User Manual
(UM)

Field Progress Web App

Team 04

Uche Uba: Project Manager and Frontend Lead
Akanksha Diwedy: Operational Concept Engineer
Mayank Kulkami: Requirements Engineer and Backend Lead
Madhavi Shantharam: Life Cycle Planner
Sahithi Vlema: Software Architect
Aisharya Joisa: Feasibility Analyst
Kevin Grimes: IIV & V and Quality Focal Point Engineer

08/12/2019

The following are a set of steps to follow in order to download and use the application on one's local machine:

1. Downloading the repository

Currently, in order to utilize this product, the user needs to clone the GitHub directory where the source code is located. To do this, type the following command in the terminal:

```
git clone https://github.com/mayankkulkarni/Field_Progress_Map.git
```

Once the repository is on your local machine, change directory into it

```
cd Field_Progress_Map
```

2. Installing Dependencies

While inside the field progress Map directory, create the python virtual environment with the following command:

```
python3 -m venv venv
```

Then activate the virtual environment with the following command:

```
source venv/bin/activate
```

Once the virtual environment is activated, you will see (venv) to the left of the command line. It is now time to install all the python dependencies used for the project. Do so by executing the following command:

```
pip install -r requirements.txt
```

This installs all the python dependencies specified in the requirements.txt file.

(Optional): If using the frontend application for visualization, otherwise the algorithm module in the backend just returns the required geoJSON used for cluster visualization, and the next steps in this section are optional

Next, change directory into the FP_Frontend folder with the following command:

```
cd frontend/FP_Frontend
```

This takes you to the frontend directory located two layers down from the main directory. Then install the dependencies required for react to run with the following command:

```
npm install
```

3. Starting the Python Server

To start the python server, navigate back to the Field_Progress_Map directory and run the following command:

```
python manage.py runserver
```

If for any reason python says it cannot recognize some modules, then just pip install said modules. When the python server runs successfully, you should see the following:

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
December 10, 2019 - 06:22:44
Django version 3.0, using settings 'backend.settings'
Starting development server at http://127.0.0.1:8000/
    Quit the server with CONTROL-C.
```

This indicates that the server is now up and running on port 8000 on your local machine. To test it, navigate to the following URL:

```
http://localhost:8000/api/results/
```

It should render a geoJSON of the initial csv voter list passed to the application

4. Starting the React Server

To start the react server, navigate back into the FP_Frontend server with the same command as previously listed:

```
cd frontend/FP_Frontend
```

once in the front end folder, start the react development server with the following command:

```
npm start
```

if for any reason, it shows some modules were not found, then manually npm install said modules. When the server is running, you should see

```
Starting the development server…
```

Then a browser page should automatically open with its URL set to:

http://localhost:3000

On this page, you should see the react application rendered

5. Using the backend API endpoints

The Django application have the following API endpoints:

i. http://localhost:8000/api/results/
ii. http://localhost:8000/api/clusters/

The first URL send a GET request to the (/api/results) endpoint on the Django server. The response data is a geoJSON containing the initial csv of voter list used in the application. It is basically just a transform of csv to json.

The second URL sends a post request to the (/api/clusters/) endpoint on the Django Server. It takes as an argument a json object with two properties. One being the precinct ID you want to view, and the other being an array of volunteer objects with 2 properties, namely; 'volunteerName' and 'availability'.

In essence the JSON object being sent to the POST request would look like this

```
{
        precinctID: (code of the precinct you want to see its cluster e.g. 130960)

        volunteers: [
                        {volunteerName: 'John Doe',  availability: '2'},            //array of objects
                    ]

}
```

Note: The algorithm uses googles OR-tools Vehicle Routing Problem (VRP) approximate solutions, to produce a result for turf cutting applied to a set of voters within a precinct. However, the number of points that can be passed to the VRP API to produce a free result is 25 points, so in order to return a result for no cost, the voters per precinct should stay within that limit. However if more voters are to be passed to the algorithm, a google access token linked to an account that can be charged, should be included in the file. Currently with the data provided, some precincts that have 25 voters or less are: 130960 and 130970. These can be used to test the optimality correctness

6. Using the frontend for visualization

The frontend application has different components to the UI, and they are as follows:

- Side Pane: This houses the forms, where the volunteer data can be input. For each form, a volunteer name and availability should be input. To create an additional volunteer form, the “Add Volunteer Button should be clicked”. This pane can also be toggled with the Toggle Input button. Lastly, it houses the cut turf button which sends the post request to the backend API

- Precinct Dropdown: This contains a dropdown of precinct codes that are within the precinct data. This information is passed as a parameter to the post request.

- Interactable Map: It contains a map that the user can interact with. The map utilizes various deckGL layers for visualization purposes. Mainly the geoJSON layer for viewing the precinct, and the scatterplot layer for viewing the voters.

- Scatterplot Radius Slider: This slider increases the radius of each individual voter rendered by the scatterplot layer. Aids in better visualization

Once a (valid) POST request is sent, the result rendered by the scatterplot is a set of points that are colored (original voter points are black). These points are also interactable, and once hovered over, would show to the top right of the map a tool tip which displays various information like the name of the voter, their location, which cluster they belong to, and the next house to be visited from that location.