Team Name: Magnum Opus 2.0

Team Members and Emails:

- Michael Bradley mbrad012@ucr.edu
- Edward Carrasco ecarr024@ucr.edu
- Ross McGuyer rmcgu002@ucr.edu
- Benny Pham bpham020@ucr.edu
- Ryan Villena rvill028@ucr.edu

Project: Rooty Tooty Gas Computey (GasGirl): A Better Gas Buddy

Communication Channel: Discord

Presentation Topic: Unity

Scrum Sheet Link:
https://docs.google.com/spreadsheets/d/1CTqRkdWDNTmJhaIrVsp-3uUTMQUsjPpF3NL_3rwYWe4/edit

Rooty Tooty Gas Computey

Description: The app allows the users to find the cheapest gas price based on their car year, make, model, location, and other filters. Users can confirm the gas prices of nearby gas stations.
Links: https://www.gasbuddy.com/App
User Stories:

SPRINT 1:

1. **Show Simple Location as Text**: As a user driving around town, I want to be able to use my location to find the cheapest nearby.
2. **Dropdowns to Select Car Options**: As a user on the go, I want to be able to select my car from a list rather than enter my MPG directly.
3. **Gas Brand Filter:** As a user with reward points for certain brands, I want to see a list where I can filter out gas stations by company
4. **Create Account with Cookies**: As a frequent user of the app, I want to create an account that saves my relevant information such as, vehicle info and recent gas stations.

SPRINT 2:

5. **Smart Location**: As a user more concerned about price rather than time, I want to know if it's worth it to drive to a farther location if it's cheaper at that location based on my miles per gallon.
6. **Gas Stations Map**: As a user that is bad with directions, I want to view the gas stations (as pins) on a map and be able to get the directions with my phone's prefered Map app.
7. **Gas/Location Calculator**: As a user that frequently drives their vehicle until I am nearly out of gas, I would like to find the cheapest gas station based on how much farther I can go.
8. **Custom Filters**: As a user that enjoys customizability, I would like to mix and match the various filter types to fit my personal preferences.

SPRINT 3:

9. **Gas Type Reminder**: As a user unfamiliar with the nuances of vehicle maintenance, I want to be reminded what kind of gas optimizes my vehicle's performance.
10. **Report Price**: As a user that wants to improve the app, I want to report the price of a gas station, so that the prices are updated regularly.
11. **Savings Tab**: As a user that enjoys saving money, I want to see how much money I have saved since I started using the app.
12. **Rush Mode**: As a user that is in a rush, I want the app to suggest the gas station that is fastest to get to.

Design Document

Design document UML can be accessed here:
https://www.draw.io/#G1T_gITxvP_knMkVoB5zUTBetDjoH776hC

Access to the app is gained by typing https://rooty-tooty-gas-computey.herokuapp.com/ into the search bar.

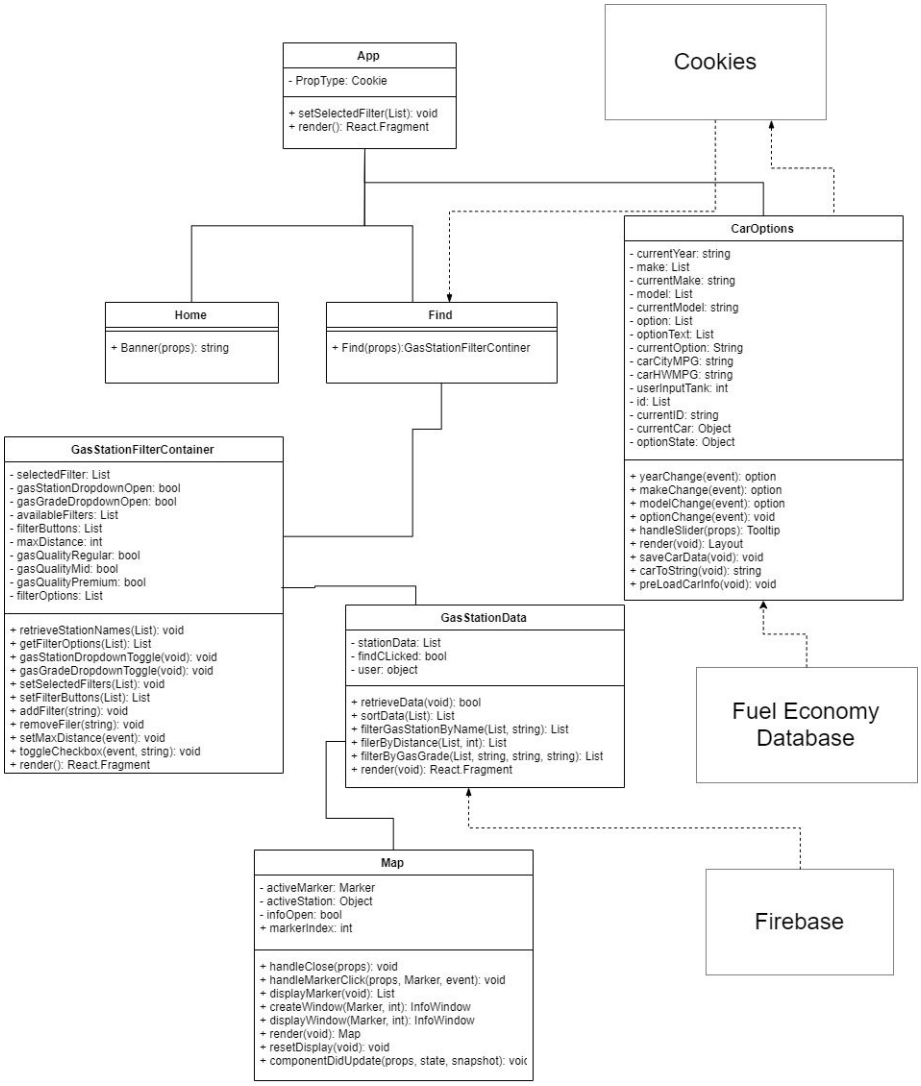
The project is to make a website that is connected to external databases via their APIs. The website will contain inputs such as buttons and dropdown menus to request data from the various databases. Databases that we will use include one that contains the entire list of year, make, model of cars. Another database is Google Maps. We will use React for the front-end.

We will start by compartmentalising each module by their function. The Map Controller module will fetch the necessary data about the location of the user and the location, price, brand etc of each gas station. Our Filter modules accepts a collection of Gas Stations and applies each filter algorithm, pruning the list of relevant gas station;. The the pruned list is past to the display module, which populates the user's screen with images that convey the information they requested.

Module Specifications

Each module is confined to its own JavaScript file. All React components and their functionality will be grouped together. This way we can import only the functionality and modules required rather than having all code in a single script. We want to maintain a loose coupling.

Modules



(This may not always be up to date. Check the current diagram here.)

Recent Changes

Sprint 1 (Weeks 3-4)

- Set up repository and Heroku deployment.
- Add ability to request user location.
- Add list of gas stations to web page.
    - Note: Gas stations are dummy data generated by us.
- Add a textbox to the webpage with the label “Max Data”.

    - Note: This input box was not connected to anything and merely allowed a user to enter numbers.
- Add a selection box which displayed Gas Station names.
    - Note: Selecting a Gas Station name did not filter any of the data.
- Add dropdowns displaying car years, makes, models, and options (trims). Each has a submit button that refreshed the page when clicked.
    - Note: Car year, make, models, and options were dummy data generated by us.
- Add placeholder text to the web page for MPG.

Sprint 2 (Weeks 5-7)

- Change layout of website
    - Add new pages (Home, Find, Car Options)
    - Move gas station list and filters and MPG text to Find page
    - Move dropdowns to Car Options page
- Add API calls to a gas station database.
- Change gas station list to use fetched data.
- Change filter list to a dropdown.
- Change displayed filters to match a few names from the database.
- Add functionality to filter dropdowns: Clicking the filters now changes which gas stations are displayed.
- Add module to fetch car data including year, make, model, options and MPG from a Government database.
- Fill dropdowns with car data from Government database.
- Add a Google Map to Find page. Contains pins and popups with data for each gas station.
- Add car mpg to MPG text
- Add slider to enter current gas fill of the user's tank to Find page
    - Note: the slider is not connected to anything and is merely aesthetic.

Sprint 3 (Weeks 8-9)

- Implement cookies to save user's car data between pages and uses.
- Add functionality for gas fill slider: Now saves to cookies.
- Add gas tank size input field to Find page and save to cookies after entry.
- Polish website making buttons more intuitive to use.
- Add Get Direction button that sends the user to Google Maps.
- Finish implementation of all filters: All gas station names, max distance, gas grade.
- Add multiple sorting functions for the gas station list: price, distance, and Smart Calculation™.
- Add display for Unleaded Pro and Premium gas prices for each gas station in list.
- Fix popups to change after filtering gas station data.
- Add ability for the Google Map to open the top gas station as soon as it loads.

Project Logistics

Repository

Branching

When creating a new branch, it's useful to know who that branch belongs to. A new branch should begin with your GitHub username followed by a slash and a descriptive branch title. For example, if I were to create a branch to work on user input, my branch would be: `MichaelJBradley/user-input`. This way we can keep all of our branches organized.

Commits

There is a fantastic article about writing good commit messages that was linked in the first lab of our CS 100 class. I'm sure that everyone read it start to finish, but in case anyone needs a refresher, here it is.

TL;DR

- Use the imperative form of the action word: Add, Change, Implement
- Use the first line as the subject, skip a line then add more details to the body if necessary
- Keep the subject descriptive yet concise, no more than 50 characters
- Limit body text to 72 characters per line, so it is easy to read on GitHub
- Doing a `git commit` without the `-m` will open up an editor and provide you with details of your staged changes.
- Here is an example:

```
Configure Jest to work with Babel

Babel provides Jest with the ability to recognize the ES2016 proposal
syntax.
```

Pull Requests

I think it would be beneficial to us, and keep our builds from catastrophic failure, to use the pull request feature on GitHub. This way we can review each other's code before merging. It has the added benefit of allowing everyone to know when a change has been made, rather than just merging and pushing directly to master.

I realize as the project progresses, we will be less inclined to review each other's code. I'm sure there will come a time when we just need to get features in and this project done. Still, it would be best to request code reviews from other members when creating a new PR. This way other

team members can become familiar with the code they did not write themselves. Plus, a second set of eyes might find some bugs that the author missed.

A few notes on writing pull requests:

- Keep the title descriptive yet concise, similar to a commit subject
- Capitalize each important word of the title. For example: Repository and Project Setup
- List the major changes in the comment section For example:
    - Update README
    - Install Jest
- A good rule of thumb for a PR is about 400 lines of code

After the PR is written, choose one or more people to review your code and assign yourself to the PR.

Testing

Jest

The project will use Jest as its testing framework. Jest is extremely easy to learn, but has the capability to create powerful, robust tests. The Getting Started page provides a brief example of how simple writing a test can be. From the page:

Code

```
function sum(a, b) {
  return a + b;
}
module.exports = sum;
```

Test

```
const sum = require('./sum');
test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});
```

Then the only thing left to do is type `npm test` in the terminal to run it.

Each test suite should relate to a single module and be named accordingly. For example, if we wanted to write a test suite for the `MovieDetails.js` module, from our React Starter Project, we would name it `MovieDetails.test.js`. Jest is magic, and it will know to which files to test.

Additionally we can group our tests with a `describe` function call. It is formatted the same way as the `test` call. For example:

```
const sum = require('./sum');
describe('adds', () => {
    test('1 + 2 to equal 3', () => {
        expect(sum(1, 2)).toBe(3);
    });
    test('1 + -2 to equal -1', () => {
        expect(sum(1, -2)).toBe(-1);
```

```
    });
});
```

Keep tests as atomic as possible. That is only test one functionality, value, etc. per test. For example, write one test for a valid value, one test for an invalid value, etc. Jest will report a which point the test failed, but it makes it much easier to determine what the issue is when we can pinpoint the value that caused it to fail.

Jest can get very specific with its `expect` calls. Here is a useful page for writing tests. There are a few idiosyncrasies that you have to get used to with Jest (Google “Jest toBe vs toEqual”), but nothing too difficult to overcome.

I hate writing tests. I hate it so much. But tests *are* useful and they *do* catch bugs that we would miss otherwise. They’re also useful to make sure that we don’t break old code when we add new features or modify old ones.

Travis CI

Travis is a continuous integration service that we attach to our repository to automatically run the tests and automate scripts. Every time we push code to the repo, travis will create a build and run the tests. If everything passes, then we receive a little green checkmark, and we’re good to go.

Most of the time we’ll know a build is passing because we’ve run the tests locally (with `npm test`). Every once in a while there will be an issue where the tests pass locally, but they won’t when Travis runs them. This is usually due to a mismatch in Node versions or an automated script we wrote that’s failing. If a build fails, you can click on the little red X mark and follow the link to the Travis website to look at the logs.

I know it’s tedious. I hate reading logs too, but 99% of the time the problem is right there in the logs. You just have to take the time to look.

Deployment

I’m not yet sure what whether we’re going to be provided a server on which to run our website or if we will do everything locally. If no server is given to us, we can use Heroku to deploy our web app. It can even use our repo as the source so it will always be up to date with the master branch.

Heroku is a service that provides hosting for web applications and other cloud services. Now I don’t know much about web development, but I do know Heroku makes things much easier and there’s plenty of documentation about setting up a Node application. Plus it’s free up to a certain amount of traffic, which I can nearly guarantee we will never hit.

Code Documentation

Each function should have documentation describing what it does. This is especially necessary since we have five people working on the project, so not everyone will know what each function does just by looking at the header. The npm package JSDoc provides a simple way to generate HTML pages containing the documentation.

While we might not actually use this functionality, JSDoc is the standard format for writing inline documentation. It's similar to Javadocs or C#'s inline documentation. If you are using an IDE, then typing `/**` followed by return/enter should auto generate the documentation for that function. If the function is already written, then the auto generated docs should include the parameters and return types as well.

This style guide provides a good overview of the things that go into JSDoc comments. While I suggest giving the whole thing a read (it's not that long), the most useful bits, for us, start at the Function section.

Retrospective

Sprint 1:

Had discussion about communication. Not all group members were forthcoming in communicating their schedules and their progress on user stories. We resolved to fix this issue in the coming sprint. Decided that Wednesday and Saturday of each week will be used to meet up as a whole group and work on the application.

Had issues completing every task assigned this sprint. This was due to underestimating the workload we had assigned this sprint. Also, some tasks that were labeled complete did not actually meet all of their acceptance criteria. We were advised to assign less tasks and to pay attention to what we actually wrote down on our scrum sheet.

We should have planned the project better overall during this phase. We assumed some APIs would be easy to access, but that was not the case which set us back slightly. In addition, some user stories such as allowing users to update gas prices was too big of a project to tackle in addition to everything else, so we ended up dropping that user story.

Sprint 2:

Even though we created the dropdown menu in the first sprint, we weren't satisfied with the look. Instead, we decided to use Reactstrap which is Bootstrap and React combined. This made our UI look exponentially better. We as a group don't have any artistic design skills so we

use what was available. If we were to redo this again, we would use Reactstrap from the very beginning.

Toward the end of the sprint, we discussed semantics of code placement within project. We had been filtering gas station data in GasStationData.js instead of FilterGasStationData.js, and really should move filtering to the latter file so that our file names actually make sense. Time constraints prevent us from refactoring this right now, so we will do this at the beginning of Sprint 3.

As we continue to add features and functionality, it's really apparent how scalable it's been to force any contributions to go through member-approved pull requests. With team members working on different features all at the same time, having team members push directly to master would have been chaotic and completely unmaintainable. Also, automated testing with Travis saved us a few times when it caught build errors we'd have otherwise pushed to master.

Sprint 3:

We had problems with most group members showing up on time to meetings. However, we set enough time for our meetings (around 6-8 hours per weekend) for all of us to meet at the same time and work together.

We didn't finish all user stories that we set to do at the beginning of the project. We should have accounted better for how long each user story would take. In general, at the beginning, user stories took about twice as long to complete due to having to learn JavaScript and React too. We got better at completing user stories on time as time progressed.

If we were to retake the class again, we would have chosen easier user stories where we can accomplish in three sprints. Looking back, we have three user stories we weren't able to do. As a group, we want to be the best we can and not being able to do three user stories definitely made us wish we can change those three user stories.