Application Pass Down
Oregon Gardening Application
Frost Date Finder

Creators

Kirsten Corrao (kirstencorrao@gmail.com, https://github.com/kcroo)
John Lebens (lebensjohns@gmail.com, https://github.com/lebensjohns)
Andrew Clos (aclos3@gmail.com, https://github.com/aclos3)

Note:

Initially, this project was called "Oregon Gardening Application." At the request of our client, the name was changed to "Frost Date Finder."

Purpose:

The purpose of this document is to assist future developers and maintainers of the Frost Date Finder by serving as a resource to familiarize themselves with the structure of the program.

Background:

The project was started in September 2020 by a group of three undergraduate Computer Science students at Oregon State University (OSU). The program was delivered to the client in early December 2020 as a functioning web, Android, and iOS application.

The application was requested by Dr. Gail Langellotto, a professor in the OSU Urban and Community Horticulture Extension Department. The goal of the application was to provide users in the state of Oregon with growing season data based on their location. The data would inform the user of the last frost date in the spring, the first frost date in the fall, and the length of the growing season. The probability of these occurrences is displayed for mild (32F), moderate (28F) and severe (24F) freezing temperatures. The raw data is taken from the NOAA Climate Normals (1981-2010) and allows anyone in the US to view their local frost dates.

The app was created with the Ionic framework, which enables simultaneous web, Android, and iOS development. Using this framework has the distinct advantage of only having to create and maintain one codebase but building three separate products. It also uses common web languages and technologies, which the developers were already familiar with given previous coursework.

Program Flow:

The logical layout of the program can be seen in the flow diagram below. On first load, the user is greeted with a welcome screen that displays the Oregon State University logo and the name of the application. Next, the homepage (HomePage.tsx) appears that lets the user choose to use the device location (browser or phone GPS) or enter a city/state or zipcode. Upon entering text, input validation occurs and errors are alerted to the user.

Once valid input is received, a new page (ResultsPage.tsx) is rendered. Depending on the type of location information provided, one of four routes is taken.

1. If device location is used and elevation data *is* included (phone GPS) the resultant latitude, longitude and elevation are used.
2. If device location is used and elevation data *is not* included (web browser or non-mobile), an API call is made to the USGS lat/long to elevation service to get the elevation.
3. If a zip code was entered, an API call is made to OpenDataSoft's Zipcode to Lat/Long service and the returned latitude and longitude are used to make a call to the USGS lat/long to elevation service.
4. If a city and state were entered, an API call is made to OpenDataSoft's Cities and Towns service to get latitude, longitude and elevation.

Next, the NOAA climate normals stations are sorted by distance from the user's location and then the closest station is checked to see if it contains the type of data required. If data is not found, the next closest station is chosen until the appropriate data is found. This station information, including: frost dates, frost-free days, station name, station elevation, and station distance from the user are displayed.

On both the home page and results page, there is a question mark button in the upper right corner of the screen that will take the user to an 'about page (AboutPage.tsx) which gives additional information about the project and the data used. From the results page or about page, a back arrow button appears in the upper left corner of the screen. Built-in device back buttons will also return the user to the previous screen.

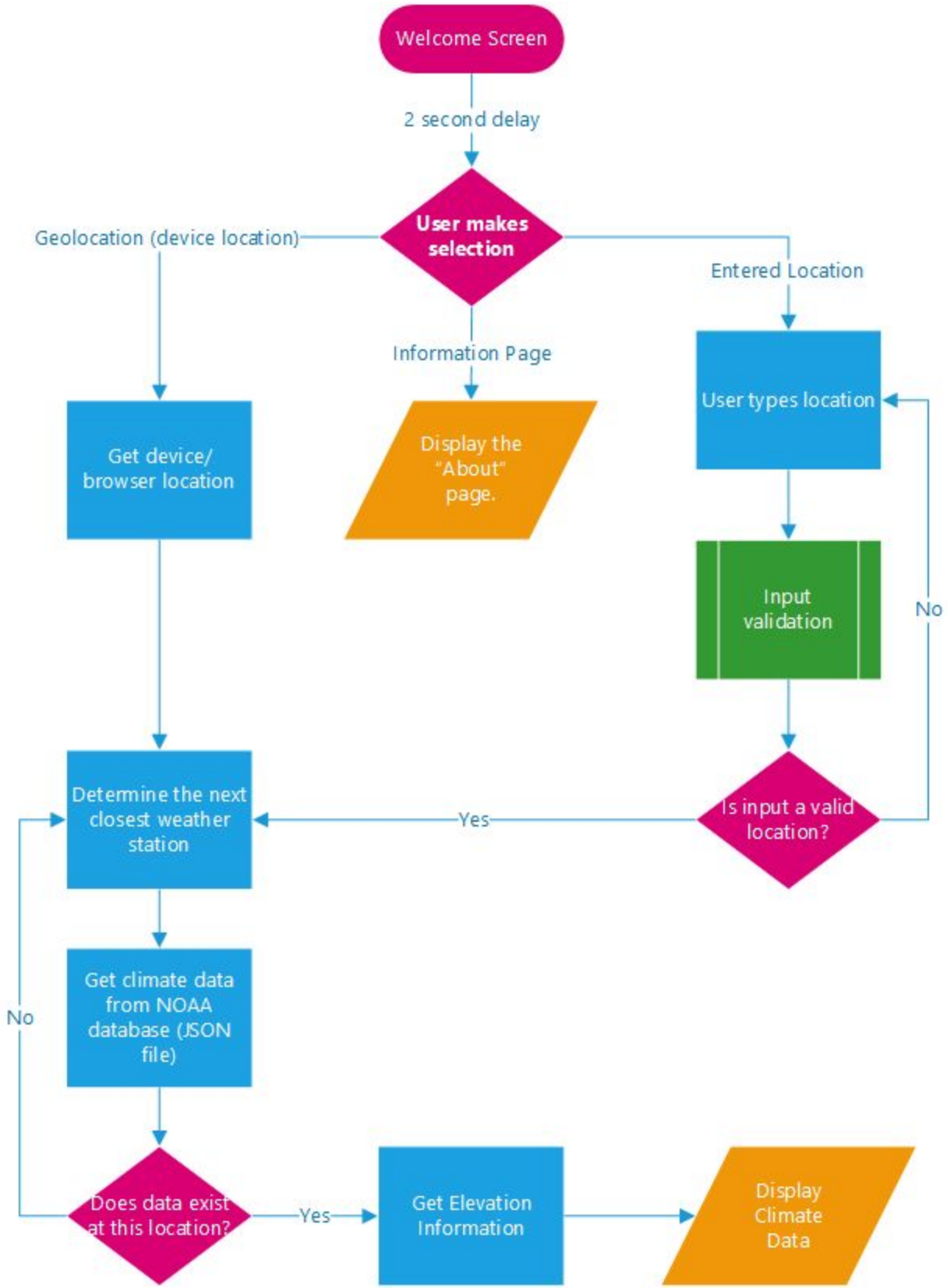

Figure 1. Frost Date Finder Flow Diagram

Code:

The Frost Date finder was built using the Ionic Framework, version 6.11.10. The React Javascript library was used to build the user interface, and the program was coded using the TypeScript language. Throughout development, testing was done primarily within the Chrome internet browser on both PC and Mac computers. Additionally, testing was regularly performed on physical and emulated Android devices. Finally, periodic testing occurred using XCode and emulated iOS devices.

A folder tree with the most relevant files highlighted in blue is shown below.

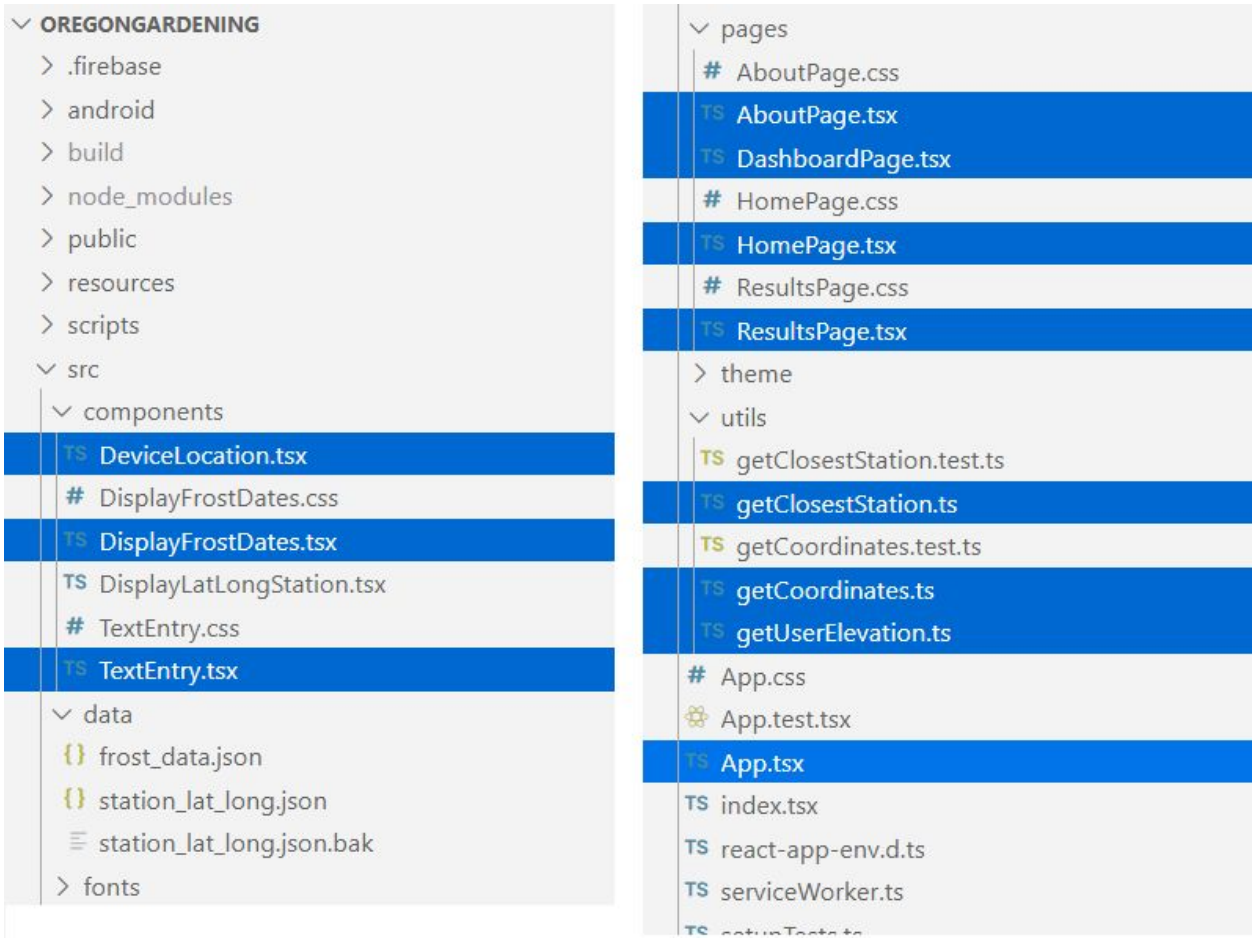

Figure 2. Folder tree with significant files highlighted in blue.

File	Description
DeviceLocation.tsx	Component that gets the user's location and elevation using their device's GPS.

DisplayFrostDates.tsx	Component that displays frost dates and growing season length for a single frost severity.
TextEntry.tsx	Component that gets text input for the user's zip code or city and state, then calls an API to get their latitude, longitude, and elevation.
AboutPage.tsx	Page that displays information about the app, such as the data source and development team. The home and results page both have links to the about page.
DashboardPage.tsx	Page that serves as the launching point for the app. It creates routes for the home, results, and about pages.
HomePage.tsx	Page that uses the DeviceLocation and TextEntry components to get the user's location. When a location is found, it sends the user's latitude, longitude, and elevation to ResultsPage.
ResultsPage.tsx	Page that gets the user's frost dates and displays them. It receives the user's latitude, longitude, and elevation from HomePage. It finds the closest weather station, then displays that station's frost dates and growing season lengths for low, moderate, and severe frost severity.
getClosestStation.ts	Determines the closest weather station to the user's location and returns the station information and its frost data.
getCoordinates.ts	If the user enters a zip code or city/state pair, these functions call Opendatasoft APIs to determine their latitude and longitude. The city/state API also returns the user's elevation, but the zip code API does not.
getUserElevation.ts	If the user entered their zip code, this function calls a USGS API to get their location's elevation.
App.tsx	This is the entry point for the app. It creates the entry routes and renders the DashboardPage.

Table 1. Files containing significant code and a brief description.

Getting the program running:

1. To get the program up and running in your environment, it is highly recommended that you follow the "Getting Started" portion of the Ionic Framework documentation found here: https://ionicframework.com/docs.

2. Once your environment is ready with the latest versions of Node.js and npm installed, clone the following GitHub repository to your local machine.
   https://github.com/aclos3/oga
3. Open the command prompt or terminal of your choice and install the Ionic Command Line Interface:
   ```
   npm install -g @ionic/cli
   ```
4. From the terminal, navigate to the directory cloned from GitHub. You should be in the directory that contains package.json and the 'src' folder among other items. Run the following command:
   ```
   npm install
   ```
5. Build the program on your local machine with the following command:
   ```
   ionic cap sync
   ```
6. If errors occur during the build, read them carefully and install any missing packages or plugins and retry step 5.
   Note: If many errors persist, delete the node_modules folder and package-lock.json file and re-run npm install.
   Note: refer to the Package Versions section below and the package.json file if you are having dependency and plugin version issues when building the application.
7. Once the build has succeeded, launch the application in a web browser with the following command:
   ```
   ionic serve
   ```
8. The application should launch in a new browser tab in your default web browser, but if it does not, open your browser (Chrome is recommended) and navigate to the following URL:
   http://localhost:8100/
9. To build the application in Android or iOS, refer to the Ionic documentation for Capacitor and React here:
   https://ionicframework.com/docs/developing/android or
   https://ionicframework.com/docs/developing/ios
   Commands:
   ```
   ionic cap build android
   ionic cap build ios
   ```

Package Versions

It is best to allow the framework to perform the plugin installations during the build step, but if manual installation becomes necessary, information about some of the important plugins can be found below.

- cordova-plugin-geolocation

- The app uses version 4.0.2. The latest version (currently 4.1.0) causes a problem with the Android manifest that is unresolved as of 11/28/2020.
- Make sure you are running version 4.0.2 of the cordova-plugin-geolocation

```
npm install cordova-plugin-geolocation@4.0.2
```

NOAA API

NOAA's weather station is stored on the National Centers for Environmental Information (NCEI), located here: https://www.ncdc.noaa.gov/data-access/land-based-station-data/land-based-datasets/climate-normals/1981-2010-normals-data

Documentation on the meaning of these data types and methodology used to create them can also be found in \data_info\ of the repository.

Zip API

Opendatasoft.com's US Zip Code Latitude and Longitude API was used to convert a user's entered zipcode into a latitude and longitude. A query is made in the following format:

https://public.opendatasoft.com/api/records/1.0/search/?dataset=us-zip-code-latitude-and-longitude&q=zip=97331

Results are returned in the following format:

```
{
    "nhits": 1,
    "parameters": {
        "dataset": "us-zip-code-latitude-and-longitude",
        "timezone": "UTC",
        "q": "97331",
        "rows": 10,
        "start": 0,
        "format": "json",
        "facet": [
            "state",
            "timezone",
            "dst"
        ]
    },
    "records": [
        {
            "datasetid": "us-zip-code-latitude-and-longitude",
            "recordid": "702b407c3ef9824912b30b4f23ccbc02d8b41b19",
            "fields": {
                "city": "Corvallis",
                "zip": "97331",
                "dst": 1,
                "geopoint": [
                    44.565648,
                    -123.27906
                ],
                "longitude": -123.27906,
                "state": "OR",
                "latitude": 44.565648,
                "timezone": -8
            },
```


City, State API

Opendatasoft.com's US Cities and Towns of the United States API was used to convert a user's entered city and state into a latitude, longitude and elevation. A query is made in the following format:

https://public.opendatasoft.com/api/records/1.0/search/?dataset=cities-and-towns-of-the-united-states&q=name=corvallis&refine.state=OR

Results are returned in the following format:

```
{
    "nhits": 1,
    "parameters": {
        "dataset": "cities-and-towns-of-the-united-states",
        "timezone": "UTC",
        "q": "corvallis, 'or",
        "rows": 10,
        "start": 0,
        "format": "json",
        "facet": [
            "feature",
            "feature2",
            "county",
            "state"
        ]
    },
    "records": [
        {
            "datasetid": "cities-and-towns-of-the-united-states",
            "recordid": "61b4251ab7e2851876671a20c4ac5653180e7b6c",
            "fields": {
                "popplat": 44.5645659,
                "elev_in_m": 69,
                "feature2": "County Seat",
                "name": "Corvallis",
                "county": "Benton",
                "gnis_id": 1140162,
                "ansicode": "2410237",
                "longitude": -123.2620435,
                "geo_point_2d": [
                    44.564565900000105,
                    -123.26204349999989
                ],
                "state_fips": "41",
                "latitude": 44.5645659,
                "state": "OR",
                "elev_in_ft": 226,
```


Icons, Favicon, and Splash Screens

These images were made in Grep. The Grep files end in .xcf and are located in the /resources folder. The full-sized images are also located there, as well as the original snowflake and OSU logo. The resources/android and resources/ios folders contain copies of all the icons and splash screens needed for the respective device.

To create new icons and splash screens, create the images, name them icon.png and splash.png, and save them in the /resources folder. The icon image should be 1024x1024 pixels

and a PNG file. The splash screen image should be 4096x4096 pixels and a PNG file. Once the icon and splash images are finished, run "npm run resources", which runs cordova commands and a script that creates all of the necessary sizes of icons and splash screens and saves them to the appropriate folders for Android and iOS. Make sure that builds for both Android and iOS exist--otherwise the script won't have anywhere to save the final images.

To create a new favicon, just replace the favicon.png file in /public/assets/icon. The current image is 64x64 pixels.

The current icon, splash screen, and favicon use this public domain snowflake and the OSU logo.

To make changes to the splash screen settings (duration, spinning wheel, etc), refer to this Capacitor splash screen documentation. Remember to run "npx cap copy" after editing splash screen settings in capacitor.config.json.

Data

The data for this application comes from the NOAA Climate normals 1981-2010 dataset. Since irregularities of first and last frost dates can often be jagged or "noisy", a bootstrapping method is employed to account for this. This is accomplished by running 10,000 simulations on a series of dates for each station. If within that set no frost event occurs, then a percentage of the simulations in which the event occurs is taken instead as 10, 20, 30, 40, 50, 60, 70, 80, or 90 percent. It is good to note that this percentage only affects a minority of stations in warmer climates. Our dataset uses the 30th percentage which means that only in warmer regions where 100 is not available, NOAA will provide the 30 percent data.
More information is available here:
https://www.ncdc.noaa.gov/data-access/land-based-station-data/land-based-datasets/climate-normals/1981-2010-normals-data
More specifically, the publication on the frost data can be found here:
https://www1.ncdc.noaa.gov/pub/data/normals/1981-2010/supplemental/documentation/agricultural-normals-methodology.pdf
The data is available via API calls or FTP. The original implementation of this program made API calls to fetch data for the desired stations, but this introduced significant lag, especially if more than one station would need to be queried. Based on this, the decision was made to download all of the data via FTP and format it to two JSON files.

station_details.json: This file contains information about all of the stations in the NOAA Climate normals database. An example of the formatting is shown here:

```
{
  "id": "USC00351862",
  "latitude": 44.6342,
  "longitude": -123.19,
  "elevation": 68.6,
  "state": "OR",
  "city": "CORVALLIS STATE UNIV"
},
{
  "id": "USC00351877",
  "latitude": 44.5078,
  "longitude": -123.4575,
  "elevation": 180.4,
  "state": "OR",
  "city": "CORVALLIS WATER BUREAU"
},
```


frost_data.json: This file contains the temperature probabilities for first frost, last frost and frost free days for probabilities 10%, 20%, …. to 90%. Based on the needs of our client, the application uses the 30% probability for 24℉, 28℉, and 32℉ temperatures. While this file contains far more data than is currently used, storage and performance impact was minimal and this will allow for future expansion options (selectability of different probabilities and temperatures). An example of the formatting of this file is shown here:


```
{
  "station": "USC00319423",
  "ann-tmin-prbfst-t24Fp10": "11/17",
  "ann-tmin-prbfst-t28Fp10": "11/02",
  "ann-tmin-prbfst-t32Fp10": "10/23",
  "ann-tmin-prblst-t24Fp10": "03/18",
  "ann-tmin-prblst-t28Fp10": "04/03",
  "ann-tmin-prblst-t32Fp10": "04/14",
  "ann-tmin-prbgsl-t24Fp10": "  320",
  "ann-tmin-prbgsl-t28Fp10": "  281",
  "ann-tmin-prbgsl-t32Fp10": "  248",
  "ann-tmin-prbfst-t24Fp20": "11/24",
  "ann-tmin-prbfst-t28Fp20": "11/08",
  "ann-tmin-prbfst-t32Fp20": "10/29",
  "ann-tmin-prblst-t24Fp20": "03/10",
  "ann-tmin-prblst-t28Fp20": "03/29",
  "ann-tmin-prblst-t32Fp20": "04/09",
  "ann-tmin-prbgsl-t24Fp20": "  308",
  "ann-tmin-prbgsl-t28Fp20": "  271",
  "ann-tmin-prbgsl-t32Fp20": "  239",
  "ann-tmin-prbfst-t24Fp30": "11/29",
  "ann-tmin-prbfst-t28Fp30": "11/13",
```

A python script called “frost_adding.py” is included in the /src/utils that will connect to NOAA’s NCDC database, download the raw agricultural normals files, combine and format them into a new frost_data.json file that is readable by this application. If NOAA data is added or updated, this script can be used to update the application data. Ensure that Python version 3.7+ is installed on your system before running the script.