Group 1: Cordelia Notbohm, Sydnee Haugen, Abhishek Muthabatulla
CSS 481 Web Development: Project Proposal

Happy Hacker Habits

A programming-themed habit tracker/self-care app that helps users create meaningful habits and improve their mental health.

Table of Contents

Development Team

This section describes our team members and some of the skills that make them qualified to develop this application.

Cordelia Notbohm

Cordelia Notbohm is a senior computer science student at the University of Washington Bothell with experience in Java, C/C++, Python, Javascript, CSS, HTML, and SQL. Academically, she has taken classes in data structures/algorithms, databases, computer networks, distributed computing, cloud computing, and web development. Professionally, she did an internship at Hearsay Systems, during which she created a serverless, event-driven feature using AWS services to help compliance directors improve their post-flagging criteria. In her free time, she mentors a local robotics team where she teaches high schoolers programming skills.

Sydnee Haugen

Sydnee Haugen is a senior computer science student at the University of Washington Bothell with experience in Java, C/C++, Python, Javascript, CSS, and HTML. Academically, she has taken classes in data structures/algorithms, computer vision, cloud computing, and web development.

Abhishek Muthabatulla

Abhishek Muthabatulla is a senior computer science student at the University of Washington Bothell with experience in Java, C/C++, Javascript, CSS, HTML, and Figma. Academically, he has taken classes in data structures/algorithms, human-computer interaction, and web development.

Project Motivation

Many applications are available to help users accomplish goals and improve their mental health. However, most of these applications only provide some of their features for free. The development team found that none of the current applications provide all the features they want, for free, with a focus/theme that motivates them to accomplish their goals.

As future software engineers, the development team is motivated by committing code and deploying successful projects. They also love using tools, like Visual Studio Code, that include many extra features that help them develop their programs.

This project will take advantage of these motivations and appeals. Happy Hacker Habits will be a programming-themed habit tracker, focused on creating goals and accompanying habits to achieve those goals. Taking inspiration from an IDE, it will also include other side features like daily quotes, emotion logging, and journaling to help users accomplish their goals.

Comparison of Competing Applications

This section compares our project with other competing applications, emphasizing our project's improvements.

Habitify: Powerful Habit Tracker

Habitify is a habit tracker that allows users to create habits, organize habits into categories, and schedule habits for different days/times. It includes a rich history function to keep track of completion statistics and streaks. Habitify even includes an integration feature with Apple and Google Calendar/fitness. However, Habitify's free tier does not include many of these features and limits the number of habits you can create. Furthermore, Habitify is focused on habits, and it does not include many other self-care features.

Our application Happy Hacker, will be a free application that takes inspiration from Habitify's habit features and allows users to create limitless habits. It will maintain the ability to categorize and schedule habits. Happy Hacker will improve on Habitify by including other self-care features like daily quotes, mood tracking, and a debugging duck chat feature.

Finch: Self-Care Pet

Finch is a self-care app that allows users to create habits and tasks that, when completed, help to raise a virtual pet finch. In addition, it includes other self-care features like daily quotes, mood tracking, and journaling. Most of these features are free. However, Finch focuses on the virtual pet, with many features centered around customizing your finch. Finch is also less focused on task completion and doesn't provide as many statistics.

With all the features one might want to take care of your mental health, Finch is closer to what the developers envision for Happy Hacker. However, for users who are less motivated by a virtual pet, and are more motivated by statistics, Happy Hacker will be focused on how many goals have been "deployed" and how many days you have "committed" a habit. Happy Hacker's programming theme will also appeal to a more STEM-based audience.

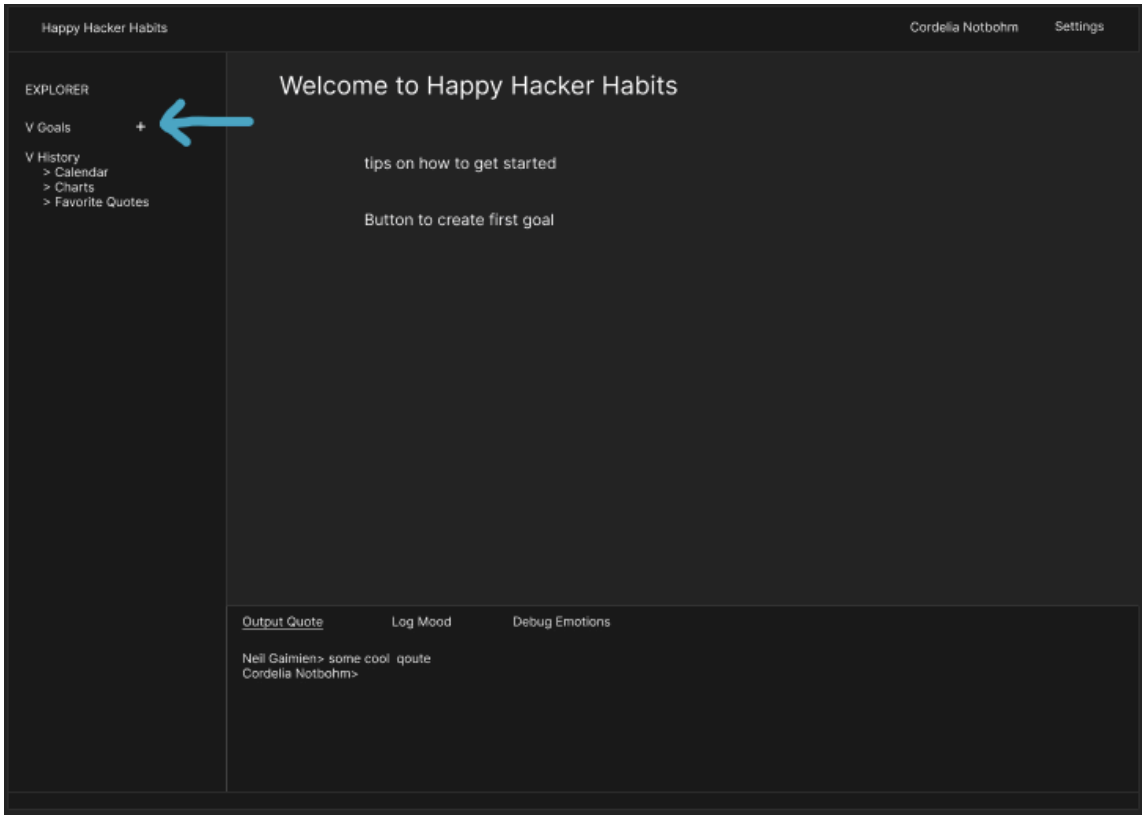
User Scenarios

This section describes the tasks users can accomplish using our application. For each task, there is an accompanying set of visual mockups that show each "click" along with what the user experiences as a result of the interaction. The visual mockups used to design the project can be viewed here.

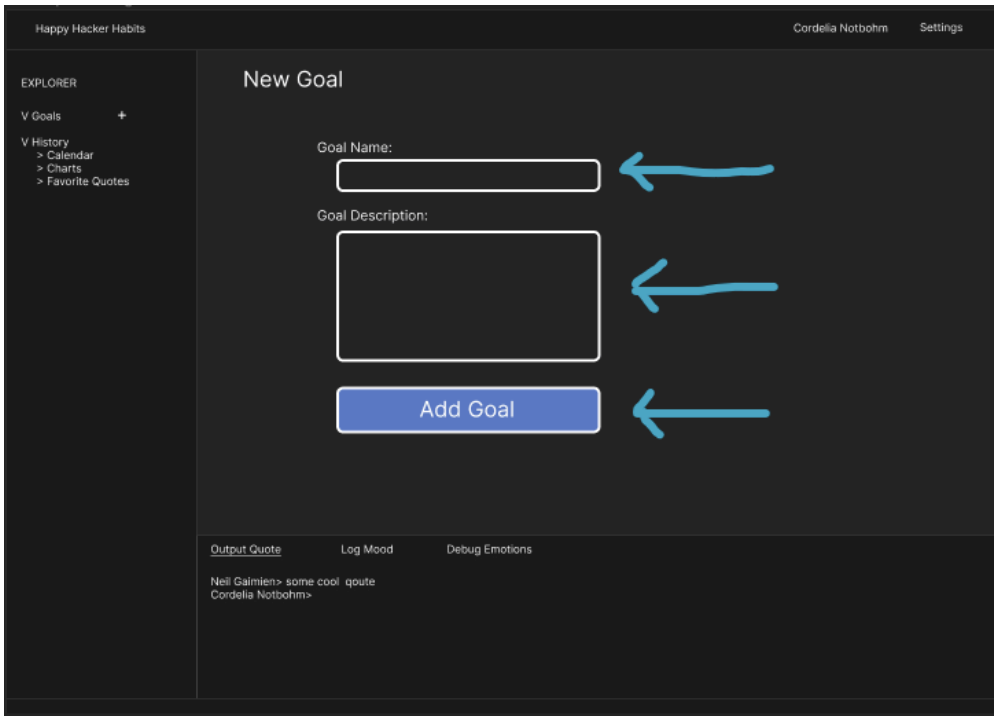
Scenario 1: Create a Habit

Happy Hacker allows users to categorize habits into separate “goals.” This flexible categorization mechanism allows users to define goals however they see fit and then create habits under each goal.

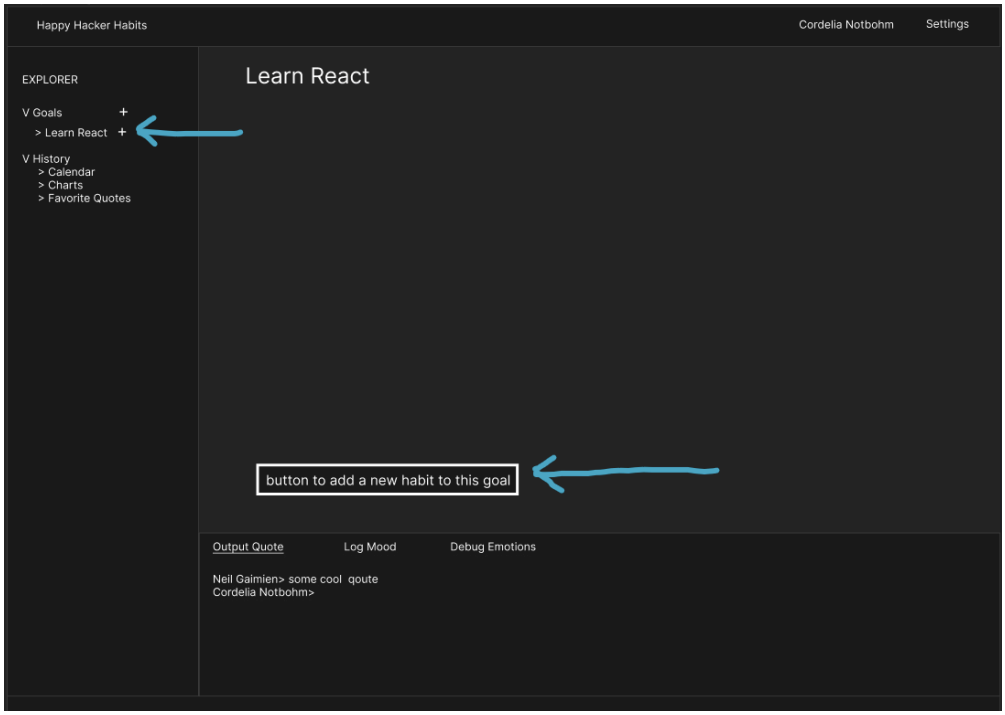
The first step to creating a new habit is to add a new goal. To add a goal the user clicks the plus icon next to the “Goals” section of the “Explorer” component.



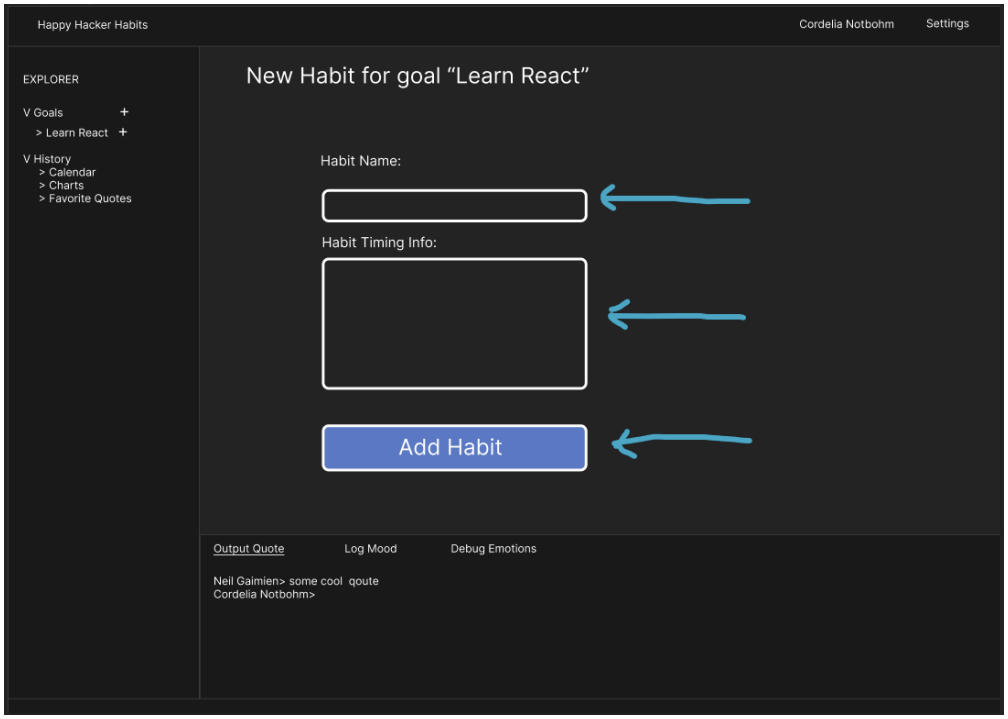
After clicking the plus icon, a “New Goal” component pops up on the main code editor screen, allowing the user to add details about the new Goal. After giving the goal a name and a description, the user clicks “Add Goal.”



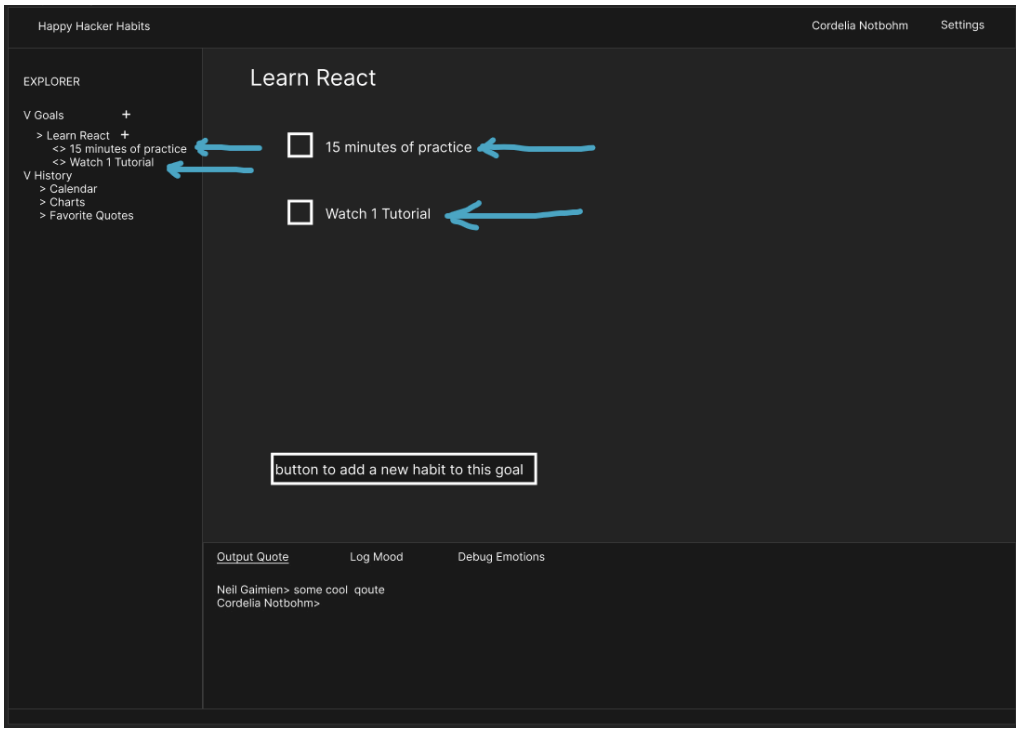
The user is then brought to the “Goal Page” for the newly created goal. It is initially empty and contains no habits. At the bottom of the screen, the user can click the “Add Habit” button, or the user can click the plus button next to the new goal in the “Explorer” component.



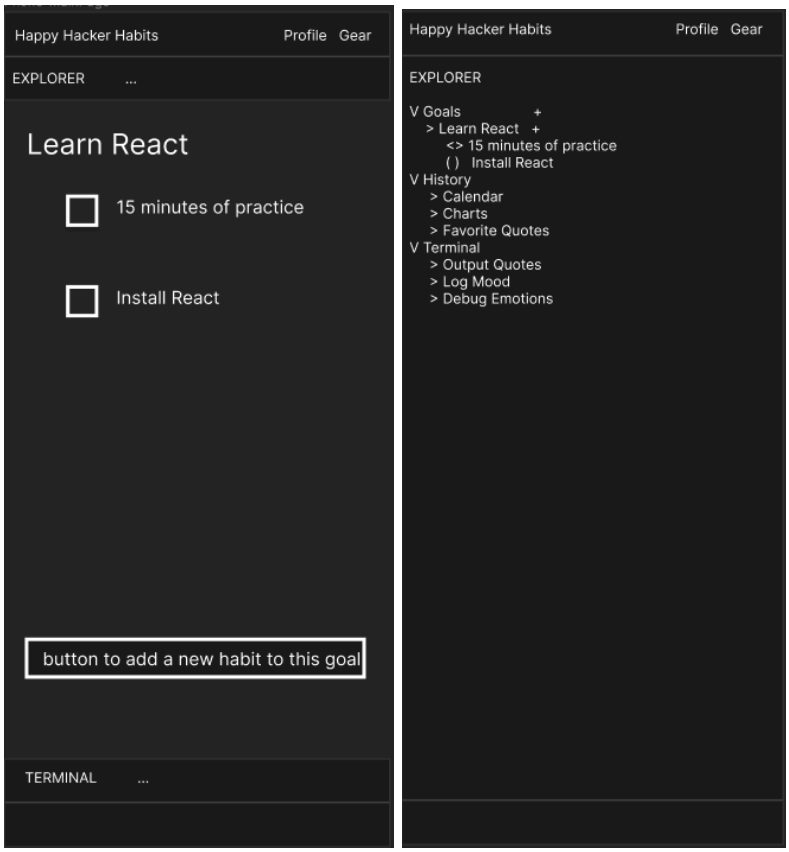
Clicking one of the “add habit” buttons brings the “new habit” page onto the main code editor, allowing the user to add details about the habit, like how often it should be performed.



After clicking “add habit” the habit is added to the page for the corresponding goal and is listed under that goal in the “explorer component”

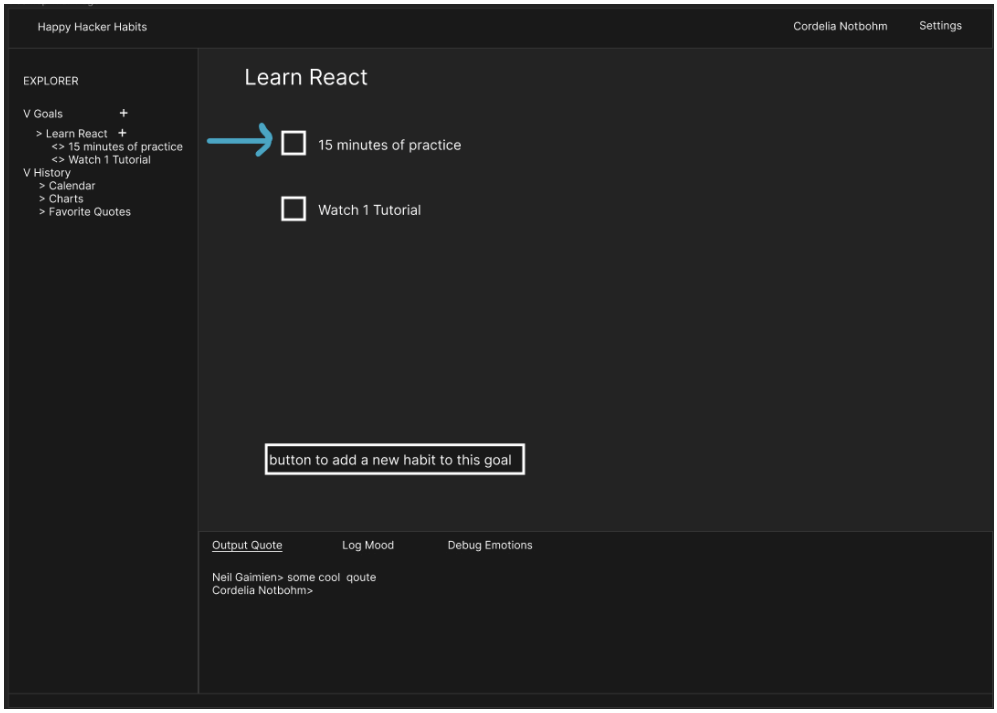


The mobile version has the same steps but with a slightly simplified interface. Only one of the terminal, explorer, or code editor components is open at a time. The user clicks the “explorer” button to bring up the explorer, the “terminal” button to bring up the terminal, and the “code editor” button to bring up the editor.

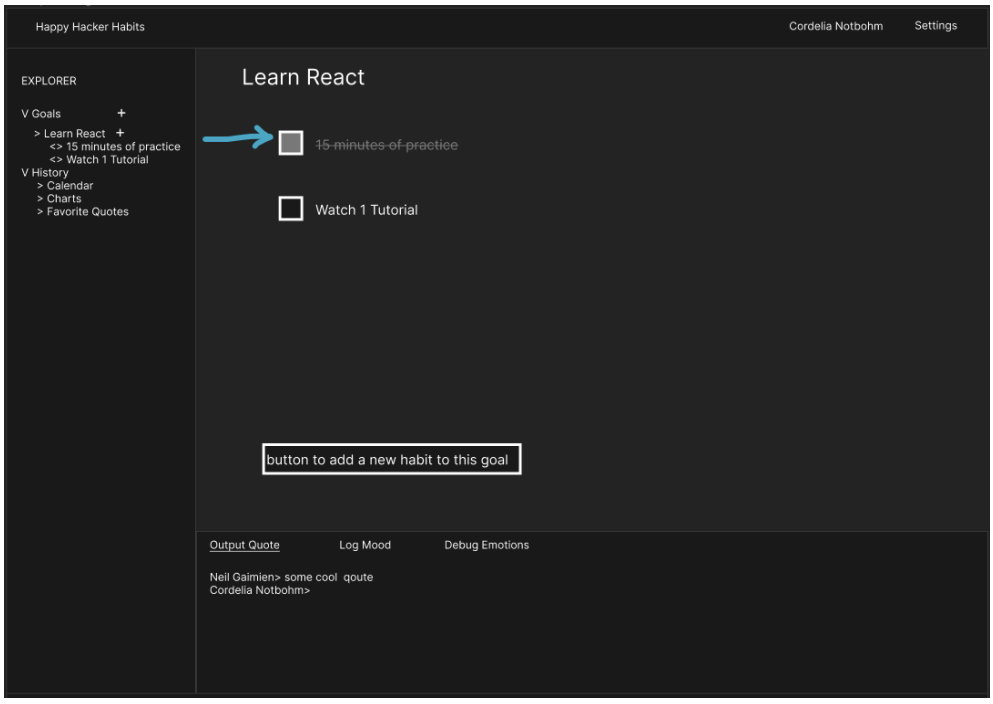


Scenario 2: Commit a Habit

To “commit” a habit for the day the user must first navigate to the page for the goal that the habit is organized under. On that page, there is a checkbox next to each habit for that goal.

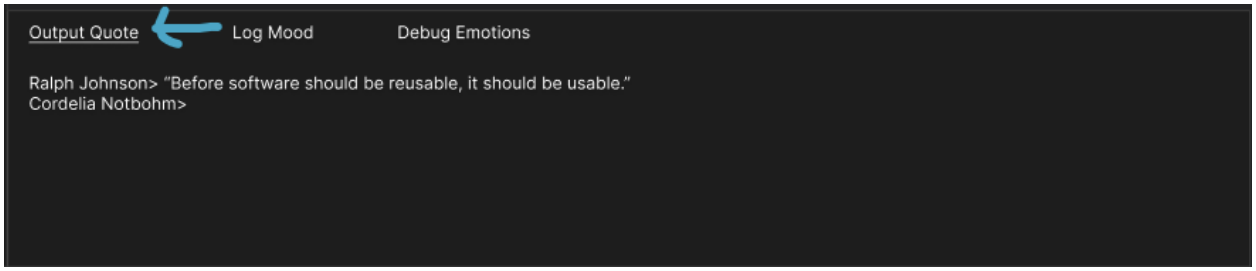


To commit the habit, the user clicks the checkbox or the text to the left of the checkbox. After clicking, the text dulls, a strikeout decoration is added, and the box is changed to signify the habit has been completed for the day.

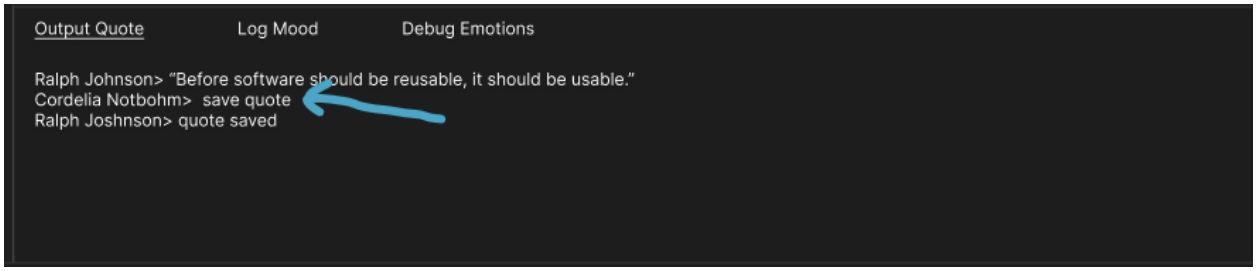


Scenario 3: View Daily Quotes

Happy Hacker shows users a daily technology-related quote. Quotes are shown in the “terminal” component under “output quote.” Output quote is the default view in the terminal. Whenever the terminal returns to this view, the daily quote is displayed on the screen using a transition where each character is added to the screen one at a time to look like the quote is being typed on the screen.

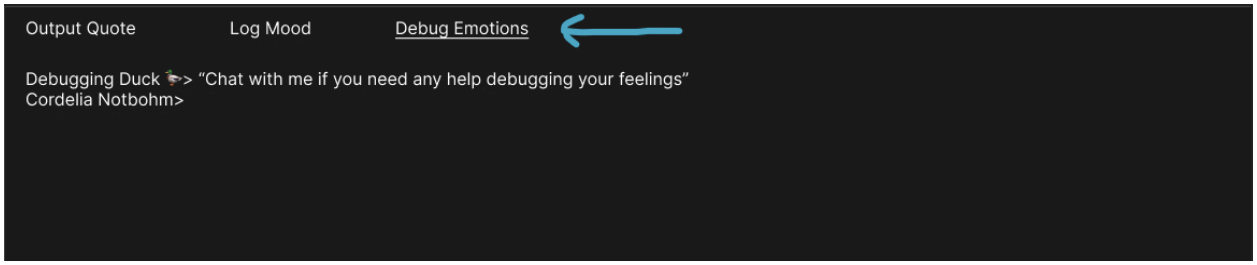


To save the quote as a favorite, the user can type “save quote” and press enter. A message letting the user know they can do this is displayed after the quote. The terminal informs the user when the quote is successfully saved. If the user enters anything else in the quote terminal then nothing happens.



Scenario 4: Chat with AI-Generated Debugging Duck

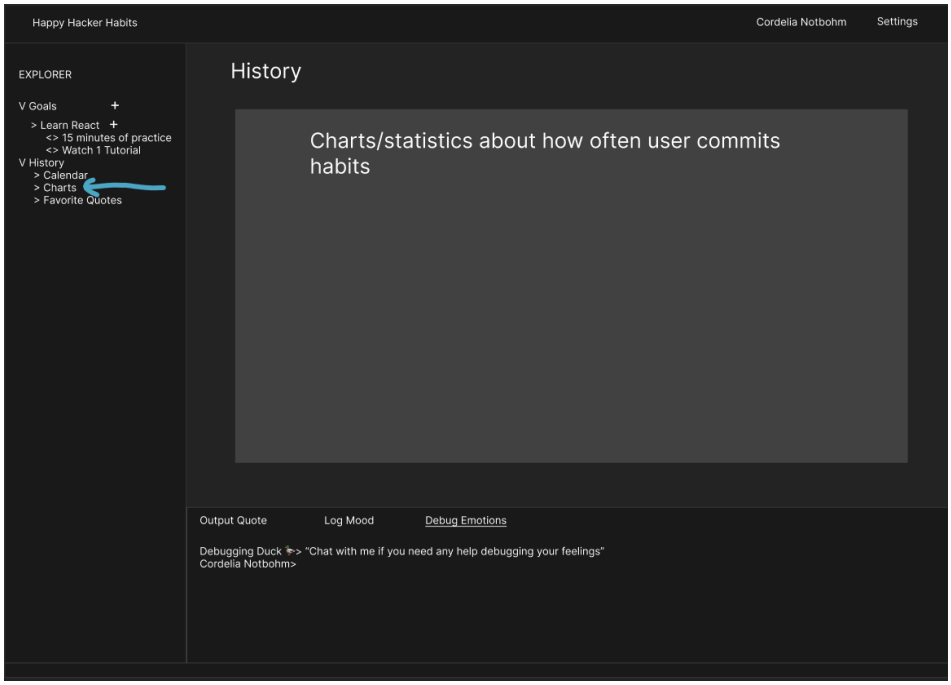
IF we have time, we will implement an AI-generated chat feature, inspired by a debugging duck, to help debug the user’s emotions. To view the chat, the user clicks on the “debug emotions” button at the top of the terminal component, which changes the terminal to chat mode.



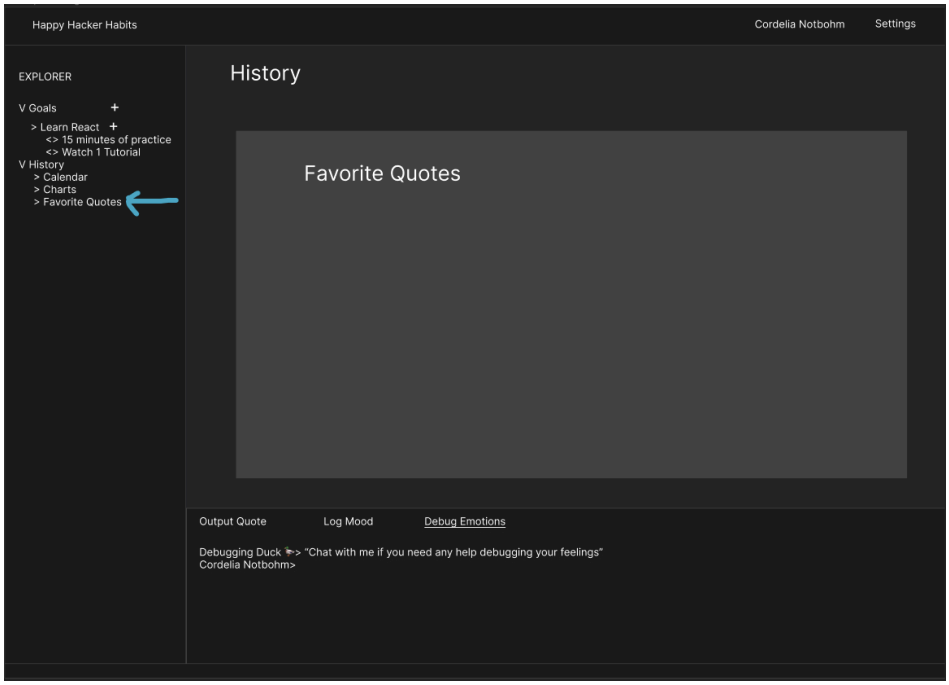
Scenario 5: View History

IF we have time, we will implement a history function that shows users details about how often they complete habits. This history feature could also show quotes that the user chooses to save and show the exact details for a specific day.

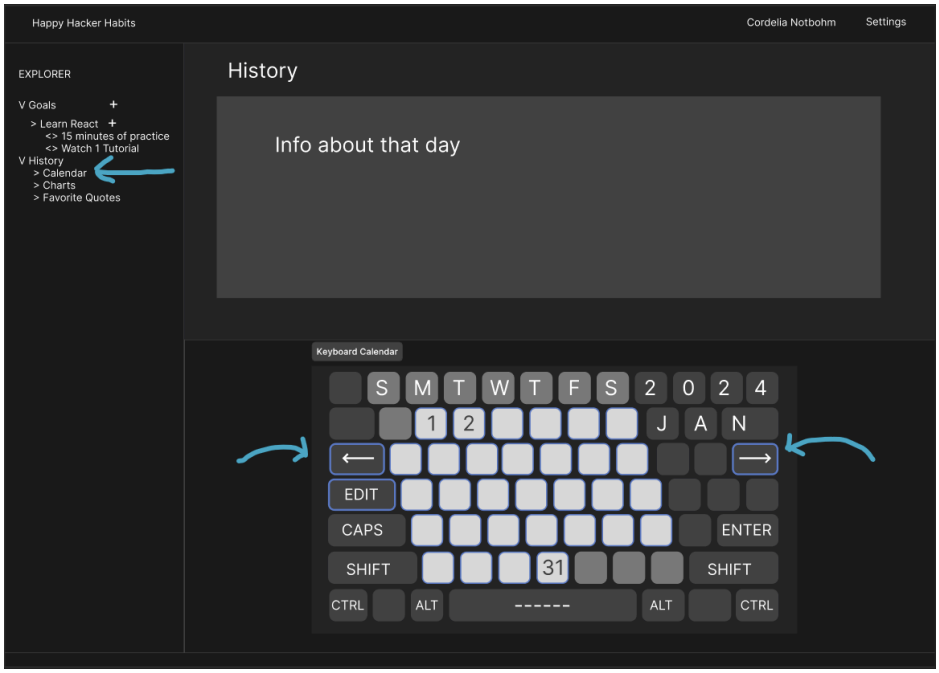
To view the statistics about how often a user completes habits, the user clicks “charts” under “history” in the explorer component. A statistics page then comes up in the editor component.



To view the saved quotes, the user clicks “quotes” under “history” in the explorer component. A quotes page then comes up in the editor component.



To view the history for a particular day, the user clicks “calendar” under “history” in the explorer component. A page for the details of the previous day then comes up in the editor component. The terminal switches to “keyboard mode” where the user can choose what day they want to look at using a calendar that looks like a keyboard. Clicking the button for a date brings up the details of that day.



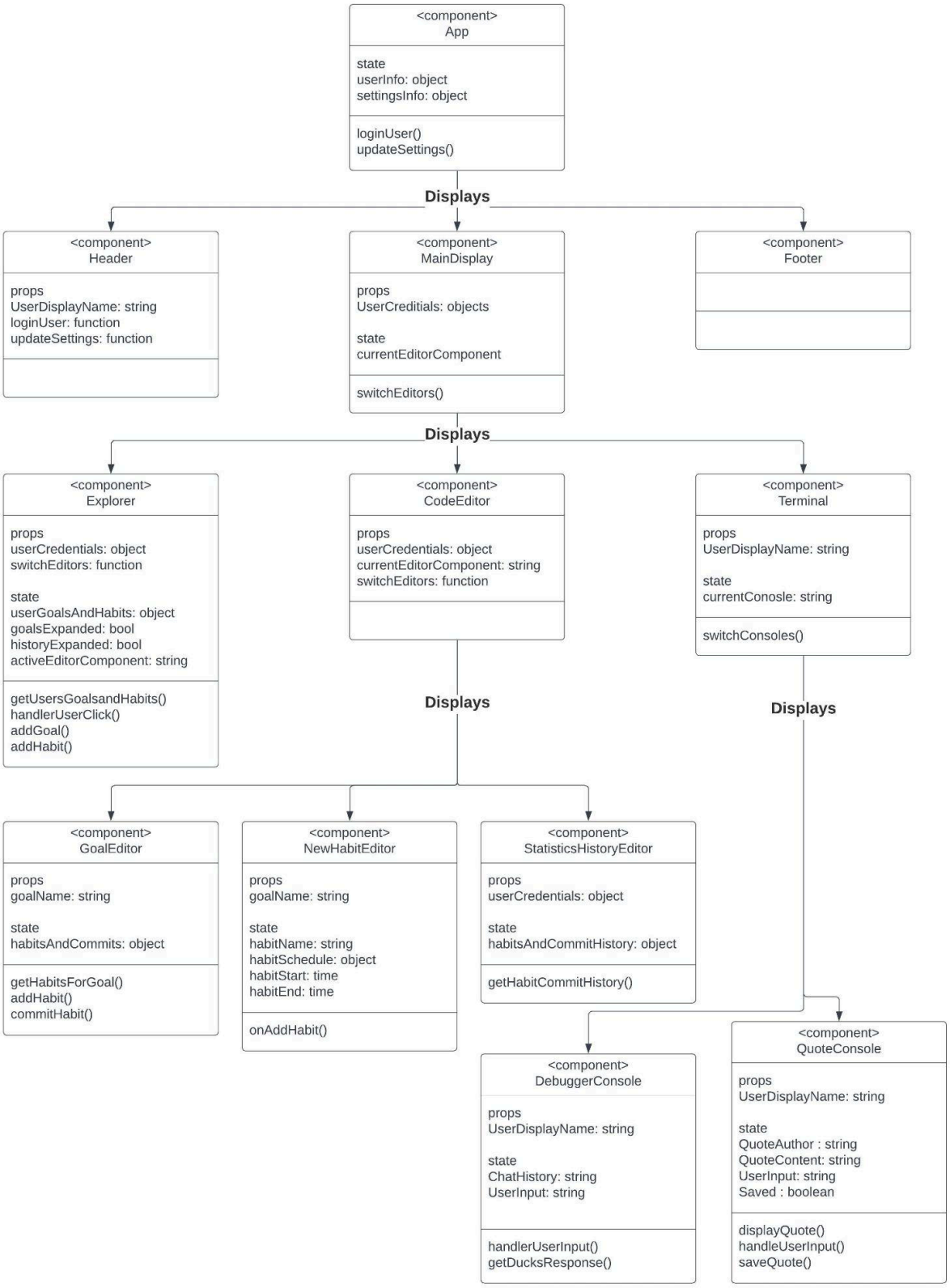
Architecture Diagrams

This section contains the architectural diagrams used to design our application. The diagrams can also be viewed here.

Class Diagram

Figure 1 displays a class diagram with the structure of Happy Hacker's presentation tier. It organizes the app into different logical components that can be developed separately. The figure also considers what props, state, and functionality belong to each component. It is likely that some components will need to be modified once we begin development, but this diagram should be a great starting point for organizing the presentation tier.

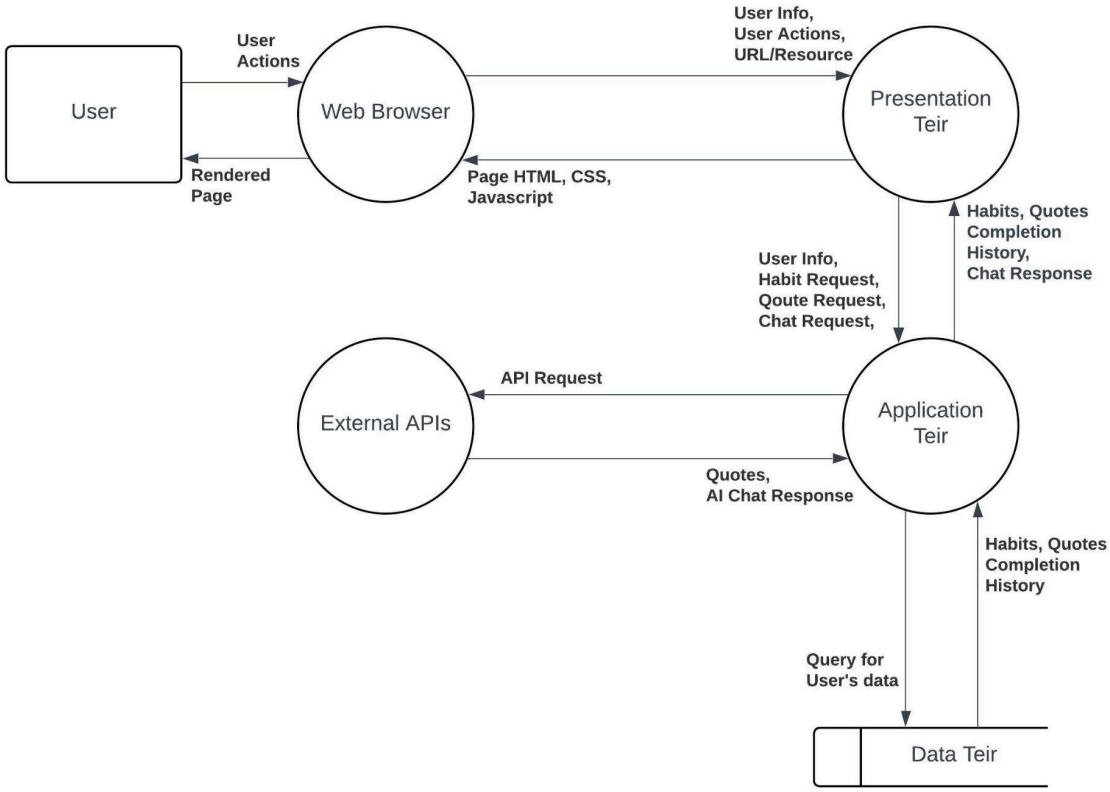
Figure 1: Initial structure of Happy Hacker Habit's presentation tier



Data Flow Diagram

Happy Hacker will use a three-tier architecture, meaning the server is broken into three logical parts. The presentation tier is responsible for generating the HTML, CSS, and JavaScript that the browser can display to the user. The presentation tier calls APIs exposed by the application tier. The application tier responds to API requests, deciding which data to return in a request. The application tier sends API requests to eternal APIs/services that provide useful data for our application. It also sends queries to the data tier to retrieve data stored by our application. The data tier is responsible for storing application data long term. The data flow between these three tiers can be viewed in Figure 2.

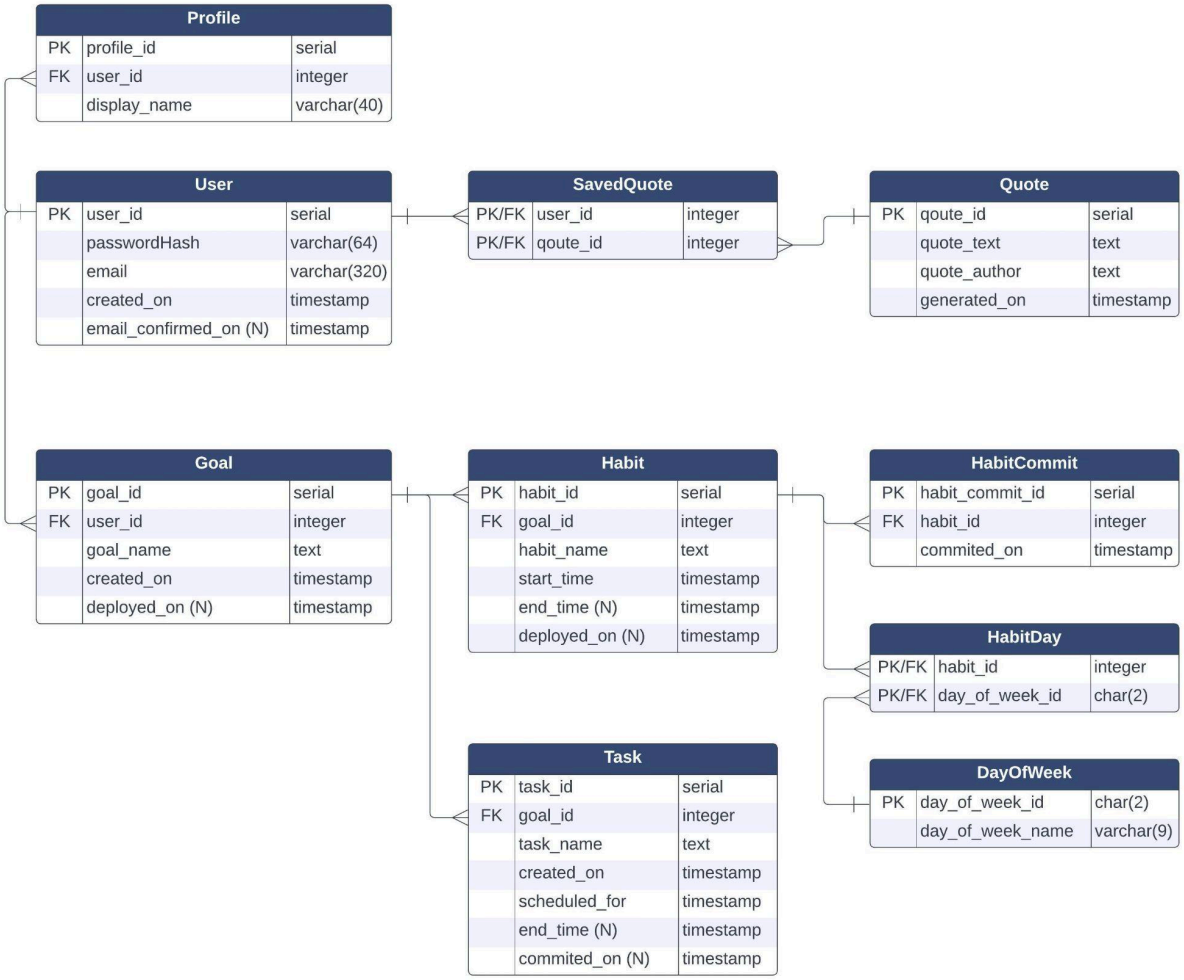
Figure 2: Data Flowing from Happy Hacker Habit's three tiers



Database Schema

Figure 3 displays the schema for the tables and data needed in our application. Note that the schema was designed with growth in mind and supports features like specifying which days of the week a habit repeats or adding tasks (non-repeating items). Some tables displayed here will not be used/needed for the MVP.

Figure 3: Data used in Happy Hacker Habits



Tools and Technologies

This section describes the pros and cons of different technology options for various parts of the application and proposes the tools the developers suggest.

Comparison of Potential Tools and Technologies

Tables 1 through 5 compare the technologies and tools we are considering for the following areas of the project: presentation tier, application tier, data tier, software development tools, and deployment services.

Table 1: Presentation Tier Technologies

Frontend Technology	Pros	Cons
Plain HTML, CSS, Javascript	• Simple, easier to learn and understand, especially for junior developers • Easier to create websites that are screen reader friendly and are optimized for search engines	• Organizing and managing the code without a framework is difficult, making the code harder to maintain • Fewer mechanisms for creating reusable components • Managing state with plain Javascript is difficult • Manipulating the DOM in javascript is less efficient compared to using a framework
React	• A vpopular library, which makes it easier to find support/documentation and valuable to learn for future projects • Has tools to take managing application state easier • Is component-based which makes it easier to write reusable and maintainable code • React's virtual DOM allows it to efficiently update and rerender the browser DOM, increasing performance • Its use of JSX allows writing with HTML-like structure which makes it easier to visualize a components structure	• Slightly more complex than plain javascript which increases project setup time, and makes it slightly harder to learn • Requires a solid understanding of javascript, HTML, and CSS to learn React • Usually results in websites that are harder for screen readers/assistive technologies to parse and are less optimized for search engines

Table 2: Application Tier Technologies

Backend Technology	Pros	Cons
Node.js + Express	• Having both the presentation and application tiers based on javascript should make the integration of the tiers easier • Designed for asynchronous and non-blocking programming, so it is great at handling concurrent connections • It's event-driven architecture makes it great at I/O bound operations, which are the main operations of our server • Has a large number of libraries and modules to help with backend programming • Our developers want more experience with javascript	• The asynchronous nature of javascript can lead to a large number of callbacks that make the code hard to read/maintain. Can be mitigated by using await and promises • Would not be an efficient language for more CPU-bound operations. Not as problematic for our I/O bound server.
Python	• A very versatile language that can used for other domains, like data science and machine learning. Could potentially be more helpful for future projects • Has a large number of libraries and modules to help with backend programming • Python's syntax is very simple and readable, potentially leading to more maintainable code	• Python does support asynchronous programming, but is not as efficient as javascript and it can be harder to learn and work with than asynchronous programming in javascript • Since python is not native to the browser, we would need to switch language for the different tiers, which might be harder for our developers

Backend Technology	Pros	Cons
Firebase	• MongoDB-based database services provided • Could be used for both the application and the data tier, greatly simplifying database management • Provides services for third party authentication, greatly simplifying user account management • Provides simple interface and many SDKs for diff languages, potentially speeding up development	• Would probably require GitHub actions to set up CI/CD integration • Less control over the backend • Most likely not as cost effective as a self-managed backend • Could lead to vender lock in making it harder to move to other services in the future (like AWS services) • Another technology to learn, potentially taking longer for this project since we don't have experience with it
Supabase	• PostgreSQL-based database services provided • Could be used for both the application and the data tier, greatly simplifying database management • Provides services for third party authentication, greatly simplifying user account management • Provides simple interface and many SDKs for diff languages, potentially speeding up development	• Would probably require GitHub actions to set up CI/CD integration • Less control over the backend • Most likely not as cost effective as a self-managed backend • Could lead to vender lock in making it harder to move to other services in the future (like AWS services) • Another technology to learn, potentially taking longer for this project since we don't have experience with it

Table 3: Data Tier Technologies

Data-Tier Technology	Pros	Cons
PostgreSQL (Database)	• One of the most popular open source SQL databases • Is mature, stable, and has a strong community • Supports complex SQL queries, powerful query optimizing, and various indexing options • Is ACID compliant and has great data consistcy/integrity • One of our developers have experience working with PostgreSQL	• Is not as scalable and flexible as NoSQL options, making it hard to support a rapidly growing application • Making schema changes can be difficult and time consuming
MongoDB (Database)	• One of the most popular open-source document based NoSQL databases, especially for Node.js developers • stores JSON-like documents which provides a natural maping to python and javascript objects • Proivides schema flexibility and horizaontal scalability	• Is not ACID compliant, which sacrifies some consitency • Does not have as much support for complex query options as many SQL databases • Our developers do not have any experience with MongoDB, or other document-based databases
Sequalize (ORM)	• Library for node.js that supports multiple SQL databases • Has many advanced features to help with schema creation, query building, transactions, and migrations • Is mature, stable, and has a strong community	• Only has support for SQL databases • Does not have native type-saftey features • Compared to pure-SQL, it has many complex features that can make it hard to learn for developers who are new to ORMs

Data-Tier Technology	Pros	Cons
Mongoose (ORM)	• Library for node.js that allows schema validation and easy query building for MongoDB • Designed specifically for MongoDB which makes it simpler and easier to use than some ORMs	• Only has support for MongoDB, so less applicable to future projects • MongoDB's document flexibility can make maintaining consistent schema designs challenges
Prisma (ORM)	• Library for node.js that allows the creation of schemas and easy query building for both SQL databases and MongoDB • Provides some newer features that older SQL ORMs lack like type-safety and better query optimization • Seems that the industry is moving towards Prisma, so it could be more applicabnle to future projects than Sequalize and Mongoose	• Newer ORM so is not as stable and mature as older options and may lack some advanced features provided by older ORMs • Compared to pure-SQL, it has many complex features that can make it hard to learn for developers who are new to ORMs
NOTE: If we use Supabase or Firebase in the application tier, we won't need a sperate data tier **NOTE:** the ORMs compared here assume the application tier will use Node.js. If a different language is used, we would need to asses other ORMs for those languages.		

Table 4: Software Development Tools

Development Tool	Pros	Cons
Visual Studio Code (IDE)	• Has support for many languages and frameworks making it applicable for future projects • Has many extensions for helpful development tools like integration with version control, peer programming, live previews, calling APIs, Intellisense, and more • Our developers have lots of experience with this IDE	• Is more complex than many IDEs which can lead to a harder learning curve and an overwhelming interface.

Development Tool	Pros	Cons
Reactide (IDE)	• A IDE developed specifically for developing React projects • Integrated tools for managing react components, state, debudding, and live previews, making it easier to refine the UI	• Only has support for React, so learning how to use Reactide does not transfer to as many future projects • Would only be helpful for developing front-end and would need to use a separate IDE for the backend • Our developers are not familiar with Reactide and would need time to learn how to use its features
Git + GitHub (Version Control)	• Large popularity and community meaning it is easy to find help, highly applicable to future projects, and a better platform to showcase projects • Provides integration services with a range of third party CI/CD, code review, and project management tools • Our developers have lots of experience with this hosting platform	• Has limitations on private repositories, and advanced tools for free-teir • Its premium features are more pricey compared to other alternatives like GitLab
Git + GitLab (Version Control)	• An all in one plateform that contains a range of CI/CD, code review, and project management tools • Provides a free self hosted community version that has full access to premium features without the high cost of alternatives like GitHub	• Less popular meaning it can be harder to get support • Its extensive features make it harder to learn than simpler alternatives like GitHub

Table 5: Deployment Services

Deployment Service	Pros	Cons
Netlify	• Simple and easy way to deploy static web apps • Built in CI/CD Integration with git • Has a free option for deployment	• Only supports static files, all other aspects of the app would need to be hosted elsewhere • Does not provide any database services, so a different database hosting service would be required • Cannot deploy docker containers • Has limited customization options compared to web services offered by AWS and Azure
Heroku	• Could host all tiers of the applicaiton with static web app, web services app, and postgres database services • Is simple and easy to use, meaning shouldnt take too long to set up • Built in CI/CD Integration with git • Can be used to deploy containerized apps	• No longer has a free option for deployment • Is more costly then some Azure/AWS options • Has limited customization options compared to web services offered by AWS and Azure

Deployment Service	Pros	Cons
Render	• Could host all tiers of the application with static web app, web services app, and PostgreSQL database services • Is simple and easy to use, meaning shouldnt take too long to set up • Built in CI/CD Integration with git • Can be used to deploy containerized apps • Has a free option for deployment	• If the application needed to scale beyond the free tier its pricing model is more costly then some Azure/AWS options • Has limited customization options compared to web services offered by AWS and Azure • Free tier can spin down a web service after inactivity, and can suspend a web service after using allotted resources, so we would need to make sure to wake it up before the presentation • Free tier postgres will only hold data for 90 days, then the database will removed unless you upgraded it
AWS EC2/Beanstalk	• Could host all tiers of the application with static web app, web services app, and many different database services • Provides as much or as little control of your application as you want • Can connect to a wide range of AWS services like database and loging services • The most popular cloud provider meaning there is a large community and extensive documentation • Very applicable to future projects and would be a great provider for our development team to learn about • Can be used to deploy containerized apps	• Would probably require GitHub actions to set up CI/CD integration • Would probably require dockerizing the react app to easily deploy it on AWS • Easier to accidentally use more resources than the free tier provides and get charged more money than we anticipate for the service • Our developers do not have much experience with dockerizing applications nor deploying on AWS beanstalk/container hosting services, so this would take more time

Proposed Tools and Technologies

Considering the above comparisons, the development team proposes that we use the PERN stack, specifically the following technologies:

- React for the Presentation tier. Since the presentation tier is our main focus for this project, we would like to learn about a new Javascript library. If we struggle to work with React, our backup is to use plain old HTML, CSS, and Javascript.
- Node.js + Express for the application tier. Since our developers want more experience with javascript we would like to use javascript for both the presentation and application tiers. If we struggle to create a backend with Node.js, our backup is to try using Supabase.
- PostgreSQL and Prisma for our data tier. While it would be great to get experience with a new database, we do not feel it is worth experimenting with a new database for this project as the data tier is not our main focus. If we struggle to work with Prisma, our backup is to use plain old SQL queries to interact with our database.
- Visual Studio Code as our IDE, and Git + Github as our version control. We do not feel it is worth experimenting with a new IDE or repository hosting service for this project since we will already be experimenting with React and Node.js.
- Render as our cloud deployment provider. We would like all tiers of the system to live on one cloud provider and don't want to spend any money. While it would be great to dockerize, set up GitHub actions, and deploy on AWS, we probably don't have time to do it well. In the future, it would be great to explore moving to AWS.

Data Providers

This section describes the data providers/APIs that will help us build our project, with emphasis on what features the API will help with.

Quotable API

Quoable provides APIs for searching for quotes and retrieving a random quote that matches certain parameters. The documentation for this API can be viewed here.

This API will help support our daily quote feature. To generate a daily technology-related quote, we can use the random quote endpoint and use the tag "technology." All quotes will be stored in our database with a timestamp of when we retrieved them. We can then check the database before generating a quote so that there is only one quote generated per day.

Gemini API

Gemini (formally called Bard) is Google's AI Model. It has a developer API with a free plan that allows 60 requests per minute. It provides SDKs for different functionalities like generating text

from a text prompt or building multi-turn conversations. The documentation for this API can be viewed here.

This API will help support our debugging duck chat features. To allow users to have a debugging conversation about their feelings, we can use the multi-turn conversation functionality so that Gemini will store the chat history context needed to maintain the conversation. Then we won't need to store any chat history or create code to facilitate generating a conversation.

60 requests per minute won't scale well, but since it is the only free AI offering we found, it will work great for testing and development. In the future, we will research other AI model APIs to determine which one would work best for our needs.

Other Resources

This section contains a description of the other resources we plan to use, including the color palette, fonts, and any images that will be present in our application.

Color Palette

We plan to use the following color palette as our main 6 colors. The background colors will be used for the application's background. The accent colors will be used for buttons and text, in a way that looks similar to a code editor.

Fonts

We plan to use Consolas as our main font, as it looks similar to text in a code editor.

```
This is what the font looks like.

This is what the font looks like.

This is what the font looks like.

This is what the font looks like.
```

Emojis

We plan to use the following emojis, in place of finding other assets like pictures/icons:

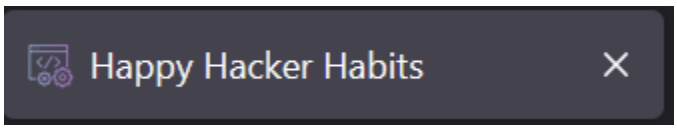
🗃️ 📁 👩‍💻 💻 🖥️ ⌨️ 💾

💿 🖲️ 🖱️ 🦆 ✉️ 🖊️ 🗑️ ⚙️

Emojis are in the creative commons, so are free to use. Emojis can be placed in components as text, which should speed up development. A disadvantage of using emojis is that they do not look the same in all browsers, since some browsers render emojis differently. In the future, we would replace the emojis with consistent icons, but for now, we want to focus on developing the app's functionality.

Images/Icons

We plan to use the following icon as the main image for our app. It will also be the app's favicon.



Development Timeline

Table 6 describes the major tasks and milestones of the project. It also contains estimated deadlines for those milestones.

Table 6: Major development tasks and milestones

Week	Tasks	Milestones
Week 6	Generate ideas about the project. All ideas are good ideas. Look into what existing services might help with those ideas. Research technology options. With a focus on what key technologies we want to get experience with and what other options work well with those key technologies. Outline key user scenarios and exactly how the user will interact with our system. Wireframe what the user scenario interactions and UI design will look like. Diagram and design the Presentation, Application, and Data tier architectures that support our scenarios.	2/4/23 - Initial project motivation, technology decisions, and other resource decisions rough-drafted 2/9/23 - All user scenarios, wireframes, and design diagrams rough-drafted 2/11/23 - Proposal Finished

Week	Tasks	Milestones
Week 7	Get the proposal checked by our stakeholders/clients (Ethan). Ensure our ideas are reasonable, check that our technical decisions seem viable, and get some advice about any aspects of the project we are unsure about. Make any needed final edits to the proposal. Start the presentation tier by working on the navigation bar that moves to different pages and can be used to create goals/habits. Also, work on any other pop-up pages that are required for creating new goals and habits. Start working on the main page that displays habits and allows you to complete a daily habit. Complete the data tier by using an ORM to generate the database with the proper schema. If an ORM ends up being too complicated, create a database build script with plain SQL. Start the application tier by creating APIs that can be used to persist and retrieve user data to the database.	**2/14/23 - Proposal Due** 2/15/23 - Page components for creating and viewing habits exist. (doesn't need to look perfect yet but are functional) 2/15/23 - finish an ORM schema or a script for generating the database 2/18/23 - All APIs for accessing database data exist 2/18/23 - Page components for main habit pages and habit completion are partially implemented. (doesn't need to look perfect yet but are planned out)

Week	Tasks	Milestones
Week 8	Finish the application tier by creating APIs for the client to use to call the third-party quote API we are using. Continue the presentation tier by finishing the main page that displays habits and allows you to complete a daily habit. Continue the presentation tier by working on the “terminal” component that shows off quotes and can navigate to other “future” features. Finish the presentation tier by improving the styling of the pages, and make the front end look polished. Begin integrating presentation and application tiers with the daily quote feature (this is the simplest interaction so it should be the best feature to get the two tiers talking).	2/22/23 - Page components for completing a habit for the day finished (doesn’t need to look perfect yet but are functional) 2/22/23 - Page components for terminal and daily quotes finished (doesn’t need to look perfect yet but are functional) 2/25/23 - all presentation tier pages have been fully styled 2/25/23 - All tiers finished and working independently 2/25/23 - Daily quote feature fully functional from presentation to data tier
Week 9	Complete integrating the presentation and application tiers by having the presentation tier call application tier API’s that modify and retrieve data from the application instead of using hard-coded data for habit creation and completion. Deploy the application to a hosting service, so that it is publicly accessible. Test the application in the hosting service and debug everything that breaks. Make presentation slides for the presentation. Plan who will say what. Plan out exactly how we will demo our project (create a demo script with the order of features we will show off).	2/29/23 - All tiers are completely integrated. For habit creation and competition especially 3/1/23 - MVP Deployed, publicly accessible, and functional 3/1/23 - Presentation Prepared (all documents and roles are ready and we could reasonably present and be fine)

Week	Tasks	Milestones
Week 10	Practice presentation. Optimally meet in person, at the school before class on 3/4/23 to run through the presentation and demo so we are confident and ready to go for presentation day. Debug any minor bugs we are aware of but have not fixed yet. (Assuming we aren't behind and have a functional MPV) Start implementing wants: AI-generated chat duck, More complicated history UI that displays cool stats, and emotion logging.	**3/4/23 - Presentation Due** 3/7/23 - The application is in great working shape, we are happy/proud, and have submitted the code. Have a pizza party to celebrate. 🦆 **3/8/23 - Implementation Due**