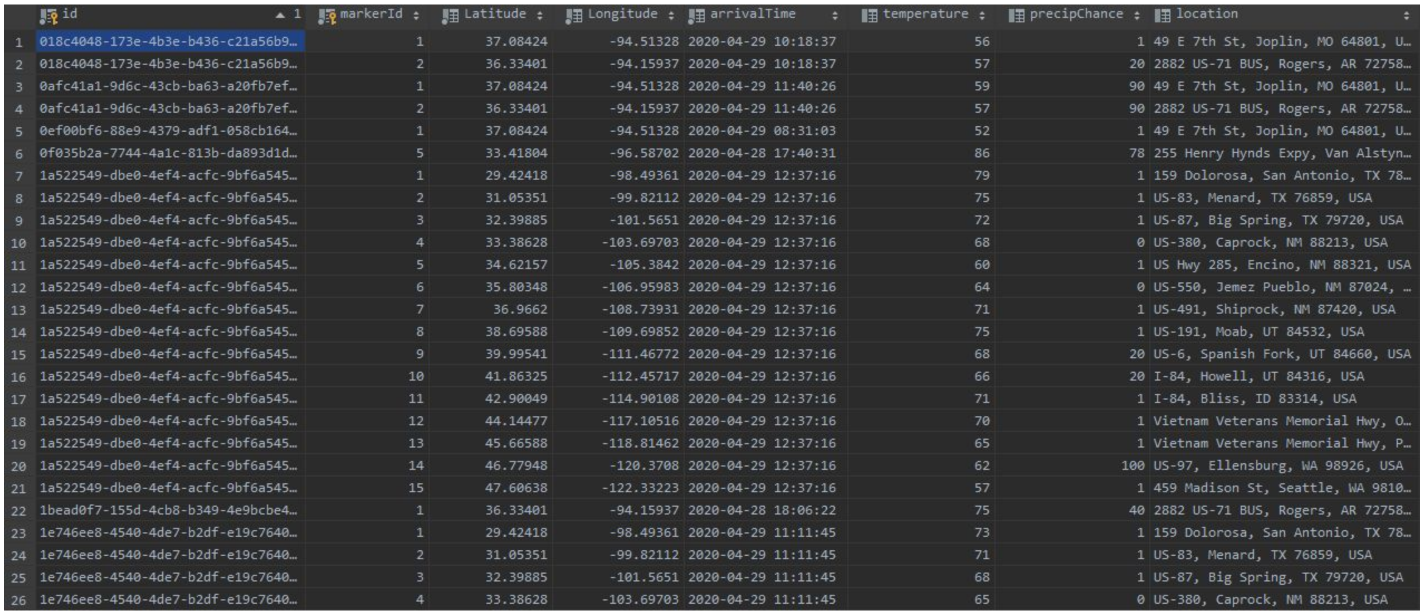
For this project there is one central use case. Therefore, the app will be streamlined to the application, making it easy to find relevant information quickly. Using the app, users should be able to make a route from their location to their destination and the app will track them along this route. This route information will be used to define points along the route. These points will be matched to locations where information from the national weather service is being collected. This information will then be used to predict weather along the user's route. The app should be streamlined to make it easy to start a new route, delete an old route and have continuously updating weather information as predictions change.

There are five requirements for this app to be considered fully functioning. The first is that it must be able to design a route from the user to their destination. Second, it must be able to show an hourly forecast based on the route's timing and location predictions. Third, the information in the forecast must be updated regularly along the route to keep weather up to date. Fourth, this app should use the Google Maps API to give users directions while a weather icon and brief weather information is displayed in a header. This header will usually show the next forecasted weather location and the time when the change will occur. However, if there is a weather emergency the emergency weather alert will replace the normal header. Finally, this app should be functional on common Android devices.

4.2a Detailed Server Architecture

Before beginning development on the back-end components of this project, we had to decide what kind of technology stack would work best for our application and what we hoped for it to accomplish. For the Weather Ways application, we first had to decide what operating system we wanted to utilize for the development of the application, in the end, we chose to use the Android OS. For our runtime environment, we choose to use Node.js to create an event-driven, non-blocking I/O model that makes it lightweight and efficient, which is perfect for data-intensive real-time applications, and typescript as our primary programming language throughout the server code. MySQL was the database we used for the application, MySQL is a free-to-use and open-source database. Once we had decided upon our technology stack, we were able to start on the development of the database.

For this application in our MySQL database, we only have one table, the *Markers* table, it is the responsibility of the table to store the individual markers along the user's route. The data saved inside the table is the user's identification, marker identification, latitude, longitude, the user's arrival time to the marker, the temperature at the marker, the precipitation chance at the marker, and finally the location of the marker. Both the latitude and longitude attributes are defined as floats, while the temperature and precipitation chance attributes are defined as integers. Our arrival time attribute is defined as a timestamp which is used for values that contain both date and time parts which for an application such as this would be necessary especially if a user would be traveling over multiple days along the same route. The table's primary key requires that any attempt to parse data within the database requires the referencing of the user's identification and the marker identification. Both of these attributes are integers that are not allowed to be NULL while all other attributes can be NULL. The user's identification, marker identification, latitude, longitude, and the user's arrival time are provided to the database through the server's communications with the client, whereas our precipitation chance and temperature attributes are provided to the database through the server's connection with the OpenWeather API. An example of what the *Markers* table will look like when it contains real marker data can be seen in the image below:



	id	markerId	Latitude	Longitude	arrivalTime	temperature	precipChance	location
1	018c4048-173e-4b3e-b436-c21a56b9…	1	37.08424	-94.51328	2020-04-29 10:18:37	56	1	49 E 7th St, Joplin, MO 64801, U…
2	018c4048-173e-4b3e-b436-c21a56b9…	2	36.33401	-94.15937	2020-04-29 10:18:37	57	20	2882 US-71 BUS, Rogers, AR 72758…
3	0afc41a1-9d6c-43cb-ba63-a20fb7ef…	1	37.08424	-94.51328	2020-04-29 11:40:26	59	90	49 E 7th St, Joplin, MO 64801, U…
4	0afc41a1-9d6c-43cb-ba63-a20fb7ef…	2	36.33401	-94.15937	2020-04-29 11:40:26	57	90	2882 US-71 BUS, Rogers, AR 72758…
5	0ef00bf6-88e9-4379-adf1-058cb164…	1	37.08424	-94.51328	2020-04-29 08:31:03	52	1	49 E 7th St, Joplin, MO 64801, U…
6	0f035b2a-7744-4a1c-813b-da893d1d…	5	33.41804	-96.58702	2020-04-28 17:40:31	86	78	255 Henry Hynds Expy, Van Alstyn…
7	1a522549-dbe0-4ef4-acfc-9bf6a545…	1	29.42418	-98.49361	2020-04-29 12:37:16	79	1	159 Dolorosa, San Antonio, TX 78…
8	1a522549-dbe0-4ef4-acfc-9bf6a545…	2	31.05351	-99.82112	2020-04-29 12:37:16	75	1	US-83, Menard, TX 76859, USA
9	1a522549-dbe0-4ef4-acfc-9bf6a545…	3	32.39885	-101.5651	2020-04-29 12:37:16	72	1	US-87, Big Spring, TX 79720, USA
10	1a522549-dbe0-4ef4-acfc-9bf6a545…	4	33.38628	-103.69703	2020-04-29 12:37:16	68	0	US-380, Caprock, NM 88213, USA
11	1a522549-dbe0-4ef4-acfc-9bf6a545…	5	34.62157	-105.3842	2020-04-29 12:37:16	60	1	US Hwy 285, Encino, NM 88321, USA
12	1a522549-dbe0-4ef4-acfc-9bf6a545…	6	35.80348	-106.95983	2020-04-29 12:37:16	64	0	US-550, Jemez Pueblo, NM 87024, …
13	1a522549-dbe0-4ef4-acfc-9bf6a545…	7	36.9662	-108.73931	2020-04-29 12:37:16	71	1	US-491, Shiprock, NM 87420, USA
14	1a522549-dbe0-4ef4-acfc-9bf6a545…	8	38.69588	-109.69852	2020-04-29 12:37:16	75	1	US-191, Moab, UT 84532, USA
15	1a522549-dbe0-4ef4-acfc-9bf6a545…	9	39.99541	-111.46772	2020-04-29 12:37:16	68	20	US-6, Spanish Fork, UT 84660, USA
16	1a522549-dbe0-4ef4-acfc-9bf6a545…	10	41.86325	-112.45717	2020-04-29 12:37:16	66	20	I-84, Howell, UT 84316, USA
17	1a522549-dbe0-4ef4-acfc-9bf6a545…	11	42.90049	-114.90108	2020-04-29 12:37:16	71	1	I-84, Bliss, ID 83314, USA
18	1a522549-dbe0-4ef4-acfc-9bf6a545…	12	44.14477	-117.10516	2020-04-29 12:37:16	70	1	Vietnam Veterans Memorial Hwy, O…
19	1a522549-dbe0-4ef4-acfc-9bf6a545…	13	45.66588	-118.81462	2020-04-29 12:37:16	65	1	Vietnam Veterans Memorial Hwy, P…
20	1a522549-dbe0-4ef4-acfc-9bf6a545…	14	46.77948	-120.3708	2020-04-29 12:37:16	62	100	US-97, Ellensburg, WA 98926, USA
21	1a522549-dbe0-4ef4-acfc-9bf6a545…	15	47.60638	-122.33223	2020-04-29 12:37:16	57	1	459 Madison St, Seattle, WA 9810…
22	1bead0f7-155d-4cb8-b349-4e9bcbe4…	1	36.33401	-94.15937	2020-04-28 18:06:22	75	40	2882 US-71 BUS, Rogers, AR 72758…
23	1e746ee8-4540-4de7-b2df-e19c7640…	1	29.42418	-98.49361	2020-04-29 11:11:45	73	1	159 Dolorosa, San Antonio, TX 78…
24	1e746ee8-4540-4de7-b2df-e19c7640…	2	31.05351	-99.82112	2020-04-29 11:11:45	71	1	US-83, Menard, TX 76859, USA
25	1e746ee8-4540-4de7-b2df-e19c7640…	3	32.39885	-101.5651	2020-04-29 11:11:45	68	1	US-87, Big Spring, TX 79720, USA
26	1e746ee8-4540-4de7-b2df-e19c7640…	4	33.38628	-103.69703	2020-04-29 11:11:45	65	0	US-380, Caprock, NM 88213, USA

The server utilizes a package known as sequelize to facilitate the communication between itself and the client side. This package has all the features we needed in order to relay information to the client as well as retrieve information from the database. Sequelize's main feature we use is the promise and resolve feature. It utilizes this in order to wait for responses from the client or the database. This feature is used throughout the server in all of the command functions.

The command functions in the server include: create, delete, query, queryall, and update. The create command receives a MarkerID, id(user identification), longitude, latitude, and arrival time from the client. Once it receives this information it checks for illegal conditions on the MarkerID and id(user identification). It takes the longitude and latitude and queries the OpenWeather database for precipitation chance and temperature at the marker. This function also implements a check to verify if a MarkerID/id(user identification) combination already exists. Once it has passed all these steps it creates a transaction to actually create a marker. The *Markers* table within the database receives this transaction as a promise and saves the collected information. Once the database has saved, it resolves the promise and finishes the transaction.

The delete command takes the id(user identification) provided by the client, and then destroys all associated MarkerIDs within the database related to the specified id(user identification). If no markers exist with that id(user identification) then the server will ignore the delete command, the server will not send out an error message about the command. Once the command has passed the check the server creates a transaction, then the database will receive the transaction and delete the appropriate markers. Once the markers are deleted the database resolves the promise and the transaction finishes.

The query command can be used by the client in order to verify the integrity of the database in case they cannot receive values from the other queries. It creates a transaction asking for all the information in the database. The server sends this command out in the form of a promise, the database returns all of the entries held within the *Markers* table and then resolves the promise.

The queryall command utilizes the received id(user identification), bringing back all the associated markers, allowing the client to display the markers required for traveling. The first thing the server does is verify that the id(user identification)exists within the database, once it has verified the user exists it creates a promise asking the database for all the markers associated with the provided id(user identification). The database sends the information and resolves the promise on the server side, allowing the server to send the necessary marker information back to the client. The update command is called by the client in order to check that nothing has changed along the route, or to communicate updated information to the server regarding the data held within the *Markers* table.

The server first checks that the MarkerID/id(user identification) combination exists, then once it verifies that the combination exists it creates a transaction. If the server is updating information within the *Markers* table then it will take the arrival time provided by the client and create a promise and request new information from OpenWeather and the database on what is occuring at the marker's location. The database and OpenWeather send back their respective information and the promise is resolved. However, if nothing has changed along the route then the server will send the original marker information back to the client.

We chose to use an external service to collect the weather data, OpenWeather is an online service that provides live weather data and weather forecasts in real time. We chose to use OpenWeather both for its convenience and its scalability. It has a free tier that is ideal for use with the proof of concept that we are implementing. It also has a paid tier that can handle a more demanding load and provide more detailed information that would be perfect for use on a commercial final product.

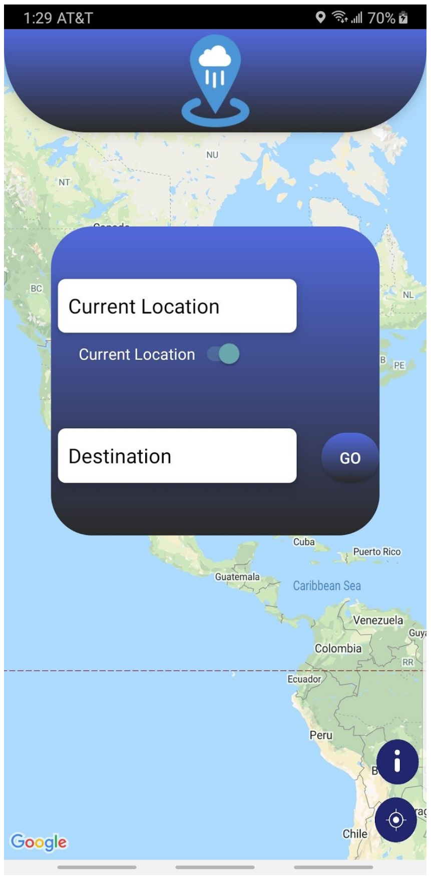
OpenWeather provides weather data using API subscriptions. Through the use of a secret key and the API address, external programs have the ability to request data from the OpenWeather API. The format of the returned data is based on the parameters passed into the API call. In this application, the information request occurs whenever a new marker is made so that the weather data can be stored in the database along with the other marker information. The server uses latitude and longitude values of the marker and passes them as parameters. The API automatically finds the nearest weather data location using Euclidean distance. Imperial units are also passed as a parameter into the API call so that the returned temperatures are in fahrenheit instead of the default kelvin units. The data returned from the API is in three parts, but the section with weather data is called the body of the message. This body is a string in the format of a JSON file, and is parsed into an object once it is received by the server. This object contains all of the weather data from the requested location that the API can provide. Using this object, the temperature and precipitation chance data is saved into the database with the new marker. This process is repeated every time the client uses the update marker command, allowing the weather data to stay up to date. An example of the JSON provided to the server can be seen below:

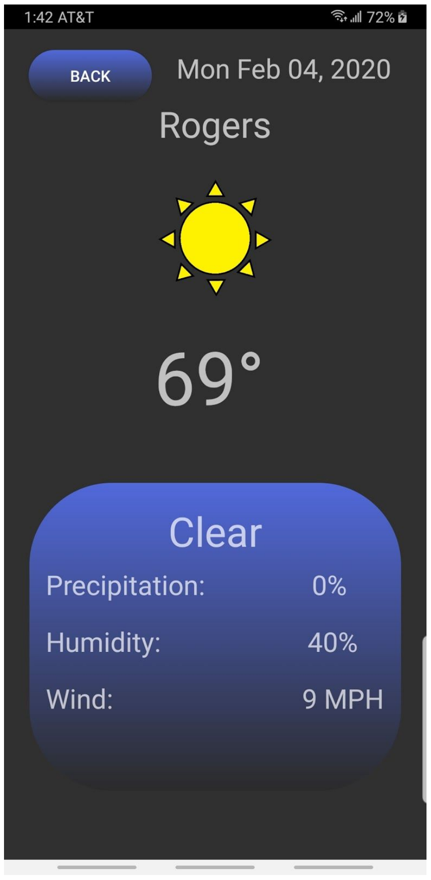
```
2020-04-27T23:26:16.115949+00:00 app[web.1]: body:
{"coord":{"lon":-94.18,"lat":36.39},"weather":[{"id":501,"main":"Rain","descrip
tion":"moderate
rain","icon":"10d"}],"base":"stations","main":{"temp":288.77,"feels_like":286.6
9,"temp_min":288.15,"temp_max":289.15,"pressure":1017,"humidity":82},"visibilit
y":16093,"wind":{"speed":4.1,"deg":100},"rain":{"1h":2.15},"clouds":{"all":90},
"dt":1588029680,"sys":{"type":1,"id":6160,"country":"US","sunrise":1587986834,"
sunset":1588035662},"timezone":-18000,"id":4101260,"name":"Bentonville","cod":2
00}
```

4.2b Detailed Client Architecture

The client-side application is divided into three main pages, requiring detailed weather information to display in the specific format of each page. There is a map page, which contains all of the Google API functionality, a route view of the weather page, which displays weather along the entirety of the route, and specific location page, which displays more detailed weather information regarding a certain location.

The user begins the application and is directed to the maps page, wherein the user can input a beginning location and a final location. Our implementation of Google's Maps API is rather basic, as the only features required to us were the option to create markers at a specific location on a map, and the ability to track the user's current location. Most of this is set up immediately when beginning a Maps project. Different permissions had to be given in order for our application to run, such as location. After the go button is pressed in the directions popup from the Map View, the application creates markers along a route between the entered locations at equidistant intervals. This value can be set manually in the code, though we have future plans to instead split the routes up by time, dependent on the length of the route. To create a route, we had to access the Directions API. A rather simple API, it can take in several different kinds of location based info and return a route between the locations. We required the use of the latitude and longitude variant. Supplying the request URL with two different markers returns a route which is drawn on using a polyline. The markers added to the route are then utilized by sending the coordinates of the marker to the database and sending the information to OpenWeather. The application then pulls the information the server creates for precipitation, arrival time, and temperature, and pulls further detailed information from OpenWeather itself. The data received from Open Weather outside of the server includes wind velocity, weather descriptions, and humidity. This was accomplished by looping over our ArrayList of markers. In order to retrieve this data, longitude and latitude values for each of the created markers is required. The information which has been received is then used to create an ArrayList of weather data objects which will be the main objects for passing information. Once all of the data points have been hit, the toolbar at the top of the page is updated with current weather information for the user's first marker. These new additions can be selected to move to a new page.

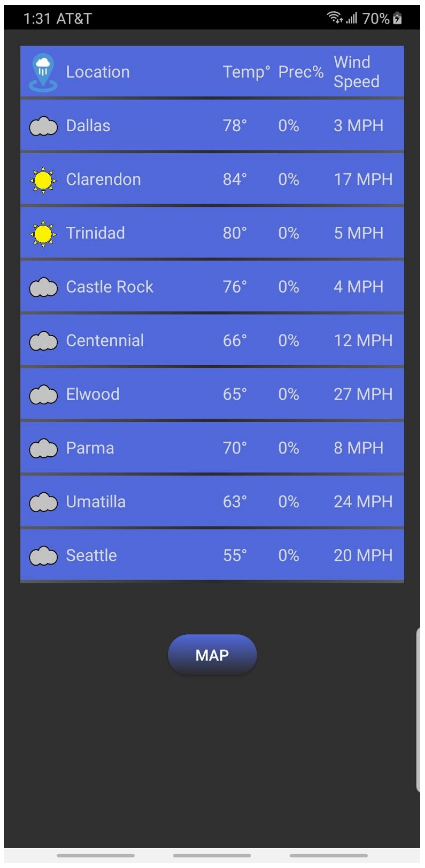




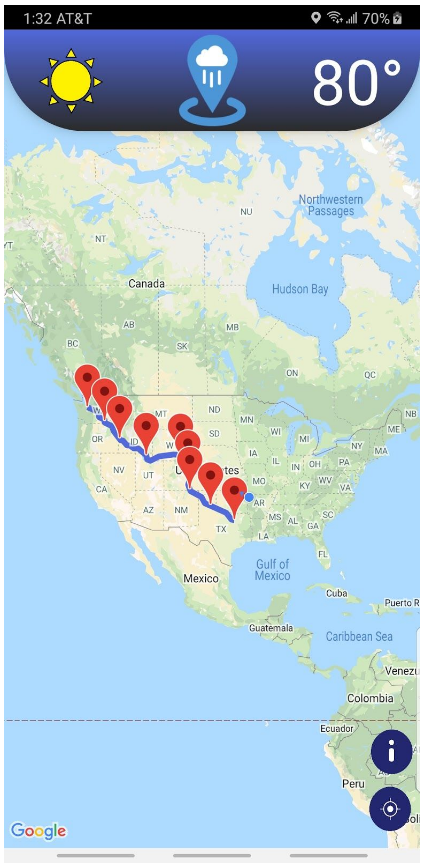
Precipitation:	0%
Humidity:	40%
Wind:	9 MPH

When the user presses the button to access the weather list page, the weather data objects are parceled and become an ArrayList of parcels holding each marker's information. This list is then passed as an intent to the list page, and in the onCreate method of the list page the ArrayList is received as an object. This list of parcels is then saved and the information is read into a separate ArrayList which functions as the list for displaying into the ListView object. If the user then clicks on an object in the list, in order to ensure data is not lost when advancing from the list and to the detailed view page and back, the parcel list is passed into an intent object along with the specific marker information the user pressed. This intent is then read by the detailed view page and the first object, the marker the user clicked, is displayed on the page. The parcel list is simply saved in order to be sent back to the list page when the user finishes on the in depth page.

The In-Depth View contains additional information, not presented in the List View, about a given location along the user's route. To access this page, the user must choose a location from the List View or click on a location on their route from the Map View. The data presented on this page is passed from the List View page, when the location is chosen from the List View, or it is passed directly from the Map View when a location marker is clicked in Map View. The In-Depth page consists of extra information not presented on the List View page, such as the date, a description of the weather, the precipitation, and the humidity. The In-Depth page also contains the same information found on the List View page; the location, an icon representing the weather conditions, the temperature, and the precipitation. In addition to the information presented to the user, there is a button labeled "BACK" in the top left corner of the screen, so the user is able to return to the List View page.


1:31 AT&T 70%

Location	Temp°	Prec%	Wind Speed
Dallas	78°	0%	3 MPH
Clarendon	84°	0%	17 MPH
Trinidad	80°	0%	5 MPH
Castle Rock	76°	0%	4 MPH
Centennial	66°	0%	12 MPH
Elwood	65°	0%	27 MPH
Parma	70°	0%	8 MPH
Umatilla	63°	0%	24 MPH
Seattle	55°	0%	20 MPH



The architecture of the client-server communication from the client's side consists of sending the server side information retrieved from Google Maps API. Once the user has determined the route they will be taking, locations along the route have been chosen for markers, user's ID, marker ID, the latitude, longitude, and arrival time of each marker is sent to the server. In order to send information to the server, the Weather Ways application utilizes the built-in Java HTTP API. This library provides the needed functionality to send, receive, and delete information from the database. The passing of information between the client and server is done by passing the information in JSON format. When the client side sends this information to the server, the server-side is able to return forecasted weather information about the given location. The information received from the server consists of the user's ID, the marker ID, location, latitude, longitude, arrival time, temperature, and precipitation. Once the client-side obtains the requested information from the server, the information can then be moved between the application pages as needed. To aid in receiving data from our API’s, we began using the Volley API. This was not initially a feature to be included in our application and thus was added late in the semester. It greatly reduced the stress involved with making requests and receiving responses without desynchronizing the rest of the code. Specifically, the Volley Callback feature was implemented as a means to ensure we retrieve any data requested prior to using it.

Two other API's that were not immediately mentioned already, but played a vital role in our architecture, are the Geocoding and Distance Matrix API’s. The Geocoding API was used to retrieve physical addresses from our marker. We simply needed to pass in the longitude and latitude values as parameters, and it would return an Address type, which we converted to a string to send to the server. The Distance Matrix API was used to gather the travel time from our initial location. This travel time was used to calculate the arrival time to each of the markers. The arrival time currently has limited functionality due to the weather API’s we had access to. With access to a higher grade API, this feature would be used in much more effective ways, as we could use specific times to grab weather data at that time.

4.3 Future Work

As the semester comes to a close, and our work on this project comes to an end there are definitely some aspects of the project that we would expand upon if we were to continue work on this project. Some of the changes would be expanding the marker object to contain more weather data, for example we would want to add in highs and low temperatures for the marker, forecasted weather over time, etc. Another feature we would add if we were to continue work on this project would be weather alerts to the application so that while a user was driving they would be able to receive weather alerts either about their current location or areas they will soon be entering. For example if they are driving and they are going into an area where there is a frost warning they would get a notification of that warning so they could better decide how to proceed with their trip. Another set of features we would add in the future would be better icon support and more detailed weather descriptions.

4.4 Risks

Risk	**Risk Reduction**
Tied to a single platform for development	Research options and choose a platform that will allow growth and support
Depend on service to provide weather data	Choose a weather API with a long history and good reputation

4.5 Tasks

1. Research:
    a. Determine if the idea has already been created by someone else.

        i. Look for similar products, and what they did differently.
    b. Target Platform.
        i. Decide on target device.
    c. Programming Language
        i. Programming language should be used to create the application.
    d. Frameworks
        i. Open source vs. proprietary frameworks available.
        ii. Locate a mapping framework for the basis of the application.
        iii. Locate a weather forecasting framework that will provide a database of forecasted weather conditions.
    e. Database
        i. Determine the need for a database
    f. Source Control
        i. Determine how source code will be maintained.
    g. Determine if the idea is conceivable.
        i. Determine if it can be produced in the timeline.
        ii. Possess the needed skills to produce the application.
    h. Target Audience
        i. Desired users.
2. Software Development Approach:
    a. Decide on software process model for the application (Agile, waterfall, etc.).
    b. Assign roles for each member of the team.
    c. Determine meeting times and dates.
3. Design:
    a. User Interface.
        i. Greeting screen.
        ii. Loading Screen.
        iii. Menu Design.
        iv. Main Screen.
    b. Determine application architecture.
4. Implementation:
    a. Divide work between team members.
    b. Construct the application.
        i. Create the smaller components of the application
        ii. Merge smaller components, ensuring there is not issues.
5. Testing:
    a. Determine the application works as intended.
    b. Obtain user feedback.
    c. Determine need for improvements.
6. Documentation:
    a. Create a report outlining the application
        i. The goal of the application
        ii. The process of creating the application
        iii. The desired end user

4.6 Schedule

Task	Timeframe
Research: • Discuss the overall goal of the project • Determine the application does not already exist • Look into Frameworks needed • Decide on programming language • Determine target device for application • Decide if the application is conceivable, in the given time frame and with the team's current skill set	01/13/2020 - 01/27/2020 01/13/2020 – 01/20/2020 01/20/2020 – 01/27/2020
Software Development Approach: • Determine the software model for the project • Assign team member roles for the project • Discuss scheduling for future meetings	01/27/2020 – 02/03/2020 01/27/2020 – 02/03/2020
Design: • Determine how the application should look when complete • User interface • User interaction with the application • Navigation within the application • Decide how the application architecture should interact	02/03/2020 – 02/10/2020 02/03/2020 – 02/10/2020
Implementation: • Meetings to discuss progress with individual parts • Decide how the work will be divided • Construct the application • Converge completed modules of the application as applicable	02/10/2020 – 03/09/2020 02/10/2020 – 02/17/2020 02/17/2020 – 03/09/2020
Testing: • Test application for bugs • Perform real world testing • Get user feedback from testers	03/09/2020 – 03/16/2020 03/09/2020 – 03/16/2020
Documentation: • Document the application • Describe its purpose • Explain how the application was designed and created • Illustrate the need for the application	03/16/2020 – 04/03/2020 03/16/2020 – 04/03/2020

4.7 Deliverables

- Android Studio Java code
    - This will include code for all three pages of our application and the helper classes that they use. These different classes include those to aid in client and server communication, object classes (primarily for location markers), interfaces, layout files, etc.
- Heroku server
- Final report
- Website code

5.0 Key Personnel

Nicholas Brinkley - Brinkley is a senior Computer Engineering major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed relevant courses including Software Engineering, Database Management, and Big Data Programming. Brinkley will be on the backend team with a focus on setting up the Heroku server. He will be the account holder for the Heroku free tier and help support any other backend functions.

Zachary Cantrell — Cantrell is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas and has completed a minor in Mathematics at the University of Arkansas. He has completed Software Engineering, Programming Paradigms, and Computer Networks. He has experienced socket programming, mobile programming, and database programming. Cantrell will be on the frontend team, focusing on getting and sending information with the server to ensure that the weather data is up to date, along with facilitating and other frontend projects.

Madison Galloway – Galloway is a senior Computer Engineering major in the Computer Science and Computer Engineering Department at the University of Arkansas and will also be completing a minor in Mathematics through the University of Arkansas. She has completed relevant courses including Software Engineering, Database Management, Information Security, and Big Data Analytics and Management. She has experience in Java and C. This experience comes from both school projects and personal projects. Galloway will be the backend lead, writing code for the server and working as the dedicated liaison with the frontend team.

William Henness - Henness is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed relevant courses including Software Engineering, Database Management, Programming Paradigms, Information Security, and Mobile Programming. He has experience with java programming, having completed an internship with Cerner on the FetaLink desktop team and several school projects. Additionally, he has experience with C++ and Python, mostly from personal and course projects. Henness will be the frontend lead, focusing on the Google Maps API and working as the dedicated liaison with the backend team.

Audrey Timmerman – Timmerman is a senior Computer Engineering major in the Computer Science and Computer Engineering Department at the University of Arkansas. She has completed Software Engineering along with Programming Paradigms and has experience in Java and writing simple apps that interface with a server. Timmerman will be the team lead and be responsible for overall organization as well as being on the backend team.

David Turnbough - Turnbough is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed relevant courses including Software Engineering, Database Management, Cloud Computing, and Information Security. Turnbough will be on the frontend team, focusing on the user interface and testing as well as supporting other aspects of the frontend.

6.0 Facilities and Equipment

The only equipment we needed for this was our own personal computers and Android Studio code development software. We also made use of a Heroku server to store user data if we decide to go that route.

7.0 References

"Google Maps Platform Documentation." *Google Maps Platform | Google Developers*, Google, 2019, developers.google.com/maps/documentation.

Hall, Julie. "AAA: Nearly 100 Million Americans Will Embark on Family Vacations This Year." *AAA NewsRoom*, 20 Mar. 2019, newsroom.aaa.com/2019/03/100-million-americans-will-embark-on-family-vacations/.

"Route Weather." *Morecast*, Morecast, 2019, morecast.com/en/plan-your-route.