COLORADOSCHOOLOFMINES
EARTH • ENERGY • ENVIRONMENT

CSCI 370 Final Report

Trash Titans

Sean Keeney
Connor McGovern
Jarrett Daffern
Amber Gehan

Revised June 16, 2023

CSCI 370 Summer 2023

Prof. Kathleen Kelly

Table 1: Revision history

Revision	Date	Comments
New	05/21/2023	Sections I-V, initial references, and initial key terms added
Rev – 2	05/28/2023	Section VI, updates to Sections I-V
Rev – 3	06/4/2023	Sections VII-VIII, Updates to Sections I-VI
Rev – 4	06/11/2023	Updates to Sections I-VIII
Rev – 5	06/14/2023	Finished the remaining sections and updated all others

Table of Contents

I. Introduction

We worked with Clean Planet Project for our project (https://www.cleanplanetproject.org). The goal was to develop a web app which will encourage and promote litter pickup and recycling by enhancing features of their current web app. Clean Planet Project is a non-profit organization dedicated to the reduction of litter globally. The web app that we made consists of a map and a leaderboard. The leaderboard shows user profiles connected to the web app as well as how much litter each user picked up, all ranked in order by those who have picked up the most litter. In addition, the leaderboard also has a variety of filters which will allow users to see those who picked up the most litter by region, time, and email domain which will allow companies and other organizations to hold competitions. The map shows markers that represent the areas in which the litter pickups happened in the world, clustered by region with more in-depth clusters as the user zooms in. Once connected to real-time data, as users pick up litter and provide the web app with the data, a new dot or marking will appear on the map for their pick-up zone, and their leaderboard location will update to reflect the actual amount of litter they have picked up over time, and if they have earned it, they will move up the global rankings.

II. Functional Requirements

The client required that our team develop a web application that displays a global leaderboard by trash count and time period. An interactive map of the cleanups is also to be made on the web app. The litter leaderboard should be able to filter the cleanups by a circular region defined by latitude, longitude and radius. Furthermore, the web app should be able to filter cleanups on the leaderboard by email domain, such as “gmail.com”, and time period. In addition, the map should also implement geo-locational queries to filter on. This web app UI should be compatible with desktop and mobile web browsers. Viewing or interacting with the map should not be laggy on an average laptop, defined as 4 GB ram, i5 processor, and should receive filtered data in less than 1 second for a typical 50 Mbps WiFi connection.

III. Non-Functional Requirements

The non-functional requirements outlined by the client for this web application project include using specific [1] APIs (Application Programming Interface), using plugins to minimize cost and tests that they wish to see in the implementation. More specifically, our team will be required to use MongoDB for storing cleanup data [2]. This is consistent with data storage used in other aspects of Clean Planet Project as well as what was given to Clean Planet Project 2. Additionally, we will be using Flutter for the front-end design of the web application [3]. This requirement will ensure that the Clean Plant Project team or future field session teams can work on the web application to add new functionalities. We were also directed to use Node.js for the backend [4]. Specifically, NestJS + TypeScript is preferred by the client [5][6]. To minimize costs, we were advised to use free plugins for the map and to look for other opportunities to reduce cost. Clean Planet Project also requires that we build unit tests for at least the server side logic functions, though we have tested at every stage of the front and back end of the project. The bulk of the tests ensured quality in the back end of the application, checking query results and making sure various functions work correctly. Tests for the front end however are manual, checking display functionality, ensuring the web app is responsive to different screen sizes and that features such as filters and lists work as expected. Examples of front end testing includes checking the map clustering points or even what is returned to a user when they use a filter that returns no results.

IV. Risks

The largest risk came from technical skill and implementation. Some of the team members have worked in app development and with some of the technology we planned to use, but others did not have much or any experience with app development. The possible negative result of this risk was an unfinished or poor quality final product due to the inexperience of the group with full stack development. Members of the team set aside time in order to research and better understand the technologies we would be using, as well as gain a further understanding of how tasks will be completed to mitigate the risk mentioned above. There also was some risk in our choices for technology used. Again, the risk pertains to the quality of the final product, as well as the client’s ability to further enhance or integrate our web app into their systems and existing website. Our client gave us a general overview of recommended technologies, but ultimately the majority of the decisions were left to us. We did largely go with the client’s recommendations to mitigate the risk of poor integration into existing systems. Additionally, we are handling user data, such as emails and location data. There is some risk to handling such data because malicious users could try to collect this information from faults in our software. To mitigate this risk, we did not display names with locations. Only usernames are displayed on the leaderboard and the map markers are only populated with the location data, not user data. Names or email addresses

are not accessible from the front end. Emails are only used in backend logic to filter the list of usernames, rankings, and trash counts on the leaderboard. A final risk, which is in every project, was the time requirement. There was a lot to accomplish, and of course there were stretch goals, but at the end of the day we had a set limit of time for this project. We had to ensure we completed what we could given the time, but still create a product that the client can be excited about. This also included ensuring the functionality is on point with the client's wishes and the initial requirements detailed in sections II-III.

V. Definition of Done

We are done working on the project when we have the main outlined features complete: a global leaderboard, filtered leaderboards, and a map displaying cleanup data, all while hitting the requirements of speed, cost, and format. In our final product, we hoped to have the functionality of the web app complete with a user interface most likely also completed. In terms of delivery, we have already established a connected GitLab with the client which will hold multiple repositories for parts of the project [7]. Using this GitLab, as the project is over, the client automatically has all of our work [7]. The project was considered 'done' when the client had access to a *running, fully functional web application that follows the requirements defined above in sections II and III and exhibits thoughtful UI design.* This concrete definition of done helped us plan our sprints and be able to achieve measurable progress towards a final product.

VI. System Architecture

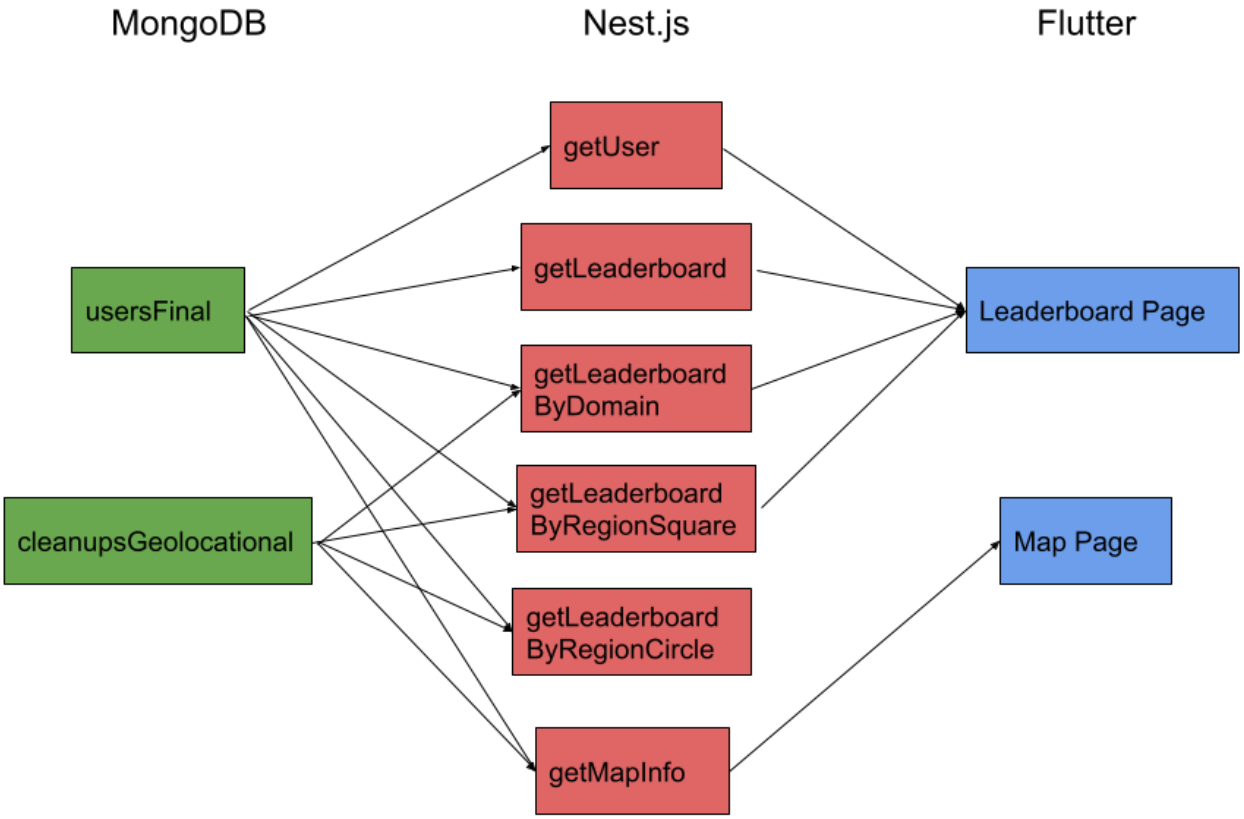

Figure 1: System Architecture and Internal Connections

Our system architecture is as shown in Figure 1 above. The web app is run with Flutter which is used to display data and make the overall design for the app. The Flutter code has a connection to our backend server which is a Nest.JS app, which builds upon Node.js features. Our Nest app provides six different GET routes, and a seventh getHello route for testing purposes. This server serves as a privacy connection so that there is not a direct connection from the web app to the database. Our backend server connects a database hosted on MongoDB Cloud with two different collections. Through this linear connection, we are able to keep the users information secure while also having an efficient full-stack website.

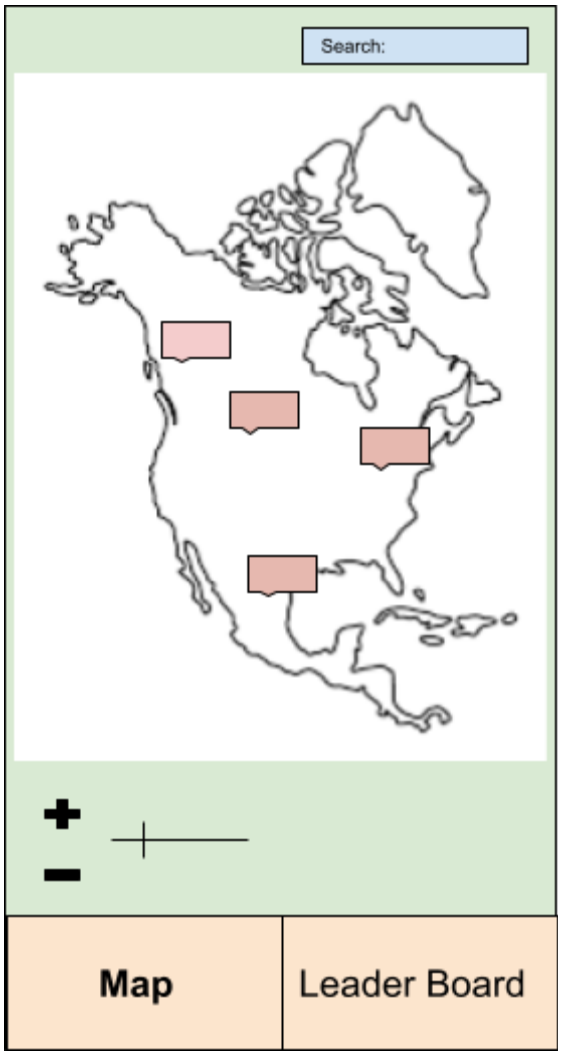

Figure 2: Sample Design for Map Section of Web App

Beyond the architecture itself, our system design was initially planned to display in a similar format to Figures 2 and 3. Figure 2 displays the map functionality of the web app. The bottom of the display is a set of buttons to switch sections, but above that is where the map itself is done. The map was a world map with a series of clustered points, spreading into further points as users zoom in using the bottom slider/zoom buttons. At the top, there was also a small search bar for the user to get information about a more specific area that the user would be able to enter and the web app will reconfigure to that unique location. The search bar would also be able to take in specific users and display the location data for that searched user's pickups.

In Figure 3 below, the functionality associated with the leaderboard section of the web app is shown. Similar to the map section, there is still a set of bottom buttons to navigate between the map and leaderboard. The leaderboard itself is a scrolling platform which shows the web app users organized by their rank (determined by trash count). Within the ranking, other users are only shown the numerical rank itself, the user with that rank's username, and the actual number of pieces of trash that user picked up. Lastly, in the top corner, we planned on creating an interface to allow searching and filtering through the data. The goal in mind for these filters was for the user to see top users of a specific area, or to only see users with a specific email address tag which could be used for school or company competitions. The user would also be able to filter by date ranges or even a specific user to see that person's rank and pickup number.

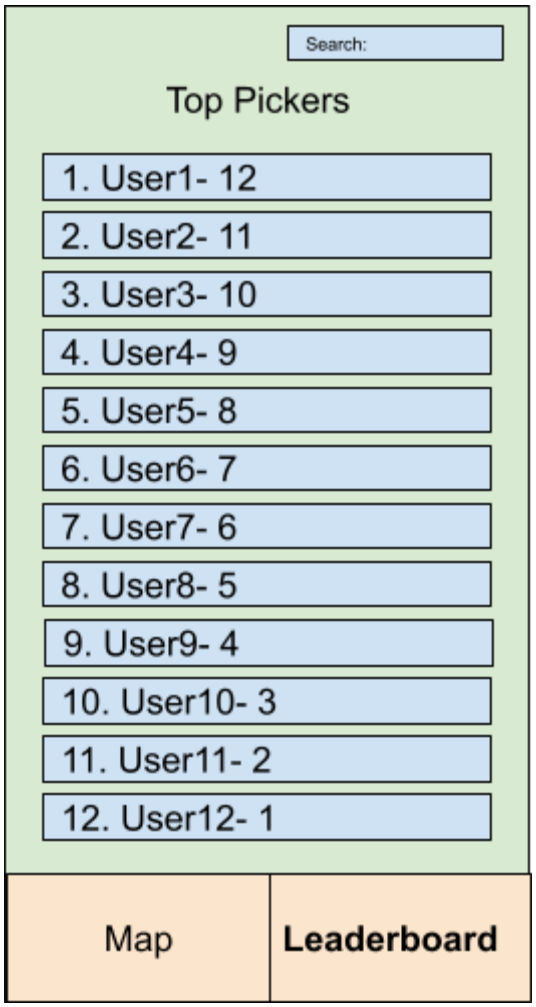

Figure 3: Sample Design for Leaderboard Section of Web App

VII. Software Test and Quality

In order to have the highest quality that we can for our code, there is a series of tests that were implemented. The first area of testing is that of our user queries. When user data are returned by our code, the backend route ranks them in order of most pieces of trash picked up and also has the username attached. It is essential that this returns correctly for our leaderboard output. A primary form of testing was going through the returned data from the query, ensuring between lines of data that the previously seen user has a higher trash pick up amount than the current one. A potential edge case(s) for these results is with the filter implementation. The filter for the leaderboard limits the number of responses returned based on various criteria which our code needs to handle. We have a test for when filters are put in place that has no result. This comes in the form of searching along an email domain in which there are no entries or checking for the existence of a user that does not exist. For both forms of these, the only acceptable result is a correct order that has all of the data that meets the criteria done through the various filters.

In a similar vein, we also must test the pages for the leaderboard making sure that we return the correct number of users and the correct users themselves. Also, we test along the lines of tie breakers for users that have the same amount of trash picked up, the tiebreaker being the username.

Another particular area of testing is with the cleanup queries. These queries return information about the location of various cleanups which will then be output onto the map section of the web app. A necessary test for these functions is to check the geo-locational coordinates of the results to ensure that, in the form given, we can output a correct marker to our map. Accepted results for these tests would only allow results through to the function that creates the map that are actual points that can be mapped onto our system, but also have the correct format that further functions can read and use. An example of an edge case for these tests would be rounding coordinates and how many digits the Flutter built in map plugin can take to create a marker point. We worked to make sure that if a point is near the threshold of acceptance, it doesn't round that point to where it is no longer valid. This ended up not being a particular issue with our implementation.

For each of these various tests, we created a miniature form of our data, taking at random about 20 entries. Using this new set of data we were able to specifically check our software, seeing various results that we would also be able to check by hand to ensure our results are as expected. These tests were also able to be run directly on our Nest.JS server through the testing package in order to be most convenient. The new set of data was able to work for all of the tests we feel need to be run including rankings, filtering, geo-location data and more.

If time permitted, we planned to also integrate our new features with Clean Planet Project's current app. Doing so would require integration testing and to ensure things run smoothly across platforms. Unfortunately, this stretch goal of ours was not achieved. Hence, integration will fall to Clean Planet Project as they might also have updates to conduct in terms of databases used and coordinating it with their full data pipeline from their picture recognition software which counts the number of pieces of trash.

The results of our tests yielded favorable results. For our unit tests on the server side, we were able to get all tests passing, which is a sign to us that things are working. These included returning values for specific users, looking for a missing user, checking user order, and even tiebreakers in rankings. Each of these tests pass, indicating our queries and methods to move data were successful.

Tests on the front end of the application also went well with clusters being displayed successfully, accomplishing a working zoom in/out method, and proper indications for when a filter is used without results. With these various cases working and properly implemented within the web application, we have accomplished a fair set of our goals for the project as given to us by Clean Planet Project.

VIII. Project Ethical Considerations

This project includes several ethical considerations. Primarily, these come in the form of user privacy and retention of data. The largest aspect of this project is the application displaying data from a database which houses information about user's real names, their location-based data, and the user's personal email address. While this information is freely given to the app by the users, and the majority of it will not be shown on the application, there is always a concern about a malicious individual or organization gaining access to the database and acquiring the data. In order to preserve and prevent this from happening, the web application has been designed with a communication pipeline as opposed to direct connections for higher security. This ties in directly to ACM principle 2.9 [8] which asks that projects developed are secure for users.

In addition, the data we used had been partially scrubbed, removing email usernames and only holding limited information about the litter pickup, which is only the global latitude and longitude coordinates. Doing such stands to make sure we can create the application in such a way that does not directly access all of the user's information. Also, once the client has the application in their hands, they would have the capability to use a more proper database of theirs which would not have any of the information we are worried about even being accessible from certain tables. This is in

collaboration with ACM principle 3.6 [8] which states that care must be taken when modifying systems. We have created our systems with this in mind, making the transition from the given sample data to the actual Clean Planet Project database easy and efficient.

Another concern which is more out of our hands and is up to Clean Planet Project is the retention of the data used. We are given sample data to use to display the map given cleanups with user's geo-locational data. This part of the application will simply use whatever data is presented from the database, but the question to this lies in Clean Planet Project's desire of how long to keep certain cleanups. At a certain point, it would be helpful for time considerations and safety considerations to remove cleanups from the database (perhaps something like after 1 year they are deleted). Deleting cleanups would be for the best interest of Clean Planet Project due to long-term collection of geo-location data which could lead to a malicious user seeing a pattern of another user or being able to pinpoint where a user lives or works, which could be quite dangerous. This issue is directly related to ACM principle 1.6 [8] which says that the privacy of users should always be respected. It is our job as developers to prevent a potentially malicious occurrence from happening to an innocent user.

Furthermore, another aspect more on Clean Planet Project's side of things is the accessibility of the application. In this regard, we thought about most specifically foreign markets. We are not sure whether extra steps need to be taken for an application to be distributed in other countries, but that would need to be something Clean Planet Project would need to be involved with. Also, the data and output we are doing for the application comes from a database we were given where all of the entries are in English. For this application to be useful in other nations, work would need to be done into translating the English database whether from the database side by making a new table, or creating a translating program which could be put in the software to convert the English strings obtained from the current database. Accessibility in development, especially relating to this ethical consideration, is reflected by ACM principle 1.4 [8] which says that developers should create products in such a way that discrimination is not included. For people all around the world to use the application, more work definitely needs to be done for language implementation.

IX. Results

When everything was said and done with the project, we delivered a web app which met the majority of the requirements our client gave us. Primarily, we were able to have the web app have both a leaderboard section as well as a map section. In the leaderboard section, the web app successfully displays users, their rankings, and the number of pieces of trash they picked up in total. We also feature a few different filters for the leaderboard. Of the filters we were asked to create, we were able to output results for email domain, time based, and getting an individual user. Moving on to the map section, we were able to make the map display points of various user cleanups with each one marked by a pin. The map can also zoom in and out of these points. Each point is also clickable to display a small amount of information about the cleanup. Furthermore, the map points feature a cluster-based system such that when zooming out, points will show as one point with a number indicator of the number of points around there. When zooming in, these clusters disperse and show the actual points for the cleanups, each of which includes the previously mentioned features. The front end also uses a layout that is similar to the current Clean Planet Project web application as well as uses a similar color palette. Beyond the features, the front end of the web application also does not have an abundant amount of lag time. The data is shown quickly and thoroughly. The longest wait time is upon entering the map which takes roughly a second or two to initially display the points.

Our team was also able to accomplish a fair bit on the back-end. The back-end of our application features a connection to the database, querying the data and formatting it in such a way that it can then be sent to the front end for its eventual display. Though we were not able to output all of the filters on the front end of the application, the back-end has all of the routes to query the requested data complete, allowing for Clean Planet Project to simply write a

small amount of code to take that data and output it. The back-end also serves as a server point for the application, allowing for an appropriate amount of security considering the amount of user data being handled. On the back-end we also have written out a series of tests which attempt the various functions built out for the queries to ensure that they are not only correct, but also that the functions were written in such a way that there is handling for error situations or misused filters (looking up a user that does not exist, etc.).

As for the database on MongoDB, we were also able to do a fair bit of work to help not only Clean Planet Project, but also others who will work on our code. For the database, we did a complete redo, making a collection of new tables to house the information in a way that was more usable for our project. We restructured the user information table so that each user was separated from each other which allowed for efficient querying. This also probably helped with security by not directly having each user clumped together with one another. A similar restructure was done for the clean-ups table as well but in this case we had to adjust the data into the individual clean-ups with users marked inside, as opposed to the original set up which had each user marked with their total cleanups. We also chose to change the layout in this table for how latitude and longitude were used because it made the querying in the back-end easier to use the set of the numbers to create a point as well as it allowed us to do geo-locational queries for some of the map routes. An example would be to pick a certain point on the map and request the other points within a certain number of miles.

For the project, we also met the requirements given to us in terms of documentation. We ensured that the code we wrote featured comments throughout to help future developers as well as we put together a document of the dependencies we needed to use for the application to run. After the database restructure, we also wrote out a guide to how we went about it, as well as a breakdown for the new tables describing what section was what as well as why certain decisions were made.

Screenshots of the final product are included below, showcasing a global leaderboard, buttons and text fields to filter the leaderboard based on email domain and time, a search feature to grab the details of an individual user, as well as a map that displays clusters of users that dissipates into individual cleanups as users zoom into specific regions.

Figure 4: Global Leaderboard Final Design

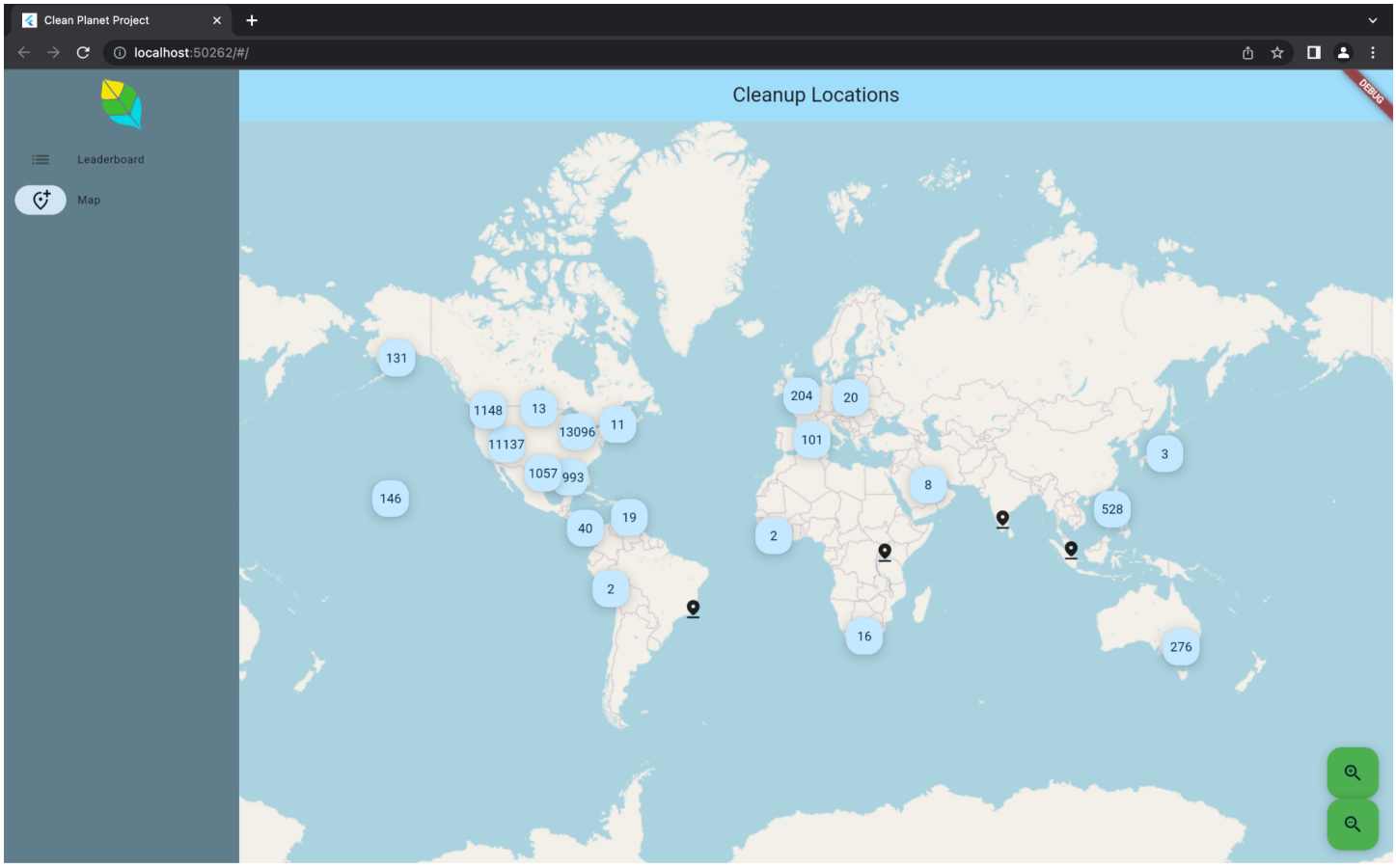

Figure 5: Global Cleanup Map Final Design

X. Future Work

Though we were able to accomplish a wide variety of the requirements we were given by Clean Planet Project, there are also a few things we were not able to get around to which will need to be work done by the client or a new team in the future.

First and foremost, some basic things that need to be done in the future are just finishing adding implementation for all of the filters for the front end. Furthermore, adding work for the geo-locational queries for both the leaderboard and map need to be worked on.

One element that could be improved is the markers for the map part of the application. We used a built-in marker point, but Clean Planet Project could have other wishes, like perhaps their logo. Each point also displays basic

information about the location right now, but the points could be adjusted to tell more about the clean-up itself like the user who did it or how much trash was picked up at that clean-up.

Another change a future team could make could be which map plugin the app uses. We used a free Flutter map plugin to keep costs down, but a future team could decide to use the paid Google Maps plugin. Such a change would require a little bit of extra time and most likely money to implement, but the Google Maps plugin has some better features that come included as well as the syntax is a little more simple. One example of a feature is the existence of zoom in/out buttons which do not need to be hardcoded into the application like we had to.

Currently, our web app stands alone on its own website that is pointed to by the main Clean Planet Project website. One of the goals for after the end of this project is that our web app will eventually be integrated into the main Clean Planet Project website. This will require a fair amount of work on Clean Planet Projects end to ensure our application is fully compatible with their own as well as some of the code will need to be adjusted and changed with emphasis towards the backend which gains information from a sample database and not Clean Planet Project's main database. An admin page specific to Clean Planet Project would also need to be created.

One possible addition that would require changes to the Litter CleanUp mobile app and the web app is the ability for users to compete in custom groups not defined by email domain. Users could create and join groups on the app and then their clean-ups would contribute both to their own total and the group's total. This would require new space on the database to store all the groups and which users are in those groups, would require UI changes to the mobile app to add buttons for creating, searching for, and joining groups, and would require a new route and UI changes to the web app to allow for showing rankings by group and rankings by group within a certain area. We cannot estimate how difficult this would be overall since we did not do any work with the pre-existing mobile app.

Another possible addition would be for a single user to filter for all map pins that represent their own trash clean-ups. This is challenging from a user security perspective. It would be too personally identifying for anyone to be able to type in someone's username and see the locations of all that user's clean-ups. This would require either some kind of login on the website, so only people who can log into the account are allowed to see all the clean-ups associated with that account, or by implementing the map on the main Litter CleanUp mobile app, and then, again, only allowing users to filter for clean-ups associated with the account they are currently logged into. Both of these solutions would be very labor-intensive for very little gain, so unless someone later down the line decides they want to integrate the map into the Litter CleanUp mobile app for another reason, adding this feature is very unlikely to be worth the effort.

XI. Lessons Learned

As a team, we learned quite a few lessons throughout the entirety of this project. One of the largest lessons that we learned came in the form of the documentation throughout the project. We learned that it is important to document often and thoroughly. The team was often split up, working on different things, so when someone would switch to something else that was partially done, it could sometimes be confusing to work on or the person would need to ask the group lots of questions.

Another lesson learned was the importance and complexity of a server in a web application. None of us had really set up a server before, which made for an interesting experience. We learned first hand just how important a server is due to its security implications but also its ability to send/obtain information within the pipeline. With this all came also a struggle and further understanding of what it takes to create a working server. We worked extensively to set up the server, taking almost a week of work to solidify a connection.

A simple lesson that perhaps could have been learned much sooner was the use of our backlog. For the start of the project, we struggled to use it, not often consulting it for work that needed to be done. However, when we set our minds to using it more, the flow of work to completion was not only smoother, but also quicker as all members had a better understanding of what to work on.

XII. Acknowledgements

We would like to thank a few individuals for their help and work with the project, where without them, we would not have been able to accomplish all that we did.

First and foremost we would like to thank our client, Josh Rands. Josh was always there when we needed help or advice. He always made sure that we knew what we were doing and strived to make sure that we did not feel pressure by the requirements or the class but rather took Field Session as a learning experience for project development.

We would also like to thank our teacher advisor Kathleen Kelly. Professor Kelly made sure we were always on track with the project and also made sure that we had an understanding of what it was that we needed to do. She was also incredibly helpful when it came to in class assignments, looking through our presentations before we gave them and also reading this document as it went through the various draft stages.

XIII. Team Profile

Sean Keeney-

Senior

Computer Science, Data Science Track

Hometown: Littleton, CO

Work Experience: Led a project to develop a python library to support the IEEE-2791-2020 standard colloquially known as BioCompute.

Hobbies: Soccer, Skiing, Cooking

Connor McGovern-

Senior

Computer Science

Hometown: Thornton, CO

Work Experience: Web Development, Outreach Manager

Hobbies: Reading, Cooking, Exercising

Jarrett Daffern-

Graduated

Computer Science

Hometown: Baltimore, MD

Work Experience: Professional Outdoor Guide

Hobbies: Climbing, Hiking, Reading

Amber Gehan-

Senior

Computer Science

Hometown: Dallas, TX

Work Experience: Wrote problems for Computer Organization and graded for Computer Networks

Hobbies: Crochet, Cooking

References

[1] “What is an API? (Application Programming Interface),” *MuleSoft*. Available: https://www.mulesoft.com/resources/api/what-is-an-api. [Accessed: May 21, 2023]

[2] “MongoDB: The Developer Data Platform,” *MongoDB*. Available: https://www.mongodb.com. [Accessed: May 21, 2023]

[3] “Development.” Available: //flutter.dev/development/. [Accessed: May 21, 2023]

[4] “Node.js,” *Node.js*. Available: https://nodejs.org/en. [Accessed: May 21, 2023]

[5] “NestJS - A progressive Node.js framework,” *NestJS - A progressive Node.js framework*. Available: https://nestjs.com. [Accessed: May 21, 2023]

[6] “JavaScript With Syntax For Types.” Available: https://www.typescriptlang.org/. [Accessed: May 21, 2023]

[7] “Try GitLab Ultimate for free.” Available: https://about.gitlab.com/free-trial/devsecops. [Accessed: May 21, 2023]

[8] “ACM Code of Ethics and Professional Conduct,” Association of Computing Machinery, Available: https://www.acm.org/code-of-ethics. [Accessed Jun. 12, 2023]

Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

Term	Definition
Flutter	*Flutter is an open-source UI software development kit created by Google. It is used to develop cross-platform applications for Android, iOS, Linux, macOS, Windows, Google Fuchsia, and the web from a single codebase*
NodeJS	*Node.js is a cross-platform, open-source server environment that can run on Windows, Linux, Unix, macOS, and more. Node.js is a back-end JavaScript runtime environment*
NestJS	*NestJS is a framework for building efficient, scalable Node.js web applications.*
MongoDB	*MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas.*
TypeScript	*TypeScript is a free and open-source high-level programming language developed by Microsoft that adds static typing with optional type annotations to JavaScript.*
APIs	*Acronym for application programming interface.*
GitLab	*GitLab is an open-core company that operates GitLab, a DevOps software package which can develop, secure, and operate software.*