Programmer's Manual

for

Edutale

Version 1.0 approved

Prepared by Cheyenne Ty, Abigail Penland, Tommy Le

Team Plasma

December 11, 2024

1. Introduction

1.1 Purpose

The purpose of this document is to define the structure of the Edutale application, as well as assist programmers in the maintenance and development of Edutale. Programming practices, essential frameworks, and the flow of control in Edutale will be defined in this document.

1.2 Scope

The Edutale application allows Computer Science students to keep track of their progress in developing skills related to the job/career that they are pursuing. The student can state their ideal career, and the application will pull the top skills that are needed for those careers. Top skills will be added to the user's account, with a progress measurement associated with each skill. Progress will be measured based on the completion of different courses, certifications, projects, etc. that the application recommends to the user, which are internally called quests. A global progress bar in the form of a leveling system will also be present, and hitting certain level milestones awards the user equipment to customize their appearance. The application will support progress check-ins on a regular basis and allow users to see their current progress through quests. Ultimately, users will be able to include the skills they have built up in their resume.

The current state of Edutale is a prototype that fulfills most goals for the scope of the future application with few exceptions. Rather than choosing any career, the prototype is limited to the career Computer Science. This was at the request of the client to ensure that development focused on reaching the educational objectives of Edutale with a game-like immersion. The prototype develops the framework for many careers to be added in the future.

2. Getting Started

2.1 Overview

This section details instructions for downloading necessary prerequisites, the Edutale application, and building the application itself. These instructions should be done in order. You will need approximately 2GB of free storage on your computer for the Edutale application and all required software.

2.2 Installing Visual Studio Code

Visual Studio Code (VSC) is an integrated development environment (IDE) that allows for the Edutale application to be built and run.

1) Navigate to the Visual Studio Code website: https://code.visualstudio.com/
2) Click on the button that says "Download for Windows". This should start downloading an installer file onto your computer.
3) After the download is complete, run the Visual Studio Code installer. Keep all default options.
4) Visual Studio Code is now installed onto your computer.

2.3 Installing Node.js

Node.js is a JavaScript runtime environment (JRE) that allows for web-based packages used by Edutale to render.

1) Navigate to the Node.js website: https://nodejs.org/en
2) Click on the button that says "Download Node.js (LTS)". This should start downloading an installer file onto your computer.
3) After the download is complete, run the Node.js installer. Keep all default options.
4) Node.js is now installed onto your computer.

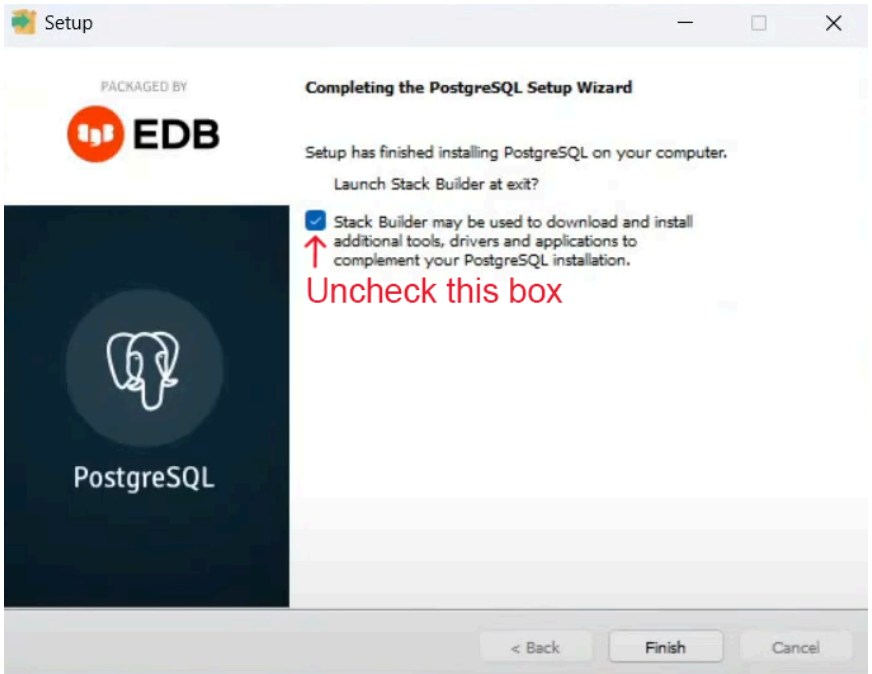
2.4 Installing PostgreSQL and pgAdmin 4

PostgreSQL and pgAdmin 4 are applications that store the backend of Edutale. PostgreSQL is the main software that stores the database, while pgAdmin 4 allows for the creation and maintenance of the database.

1) Navigate to the PostgreSQL installers page: https://www.enterprisedb.com/downloads/postgres-postgresql-downloads

2) On this installers page, click on the download icon for PostgreSQL version 17.1 (or above). This should start downloading an installer file onto your computer.

3) After the download is complete, run the PostgreSQL installer. Keep all default options selected.
    a) When asked for a password, be sure to save this password somewhere convenient. This password will be used later when setting up the backend for Edutale. Although this password will only be used internally by the Edutale application, it is still recommended to use a strong password.
    b) After setting a password, follow the rest of the PostgreSQL installation instructions with all default options selected.
4) After the installation is complete, uncheck the Stack Builder checkbox and then click on "Finish".



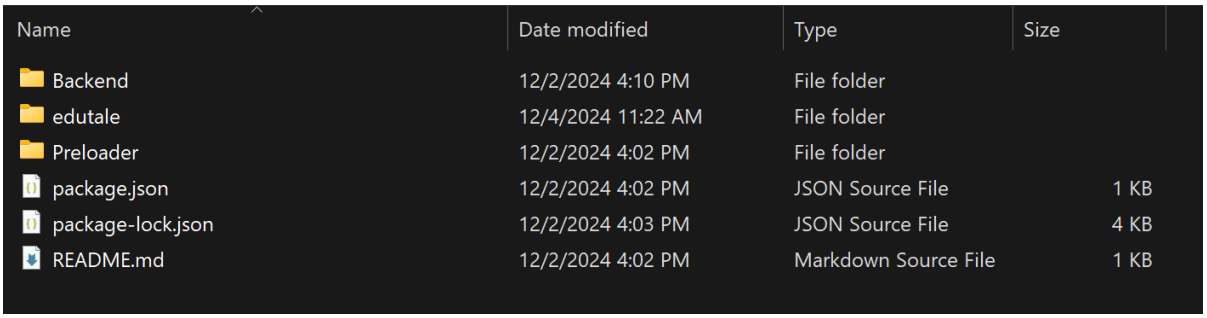
5) PostgreSQL and pgAdmin 4 are now installed onto your computer.

2.5 Downloading Edutale

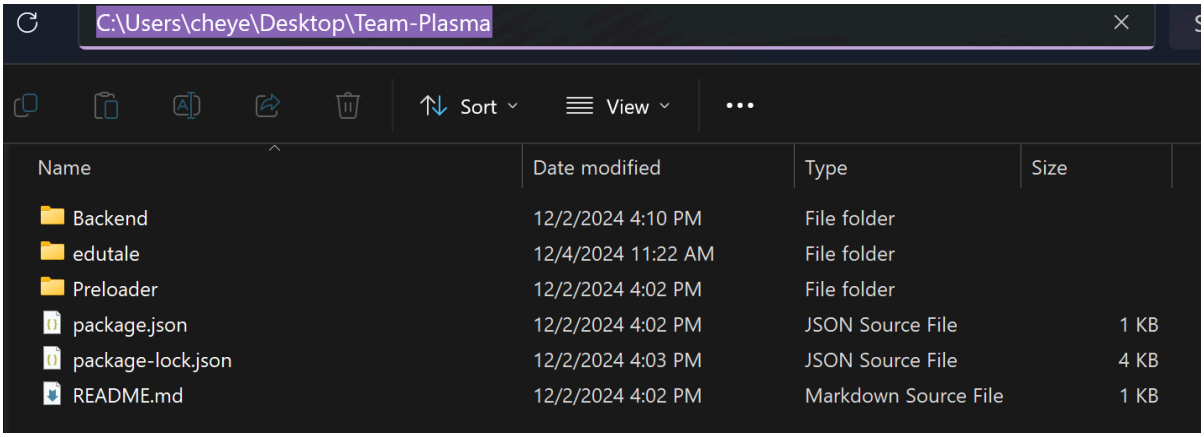
The Edutale files can be accessed through the Edutale Github repository: https://github.com/Edutale/Team-Plasma. You do not need to have a Github account nor Git installed to obtain these files. If you already have the ZIP file downloaded, please begin this section at step 4.

1) Navigate to the Edutale Github repository: https://github.com/Edutale/Team-Plasma.
2) Click on the green "Code" button.
3) In the dropdown menu, click on "Download ZIP".
4) After the ZIP file has been downloaded, navigate to your computer's Downloads folder and unzip the file. Make sure that all unzipped files are within their own folder.

For future steps, the **root** of the project will be the path to the folder that has the following contents:



Since this root path will be different for every user, we recommend saving the root path somewhere convenient. For example, the highlighted portion of the following image would be the **root** path for an example user, since they see the correct contents:

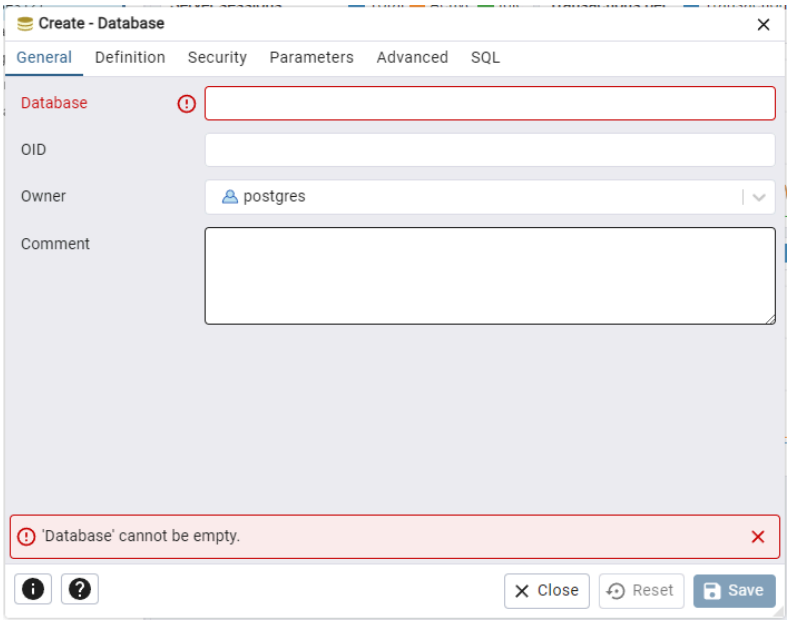


When extracting the ZIP file for the project, the folder containing all the project files may be called `Team-Plasma` or `Team-Plasma-main`. Either name is appropriate, and no setup/installation steps will differ between the two names.

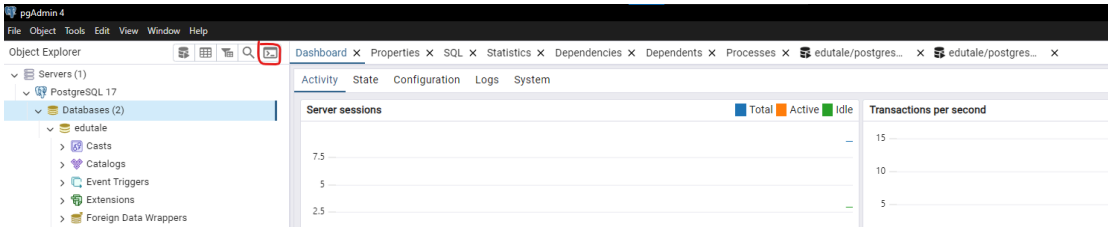
2.6 Setting up Edutale's Backend in pgAdmin 4

Some Edutale files need to be run within pgAdmin 4 to create the database for the application.

1) Open the pgAdmin 4 application on your computer.
2) Upon opening, you will be asked to set a master password for pgAdmin. After entering your password, click on the blue "✓ Ok" button.
    a) This is necessary to secure and unlock other credentials used within the application. Be sure to save this password somewhere convenient. Although this password will only be used internally by the Edutale application, it is still recommended to use a strong password.
3) In the left side of the application, you will see a dropdown menu arrow followed by the header "Servers". Click on the dropdown arrow.
4) You will then be asked to enter the password to connect to the PostgreSQL server. Enter the same password and then click on the blue "✓ Ok" button.
5) Click on the dropdown menu followed by the header "PostgreSQL 17" (17 could be changed to whatever version number was installed on the device).
6) Right click on the header "Databases".
7) This will popup a new menu. From this, click on the option "Create". An additional popup will appear. From this, click on the option "Database…".
8) You will be presented with many options to create your database. For the Edutale application, you will only need to input the database name in the input box highlighted red (pictured below). Input the word "edutale", all lowercase and without the quotations. Then, click on the blue "Save" button.



9) There should now be a subheader called "edutale" under the "Databases" header. Click on the "edutale" header on the left sidebar.
10) In the top left of the menus, click on the PSQL Tool (circled in red below). A command-line interface should be displayed on the right side of the application.



11) At the top-level folder of the project, At the prompt line `edutale=#`, type in the following command:

    `\i` **`root`**`/Backend/src/config/main.sql;`

    Below is an example of what this command can look like. On this machine, the **root** path is `C:/Users/cheye/Desktop/Team-Plasma`. Ensure that all slashes in the file path are forward slashes; the beginning `\i` should be the only place that has a backslash.

    ```
    \i C:/Users/cheye/Desktop/Team-Plasma/Backend/src/config/main.sql;
    ```

    a) Note: If this is the first time you are setting up the Edutale database, you may see an error that looks similar to the following:

    ```
    edutale=# \i C:/Users/cheye/Desktop/Team-Plasma/Backend/src/config/main.sql;
    psql:C:/Users/cheye/Desktop/Team-Plasma/Backend/src/config/drop.sql:3: ERROR:  table "student_skill" does not exist
    psql:C:/Users/cheye/Desktop/Team-Plasma/Backend/src/config/drop.sql:4: ERROR:  table "student_progress" does not exist
    psql:C:/Users/cheye/Desktop/Team-Plasma/Backend/src/config/drop.sql:5: ERROR:  table "career_skill" does not exist
    psql:C:/Users/cheye/Desktop/Team-Plasma/Backend/src/config/drop.sql:6: ERROR:  table "student_quest" does not exist
    psql:C:/Users/cheye/Desktop/Team-Plasma/Backend/src/config/drop.sql:7: ERROR:  table "skill_quest" does not exist
    psql:C:/Users/cheye/Desktop/Team-Plasma/Backend/src/config/drop.sql:8: ERROR:  table "quest_resources" does not exist
    psql:C:/Users/cheye/Desktop/Team-Plasma/Backend/src/config/drop.sql:9: ERROR:  table "student_inventory" does not exist
    psql:C:/Users/cheye/Desktop/Team-Plasma/Backend/src/config/drop.sql:10: ERROR:  table "student_career" does not exist
    psql:C:/Users/cheye/Desktop/Team-Plasma/Backend/src/config/drop.sql:11: ERROR:  table "student" does not exist
    psql:C:/Users/cheye/Desktop/Team-Plasma/Backend/src/config/drop.sql:12: ERROR:  table "skill" does not exist
    psql:C:/Users/cheye/Desktop/Team-Plasma/Backend/src/config/drop.sql:13: ERROR:  table "career" does not exist
    psql:C:/Users/cheye/Desktop/Team-Plasma/Backend/src/config/drop.sql:14: ERROR:  table "quest" does not exist
    psql:C:/Users/cheye/Desktop/Team-Plasma/Backend/src/config/drop.sql:15: ERROR:  table "resources" does not exist
    psql:C:/Users/cheye/Desktop/Team-Plasma/Backend/src/config/drop.sql:16: ERROR:  table "inventory" does not exist
    ```

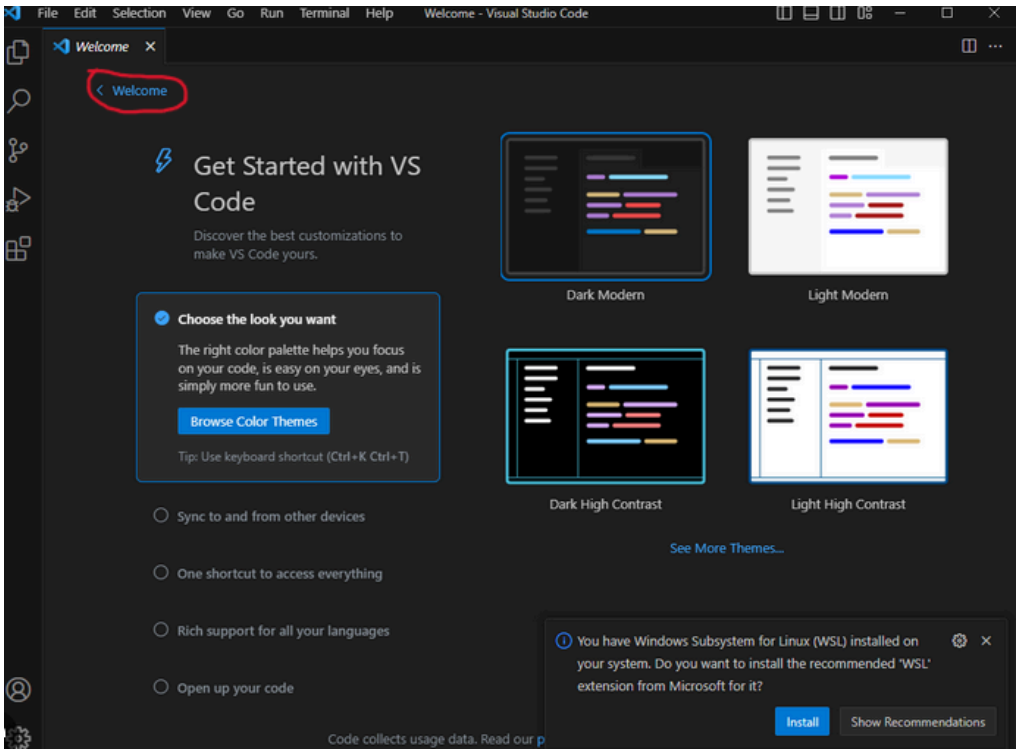
    These error messages are normal, and only appear because this is the first time the Edutale database is being set up. You can safely disregard these messages. If the `main.sql` file is run again in the future, these error messages will no longer appear.

2.7 Setting up Edutale's Frontend in Visual Studio Code

In order to set up the frontend of Edutale, we will use the terminal within Visual Studio Code. We will also set up a crucial file in order to allow the frontend of Edutale to communicate with the backend database we set up in Section 2.6 (Setting up Edutale's Backend in pgAdmin 4).

1) Open Visual Studio Code.
2) At the center of the screen, click on the "Open Folder…" option. This should bring up your file explorer.

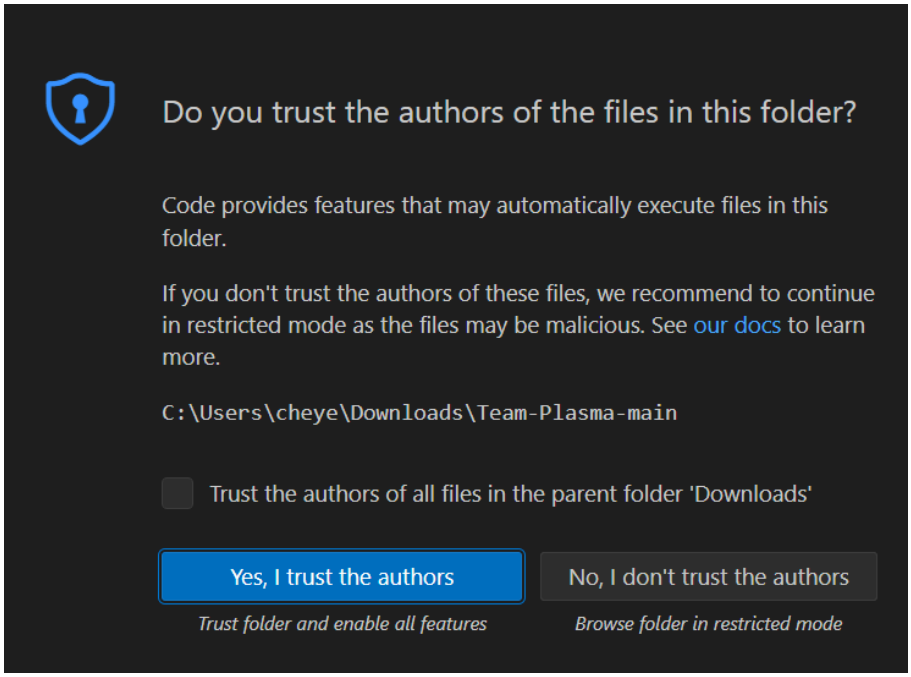
a) If the "Open Folder…" button is not at the center of the screen, look at the top left corner for a "< Welcome" button. If you can see this button, click it to go back to the Welcome screen.



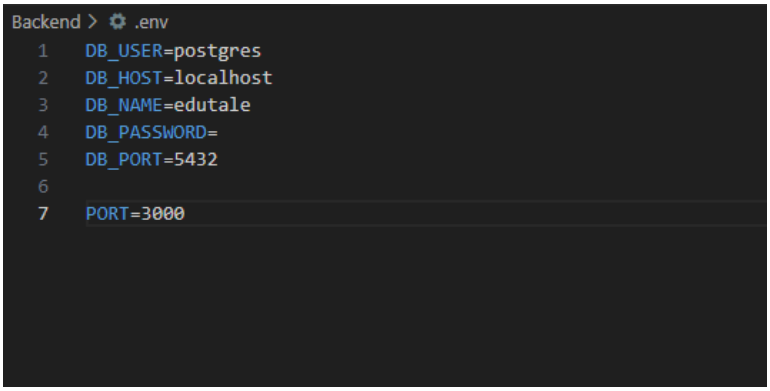
3) Find and click on your **root** folder which is where you extracted the ZIP file to. This would be named either `Team-Plasma` or `Team-Plasma-main`. VSC should then open up a menu on the left side of the screen that displays all of the folders and files seen within the **root** folder.

a) VSC may ask you if you trust the files in the opened folder. If prompted, click "Yes, I trust the authors", but leave the checkbox empty. If you are concerned

about the security of this project, please refer to Section 1.3 (Security).

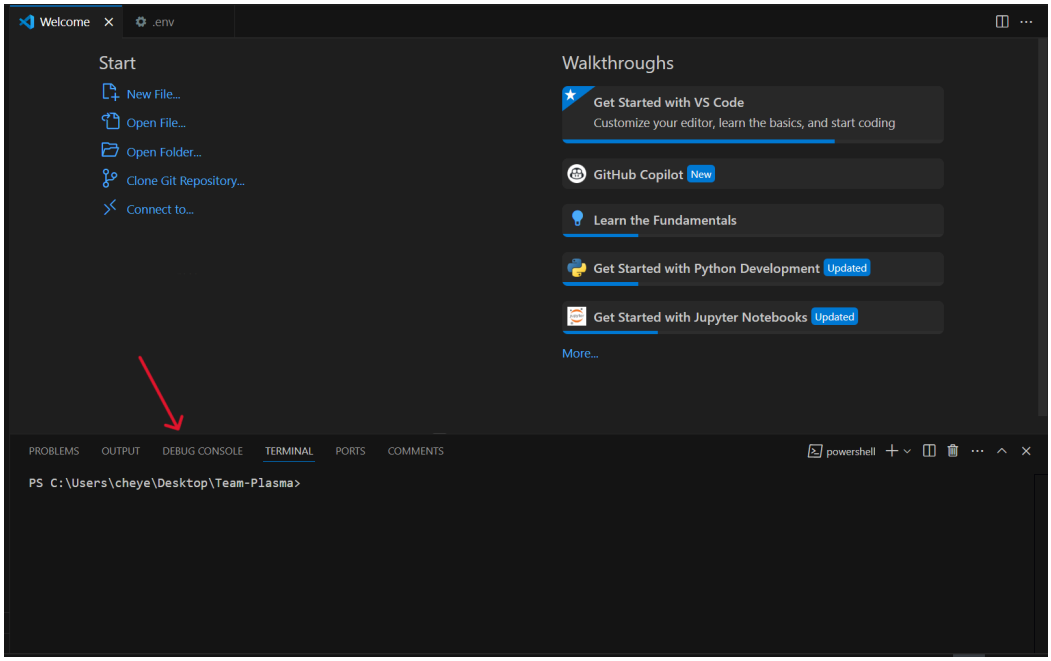


4) Open the "Backend" folder. While this folder is selected, click on the "New File" button to create a new file in the "Backend" folder. Name this file `.env`.
5) Enter the following inside of the `.env` file. For the value for `DB_PASSWORD=`, put the master password you created in Section 2.6 (Setting up Edutale's Backend in pgAdmin 4).



6) At the bottom of the VSC window, there should be a Terminal tab. If this is not on your screen, press Ctrl + ` to bring up the Terminal (the ` button is often located

directly above the Tab button on a keyboard).



7) Essential Node.js packages must be installed in order to successfully set up Edutale. To do this, specific commands must be run within the terminal. To install these packages, enter the following commands in the order that they are listed. Be sure to enter the next command when you are *sure* the current command has finished; you will know when you can enter the next command when `PS root>` appears again in the terminal.
    a) `cd Backend`
    b) `npm install`
    c) `cd ../edutale`
    d) `npm install`
    e) `cd ../`
    f) `npm install`

    At any point while installing the packages, if vulnerabilities are found please run the following command to fix the vulnerabilities:

    ```
    npm audit fix
    ```

    Below is an example of this message. You can continue with the steps above as

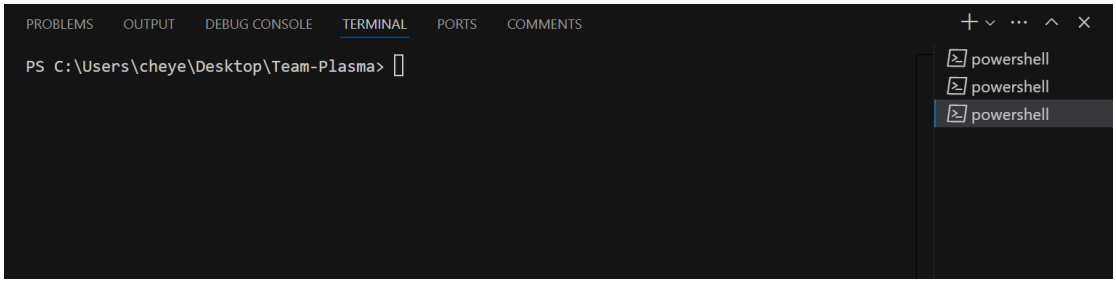
normal.

```
PS C:\Users\cheye\Desktop\Team-Plasma-main\edutale> npm install

added 331 packages, and audited 332 packages in 9s
112 packages are looking for funding
  run `npm fund` for details

3 vulnerabilities (2 low, 1 high)

To address all issues, run:
  npm audit fix
Run `npm audit` for details.
```

3. Running Edutale

This section details the steps done to run the Edutale application on your machine. Ensure that you have followed all of the instructions in Section 2 (Getting Started) before attempting to run Edutale. Any time you want to run Edutale again, you must complete all of the following steps again.

1) Open Visual Studio Code.
2) On the center of the screen, click on the "Open Folder…" option. This should bring up your file explorer.
3) Find and click on your **root** folder. VSC should then open up a menu on the left side of the screen that displays all of the folders and files seen within the **root** folder.
4) If there is not one already, open a Terminal at the bottom of the VSC window by pressing Ctrl + `.
5) At the right side of the Terminal window, there should be a "+" button. This button opens more Terminal tabs. Click the "+" button until there are a total of three tabs. The Terminal tab should now look similar to the following:

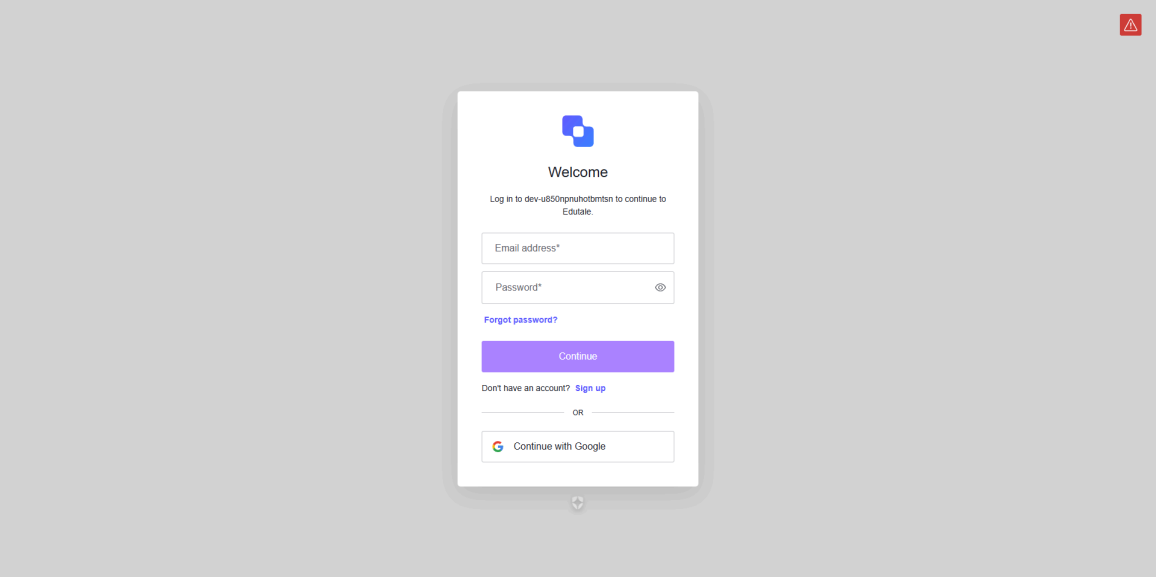


6) Now, you must enter a set of commands for *each* terminal. It is okay to change the order in which each terminal is set up, but the instructions within *each* terminal must be followed in the order that they appear.

a) First terminal:
```
cd Backend
npm run dev
```
b) Second terminal:
```
cd edutale
npm run dev
```
c) Third terminal:
```
cd edutale
npx http-server --cors
```

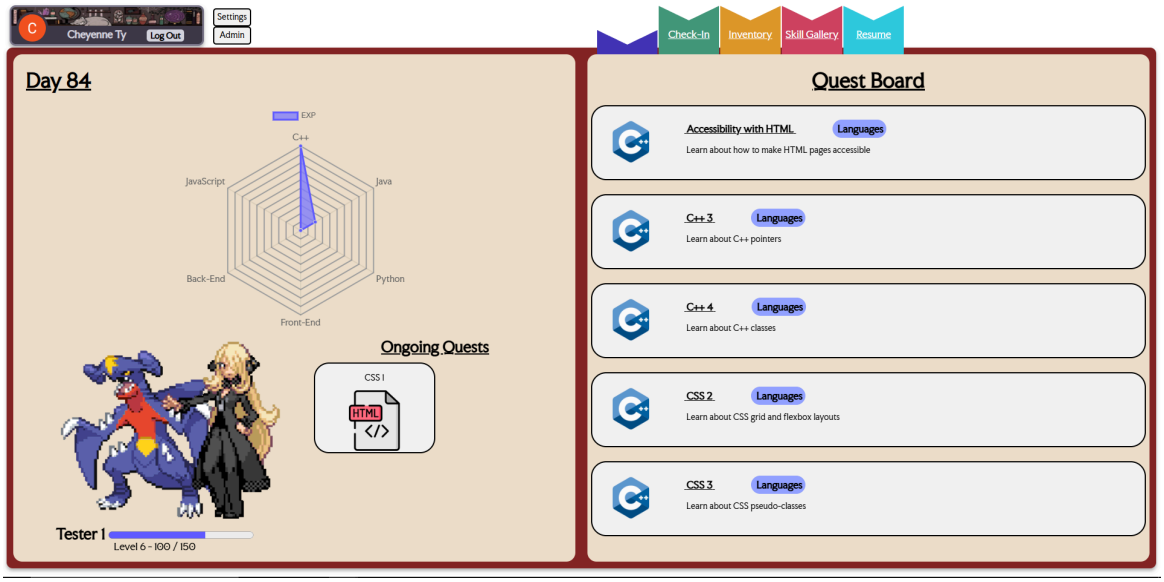
7) After all three terminal windows are set up, you can now access the Edutale application using a web browser of your choice. Within your web browser's address bar, type in `http://localhost:5173` and press enter.
8) If all steps have been successfully completed, you should now see the Welcome page:

9) Click on "Connect". You will be brought to the screen pictured below. We recommend using the "Continue with Google" option.



a) Note: If you have run Edutale before, you may already be logged in. In this case, you will see the Mainpage instead.

10) After authenticating, you should now see the Mainpage of Edutale, similar to what is seen below. This means that Edutale has been successfully run.



4. File Descriptions

Due to the significant number of subfolders within the Edutale application, some sections below describe the files within a folder as well as the files within its subfolders.

4.1 Backend Folder

4.1.1 Setup SQL Scripts

These scripts are run when the Edutale backend is initialized for the first time. These scripts include the database schema as well as any PL/SQL procedures or functions used within other scripts or the API.

4.1.1.1 `main.sql` - Master script that runs all other setup scripts for Edutale, setting the database up to its default prototype state. This script also runs `drop.sql`, which will destroy the database in order to repopulate it with testing data.

4.1.1.2 `schema.sql` - Script that defines the database schema.

4.1.1.3 `test_plan_data.sql` - Script that populates the database with the data required to carry out the Test Plan for Edutale. This data can also be treated as prototype data.

4.1.1.4 `drop.sql` - Script that drops all tables within the Edutale database.

4.1.1.5 `accept_quest.sql` - PL/SQL procedure that handles the acceptance of quests within the application.

4.1.1.6 `buy_item.sql` - PL/SQL procedure that handles buying items within the application.

4.1.1.7 `calc_quest_exp.sql` - PL/SQL function that calculates the amount of EXP a quest will give, given the quest's difficulty level.

4.1.1.8 `checkin_submit.sql` - PL/SQL procedure that handles all updates to the database when a student submits a Check-In.

4.1.1.9 `complete_quest.sql` - PL/SQL procedure that handles completing a quest within the application in places other than the Check-In form.

4.1.1.10 `equip_item.sql` - PL/SQL procedure that handles equipping an item within the application.

4.1.1.11 `quit_quest.sql` - PL/SQL procedure that handles quitting a quest within the application.

4.1.2 API SQL Scripts

4.1.2.1 `get_career_quest_list.sql` - Given a `career_id`, returns all skills for that career, all quests for that career, and all resources associated with each quest.

4.1.2.2 `get_career_skills.sql` - Gets the skills associated with a career given its ID.

4.1.2.3 `add_new_item.sql` - Adds a new item into the `Inventory` table.

4.1.2.4 `delete_item.sql` - Deletes the item with the given `item_id` from the `Inventory` table.

4.1.2.5 `get_inventory.sql` - Gets all rows and all columns from the `Inventory` table.

4.1.2.6 `get_item_details.sql` - Given an `item_id`, obtains all details for the specified item.

4.1.2.7 `grab_itemId.sql` - Obtains the most recent `item_id` inserted into the `Inventory` table. This is used to determine the next `item_id` number to increment to.

4.1.2.8 `update_item.sql` - Given an `item_id`, updates the specified item's type, name, PNG file name, and price.

4.1.2.9 `add_new_quest.sql` - Adds a new quest into the `Quest` table.

4.1.2.10 `add_quest_resource.sql` - Given a `quest_id` and `resource_id`, adds the specified resource to the specified quest by adding a row to the `Quest_Resources` table.

4.1.2.11 `add_quest_skill.sql` - Given a `quest_id` and `skill_id`, adds the specified skill to the specified quest by adding a row to the `Skill_Quest` table.

4.1.2.12 `delete_quest_resource.sql` - Given a `quest_id` and `resource_id`, removes the specified resource from the specified quest by deleting the indicated row in the `Quest_Resources` table.

4.1.2.13 `delete_quest_skill.sql` - Given a `quest_id` and `skill_id`, removes the specified skill from the specified quest by deleting the indicated row in the `Skill_Quest` table.

4.1.2.14 `delete_quest.sql` - Deletes the quest with the given `quest_id` from the `Quest` table.

4.1.2.15 `get_all_quests.sql` - Gets all currently available quests from the `Quest` table.

4.1.2.16 `get_quest_details.sql` - Given a `quest_id`, obtains all information on the specified quest as well as the skills and resources associated with that quest.

4.1.2.17 `get_quest_resources.sql` - Given a `quest_id`, obtains all information of the resources that are associated with the specified quest.

4.1.2.18 `get_quest_skill.sql` - Given a `quest_id`, obtains all information on the skills associated with the specified quest.

4.1.2.19 `grab_questId.sql` - Obtains the most recent `quest_id` inserted into the `Quest` table. This is used to determine the next `quest_id` number to increment to.

4.1.2.20 `update_quest.sql` - Given a `quest_id`, updates the specified quest's name and description.

4.1.2.21 `add_new_resource.sql` - Adds a new resource to the `Resources` table.

4.1.2.22 `delete_resource.sql` - Given a `resource_id`, removes the specified resource from the `Resources` table.

4.1.2.23 `get_all_resources.sql` - Gets all resources and their details from the `Resources` table, ordered alphabetically.

4.1.2.24 `get_resource_details.sql` - Given a `resource_id`, obtains all information of the specified resource.

4.1.2.25 `grab_resource_id.sql` - Obtains the most recent `resource_id` inserted into the `Resources` table. This is used to determine the next `resource_id` number to increment to.

4.1.2.26 `update_resource.sql` - Given a `resource_id`, updates the specified resource's name, link, description, and type.

4.1.2.27 `add_new_skill.sql` - Adds a new skill to the `Skill` table.

4.1.2.28 `delete_skill.sql` - Given a `skill_id`, removes the specified skill from the `Skill` table.

4.1.2.29 `get_all_skills.sql` - Obtains all skills and their details from the `Skill` table, ordered alphabetically.

4.1.2.30 `grab_skillId.sql` - Obtains the most recent `skill_id` inserted into the `Skill` table. This is used to determine the next `skill_id` number to increment to.

4.1.2.31 `update_skill.sql` - Given a `skill_id`, updates the specified skill's name and description.

4.1.2.32 `add_new_student_skill.sql` - Adds a new skill to a student's skill set by inserting a new row into the `Student_Skill` table, where the student's ID, skill's ID, and amount of EXP are specified.

4.1.2.33 `add_new_student.sql` - Adds a new student into the `Student` table.

4.1.2.34 `add_student_quest.sql` - Given a `student_id` and `quest_id`, adds a new row into the `Student_Quest` table to indicate that the student is currently working on the specified quest.

4.1.2.35 `delete_student_quest.sql` - Given a `student_id` and `quest_id`, removes the specified row from the `Student_Quest` table to indicate that the student has quit the specified quest.

4.1.2.36 `get_inventory_page.sql` - Obtains all information necessary to populate all components of the Inventory page.

4.1.2.37 `get_mainpage_stats.sql` - Obtains all information necessary to populate all components of the Mainpage.

4.1.2.38 `get_student_career.sql` - Given a `student_id`, obtains the specified student's career.

4.1.2.39 `get_student_checked_days.sql` – Given a student_id, obtains all information related to the student's progress from the `Student_Progress` table. This information is ordered in descending order with respect to the progress date.

4.1.2.40 `get_student_lvl_exp.sql` - Given a `student_id`, obtains the specified student's level, total amount of EXP, and name.

4.1.2.41 `get_student_quests.sql` - Given a `student_id`, obtains all quests that the specified student has progress in from the `Student_Quest` table.

4.1.2.42 `get_student_skills.sql` - Given a `student_id`, obtains the information of all skills the student is currently working on.

4.1.2.43 `update_frequency.sql` - Given a `student_id` and reminder frequency, updates the specified student's entry in the `Student` table with the specified reminder frequency.

4.1.2.44 `update_skill_exp.sql` - Given a `student_id` and `skill_id`, updates the specified student's EXP within the specified skill.

4.1.3 JavaScript Files

4.1.3.1 `db.js` - Script that pools all connections to the backend database.

4.1.3.2 `server.js` - Defines the main API routes and backend framework using Express.

4.1.3.3 `careersGet.js` - Defines the functions needed to perform GET requests related to careers, loading all necessary SQL files to perform these functions.

4.1.3.4 `inventoryDelete.js` - Defines the functions needed to perform DELETE requests related to the entire inventory, loading all necessary SQL files to perform these functions.

4.1.3.5 `inventoryGet.js` - Defines the functions needed to perform GET requests related to the entire inventory, loading all necessary SQL files to perform these functions.

4.1.3.6 `inventoryPost.js` - Defines the functions needed to perform POST requests related to the entire inventory, loading all necessary SQL files to perform these functions.

4.1.3.7 `inventoryPut.js` - Defines the functions needed to perform PUT requests related to the entire inventory, loading all necessary SQL files to perform these functions.

4.1.3.8 `questsDelete.js` - Defines the functions needed to perform DELETE requests related to quests, loading all necessary SQL files to perform these functions.

4.1.3.9 `questsGet.js` - Defines the functions needed to perform GET requests related to quests, loading all necessary SQL files to perform these functions.

4.1.3.10 `questsPost.js` - Defines the functions needed to perform POST requests related to quests, loading all necessary SQL files to perform these functions.

4.1.3.11 `questsPut.js` - Defines the functions needed to perform PUT requests related to quests, loading all necessary SQL files to perform these functions.

4.1.3.12 `resourcesDelete.js` - Defines the functions needed to perform DELETE requests related to resources, loading all necessary SQL files to perform these functions.

4.1.3.13 `resourcesGet.js` - Defines the functions needed to perform GET requests related to the resources, loading all necessary SQL files to perform these functions.

4.1.3.14 `resourcesPost.js` - Defines the functions needed to perform POST requests related to resources, loading all necessary SQL files to perform these functions.

4.1.3.15 `resourcesPut.js` - Defines the functions needed to perform PUT requests related to resources, loading all necessary SQL files to perform these functions.

4.1.3.16 `skillsDelete.js` - Defines the functions needed to perform DELETE requests related to skills, loading all necessary SQL files to perform these functions.

4.1.3.17 `skillsGet.js` - Defines the functions needed to perform DELETE requests related to skills, loading all necessary SQL files to perform these functions.

4.1.3.18 `skillsPost.js` - Defines the functions needed to perform POST requests related to skills, loading all necessary SQL files to perform these functions.

4.1.3.19 `skillsPut.js` - Defines the functions needed to perform PUT requests related to skills, loading all necessary SQL files to perform these functions.

4.1.3.20 `studentsDelete.js` - Defines the functions needed to perform DELETE requests related to student information, loading all necessary SQL files to perform these functions.

4.1.3.21 `studentsGet.js` - Defines the functions needed to perform GET requests related to student information, loading all necessary SQL files to perform these functions.

4.1.3.22 `studentsPost.js` - Defines the functions needed to perform POST requests related to student information, loading all necessary SQL files to perform these functions.

4.1.3.23 `studentsPut.js` - Defines the functions that load all the SQL files needed to perform PUT requests related to student information. These functions may also call prepared statements to call stored PL/SQL procedures and functions.

4.1.3.24 `careers.js` - Defines all API routes related to careers.

4.1.3.25 `inventory.js` - Defines all API routes related to the inventory.

4.1.3.26 `quests.js` - Defines all API routes related to quests.

4.1.3.27 `resources.js` - Defines all API routes related to resources.

4.1.3.28 `skills.js` - Defines all API routes related to skills.

4.1.3.29 `students.js` - Defines all API routes related to students and student information.

4.1.3.30 `package-lock.json` - File that defines the dependencies and their respective versions for the backend of Edutale to work correctly.

4.1.3.31 `package.json` - File that defines the dependencies and project information for the backend of Edutale.

4.1.3.32 `.env.example` - An example file that specifies the information needed for Edutale's `.env` file.

4.2 Frontend Folder

4.2.1 Assets Folder

This folder includes all of the images used by Edutale. The majority of the images used can be found in the Inventory page, but some also appear on the Mainpage, Welcome page, and Header. Please see Appendix A for more information on the used assets in Edutale.

4.2.2 Admin Folder

4.2.2.1 `Admin.jsx` - Driver file that defines the Admin component, which renders the Admin page in its entirety.

4.2.2.2 `AdminTabs.jsx` - Defines the tab system used to organize different admin functions (Quests, Skills, Resources, Items).

4.2.2.3 `AdminTabs.css` - Styles the Admin page tabs and layout.

4.2.2.4 `Message.jsx` - Defines the component for displaying success/error messages for admin actions.

4.2.2.5 `ItemForm.jsx` - Form component for creating/editing inventory items.

4.2.2.6 `QuestForm.jsx` - Form component for creating/editing quests.

4.2.2.7 `QuestResourceForm.jsx` - Form component for managing quest-resource associations.

4.2.2.8 `QuestSkillForm.jsx` - Form component for managing quest-skill associations.

4.2.2.9 `ResourceForm.jsx` - Form component for creating/editing resources.

4.2.2.10 `SkillForm.jsx` - Form component for creating/editing skills.

4.2.2.11 `AdminFunctionSelector.jsx` - Helper file that selects specific functions seen in the Admin page, as well as define the modal for their dialog.

4.2.2.12 `useItems.js` - Hook for managing item CRUD operations.

4.2.2.13 `useQuests.js` - Hook for managing quest CRUD operations.

4.2.2.14 `useResources.js` - Hook for managing resource CRUD operations.

4.2.2.15 `useSkills.js` - Hook for managing skill CRUD operations.

4.2.2.16 `itemServices.js` - Service functions for item API calls.

4.2.2.17 `questServices.js` - Service functions for quest API calls.

4.2.2.18 `resourceServices.js` - Service functions for resource API calls.

4.2.2.19 `skillServices.js` - Service functions for skill API calls.

4.2.2.20 `ItemTab.jsx` - Component for the Items management tab.

4.2.2.21 `QuestTab.jsx` - Component for the Quests management tab.

4.2.2.22 `ResourceTab.jsx` - Component for the Resources management tab.

4.2.2.23 `SkillTab.jsx` - Component for the Skills management tab.

4.2.3 CheckIn Folder

4.2.3.1 `CalendarHolder.css` - Defines the styling for the react-calendar displayed on the Check-In page, under the "Statistics" header.

4.2.3.2 `CalendarHolder.jsx` - Defines the react-calendar component to be displayed on the Check-In page. Includes functions that allow for different styling of checked-in days.

4.2.3.3 `ChangeFreq.css` - Defines the styling for the content under the "Schedule" header on the Check-In page.

4.2.3.4 `ChangeFreq.jsx` - Defines the `ChangeFreq` component that displays the student's current frequency timing as well as a button allowing the student to change their frequency. Also generates an iCalendar invite that reflects the frequency selected.

4.2.3.5 `ChangeFreqHandler.jsx` - Handles the change of frequency on the backend of Edutale through an API call.

4.2.3.6 `CheckInForm.css` - Defines the styling for the Check-In form portion of the Check-In page, under the "Check-In" header.

4.2.3.7 `CheckInFormContent.jsx` - Defines the content of the Check-In form, alongside all the logic behind dynamically adding and subtracting entries within the form.

4.2.3.8 `CheckInFormHolder.jsx` - Makes API calls to obtain a student's level, EXP, and ongoing quests. Also holds the `CheckInFormContent` component created by `CheckInFormContent.jsx`.

4.2.3.9 `SubmitHandler.jsx` - Handles a Check-In form submission by making an API call to the database.

4.2.3.10 `CheckIn.css` - Defines the styling for the top-level containers for the CheckIn component.

4.2.3.11 `CheckIn.jsx` - Driver file that defines the `CheckIn` component, which renders the Check-In page in its entirety. Also handles the logic of either showing the Check-In page or the Check-In summary depending on whether the student has filled out a Check-In today.

4.2.4 Header Folder

4.2.4.1 `Navigation.css` - Styles the Navigation tabs in the form of bookmarks.

4.2.4.2 `Navigation.jsx` - Defines the navigation links that are displayed on every student page. These links are hidden on the Welcome, Admin, and Settings pages.

4.2.4.3 `LogoutButton.css` - Styles the `LogoutButton` component of the Header.

4.2.4.4 `LogoutButton.jsx` - Defines the `LogoutButton` component that logs a user out of the Edutale application.

4.2.4.5 `Profile.css` - Styles the `Profile` component of the Header.

4.2.4.6 `Profile.jsx` - Defines the profile card displayed on the top left corner of the Header, which displays a student's profile picture, their name, and the `LogoutButton` component.

4.2.4.7 `Header.css` - Defines the styling for the top-level containers for the Header component.

4.2.4.8 `Header.jsx` - Driver file that defines the Header component, which renders the header seen at the top of student pages.

4.2.5 Inventory Folder

4.2.5.1 `Equipped.jsx` - Defines the `Equipped` component, which renders the student's currently-equipped items (or an empty space if the student does not have an item equipped).

4.2.5.2 `BuyHandler.jsx` - Defines the function that handles the action of buying an item by making an API call to the backend of Edutale. Also handles the case when a student does not have enough money to complete the transaction.

4.2.5.3 `EquipHandler.jsx` - Defines the function that handles the action of equipping an item by making an API call to the backend of Edutale.

4.2.5.4 `InventoryList.css` - Styles the `InventoryList` component, which is the right portion of the Inventory page.

4.2.5.5 `InventoryList.jsx` - Defines the `InventoryList` component, which populates the list of items that a user owns or is allowed to purchase.

4.2.5.6 `Money.jsx` - Defines the `Money` component, which displays the student's current amount of money on the Inventory page.

4.2.5.7 `InvUserBar.css` - Styles the level bar seen in the Inventory page.

4.2.5.8 `InvUserBar.jsx` - Creates the level bar on the Inventory page.

4.2.5.9 `Inventory.css` - Styles the top-level containers for the Inventory page.

4.2.5.10 `Inventory.jsx` - Driver file that defines the `Inventory` component, which renders the Inventory page in its entirety. Also makes the necessary API calls that obtains all the information necessary to render the Inventory page.

4.2.6 Mainpage Folder

4.2.6.1 `Avatar.css` - File that styles the `Avatar` component.

4.2.6.2 `Avatar.jsx` - Defines the `Avatar` component, which displays the student's current avatar. Currently, the image used is a placeholder due to the nature of the prototype build. The `Avatar` component is also used on the Inventory page.

4.2.6.3 `Day.css` - File that styles the `Day` component.

4.2.6.4 `Day.jsx` - Defines the `Day` component, which calculates the days since the student's join date and displays it within the Mainpage.

4.2.6.5 `CompleteHandler.jsx` - Defines the function that handles the action of completing a quest while on the Mainpage, under the "Ongoing Quests" header. This is done by making an API call to the backend of Edutale.

4.2.6.6 `OngoingQuestBlock.jsx` - Defines the `OngoingQuestBlock` component, which displays a single instance of an ongoing quest within the `OngoingQuests` component.

4.2.6.7 `OngoingQuests.css` - Styles the `OngoingQuests` component, alongside the `OngoingQuestBlock` components that are rendered as part of it.

4.2.6.8 `OngoingQuests.jsx` - Defines the `OngoingQuests` component, which displays all of the student's ongoing quests on the Mainpage. Also makes the necessary API calls to obtain the student's ongoing quests themselves.

4.2.6.9 `QuitHandler.jsx` - Defines the function that handles the action of quitting an ongoing quest while on the Mainpage, under the "Ongoing Quests" header. This is done by making an API call to the backend of Edutale.

4.2.6.10 `AcceptHandler.jsx` - Defines the function that handles the action of accepting a quest while on the Mainpage, under the "Quest Board" header. This is done by making an API call to the backend of Edutale.

4.2.6.11 `QuestBoard.css` - Styles the `QuestBoard` component, along with the components that are rendered as part of it (`QuestRow`).

4.2.6.12 `QuestBoard.jsx` - Defines the `QuestBoard` component, which displays five quests that a student can choose to accept from. Also makes the necessary API calls to obtain the quests a student has no progress in.

4.2.6.13 `QuestRow.jsx` - Defines the `QuestRow` component, which renders a single instance of a quest that can be accepted on the Quest Board.

4.2.6.14 `SkillGraph.css` - Styles the container for the Chart.js radar graph within the `SkillGraph` component.

4.2.6.15 `SkillGraph.jsx` - Defines the `SkillGraph` component, which renders a Chart.js radar graph displaying the EXP a student currently has for each skill.

4.2.6.16 `UserBar.css` - Styles the `UserBar` component.

4.2.6.17 `UserBar.jsx` - Defines the `UserBar` component, which displays a student's name, current level, and EXP towards the next level. This component is also used on the Inventory page.

4.2.6.18 `Mainpage.css` - Styles the top-level containers for the Mainpage.

4.2.6.19 `Mainpage.jsx` - Driver file that defines the `Mainpage` component, which renders the Mainpage in its entirety. Also makes the API calls necessary to obtain almost all the information required to render the Mainpage correctly.

4.2.6.20 `QuestModal.jsx` - Defines the `QuestModal` component, which renders the text on the modals seen on `OngoingQuests` and `QuestBoard`.

4.2.7 Resume Folder

4.2.7.1 `ResTemplates.jsx` - Defines the `ResTemplates` component, which takes a student's current progress and turns it into two different resume templates they can use in their resume.

4.2.7.2 `YourProjects.jsx` - Defines the `YourProjects` component, which lists the student's completed projects.

4.2.7.3 `YourSkills.jsx` - Defines the `YourSkills` component, which lists the student's EXP in their skills in descending order.

4.2.7.4 `Resume.css` - Styles the Resume page and all components within said page.

4.2.7.5 `Resume.jsx` - Driver file that defines the `Resume` component, which renders the Resume page in its entirety. Also makes the API calls necessary to obtain all the information required to render the Resume page correctly.

4.2.8 Settings Folder

4.2.8.1 `Settings.jsx` - Defines the `Settings` component to be used in the Settings page. Currently, this component is a placeholder for settings to be added in future versions of Edutale.

4.2.9 SkillGallery Folder

4.2.9.1 `QuestList.css` - Styles the `QuestList` and `QuestListHelper` components.

4.2.9.2 `QuestList.jsx` - Defines the `QuestList` component, which makes API calls in order to get the student's progress towards quests as well as the whole quest catalog. Also acts as the driver file for `QuestListHelper`.

4.2.9.3 `QuestListHelper.jsx` - Defines the `QuestListHelper` component, which populates the list of quests for a given skill. Also styles each quest correctly according to the student's current progress in that quest, as well as assigns the corresponding modals.

4.2.9.4 `SkillTiles.css` - Styles the `SkillTiles` component.

4.2.9.5 `SkillTiles.jsx` - Defines the `SkillTiles` component, which displays the student's skills in the form of tiles on the Skill Gallery page. Clicking on each skill leads to the `QuestList` component updating to display the quests for the current skill.

4.2.9.6 `SkillGallery.jsx` - Driver file that defines the `SkillGallery` component, which renders the Skill Gallery page in its entirety. Also makes the necessary API call in order to obtain the student's selected career.

4.2.10 Welcome Folder

4.2.10.1 `ConnectButton.css` - Styles the `ConnectButton` component.

4.2.10.2 `ConnectButton.jsx` - Defines the `ConnectButton` component which prompts the user to authenticate with Auth0, after which they are directed to the Edutale student pages.

4.2.10.3 `Welcome.css` - Styles the Welcome page.

4.2.10.4 `Welcome.jsx` - Defines the `Welcome` component, which displays the Welcome page that a user first sees upon signing in to Edutale.

4.2.11 Other Frontend Files

4.2.11.1 `App.css` - Styles the entire Edutale frontend. Defines styles for the app-wide font alongside classes that are used frequently, such as those for designing the book aesthetic.

4.2.11.2 `App.jsx` - Defines the `App` component, which includes all frontend routes and authentication. Can be seen as the driver file for all other driver files.

4.2.11.3 `main.jsx` - Defines the content that should be displayed within index.html. This file includes the `App` component as well as the Auth0 service.

4.2.11.4 `eslint.config.js` - Configuration file for ESLint.

4.2.11.5 `index.html` - The HTML file used to hold the entire React application that Edutale uses as the frontend.

4.2.11.6 `package-lock.json` - File that defines the dependencies and their respective versions for the backend of Edutale to work correctly.

4.2.11.7 `package.json` - File that defines the dependencies and project information for the backend of Edutale.

4.2.11.8 `vite.config.js` - Defines the configuration of Vite with respect to React.

5. Coding Standards

5.1 General

1) All constants will be written in all uppercase.
2) Double quotes will be preferred over single quotes, unless the use of single quotes is required (such as with SQL).
3) Code reviews to uphold coding standards shall be conducted at the end of each week.
4) All open and close brace pairs for functions and expressions will use the slanting style. The only exception to this is for JSX-specific syntax, such as using JavaScript syntax to declare the value of an HTML attribute:

```
<div className="pane-item">
  <h1 className="pane-header"> Your Skills </h1>
  <SkillTiles career={career} onSetSkill={setCurrSkill} currSkill={currSkill}/>
</div>
```

5.2 JavaScript / JSX

1) Indentation within JS functions and code shall be set to four spaces per indent. The only exception to this are JS objects, as well as any JSON, which can use either two or four spaces per indent.
    a) The level of indentation for JS objects and JSON should be consistent within the file that it is located within.
2) Ending semicolons are not required within JS or JSX files, and programmers may use them or omit them. However, usage of semicolons must be constant within a given file.
3) Functions that are used to directly generate React components must use PascalCase.
4) Helper functions that are not directly used to generate React components must use camelCase.
5) All variables will use camelCase. The only exception to this are variables that map directly to database table columns, since SQL does not distinguish between lowercase and uppercase within column attributes (see SQL section for more information):

```
<li key={item.quest_id + currSkill} className="sq-list-item completed">
  <p className="prog-indicator"> Quest complete! </p>
  <p> <b>{item.quest_name}</b>: {item.quest_description} </p>
```

6) All functions must be separated by one empty line.

5.3 HTML

1) Indentation within HTML code shall be set to two spaces per indent. This also applies to HTML seen within React components or JSX files.
2) Standard HTML element and attribute names shall be all lowercase.
3) All void HTML elements must have a closing forward slash (Ex. `<img />`). This includes React components.
4) All closing tags shall have a forward slash to indicate them as such (Ex. `<p> This is a sentence. </p>`).
5) All class names and IDs must use kebab-case (all lowercase with dashes as a separator). Exceptions are when attribute values need to comply with the syntax of other programming languages, such as when the value of a name needs to comply with PL/SQL syntax:

```
<select className="quest-select" defaultValue="" name="quest_id_diff"
```

6) HTML elements that get too long may be split into multiple lines. Subsequent lines must be at least two spaces indented in comparison with the beginning of the element.
    a) Some programmers may opt to instead line up HTML on continuing lines with the start of the HTML attributes; this is also acceptable, so long as subsequent lines are indented at least two spaces.

5.4 CSS

1) Indentation within CSS code shall be set to four spaces per indent.
2) Any use of colors, whether they be foreground or background, must comply with the WCAG AAA standards for ADA compliance.
    a) An example tool to uphold this standard is the Colour Contrast Checker by Snook.ca.
3) All CSS rules must be separated by one empty line.
4) All CSS declarations must be on their own line.
5) All CSS classes and IDs must use kebab-case.

5.5 PL/SQL

1) All SQL scripts, file names, columns, etc. shall use snake_case, since SQL does not distinguish between uppercase and lowercase.

a) When referring to SQL tables, use snake_case with each word having a capital first letter:

```
create table if not exists Skill_Quest(
    skill_id        char(9),
    quest_id        char(9),
    primary key     (skill_id, quest_id),
    foreign key     (skill_id) references Skill(SKILL_ID) on delete cascade,
    foreign key     (quest_id) references Quest(QUEST_ID) on delete cascade
);
```

2) All multi-line SQL statements shall have an empty line separating them from other statements.
3) The clauses for SELECT, UPDATE, INSERT, and DELETE statement shall be on their own line, such that the readability of each statement is upheld:

```
select item_id
from Inventory
where item_id like 'I%'
order by item_id desc limit 1;
```

4) Nested statements, statements with multiple AND clauses, and PL/SQL blocks shall be indented by at least four spaces.
    a) Some programmers may opt to line up PL/SQL statements on continuing lines with the start of the parameters on the initial line; this is also acceptable, so long as subsequent lines are indented at least four spaces. The following two indentation styles are acceptable:

```
update Student_Skill
set skill_exp = skill_exp + calc_quest_exp(skill.quest_difficulty)
where skill_id = skill.skill_id
      and student_id = stu_id;
```

```
update Student_Skill
set skill_exp = skill_exp + calc_quest_exp(skill.quest_difficulty)
where skill_id = skill.skill_id
    and student_id = stu_id;
```

6. Modules

6.1 Database API

6.1.1 Career Functions

6.1.1.1 **Get Career Quests** - Gets all quests given a career.
Parameters: Career ID

Request Type: GET
Route: `/api/careers/:id/quests`

6.1.1.2 **Get Career Skills** - Gets all skills for the given career.
Parameters: Career ID
Request Type: GET
Route: `/api/careers/:id/skills`

6.1.2 Inventory Functions

6.1.2.1 **Get Inventory Catalog** - Get entire inventory catalog.
Parameters: None
Request Type: GET
Route: `/api/inventory/inventory`

6.1.2.2 **Get Inventory Item Details** - Gets an inventory item's details.
Parameters: Item ID
Request Type: GET
Route: `/api/inventory/inventory/:id`

6.1.2.3 **Add New Inventory Item** - Adds a new inventory item to the inventory table.
Parameters: Item's type, name, PNG name, and price
Request Type: POST
Route: `/api/inventory/inventory`

6.1.2.4 **Update Inventory Item** - Updates an existing inventory item.
Parameters: Item ID
Request Type: PUT
Route: `/api/inventory/inventory/:id`

6.1.2.5 **Delete Inventory Item** - Deletes the specified inventory item.
Parameters:Item ID
Request Type: DELETE
Route: `/api/inventory/inventory/:id`

6.1.3 Quest Functions

6.1.3.1 **Get All Quests** - Gets all quests from the backend.
Parameters: None
Request Type: GET
Route: `/api/quests`

6.1.3.2 **Get Quest Resources** - Gets all resources related to a specific quest.
Parameters: Quest ID
Request Type: GET
Route: `/api/quests/:id/resources`

6.1.3.3 **Get Quest Skills** - Gets all skills related to a specific quest.
Parameters: Quest ID
Request Type: GET
Route: `/api/quests/:id/skills`

6.1.3.4 **Get Quest Details** - Get the details of a specific quest.
Parameters: Quest ID
Request Type: GET
Route: `/api/quests/:id`

6.1.3.5 **Add New Quest** - Adds a new quest with the given information.
Parameters: Quest name, description, if it's a project (true/false), difficulty
Request Type: POST
Route: `/api/quests`

6.1.3.6 **Add Resource to Quest** - Adds a resource to a quest.
Parameters: Quest ID, resource ID
Request Type: POST
Route: `/api/quests/:id/resources`

6.1.3.7 **Add Skill to Quest** - Adds a skill to a quest.
Parameters: Quest ID, skill ID
Request Type: POST
Route: `/api/quests/:id/skills`

6.1.3.8 **Delete Quest** - Deletes a quest.
Parameters: Quest ID
Request Type: DELETE
Route: `/api/quests/:id`

6.1.3.9 **Remove Skill from Quest** - Removes a specified skill from a specified quest.
Parameters: Quest ID, skill ID
Request Type: DELETE
Route: `/api/quests/:id/skills/:skillId`

6.1.3.10 **Remove Resource from Quest** - Removes a specified resource from a quest.
Parameters: Quest ID, resource ID

Request Type: DELETE
Route: `/api/quests/:id/resources/:resourceId`

6.1.3.11 **Update Quest Information** - Updates the name and description of a quest.
Parameters: Quest ID, quest description
Request Type: PUT
Route: `/api/quests/:id`

6.1.4 Resource Functions

6.1.4.1 **Get All Resources** - Gets all resources within the entire database.
Parameters: None
Request Type: GET
Route: `/api/resources`

6.1.4.2 **Get Resource Details** - Get the details of a specific resource.
Parameters: Resource ID
Request Type: GET
Route: `/api/resources/:id`

6.1.4.3 **Add New Resource** - Adds a new resource.
Parameters: Resource's name, link, description, and resource type
Request Type: POST
Route: `/api/resources`

6.1.4.4 **Update Resource Information** - Updates a resource's information.
Parameters: Resource's name, link, description, and resource type
Request Type: PUT
Route: `/api/resources/:id`

6.1.4.5 **Delete Resource** - Deletes the specified resource from the database.
Parameters: Resource ID
Request Type: DELETE
Route: `/api/resources/:id`

6.1.5 Skill Functions

6.1.5.1 **Get All Skills** - Gets all skills stored in the database.
Parameters: None
Request Type: GET
Route: `/api/skills`

6.1.5.2 **Add New Skill** - Adds a new skill to the database.
Parameters: Skill's name and description
Request Type: POST
Route: `/api/skills`

6.1.5.3 **Update Skill Information** - Updates the specified skill's information.
Parameters: Skill's name and description
Request Type: PUT
Route: `/api/skills/:id`

6.1.5.4 **Delete Skill** - Deletes the specified skill.
Parameters: Skill ID
Request Type: DELETE
Route: `/api/skills/:id`

6.1.6 Student Functions

6.1.6.1 **Get Student Skills** - Gets all the skills of a specified student.
Parameters: Student ID
Request Type: GET
Route: `/api/students/:id/skills`

6.1.6.2 **Get Student Quests** - Gets all the ongoing/completed quests of a student.
Parameters: Student ID
Request Type: GET
Route: `/api/students/:id/quests`

6.1.6.3 **Get Student Career** - Gets the specified student's career.
Parameters: Student ID
Request Type: GET
Route: `/api/students/:id/career`

6.1.6.4 **Get Student Checked-In Days** - Gets all days that the student checked in.
Parameters: Student ID
Request Type: GET
Route: `/api/students/:id/checkin`

6.1.6.5 **Get Student Level and EXP** - Gets the specified student's level and EXP.
Parameters: Student ID
Request Type: GET
Route: `/api/students/:id/progress`

6.1.6.6 **Get Student Inventory Page** - Gets all student information needed to populate the Inventory page.
Parameters: Student ID
Request Type: GET
Route: `/api/students/:id/inventory`

6.1.6.7 **Get Student Mainpage** - Gets all student information needed to populate the Mainpage.
Parameters: Student ID
Request Type: GET
Route: `/api/students/:id/mainpage`

6.1.6.8 **Update Student Skill EXP** - Updates the specified student's skill EXP.
Parameters: Student ID, skill ID
Request Type: PUT
Route: `/api/students/:id/skills/:skillId`

6.1.6.9 **Submit Check-In Form** - Handles all updates to the backend when the specified student submits a check-in.
Parameters: Student's ID, total EXP, net EXP, completed quest IDs, number of completed quests, total minutes, net money
Request Type: PUT
Route: `/api/students/:id/check-in-complete`

6.1.6.10 **Buying an Item** - Updates the specified student's money and inventory when purchasing an item.
Parameters: Student ID, item ID, item price, student's money amount
Request Type: PUT
Route: `/api/students/:id/buy-item`

6.1.6.11 **Completing a Quest** - Updates the student's completed quests after completing a quest.
Parameters: Student ID, quest ID
Request Type: PUT
Route: `/api/students/:id/complete-quest`

6.1.6.12 **Equipping an Item** - Updating the specified student's equipped item
Parameters: Student ID, item ID
Request Type: PUT
Route: `/api/students/:id/equip-item`

6.1.6.13 **Accepting a Quest** - Updates the student's ongoing quests after accepting a quest

Parameters: Student ID, quest ID
Request Type: PUT
Route: `/api/students/:id/accept-quest`

6.1.6.14 **Quitting a Quest** - Updates the student's ongoing quests after quitting a quest.
Parameters: Student ID, quest ID
Request Type: PUT
Route: `/api/students/:id/quit-quest`

6.1.6.15 **Update Student Check-In Frequency** - Updates the specified student's check-in frequency
Parameters: Student ID, new check-in frequency
Request Type: PUT
Route: `/api/students/:id/freq`

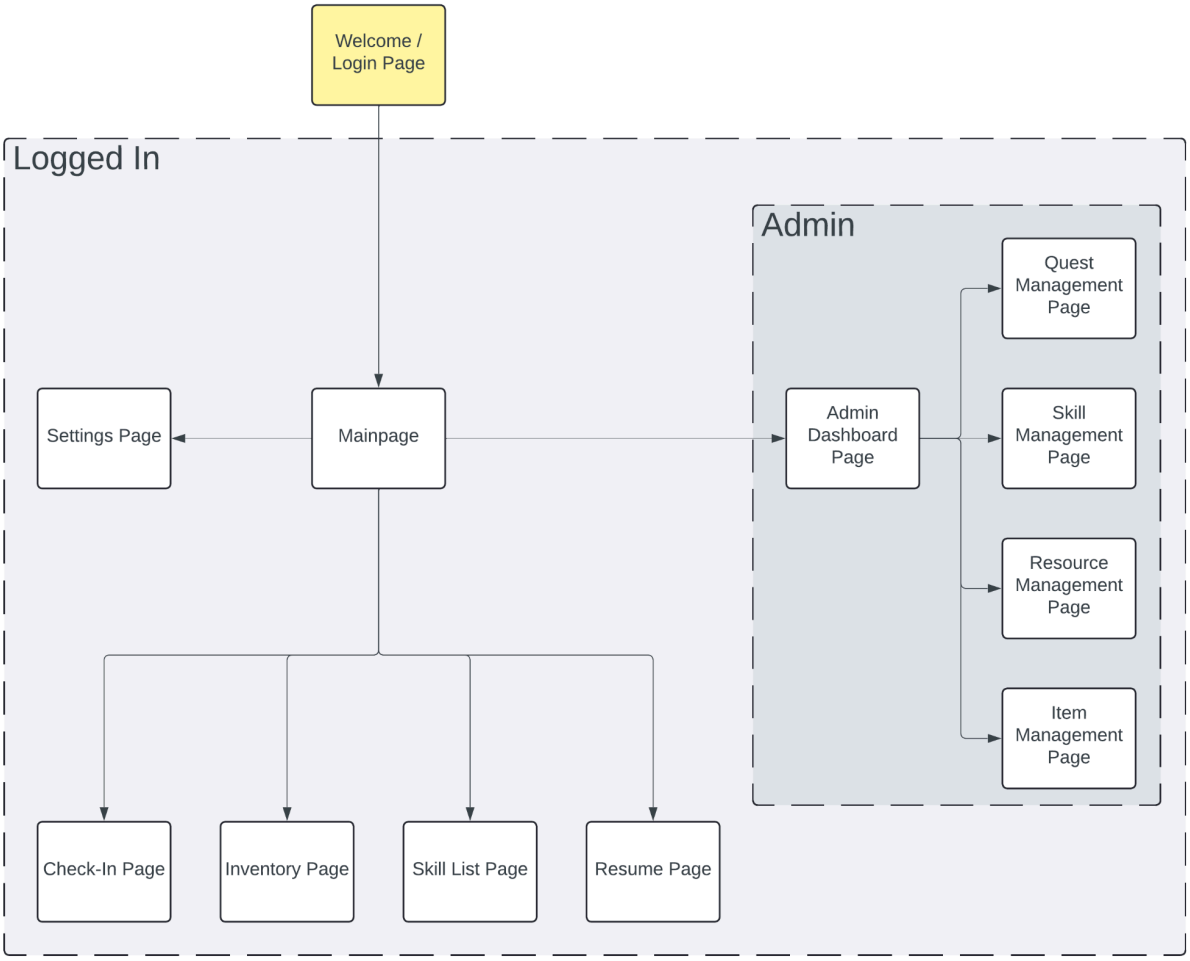
6.1.6.16 **Delete Student's Quest Progress** - Delete a student's quest progress.
Parameters: Student ID, quest ID
Request Type: DELETE
Route: `/api/students/:id/quests/:questId`

6.1.6.17 **Add New Student** - Adds a new student to the Student table.
Parameters: None
Request Type: POST
Route: `/api/students`

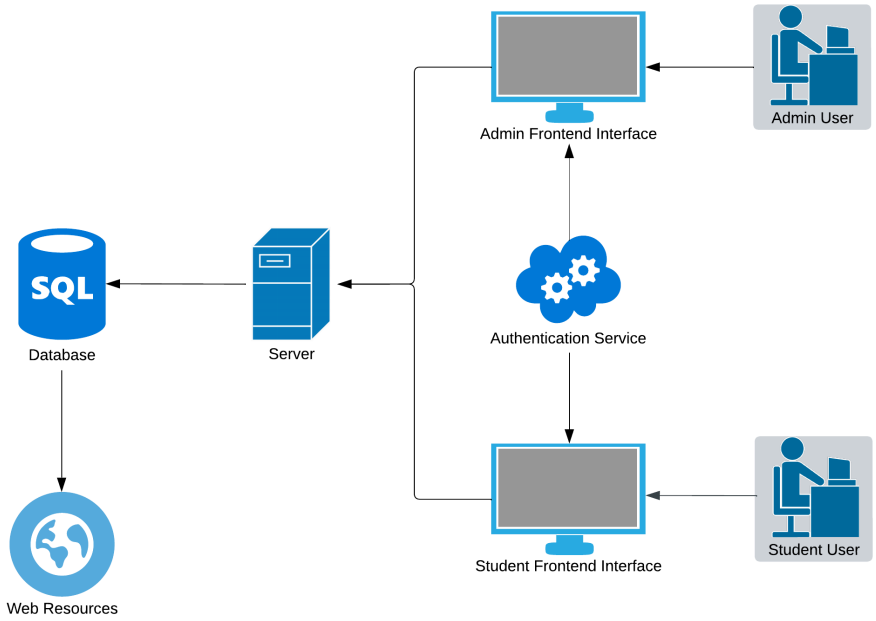
6.1.6.18 **Add New Quest for Student** - Adds a new quest for the specified student.
Parameters: Student ID, quest ID, current quest status
Request Type: POST
Route: `/api/students/:id/quests`

6.1.6.19 **Add New Skill for Student** - Adds a new skill for the specified student.
Parameters: Student ID, skill ID, skill EXP
Request Type: POST
Route: `/api/students/:id/skills`

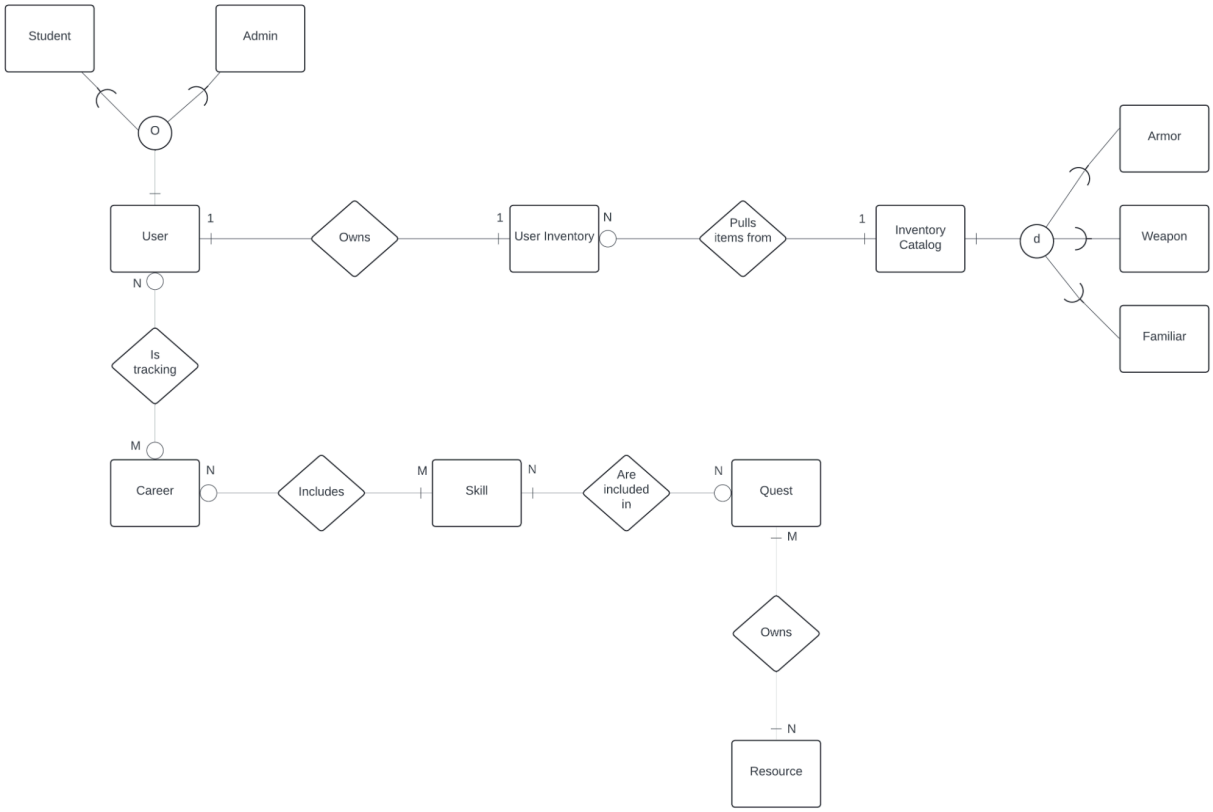
6.2 Site Map


Welcome / Login Page
Logged In
Admin
Quest Management Page
Settings Page
Mainpage
Admin Dashboard Page
Skill Management Page
Resource Management Page
Item Management Page
Check-In Page
Inventory Page
Skill List Page
Resume Page

6.3 Architectural Design



6.4 Entity Relationship Diagram



6.5 Data Dictionaries

Edutale's data dictionaries describe the core tables of the database, which are the main entities with the most essential data to the system.

6.5.1 Student Table

Column	Data type	Description
`STUDENT_ID`	char(9)	Student's unique ID
`student_name`	varchar(100)	Student's name
`student_email`	varchar	Student's email (not gathered but was included for future implementation)
`student_join_date`	date	Date of student's first login to Edutale
`total_exp`	int	Total global experience amount
`student_lvl`	int	Global level based on experience
`student_money`	int	Money to be spent on the Inventory page
`reminder_freq`	char(1)	Value of either "Daily", "Weekly", or "Monthly" used to generate an iCalendar file
`equip_weapon`	char(9)	Item ID for the weapon item the student currently has equipped
`equip_armor`	char(9)	Item ID for the armor item the student currently has equipped
`equip_familiar`	char(9)	Item ID for the familiar item the student currently has equipped

6.5.2 Career Table

Column	Data type	Description
`CAREER_ID`	char(9)	Career's unique ID
`career_name`	varchar(32)	Career's name
`career_description`	text	Description of the career

6.5.3 Quest Table

Column	Data type	Description
`QUEST_ID`	char(9)	Quest's unique ID
`quest_name`	varchar(255)	Quest's name
`quest_description`	text	Description of the quest
`quest_difficulty`	int	Value of either 1, 2, or 3 where a higher value refers to a more difficult quest
`is_project`	boolean	True or false value determining whether or not a quest is a project

6.5.4 Skill Table

Column	Data type	Description
`SKILL_ID`	char(9)	Skill's unique ID
`skill_name`	varchar(32)	Skill's name
`skill_description`	text	Description of the skill

6.5.5 Resource Table

Column	Data type	Description
`RESOURCE_ID`	char(9)	Resource's unique ID
`resource_name`	text	Resource's name
`resource_link`	text	Link to the resources source where content can be accessed
`resource_description`	text	Description of the resource
`resource_type`	varchar(20)	Value of either "Video", "Article", 'Tutorial", "Exercise", "Documentation", or "Tool" detailing the type of content the resource includes

6.5.6 Inventory Table

Column	Data type	Description
`ITEM_ID`	char(9)	Item's unique ID
`item_type`	char(1)	Value of either "A", "W", or "F" describing the differing types of items by the first letter of the type, Armor, Weapon, and Familiar respectively
`item_name`	varchar(30)	Item's name
`item_png_name`	varchar(30)	Name of PNG file for the item's digital image
`item_price`	int	Amount of the student's money it takes to purchase the item

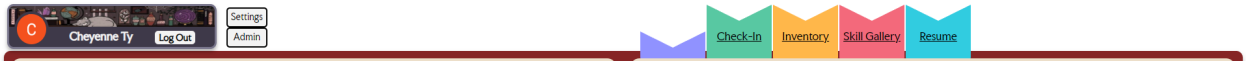
6.6 Welcome Page

This is the landing page of Edutale that users see when they are not logged in. Pressing the "Connect" button initiates authentication with Auth0. If a user is already logged in, they will not see this page again until the log back out.

6.7 Header

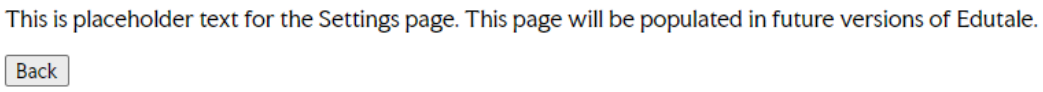
The Header consists of multiple components that appear at the top of every main student page. The Header contains navigation tabs in the form of bookmarks, as well as a profile card

in the top left corner. The Header also includes a Settings button and an Admin button. In future versions of Edutale, this Admin button will only be visible to users that are also administrators.



6.8 Settings Page

Currently, the Settings page is a placeholder for future Edutale settings. These settings will be implemented in future versions of Edutale.



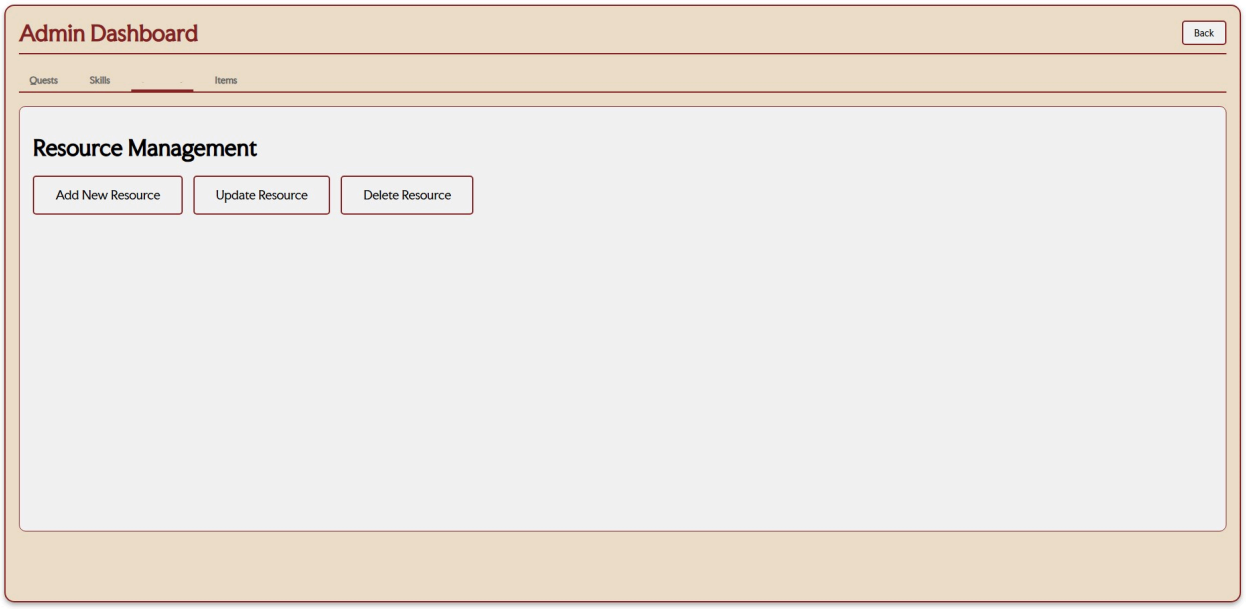
6.9 Admin Page

Administrators of Edutale will have access to the Admin Page. This page consists of four tabs, allowing administrators to manage all types of resources stored within Edutale. Each tab allows the user to alter the database involving that specific class of resource.

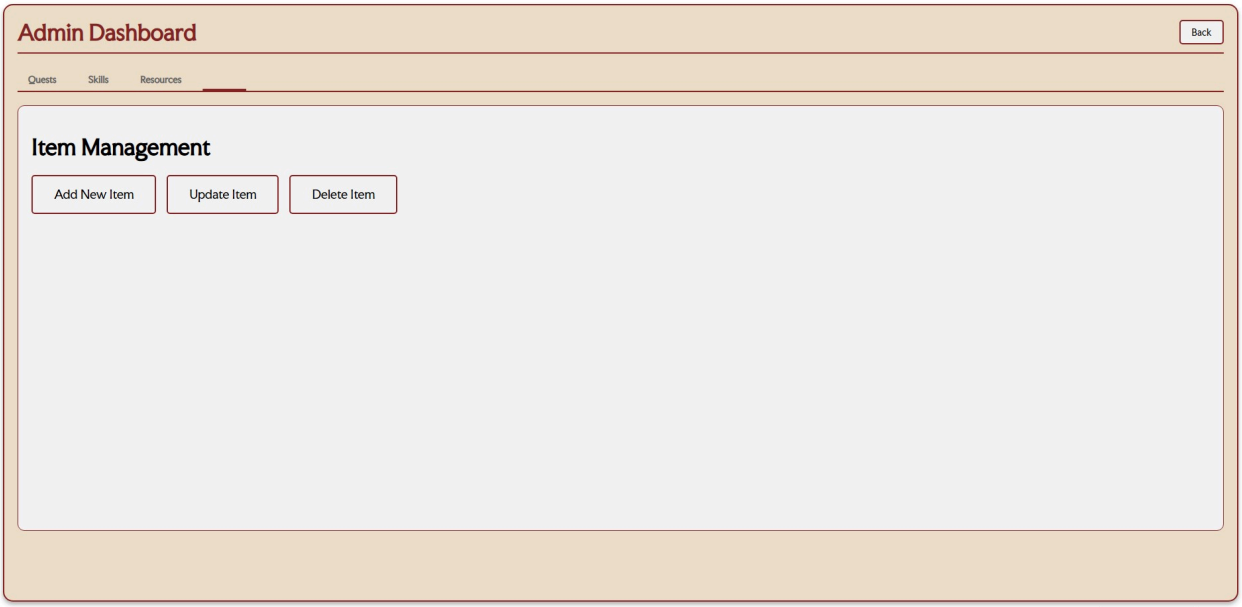
6.9.1 Quest Tab

6.9.2 Skills Tab

6.9.3 Resources Tab

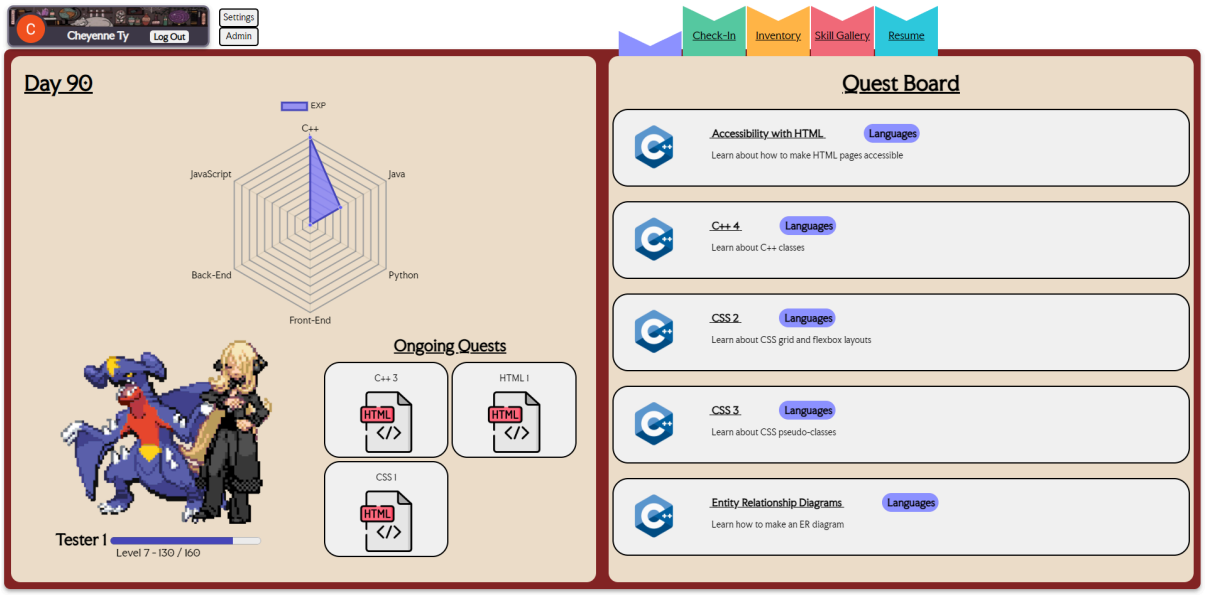


6.9.4 Items Tab



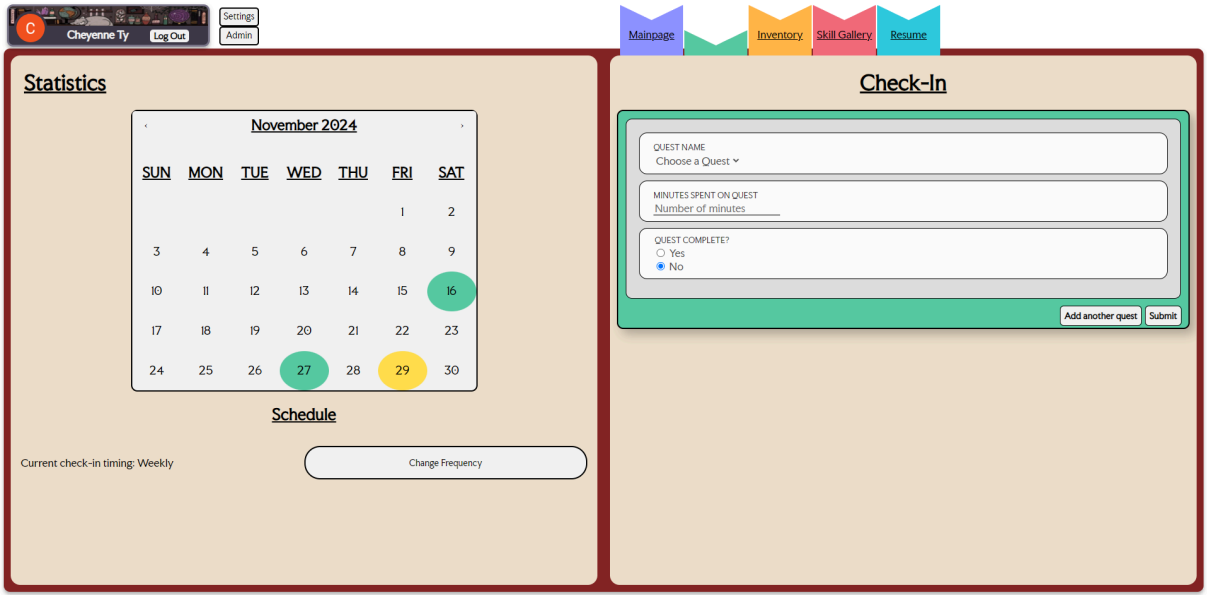
6.10 Mainpage

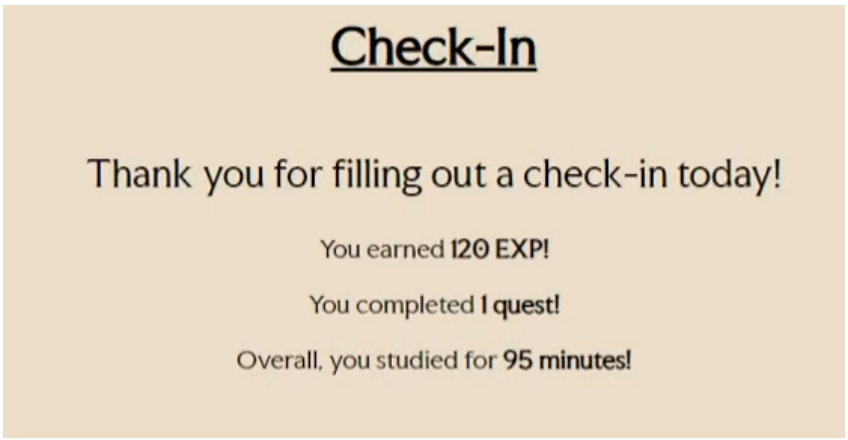
The Mainpage displays a summary of the student's account. It displays the student's current level, global EXP, skill EXP, and ongoing quests. It also offers new quests for a student to accept through the Quest Board. Quests can be accepted, marked as complete, or quit from this page through modals.



6.11 Check-In Page

The Check-In Page is the main way of recording progress in Edutale. There is a check-in form to the right that records a student's current progress towards each quest they are currently working towards. This form can only be filled out once, and a summary of the current day will be displayed in place of this form if the student has already completed it. The Check-In page also displays a student's past check-ins through the calendar, and a student can set their reminder frequency through the "Schedule" portion of the page. Edutale creates an iCalendar invite whenever a student changes their check-in frequency.





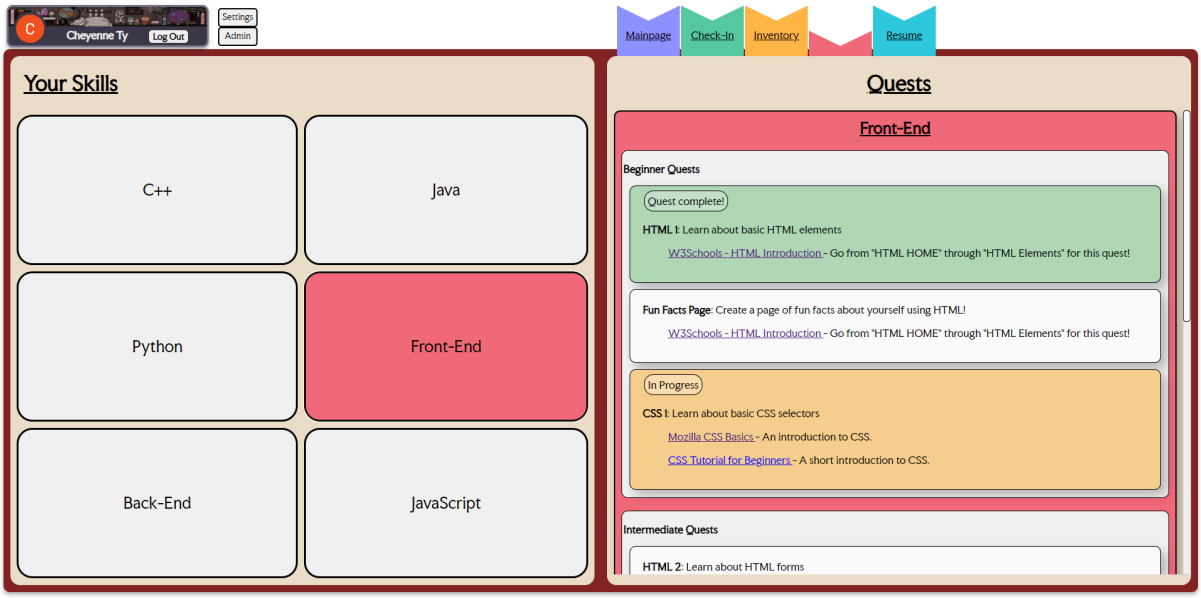
6.12 Inventory Page

The Inventory page allows for a greater incentive in using Edutale. Students obtain 100 money per level they gain, and they can spend their money through buying items from this

page. Students can then equip their bought items. In future Edutale updates, the displayed avatar will reflect the currently-equipped inventory items.

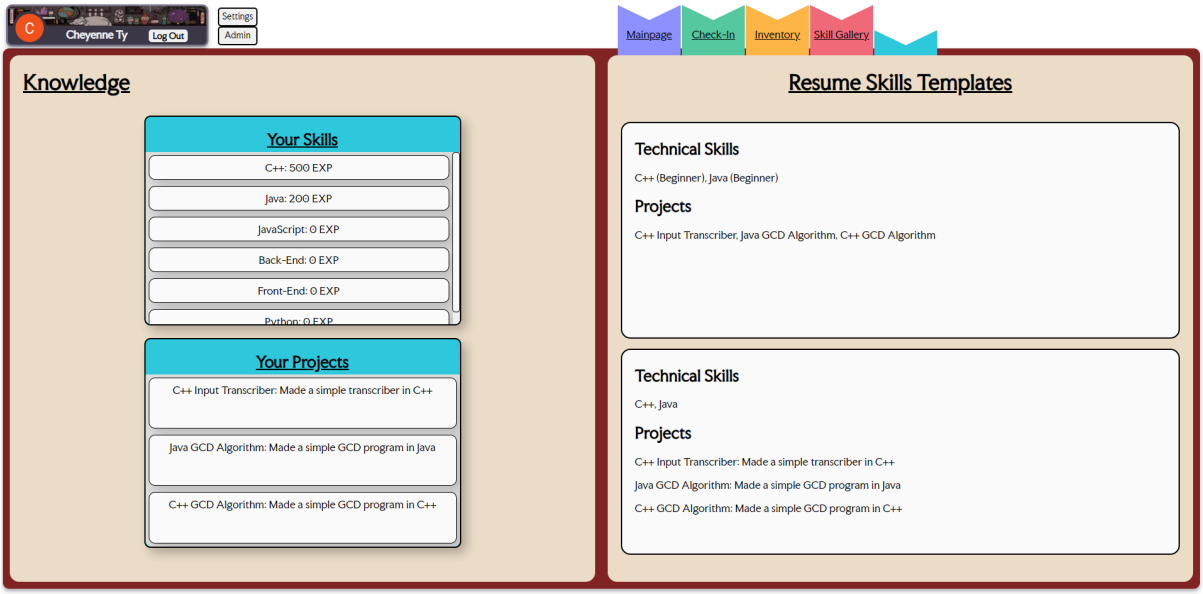
6.13 Skill Gallery Page

The Skill Gallery page displays all of the available quests for each skill that a student is currently working on. This page also shows the student's current progress towards each quest. Quests can be marked as complete, accepted, or quit on this page, using the same logic as these operations are on the Mainpage.



6.14 Resume Page

The Resume page displays the student's overall progress through two resume templates. The student's EXP per skill as well as the projects they have completed are also displayed on this page. This allows for students to easily transfer their Edutale progress from the application to their actual resumes.



7. References

The following references include key documentation for the technologies used to build Edutale.

7.1 Application Technologies

Technology Name	URL to documentation
`React`	https://react.dev/
`Vite`	https://vite.dev/
`Node.js`	https://nodejs.org/en
`PostgreSQL`	https://www.postgresql.org/
`Auth0`	https://auth0.com/

7.2 npm Packages

Package Name	URL to documentation
`react-router-dom`	https://www.npmjs.com/package/react-router-dom
`@auth0/auth0-react`	https://www.npmjs.com/package/@auth0/auth0-react
`react-calendar`	https://www.npmjs.com/package/react-calendar
`react-chartjs-2`	https://www.npmjs.com/package/react-chartjs-2
`reactjs-popup`	https://www.npmjs.com/package/reactjs-popup

Appendix A: Used Assets

Asset Name in Edutale	Asset Source	URL to where Asset was found
`avatar.gif`	Animation of Cynthia and Garchomp from *Pokemon: Black 2 / White 2*	https://tenor.com/view/animation-pixel-art-garchomp-cynthia-gif-15793261332433284025
`background.jpg`	"Creating My Dream Library"blog post by Brigitta Blair	https://brigittablair.com/blog/creating-my-dream-library/
`cpp.png`	Official C++ logo for the C++ Foundation	https://github.com/isocpp/logos
`html_icon.png`	"Html free icon" by Freepik	https://www.flaticon.com/free-icon/html_1197396
`profileBanner.png`	TK Krussow on Instagram (@tkcreates)	https://www.instagram.com/tkcreates/p/CxGQdm8rlCP/
`crimson_armor.png`	"Adamantite armor" from *Terraria*	https://terraria.fandom.com/wiki/Adamantite_armor
`dark_armor.png`	"Netherite Chestplate" from *Minecraft*	https://minecraft.wiki/w/File:Netherite_Chestplate_(item)_JE1.png
`diamond_armor.png`	"Diamond Chestplate" from *Minecraft*	https://minecraft.wiki/w/File:Diamond_Chestplate_(item)_JE1_BE1.png
`fancy_knight_armor.png`	"Knight's Helmet" from *Stardew Valley*	https://stardewvalleywiki.com/Knight%27s_Helmet
`fireproof_armor.png`	"Molten armor" from *Terraria*	https://terraria.fandom.com/wiki/Molten_armor
`ghost_armor.png`	"Spectre armor" from *Terraria*	https://terraria.fandom.com/wiki/Spectre_armor
`gold_armor.png`	"Gold Chestplate" from *Minecraft*	https://minecraft.wiki/w/File:Golden_Chestplate_(item)_JE1_BE1.png
`holy_armor.png`	"Hallowed armor" from	https://terraria.fandom.com/w

	Terraria	iki/Hallowed_armor
`iron_armor.png`	"Iron Chestplate" from *Minecraft*	https://minecraft.wiki/w/File:Iron_Chestplate_(item)_JE1_BE1.png
`leather_armor.png`	"Leather Tunic" from *Minecraft*	https://minecraft.wiki/w/File:Leather_Tunic_(item)_JE3_BE2.png
`titanium_armor.png`	"Titanium armor" from *Terraria*	https://terraria.fandom.com/wiki/Titanium_armor
`viking_helmet.png`	"Warrior Helmet" from *Stardew Valley*	https://stardewvalleywiki.com/Warrior_Helmet
`basset_hound.png`	"Dog 3" from *Stardew Valley*	https://stardewvalleywiki.com/File:Dog_3.png
`black_cat.png`	"Cat 5" from *Stardew Valley*	https://stardewvalleywiki.com/File:Cat_5.png
`brown_chicken.png`	"Brown Chicken" from *Stardew Valley*	https://stardewvalleywiki.com/File:Brown_Chicken.png
`dinosaur.png`	"Dinosaur" from *Stardew Valley*	https://stardewvalleywiki.com/File:Dinosaur.png
`duck.png`	"Duck" from *Stardew Valley*	https://stardewvalleywiki.com/File:Duck.png
`horse.png`	"Horse" from *Stardew Valley*	https://stardewvalleywiki.com/File:Horse.png
`purple_turtle.png`	"Iridium Turtle" from *Stardew Valley*	https://stardewvalleywiki.com/File:Iridium_Turtle.png
`rabbit.png`	"Rabbit" from *Stardew Valley*	https://stardewvalleywiki.com/File:Rabbit.png
`retriever.png`	"Dog 1" from *Stardew Valley*	https://stardewvalleywiki.com/File:Dog_1.png
`tabby_cat.png`	"Cat 1" from *Stardew Valley*	https://stardewvalleywiki.com/File:Cat_1.png
`turtle.png`	"Turtle" from *Stardew Valley*	https://stardewvalleywiki.com/File:Turtle.png

`tuxedo_cat.png`	"Cat 2" from *Stardew Valley*	https://stardewvalleywiki.com/File:Cat_2.png
`white_cat.png`	"Cat 4" from *Stardew Valley*	https://stardewvalleywiki.com/File:Cat_4.png
`white_chicken.png`	"White Chicken" from *Stardew Valley*	https://stardewvalleywiki.com/File:White_Chicken.png
`dark_sword.png`	"Netherite Sword" from *Minecraft*	https://minecraft.wiki/w/File:Netherite_Sword_JE1.png
`diamond_sword.png`	"Diamond Sword" from *Minecraft*	https://minecraft.wiki/w/File:Diamond_Sword_JE1_BE1.png
`gold_bow.png`	"Gold Bow" from *Terraria*	https://terraria.fandom.com/wiki/Gold_Bow
`gold_sword.png`	"Gold Sword" from *Minecraft*	https://minecraft.wiki/w/File:Golden_Sword_JE2_BE1.png
`holy_lance.png`	"Hallowed Jousting Lance" from *Terraria*	https://terraria.fandom.com/wiki/Hallowed_Jousting_Lance
`ice_bow.png`	"Ice Bow" from *Terraria*	https://terraria.fandom.com/wiki/Ice_Bow
`ice_rod.png`	"Ice Rod" from *Stardew Valley*	https://stardewvalleywiki.com/Ice_Rod
`iron_bow.png`	"Iron Bow" from *Terraria*	https://terraria.fandom.com/wiki/Iron_Bow
`iron_sword.png`	"Iron Sword" from *Minecraft*	https://minecraft.wiki/w/File:Iron_Sword_JE1_BE1.png
`katana.png`	"Katana" from *Terraria*	https://terraria.fandom.com/wiki/Katana
`trident.png`	"Trident" from *Terraria*	https://terraria.fandom.com/wiki/Trident
`wooden_sword.png`	"Wooden Sword" from *Minecraft*	https://minecraft.wiki/w/File:Wooden_Sword_JE1_BE1.png