COLORADO SCHOOL OF MINES
EARTH • ENERGY • ENVIRONMENT

CSCI 370 Final Report

No Planet B

Christine Hwang
Anastasiya Velychko
Minnie Her
Briar Martin

CSCI 370 Summer 2023

Prof. Paone

Table 1: Revision history

Revision	Date	Comments
Rev – 1	5/17/2023	Nothing substantial yet. The first week should be spent learning how to use Flutter for the project. First revision consists of creating the Requirements document.
Rev – 2	5/19/2023	Asked client detailed questions on the requirements of the product and modified the Requirements section accordingly.
Rev – 3	5/22/2023	Cleaned up the Requirements sections and added a User Interface diagram to System Architecture.
Rev – 4	5/27/2023	Added Backend diagram to System Architecture.
Rev – 5	5/30/2023	Added descriptions of diagrams in System Architecture and added. Added Software Test and Quality section and Project Ethical Considerations section.
Rev – 6	6/5/2023	Stubbed out the Results section.
Rev - 7	6/10/2023	Took into account advisor feedback to flushed out all sections. Added High Level diagram to System Architecture.
Rev - 8	6/14/2023	Cleaned up previous revisions. Modified Team Profile. Added previous screens to Introduction and new screens to the Results section.
Rev - 9	6/16/2023	Added Acknowledgments section and modified Appendix B. Putting finishing touches on report.

Table of Contents

I. Introduction

The Clean Planet Project is a nonprofit that was started by our client Josh Rands and his partner during the start of the pandemic. They had started to take walks around their apartment and began to pick up litter. They wanted something to track their progress, so they made an app, which later became the basis of the main focus of The Clean Planet Project. The main purpose of the app now is for volunteers to track how much litter they are picking up and for volunteer managers to keep track of the progress of their volunteers, with the mission to make this kind of volunteering more accessible and fun for everyone. Volunteers earn hours by uploading photos with geocache and timestamp information of every piece of litter picked up during a trip to the app/website. An hour is counted as a piece of trash being picked up at least every three seconds. Currently, there is an app (developed in Flutter and Josh would prefer we use Flutter, as well) that is pretty bare bones, and most (if not all) correspondence is done via email. A prior field session project trained a neural network named Oscar to determine how likely a picture of litter is to be real. There are two different portals, one for managers and one for volunteers. What we are tasked with creating should only be accessed by managers, as this would be the manager portal. Our project is also meant to be more like a very detailed schematic for Josh and his team to follow when updating their app. We are not making anything deployable.

Besides all correspondence being done over email, the current implementation displays numbers for things like trip times with many decimals in a table format. These tables are, overall, hard to read and aren't a very efficient use of space. Images of litter are displayed after Oscar has looked over them, but all data about that image can only be seen when the image is clicked on. This is not the most helpful, especially for trying to see the area that a volunteer cleaned up, because the location data for every photo is simply a line of text. This original UI can be seen in *Figures 1* through *3* below. Finally, Josh has been working in Amsterdam, and for the first couple weeks of our field session, he will be either in Europe or traveling. He has assured us that this will not be a problem and that we should contact him via the Google Chat he has set up for the quickest response, but during this time, video calls might be difficult. Many of our group also work at least one job or have a long commute to campus, so time management in that way could also be a concern.

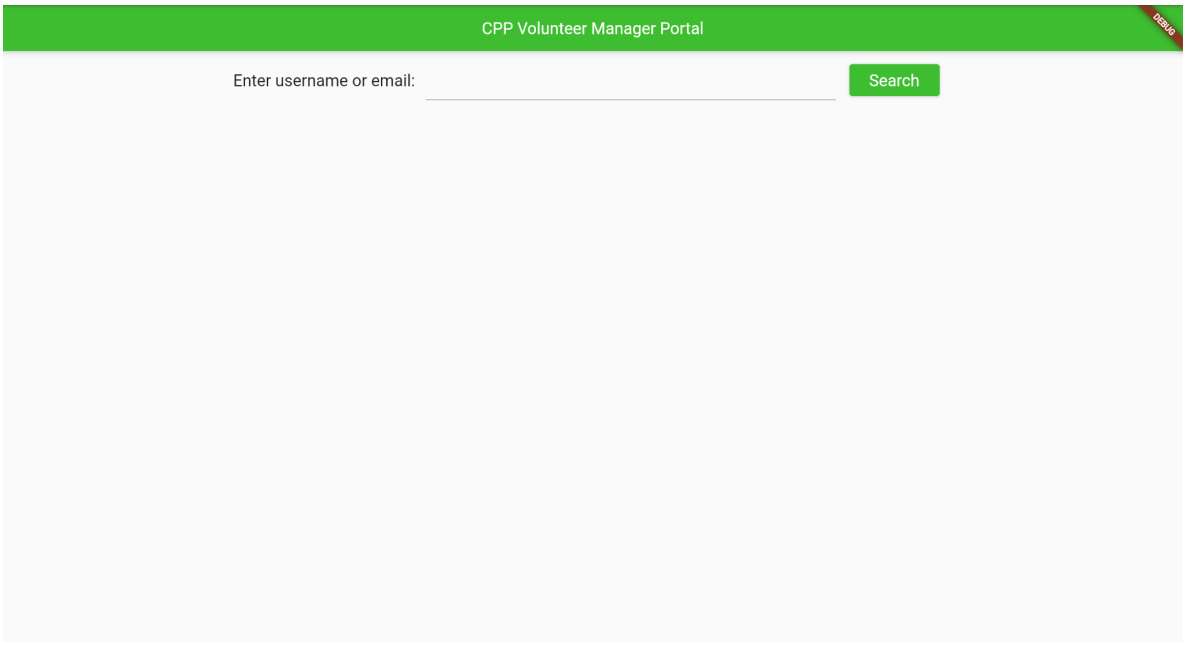


Figure 1: Current Homepage (Screen 1)

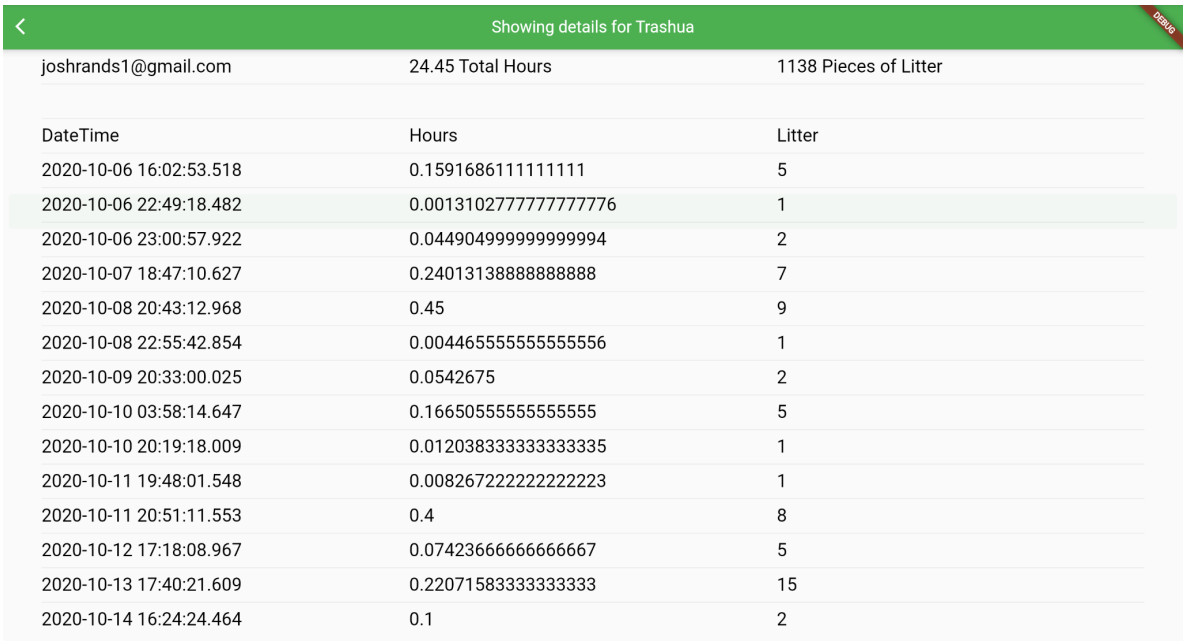


Figure 2: Curren Details Screen (Screen 2)

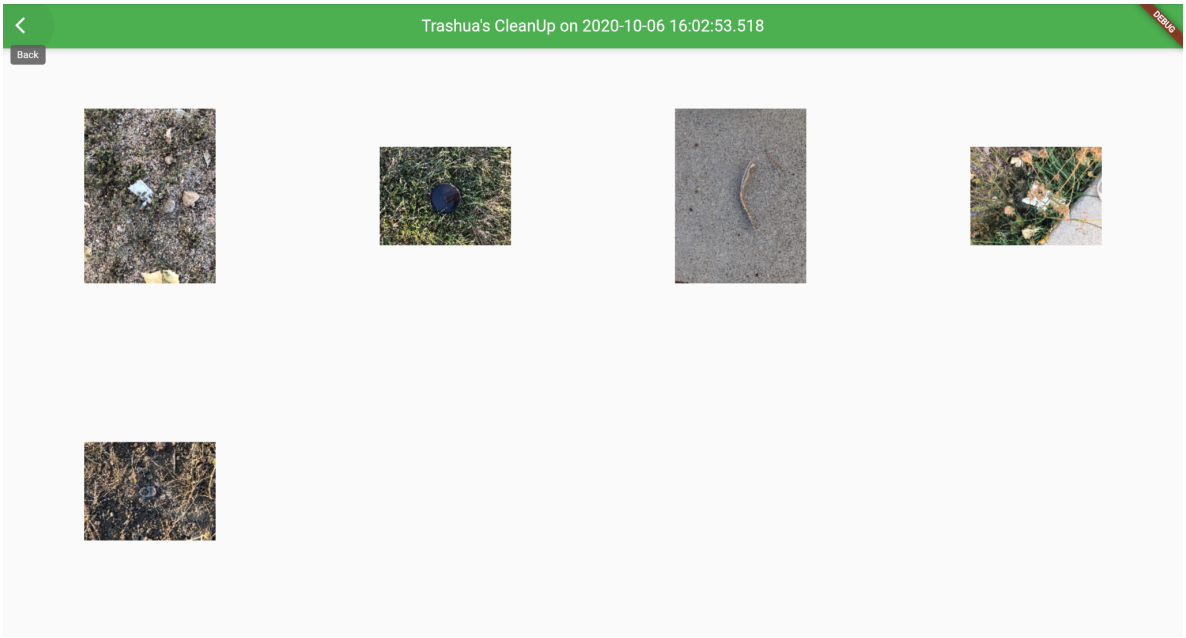


Figure 3: Current Map Screen (Screen 3)

II. Functional Requirements

We were given the description of three screens, or three different pages of the app that Josh would like us to focus on. Specifically, he would like to focus on them in the order of screen two, then screen three, then screen one, and would prefer a finished screen two over started and unfinished versions of all of the screens. The first screen would be a user dashboard, which would preferably have a way for managers to track the activity of all relevant volunteers. The second screen would show information about a specific volunteer. This would include precached data from a specific trip(s)

displayed in charts, certificates for volunteers per trip (pre-populated and made into PDFs), and a direct message (DM) system where volunteer managers can see what questions their volunteers have asked them, and reply. The DM system will, at minimum, be a mock-up. Some volunteer managers have overlapping volunteers with other volunteer managers, so each manager needs to see in which ways a volunteer has been managed by other managers. The DM feature does not exist at all in the current version of the app. The third screen would be a display of the images that a volunteer entered during a trip. This would display each location of trash picked up by a volunteer during a trip on a map that would display the image when the image is clicked on. With enough time, this could also display other information, like the verification percentage from Oscar, or the timestamp that the trash was collected. Currently, this page just displays all images from a specific trip in a grid.

Besides being developed in Flutter, the team is porting their database over to MongoDB from Firebase, so our project should interact with that as well. If we do use MongoDB, we have an opportunity to improve general interaction with this system if we have time. The server framework the current team is using is NestJS and Josh would prefer that we use that for the server framework of our app. If we use Flutter, we have been asked to use in-code documentation at a minimum level of function/class/level. If we use NestJS for our server framework, we should use Swagger documentation. We have been provided with a GitLab repository to pass code back and forth between ourselves as well as Josh.

II-a. Absolute Requirements

The following are our requirements for the project, separated by screen goals and overall goals:

SCREEN 2:

- Chart that displays volunteer statistics for the user filterable by time window.
- Prepopulate volunteer certificates as a Portable Document Format (PDF) with a start date, an end date, total volunteer hours for that period of time, volunteer name, and a current date.
- Managers can see when certificates were generated.
- Implement a non-functional DM user interface between the current volunteer and the user.

OVERALL:

- Calculate the number of hours spent for each volunteer session on the server side.
    - Calculate and store information so that it's not calculating every time something new is pressed.
    - Calculate statistics at the end of the trip and store them within the database for display later.

II-b. Nice-To-Haves

The following are our "nice-to-haves," or things we should implement if we have time. They are separated by screen goals and overall goals:

- SCREEN 1:
    - Create a home page dashboard that replaces the current login screen.
        - Create a home page dashboard system.
        - Create a home page "recent activity feed."
    - Add an "add" button that tracks relevant volunteers to a specific manager.
- SCREEN 2:
    - Make certificates accessible by both volunteers and managers.
- SCREEN 3:
    - Display information about trash on map, such as likelihood an image will be rejected.
    - Create a map of locations a user cleaned up with potentially more information about each location.
- OVERALL:
    - Increase functionality with the database.

III. Non-Functional Requirements

There are some requirements that won't be seen immediately when someone looks at our final product. These are the things specific about how we're to implement certain features. This includes things such as what our documentation should look like, how our project should interact with previous projects or existing code, what database to use, etc. These are important, especially for delivering a product that is workable and adaptable for the client as if our implementation was too far removed from the system they have now, it wouldn't necessarily be the most useful. Our client has a few things he would prefer, but not require, such as working with Flutter for our frontend, and NEST JS for our server. Additionally, he would prefer a RESTful API that we build be used and recommends using Android Studio.

Our non functional requirements are as follows:

- Use proper documentation. This means in-code commenting on all of our files, using Flutter's built in Dart Doc, and high level block diagrams for any server code.
- Use MongoDB data from a provided MongoDB database.
- Done in such a way that it interacts fully/seamlessly with the existing framework.
- No lag on an average (i5) computer. Tested using the built in configuration service in Android Studio.
- PDF generation done in JavaScript.

IV. Risks

As with any project, there are risks that come along in a variety of different ways. For example, our client would prefer that we use Flutter with Android Studio to do all of our interface testing. No one in our group has used Flutter, and only one group member has used Android Studio. Setting up a new IDE to work with a framework that implements a language (Dart) that none of us have used before, imposes a great risk. Learning a new IDE can be difficult enough, and definitely causes hiccups even when coding in a language that one is very familiar with, adding a new framework/language on top of that only further compiles the risk. If we have to spend too much time learning how to implement basics in Flutter, our final project could overall end up rushed or inefficient. Something that could potentially ease this risk is the large number of tutorials and walkthroughs on the internet, so we won't be going in completely blind.

Another risk is that none of us have worked with server frameworks before. Our client would prefer we use a NEST JS server and a RESTful API. Josh has told us that NEST JS can be hard to work with, but his preference comes from the current app using NEST JS. If we cannot get the backend to work, the app will be viewable as a preview in Android Studio, but none of the data from our provided database will make it into the project unless it's hard coded. There are a few things that could ease this concern. Josh has said if NEST JS is too hard for us to implement or is taking too much time, that we can move to Express, which should be easier to use even if no one has worked with it before. He also told us that, if connecting to the database isn't possible for us to do with the time we have, then he would also be happy if we just hardcoded in a user. A similar trend to the first two risks comes with MongoDB and PDF generation, as none of us have used MongoDB or any JavaScript PDF libraries, and none of us have any database management experience.

Moving away from the project side to the teammate/client side of risks, there are also a couple. For the first half of our field session, as stated above, our client will be traveling around Europe and moving back to the States. Having him so conventionally unavailable means that if we need something to be further explained, or we want to have a check-in to see if our progress is in the right direction, that might not be necessarily available for the first half of the session. This could result in us having to cram large changes last minute, which could lead to an overall much rougher product. Many of our teammates live a long commute away, and we are all working at least one job. Finding time to meet that isn't a financial burden on the team may prove to be difficult. Without that meeting time, we could absolutely struggle to stay on top of everything and thus fall behind.

V. Definition of Done

For our project to be done, a few things need to happen. To start, our non-functional requirements need to be met. This means that our project must be done with the MongoDB data we've been provided. This would preferably be with a connection between our app and the database, but we can also hardcode some of the data, as mentioned in Section IV. There also needs to be documentation as described in Section III. This documentation needs to be polished and thoughtful. The non-functional requirement for seamless integration should be fulfilled at least by using Flutter and NEST JS/Express. We should also have used a PDF generation library in JavaScript and our app shouldn't lag on an average (i5) computer.

All of our functional requirements should be applied cleanly. This means that, at a minimum, we must have charts that display volunteer statistics filterable by time, volunteer certificates need to be downloaded and populated with manager-selected data of the correct format, managers can see when certificates are generated, and a non functional DM user interface between the current volunteer and the manager must be present. This can be achieved through testing and code reviews, so multiple tests can be taken into account when doing UI design. Our project should be easily adapted into something Clean Planet Project can easily port over to their new system when they fully implement it. Our complete, functional project should also be fully added to our provided GitLab so that it is in the client's hands. This should include a README or a similar file explaining everything that needs to be on a machine for the project to successfully run. If we have extra time, done would also imply that we finished and/or started some of the nice-to-haves.

VI. System Architecture

$$Time\ per\ Trip\ =\ T\ +\ 3P\ minutes \qquad \text{Equation 1}$$

Equation 1 is the equation needed to calculate time counted toward volunteering. T is the earliest timestamp from a piece of litter in the trip, and P is the total number of pieces of litter collected. This determines how many minutes of volunteering a specific trip is counted for. All photos are uploaded in a batch, which is how the computer knows what counts as a specific trip.

The Clean Planet Project Manager Portal App will have four screens that navigate within each other. At this time, the app contains all four screens, however, our team will be modifying all screens except the first screen. When the app is opened, the user is presented with the login screen, which will not be modified. Once the user logins, they are taken to the home page which lists all the volunteers with their recent activity. This screen also contains an "Add" button which allows the user to add a new volunteer to the database. This button will only be a placeholder (nonfunctional). Once the user clicks on a volunteer, the user is presented with a detailed screen of the volunteer's activity. This screen contains charts for each session, a list of things the managers have done to the volunteer's profile, a button to print a certificate, and a chat button, which allows the manager to communicate with the volunteer. In this iteration, the chat feature is a mock-up (nonfunctional). When the user clicks on a chart, the user is taken to the last screen, which contains a map with pictures for the specific session. This interface is presented in *Figure 4*. For our frontend, it is the user interface itself and the backend is how it works. This includes importing the users from the database and integrating that into the server as well as making sure the features on the frontend function properly.

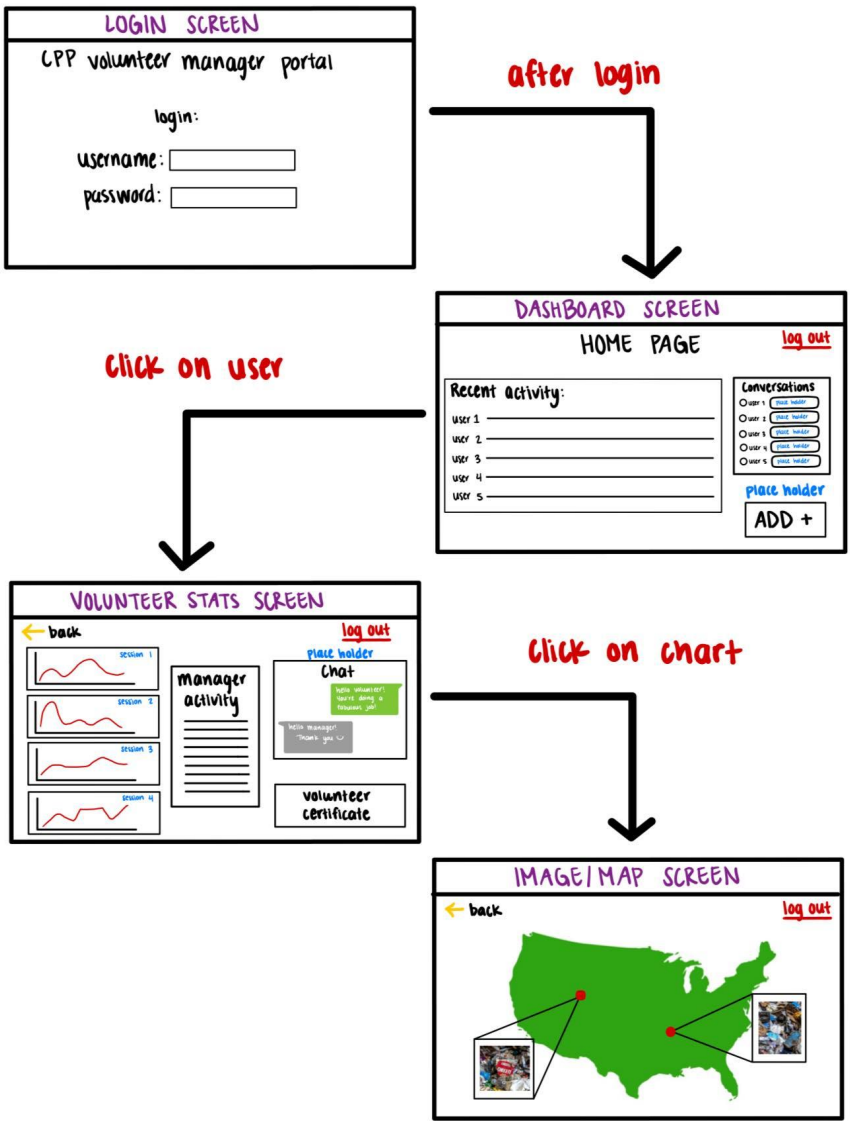


Figure 4: User Interface Flow Diagram

Our team will be using Express as our server to connect our app to our database, through integration of the server into our app's framework. For our database, we will be using MongoDB which will connect into our Flutter user interface. MongoDB is a NOSQL database that handles data in the form of collections containing unstructured data. Collections contain documents and documents contain data fields. The structure of MongoDB is meant to handle dynamic data models, which is incredibly efficient because MongoDB can scale horizontally and vertically. [1]Android Studio will be used to create the application. This can be viewed in *Figure 5*.

[1] Information from MongoDB's "Getting Started" page

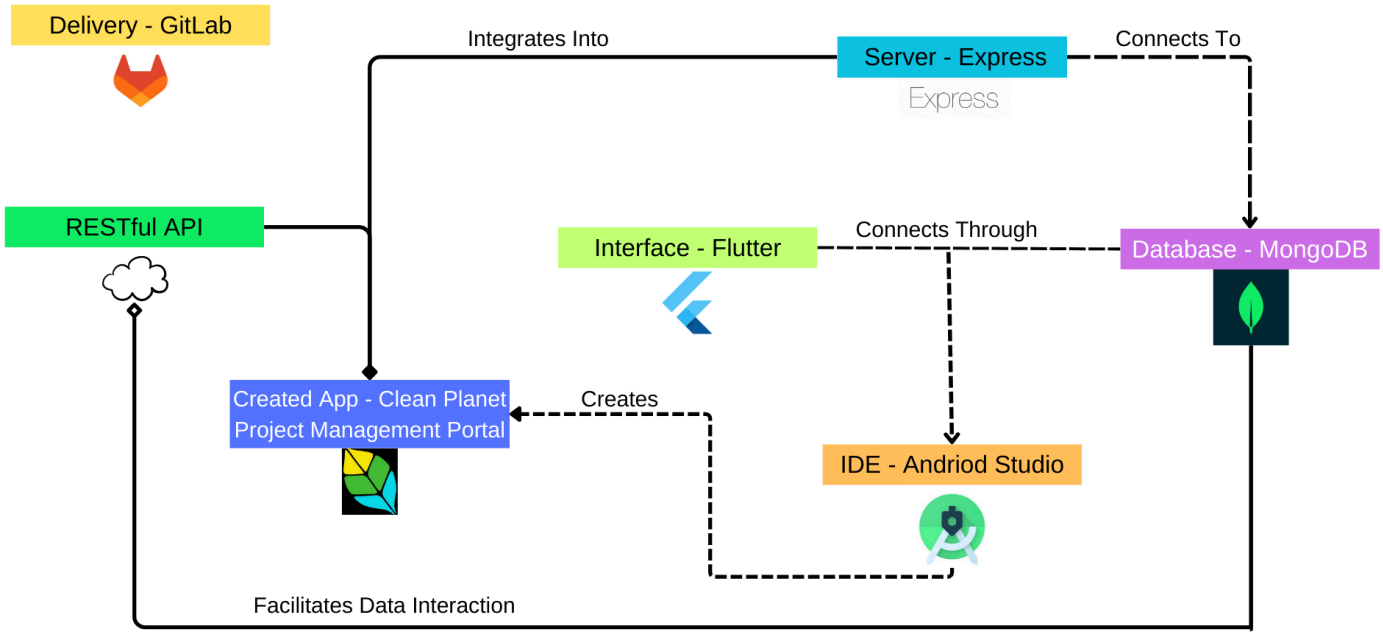


Figure 5: Tool Flow Diagram

Finally, some of the sample data for the database is shown below in *figure 6*, and it follows the schema of the database described above. We can see that the collection with a unique ID has documents named "username," "lowerUsername," "email," and "piecesOfTrash." For the specific expanded collection, we can see the corresponding data fields of "Medusa," "medusa," "fake@gmail.com," and "0."

Figure 6: Database Schema

VI-a. Technical Design

Our technical design went through a fair few iterations before landing on what we currently have. A part of the reason for this is because we got new information about Flutter and best practices as we continued to learn the framework, and a part of it is because we ended up adding more and more programming languages into our project to work with various smaller tasks, which required refactoring to best reflect said additions. To begin, we first want to cover some project structure decisions that became pertinent to our overall design, and then we want to describe our final implementation and flow.

There were a few project folder architecture conventions we had to follow because of our use of Flutter. This is mainly that all of our Dart code had to be contained in the lib folder of our project directory. This originally didn't feel like much of a problem towards the beginning of the project, but as we started working on integrating all of our moving parts together, it posed a rather large problem. One of the biggest issues with Flutter that we encountered is that, because the lib folder is the default, including anything in the project directory that isn't in the lib folder can be near impossible. This is because path and package references currently are incredibly finicky in regards to import statements. This became a problem because the JavaScript library that we chose to use was PDF-LIB. This pure JavaScript library required integration into our server to work correctly. However, because we didn't know anything about the lib folder at the time, we created and downloaded our server into a different folder named server. This means that there is no function call or other denotation that a certification is being generated on the press of the certification generation button, as we couldn't connect that code at all. This also made making the certificates personalized much more difficult. What ended up being the most efficient solution to this problem, mostly due to time constraints, was to integrate the script that was written for PDF generation into the POST request from our RESTful API to store the certificate date. This way, we have access to all of the needed parameters, and we could guarantee that the certificate was being generated on the press of the generation button and no other time, as that POST request was specifically made (and only called) when the certificate button was pressed.

Another structure convention that ended up being a significant portion of our design is the structure of the MongoDB database. As described above, this database uses collections of unstructured data with documents containing data fields. This structure means that pulling data from the database isn't a straightforward process. All of the provided collections had to be put into a modified collection that was significantly restructured from the original. The way the Clean Planet Project Team did this originally was with Python scripts. Josh provided us with these Python scripts and we used them to modify our own data for the users collection and the sessions collection. Clean Planet Project did not have a Python script to do this for the image data, which was one one the biggest reason the litter photos didn't get implemented. We did not have time to write another Python script on top of everything else, especially as the original scripts were provided to us. The scripts provided to us determined the way that we structured all of the data when we were populating the database and pulling from the database.

We used a variety of different libraries in Flutter to create the most visually appealing user interface as we could. This was especially important because a large portion of our requirements boiled down to a good looking user interface in some way. The libraries we ended up using were as follows:

- Dart convert for encoding and decoding data into JSON to communicate with our backend.
- Dart http for connecting into our backend with a URL.
- Flutter chat bubble for making our DM mockup on the Details Page.
- Syncfusion flutter charts for displaying user data as a chart on the Details Page.
- Syncfusion flutter date picker for displaying a clickable calendar for filter the charts on the Details Page and picking a date range for certificate generation in the certificate popup menu from the Details Page.
- Dart intl for handling parsing dates and times both from our calendar user input and the epoch timestamp provided in our MongoDB.
- Dart latlong for using the latitude and longitude data provided for cleanup sessions for use on the Map Screen.
- Flutter map for displaying the map needed for the Map Screen.
- Dart doc for providing beautiful, auto-generated documentation that follows all of the rules of a regular Dart API.

These libraries are all a part of the Dart SDK, so all we had to do to use these libraries is run a command in our Flutter terminal in our project directory, add an import statement to the files applicable, and update our dependencies. This dependency update is prompted automatically whenever we run the aforementioned command in Android Studio, so this process is seamless. This also ensures that anyone who is to pull our project from our GitLab only has to update their dependencies and everything will work as intended. There were a variety of other Dart libraries we tried to use at different iterations of the project, and those weren't included for a variety of reasons. The biggest was that most of them were relating to integration of Javascript in Dart, and they all required we run a web preview of the app. Because we were running our server locally and the web debug configurations in Android Studio also run locally, the server and the configuration were causing conflicts with each other, and not allowing any data from the server to be used in the app. Because of this, all of our debugging was done as if we were running a desktop application, which is what led to a lot of our previously included Dart libraries being removed from the final product.

During a lot of our original iterations, we programmed all of our classes into the main.dart library. This means that every single class we needed for our app to work was all contained within one library. As we continued coding and our project grew significantly, we realized that this was not feasible. It became incredibly difficult to differentiate what code was what, even with in code commenting and Android Studio's feature of auto commenting what brackets belong to what. Because of this, we ended up breaking every page into its own file, and every data type into its own file. This means that we have seven libraries in our final project. The libraries for pages are HomePage, DetailsPage, SessionMap, and main. The libraries for data types are User, Session, and Certificate. Those three libraries specifically allow us to engage in a lot of object oriented programming for our database data, which allows for more efficacious integration of that data into our overall final product. Our Dart Doc API can be viewed here. Next to talk about is how this actually runs on any specific machine.

The app can be accessed by running Android Studio along with the server in the background. As pictured in *Figure 3*, the server connects to the database through index.js. As the application is run, main.dart runs the Home Page screen. Home Page screen fetches the data from the user collection with the use of 'GET user/' from the user.js file, which connects to the user collection. That data is then stored as a User object. The manager is then able to click on the 'add volunteer' button that adds a new user to the user collection with the use of 'POST users/'. The manager is able to click on each user object from the UI and be navigated to the Details screen. Details screen fetches the data from the cleanups collection with the use of 'GET users/:user_id/cleanups/' from the cleanups.js file, which connects to the cleanups collection. Details screen also fetches the data from the certificate collection with the use of 'GET users/:user_id/certificate/' from the certificate.js file, which connects to the certificate collection. The manager is then able to click the 'generate PDF' button, which adds the date and time the certificate was generated to the certificate collection with the use of 'POST users/:user_id/certificate/'. The manager can click on each session object from the UI and be navigated to the Session Map screen.

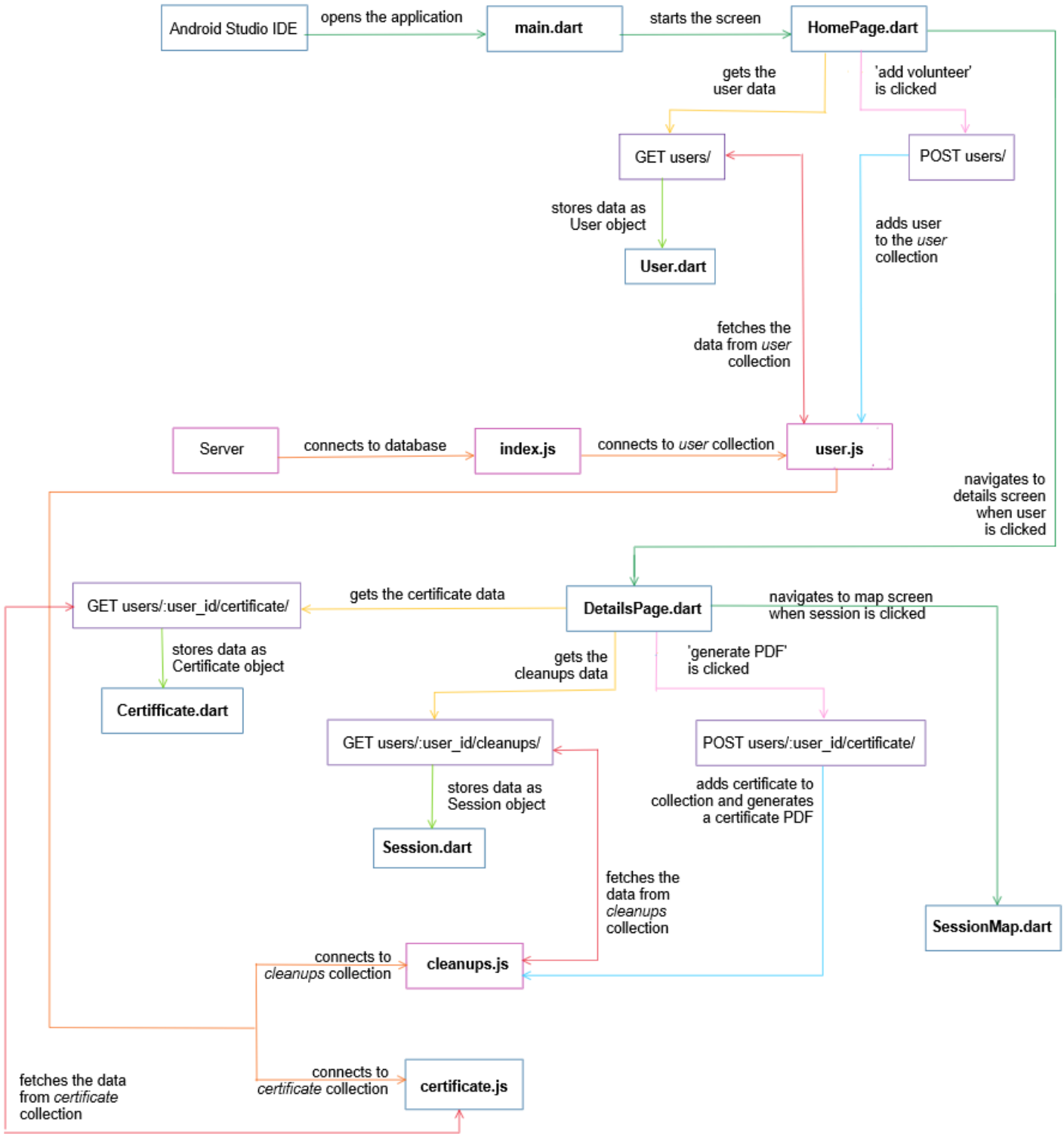


Figure 7: High Level Diagram

VII. Software Test and Quality

In order to deliver a good, clean, and polished product to our client, we need to have a good software test and quality plan. This includes many different kinds of testing and quality checks. For testing, our group chose to focus on unit testing, user interface testing, integration testing, and user acceptance testing. For our testing, it is important that we test every situation we can think of for correct behavior. This includes edge cases or cases where a user does something incredibly out of the ordinary. It is important to test these situations because there is no control over what the user might do or things that might happen outside of a testing environment so covering as many of those scenarios as possible in testing makes less work for later. For quality reviews, we've chosen to go with code reviews and certain code metrics.

The first kind of testing to look at is unit testing. Unit testing is the idea of writing tests to see if specific fragments of code work as intended in every situation. The things we focused on for unit testing are as follows:

- Time per trip is calculated correctly, based on Equation 1.
- The calculated time is stored in the database correctly.

We chose these things for unit testing because they are measurable in the way that unit tests should be. That being said, these things can also be tested with a visual inspection of the database. We chose to do both.

User interface testing for us is mostly manual. As we implement things, we are clicking through the UI to make sure that everything is doing what we expect it to do. We also clicked in ways we wouldn't expect a user to click and tried to break the app a little bit to cover some of those edge cases or things we otherwise aren't expecting. Our user interface tests are as follows:

- Clicking a volunteer's name takes you to the volunteer statistics page for that volunteer.
- The calculated time is displayed in the UI correctly.
- Certification generation dates are shown to the user correctly.
- The name populates into certificates correctly.
- Time data populates into certificates correctly.
- Date data populates into certificates correctly.
- Test if the map displays on the screen.

The above list is best suited for user interface testing because there wouldn't be a real way to implement a lot of them as a unit test. Additionally, some of what we're taking into consideration is the visual efficacy of our UI, which is something that a human needs to be in charge of unless we want to use the artificial intelligence of some kind.

The next kind of testing that is important is integration testing, which is all about our frontend interacting correctly with our backend. A lot of this testing was done by inspection, much like the user interface testing. These tests are important because proper integration is important for usability, and it is also one of our requirements. Our integration tests are as follows:

- Data displays in charts and maps from the database in the UI correctly.
- Information about manager edits is stored in the database correctly.
- Information about manager edits is displayed from the database correctly.

The first test was done by running the server and clicking through the app to see if everything is displaying correctly. We then looked through the database to make sure that everything is being correctly represented. We did this for two volunteers, as most of them didn't have cleanup data provided. We followed the same process for the third test. The second test required the use of our UI to make some edits to a volunteer profile followed by a visual inspection of the database from the MongoDB manager to verify everything was working properly.

The last kind of testing is user acceptance testing. Because we aren't in communication with any of the project managers from the Clean Planet Project, we don't necessarily have a user to test on. Because of this, we talked with our client and got the things he would be looking for as a user, and tried to implement those. Then, we had people in our lives who aren't familiar with our intentions for the project to click through our UI and tell us what they thought about using our app. The things we will be focusing on for usability are as follows:

- No lag on an average laptop while using the app. This generally means things like scrolling and switching between screens happen seamlessly.
- The app doesn't crash.
- The app does what it is intended to do.

As we tested on multiple people, we got multiple views on things such as “lag” that are more subjective which helped us narrow down on how to make the app more user-friendly.
Code reviews are important to guarantee our code has the most quality possible. We are doing this in a couple of ways. First, since we cannot meet in person frequently, we are using Code With Me to do remote pair programming. This is important because two eyes on code at once helps us to find more mistakes quicker, and allows us to clean up code as we write it easier. Secondly, because we’re working remotely and asynchronously more than is optimal, we can tend to have a lot of problems with merges. When a merge occurs, we have a protocol to review the changes and talk through fixing the merge issue with as much of the group as possible, before fixing the merge or moving to make any more major changes.
The next and final consideration here is code metrics. This is mostly defined by our documentation nonfunctional requirements, but having clean and informational documentation is important for code maintainability. This includes in-code documentation for everything that we write as well as high-level block diagrams to represent our server.

VIII. Project Ethical Considerations

Another important part of quality assurance and a responsibility of anyone making a product is ethical considerations. These considerations are very important to think through and integrate into every step of our process because we have a responsibility as engineers to create ethical products to the best of our abilities. Additionally, because we are working for an environmental justice non-profit, it is even more important for us. With so many considerations, there are some that are more likely to be violated and there are some that are more pertinent. For the remainder of this section, we focused mostly on the code of conduct from the Institute of Electrical and Electronics Engineers (IEEE), although The Association for Computing Machinery (ACM) code of conduct has many similar things to say.

To start, the specific IEEE principles that are particularly pertinent to our product, along with the reasons that they're pertinent to our project, are as follows. These are broken up into public considerations, product considerations, and self considerations:

PUBLIC CONSIDERATIONS:

- 1.03 Approve software only if they have a well-founded belief that it is safe, meets specifications, passes appropriate tests, and does not diminish quality of life, diminish privacy or harm the environment. The ultimate effect of the work should be to the public good.

Clean Planet Project is a non-profit organization, and what we are making is supposed to make it easier for both them and their users to make a difference in the world and in the climate. If our project ends up not being safe, or not being what the client wants, diminishes the quality of life of either the client or the user, and/or is detrimental to the user or the environment, then what we are doing is antithetical to the goal of the Clean Planet Project.

- 1.08 Be encouraged to volunteer professional skills to good causes and contribute to public education concerning the discipline.

In the same vein as the principle above, if we aren't volunteering our skills to a good cause and helping to educate people about this movement in some way, we are at minimum doing a disservice to the client and the movement, and at worst, working actively against it.

PRODUCT CONSIDERATIONS:

- 3.03 Identify, define and address ethical, economic, cultural, legal and environmental issues related to work projects.

Because The Clean Planet Project is an environmental justice project, it is important that we address specifically the ethical and environmental concerns that go into what we're making. It is important that we take the most ethical and least environmentally impactful approach to keep the project aligned with the goals of the client and the greater good.

SELF CONSIDERATIONS:

- 8.02 Improve their ability to create safe, reliable, and useful quality software at reasonable cost and within a reasonable time.

It is important that we use this project as a way to practice and improve our skills around creating safe, reliable, and useful quality software within a reasonable time frame because this field session is an opportunity provided to us to do just that. If we do not meet this goal, then we are being, in a sense, wasteful.

To continue the ethical discussion, there are also IEEE principles that are most in danger of being violated. These principles are as follows, along with the reasons why they are most in danger:

PRODUCT CONSIDERATIONS:

- 3.04 Ensure that they are qualified for any project on which they work or propose to work by an appropriate combination of education and training, and experience.

If we are not properly qualified to do this project and we do not communicate this with the client and adjust the project accordingly, we could end up delivering something useless or detrimental. Comparatively, if we communicate our skill gaps beforehand, we can make something that is overall more useful and satisfying to the client and more useful to our learning because we aren't just spinning our wheels trying to do something we just can't reach yet.

- 3.08 Ensure that specifications for software on which they work have been well documented, satisfy the users' requirements and have the appropriate approvals.

If we do not document our code properly, when The Clean Planet Project Team goes to move over to their new system completely, or if another field session team decides to pick up where we left off, then our code can become very difficult to understand and work with. At a minimum, it will be a large inconvenience to future people who have to work with our code, and at the worst, it could end up making our whole project unusable or basically unusable.

- 3.09 Ensure realistic quantitative estimates of cost, scheduling, personnel, quality and outcomes on any project on which they work or propose to work and provide an uncertainty assessment of these estimates.

If our team cannot assure realistic estimates of scheduling and quality of outcomes specifically, then we could end up costing our client a lot more in the long run. This is because future plans for the company can (and often are) made based at least partly on what information we have given them. If these estimates aren't true, it can throw the entire timeline for our client off, and cause many different adverse consequences.

- 3.10 Ensure adequate testing, debugging, and review of software and related documents on which they work.

If our team does not ensure that our testing, debugging, and review of everything we hand off to the client isn't at least adequate, then we run the risk of handing them many different problems down the road that they aren't necessarily equipped to solve. This isn't because they aren't qualified, but because they didn't work on it with us and just aren't familiar with the entire project.

- 3.11 Ensure adequate documentation, including significant problems discovered and solutions adopted, for any project on which they work.

If our team does not provide adequate documentation, we run into similar problems as those above. Additionally, if we don't communicate any significant problems that we encountered and how (and if) we came up with a solution, then anyone who works on this project in the future, as well as the client, won't know the pitfalls, weak points, or strong points of the project. This could make adapting, expanding, or bug-fixing difficult or potentially impossible in the future.

Continuing on, a professor named Michael Davis who worked for the Illinois Institute of Technology created a framework called the "Seven Step Method for Ethical Decision Making." This framework consists of seven tests that one should apply to their decision-making in order to make the most ethical decision possible. For our project, we chose to apply two tests to our plan, and the two tests that we picked were the publicity test and the mirror test.

The publicity test states "How would this choice look on the front page of a newspaper?" Our answer to this question is that the vision that we have would look great on the front of the newspaper. This is partially because of the initiative that we are working with, but also because many of the decisions we are making with our design line up with the software engineer's code of ethics. The product that we have right now perhaps wouldn't be the best on the front of a newspaper, but that is mostly because we're still developing and haven't gotten to a stage where that is necessarily applicable.

The mirror test says "Would I feel proud of myself when I look in the mirror?" Our answer to this question is that In a similar fashion to the publicity test, because of the initiative and the cause behind this project, working on it is

something that absolutely makes us as a group proud. As long as we complete at least the volunteer statistics screen, I think that we would also be proud of ourselves at the end of this project, especially with all of the new things we are trying to tackle all at once.

To wrap up the rest of the ethical discussion, it is important to talk about the ethical ramifications of the software quality plan discussed in Section VII not being implemented correctly. or if our plan isn't comprehensive enough. If our software quality plan is not implemented properly or is not comprehensive enough, we run the risk of making The Clean Planet Project's job more difficult, because they would need to adapt and/or extend what we have given them. In the long run, this could affect the number of volunteers that they have. This is because the app is currently not very easy to use, and if this continues for too long, or if they roll out our project as a new app and it is harder to use, then one of the core tenants of The Clean Planet Project is violated and volunteers will want to volunteer less because of the increase in difficulty. This could lead to fewer people picking up litter, especially with school projects and similar service projects that rely on this app to give them hours, which is a detriment to the environment. Additionally, if accessibility to volunteerism is lowered, fewer people could volunteer, and if that happens, all of the positive benefits that volunteering has on people and their persons isn't reaching as many people, leading to a potential decrease in virtue.

IX. Results

There are a few features from both our requirements and our nice to haves that didn't get implemented. Everything for Screen 1 was implemented. This can be seen in *Figure 8* and *Figure 9* below. When the green plus button in the bottom right hand corner in *Figure 8* is pressed, the popup dialog shown in *Figure 9* is displayed to the user. This dialog allows the manager to add a new volunteer to the database.

FIgure 8: Final Homepage Screen (Screen 1)

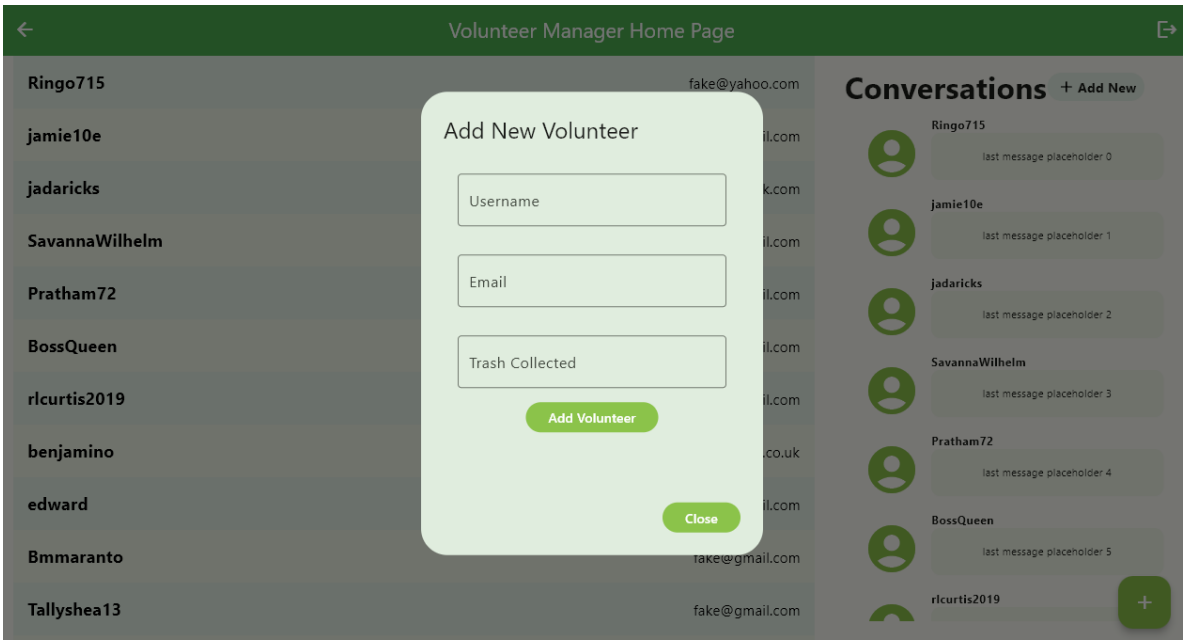


Figure 9: "Add User" Button Popup

For Screen 2, the nice-to-have chat functionality wasn't implemented. Additionally, as we ran into issues, the charts on Screen 2 aren't filterable. For Screen 3, the images and information about the images weren't implemented. Overall, every requirement was either implemented or mostly implemented, and most of our nice- to-haves were also implemented. These results can be seen in *Figure 10*, *Figure 11*, and *Figure 12*. When the "certificate button" on *Figure*

10 is pressed, the screen transitions to the dialog presented in *Figure 10.* For *Figure 11* specifically, the date line at the top of the certificate, the volunteer time, and the date range are all based on the user and the date range picked in *Figure 11*.

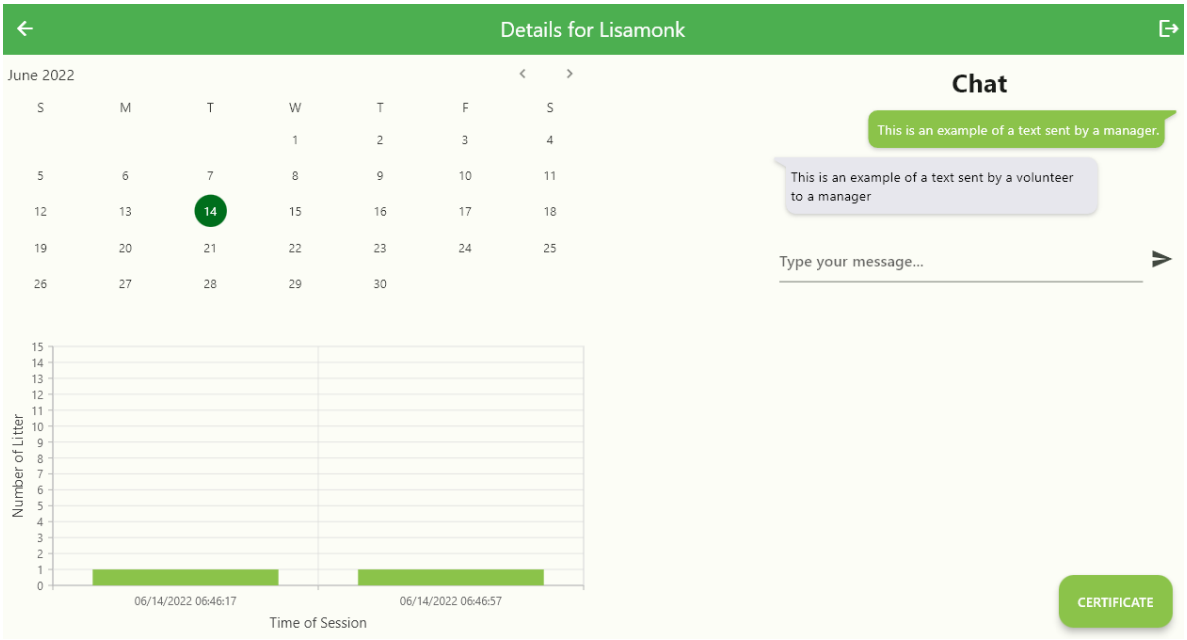


Figure 10: Final Details Page (Screen 2)

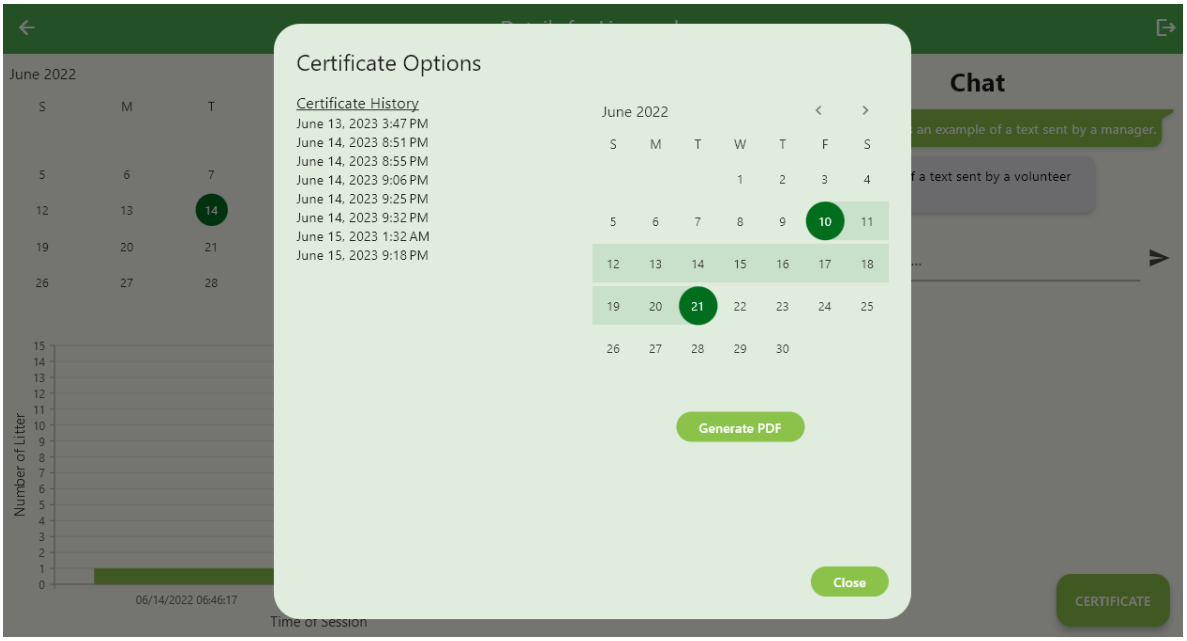
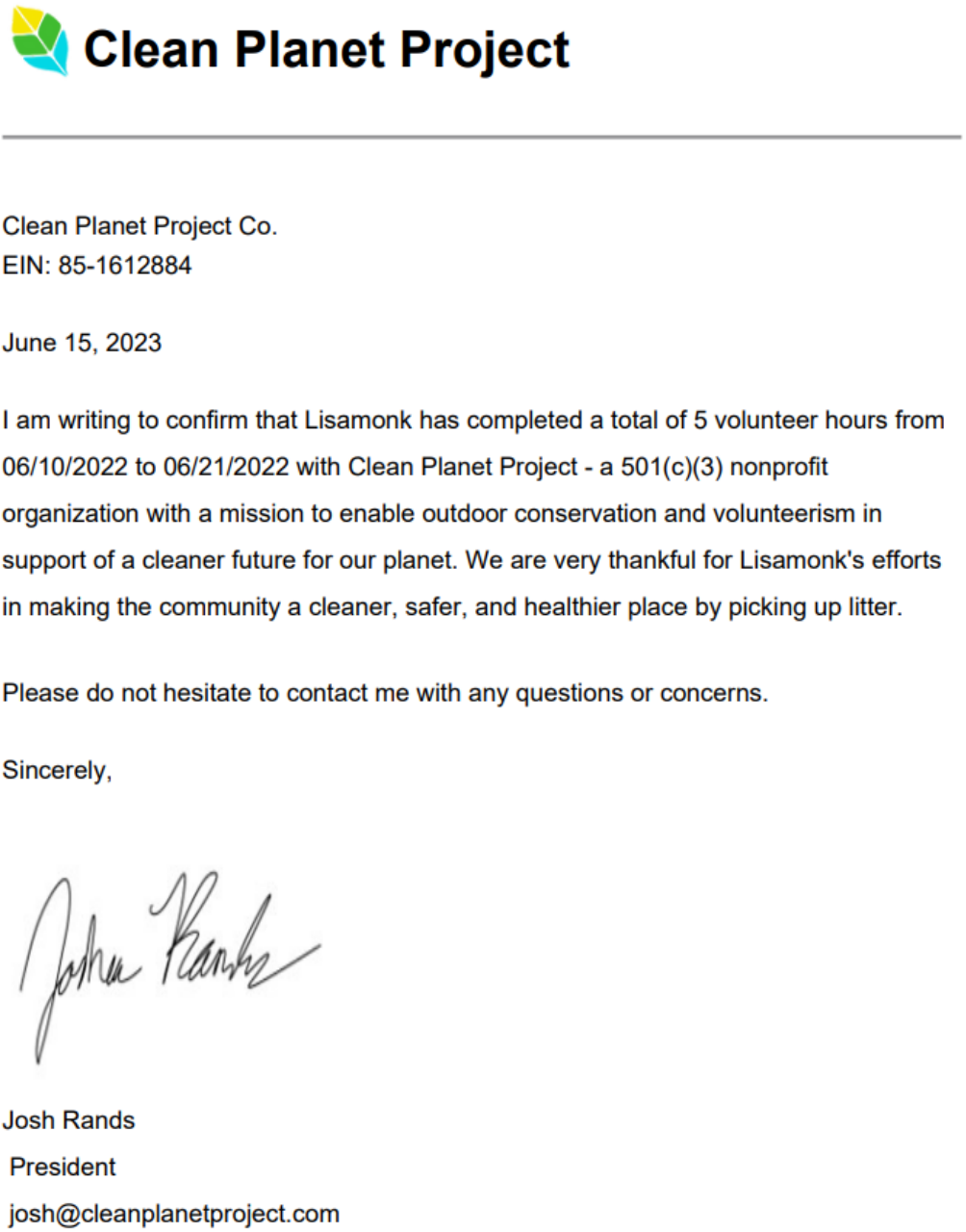


Figure 11: Certification Options Popup



Clean Planet Project

Clean Planet Project Co.
EIN: 85-1612884

June 15, 2023

I am writing to confirm that Lisamonk has completed a total of 5 volunteer hours from 06/10/2022 to 06/21/2022 with Clean Planet Project - a 501(c)(3) nonprofit organization with a mission to enable outdoor conservation and volunteerism in support of a cleaner future for our planet. We are very thankful for Lisamonk's efforts in making the community a cleaner, safer, and healthier place by picking up litter.

Please do not hesitate to contact me with any questions or concerns.

Sincerely,

Josh Rands
President
josh@cleanplanetproject.com

Figure 12: User Generated Certificate

For Screen 3, the map was implemented, but not the images and their information. Currently, the map appears zoomed into the longitude and latitude of the cleanup. This is shown in *Figure 13*.

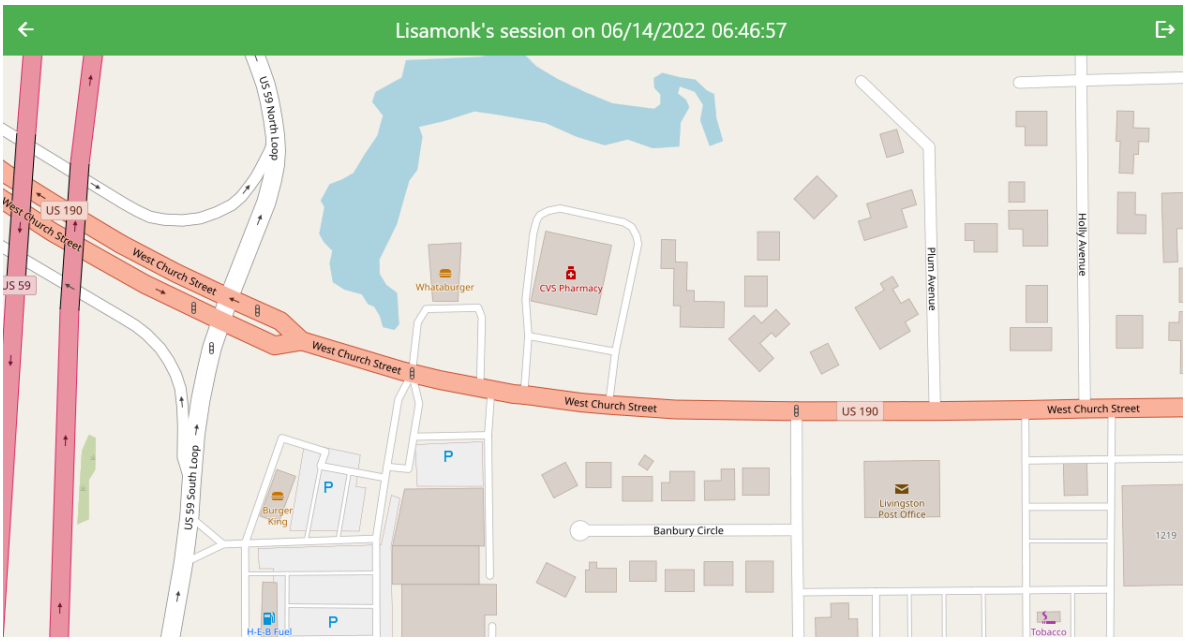


Figure 13: Final Map Screen (Screen 3)

As far as performance testing goes, there wasn't any way to run our app in something other than the Flutter preview window. On that window, the only hang ups that occurred were related to the Flutter preview window. The server doesn't hang up and once everything is loaded in, everything runs seamlessly.

Overall, we are able to properly navigate through the different pages with the back arrow. When a user is clicked on the home page, we are taken to its corresponding detailed stats page where it shows its recent activity and charts. When a specific chart is clicked, it brings us to the maps/image page properly where the litter picked up on the map would be displayed, although it is currently blank. The entire process is responsive and easy to navigate/use. This means that all of our user interface testing passes. The PDF testing did end up being more of a UI test, and that absolutely passed as well. Our timing equation also worked out as more of a unit test.

On the side of usability, there are no crashes. The user is able to click on buttons and navigate to different screens by clicking certain information. All buttons presented on the screen have functionality and work as intended. This includes generating certificates and moving throughout those windows. The UI is nice to look at and easy to navigate, making the learning process and flow of navigation intuitive.

X. Future Work

Because our project, in a lot of ways, is a very detailed and elaborate template for the current Clean Planet Project team to adapt from as the project moves forward, there is a ton of potential for future work. Since the project was presented to us through the lens of screens, it is fitting to start talking about future work for screens first. The main screen priority would be the Map Screen. As mentioned above, the current map screen has a map zoomed into a session's latitude and longitude and that is about it. The most obvious expansion here would be implementing the litter gallery to the display, as that was the main stretch goal for this screen besides the display of the map. The images would have to be anchored to the places that they are picked up, even when the manager is zooming in or out or otherwise moving around the map. This would also require the writing of a Python script to make the data usable. There is also a variety of metadata such as time collected currently connected to a picture that would be useful to display with the picture in some way. As mentioned above, Oscar also generates a number to describe if an image is going to be accepted as a valid cleanup, so that is something that could also be displayed. A byproduct of the longitude and latitude zoom is that it isn't immediately obvious where a cleanup took place. The name of the city or town that contains the longitude and latitude, along with the state that contains it, is also information that could be displayed. There are also a few ways that this information could be displayed. One way could be in another screen upon clicking the picture of the litter. Another way could be in a popup dialog box when the litter is clicked. In a different vein, it could also be displayed when the mouse hovers over the picture.

There is also the chat feature that is present on both the Home Page and the Details Page. Currently, these are both mock-ups and in that way are non-functional. There is also no volunteer portal at this moment, so that would need to be implemented and connected to the manager portal in some way before any sort of functionality could be added to the chat. A possible implementation here for the Home Page chat list specifically is to make the current chats that a manager has function a lot like the list of volunteers that they're tracking. This is because the chat exists as an embedded feature of the Details Page. It would make a lot of sense if clicking on a chat for a specific user would take the manager to their Details Page. The placeholder text would also need to be replaced with the last message sent, and would probably need to have some indication of if that last message was sent by the volunteer or the manager. This could be done with a simple identifier of "you:" at the beginning of the text if the last message was sent by the manager. As far as chat functionality on the Details Page, messages that are typed in the current message field and sent would need to generate a new message that is then displayed on the screen. The same would need to happen if a volunteer sent a message. The generation of a chat bubble would most likely come with some animation. Additionally, these messages would need to be saved into the database somehow and then subsequently fetched. That means that a schema for the original collection, a modified collection, and a Python script would most likely need to be developed for full functionality here.

On the backend, everything is currently run on a local server, so making this into something hosted on the web is also potential future work. Doing this would require integrating this app into the current Clean Planet Project's servers, which are run on NestJS and not Express, which could be a potentially large workload. This would also require the integration of the login screen into the app that we have now, and because it would be hosted on the web, it would also require proper authentication and security. This means that managers would need to login. During this login process, they would need to have their password authenticated. On top of this, all passwords would need to have some kind of encryption scheme to make them as secure as possible. Since this is a manager portal, it would also be necessary to make sure that only managers can log into this portal. This would be where the certificate generator by other managers being viewable to the current manager when there are overlapping volunteers becomes applicable and would then be good to implement.

Last but not least, there is a little bit of functionality with the charts that aren't completely implemented, as well as a few potential improvements to the charts. Currently, the filtering on the calendar doesn't work completely, and any click on the calendar will bring up all of the charts for a volunteer. Getting the date filter to work correctly and refresh on every click of the chart is something that still needs to be finished. A potential way to make session dates more clear to a manager is making those dates the only clickable dates on the chart calendar, along with changing the color of the background on those dates in some way. Additionally, adding an additional dropdown or text fill field to allow them to filter that way if it's more accessible for whatever reason could also be a potential addition to the chart feature.

XI. Lessons Learned

- We learned how to code with Flutter, meaning we also learned how to read through an API and build off of both the API and different tutorials.
- Port in a database for backend with the use of MongoDB, Node.js, and REST API.
- Taking useful software-related classes before taking Field Session would be very helpful.
- Agile is a powerful structure for software development, but it can sometimes become overwhelming if a teammate or client is incapable of timely communication for whatever reason, for a period of time. Talking through this possibility and what it means before it happens would be very beneficial for the team.
- Although it is helpful to break overall goals into smaller tasks, this can be difficult to do effectively when all of the tools you're working with are brand new to you. You should come prepared with more research than you think you need before starting to break things up to make this task less daunting.
- Working in pairs or as a whole team hinges on being able to communicate verbally and as in-person as possible. However, this isn't always realistic because of commute times and work schedules. A way to remedy this is video calls and applications such as Code With Me, that allows remote group work to simulate in-person group work as seamlessly as possible.
- There is a stigma around asking questions or asking for help and what we think it makes us look like, but asking clarifying questions is crucial. No matter how small or self explanatory a question ultimately is, even just the experience of asking a question is more than worth it. Ask questions as soon as you have them so the team can have the most useful information as quickly as possible. Ask for help when you need it because a part of developing anything is leaning on others to better yourself, your understanding, and your overall product.

XII. Acknowledgments

Thank you to Jeffrey Paone, who is a professor at Colorado School of Mines, for taking the time to meet with us weekly for the past five weeks. He was our advisor who helped us stay on track when it came to working on our project and also gave us feedback on our presentations and reports. Thanks to Dr. Paone, we were able to learn more about ethical considerations in the computer science/engineering field and the work of agile within software engineering.

Josh Rand is the founder of the Clean Planet Project, which allows for more outdoor activism in picking up litter. He was our client during this field session who was so gracious to us and allowed us work on what was possible to accomplish within these past five weeks. Josh trusted us to create and execute a volunteer web application for his company, although communication was difficult within the different time zones as he was traveling abroad. He also always made time to meet with us on a weekly basis. Thanks to Josh, we were able to complete the task he assigned to us with the many systems he provided for us to work off on.

Kathleen Kelly is also a professor at Colorado School of Mines. She was the main course instructor for Field Session this summer. Dr. Kelly did an amazing job of running the course this summer. She always gave us daily reminders of tasks for the week as we kicked off each sprint. Thanks to Dr. Kelly, we were successful in keeping each other accountable and remain enthusiastic throughout the big coding weeks. From every lecture she gave, we were able to learn the importance of ethics, software quality, unit testing and more.

XIII. Team Profile

Minnie Her
Advisor POC

Year: Senior

Skills/Experience: Mining Engineering Receptionist, Game development, music, art & poetry, ministry, Hmong language

I am passionate about ministry and hope to do missions overseas in the future.

Briar Martin
Client Liaison

Year: Senior

Skills/Experience: User centered design, Web App Development, DI&A, McBride Honors Program, Kickstart, philosophy, ethics, CAD and graphics, sports and sports management

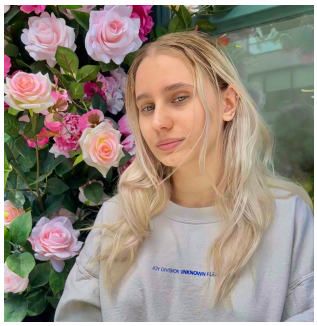
During my time at Mines, I've tried my hardest to create change in social, environmental, and educational contexts across many different facets of campus. I am very excited to direct that drive to a good cause outside of Mines, while also developing skills pertinent to my major and my individual future.

Christine Hwang
Temperature Checker

Year: Senior

Skills/Experience: Game Development, Violin, Drawing and Painting, Digital Art, Basic Chinese and Korean

I have somewhat extensive experience and am quite passionate about art and design.



Anastasiya Velychko
Team Leader

Year: Senior
Skills/Experience: Game Development, Web Applications, Mobile Apps Development, Customer Service, Food Industry, Piano, Russian Language, Drawing

I am passionate in helping the environment and I would like to get more experience in designing web applications.

References

[1] *Get started with mongodb*. MongoDB. (n.d.). https://www.mongodb.com/basics/get-started

Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

Term	Definition
ACM	*Association for Computing Machinery*
Code With Me	*A collaborative programming service made by JetBeans that allows groups to program into the same file at the same time from different machines.*
CPP	*Clean Planet Project*
DM	*Direct Messages*
Flutter	*An open source framework for building application UIs.*
GitLab	*A web-based Git repository that provides free open and private repositories.*
IEEE	*Institute of Electrical and Electronics Engineers*
MongoDB	*A source-available cross-platform NoSQL database program.*
Firebase	*A set of backend cloud computing services and application development platforms provided by Google.*
NestJS	*Frameworks for building efficient, scalable, and enterprise-grade backend applications using NodeJS.*
Oscar	*Learning method that uses object tags detected in images as anchor points to ease the learning of image-text alignment.*
PDF	*Portable Document Format*
PDF-LIB	*A pure Javascript library for PDF generation.*
Swagger	*Open source set of rules, specifications and tools for developing and describing RESTful APIs.*
UI	*User Interface; where interaction between a user and a computer occurs.*
WebApp	*Web Application; an application program that is stored on a remote server and delivered over the internet through a browser interface.*

Appendix B – Figures

Includes all figures and equations.