**Layout:**

* A main.rs file under src/bin. This is where I use rocket to redirect pages, load pages, launch the website, etc.
* A static folder: Contains my CSS and images used.
* A template folder: Contains my html.tera pages.
* In cargo.toml: I use tokio for asynchronous runtime, sqlx for backend, rocket\_auth for login system, rocket, and serde (among others as well).

**The Structs:**

For this project, structs play a huge role as they are necessary for my forms. In my main.rs, I have the following:

1. Inventory – represents the stock count of the website. More specifically, the struct keeps track of each individual food count.
2. User – From the rocket\_auth library, it keeps track of a user’s id, email, and password. Also determines if a user is an admin, but I don’t use said feature for this assignment.
3. UserStock – Keeps track of the count of a user’s inventory. Like #1, the struct keeps track of each individual food count.
4. A form for each food for trading purposes. Consists of the following: AppleForm, BananaForm, BroccoliForm, OrangeForm, GrapeForm, and WatermelonForm.

**Getting set up:**

Alongside my import statements on main.rs and my cargo.toml depencies, in Ubuntu, I installed the following – libssl-dev, pkg-config, sqlite3, cargo. Also messed around with postgresql but not used in the final submission.

Afterwards, I created a database in sqlite3 named database.db

**Running the Program:**

Upon start up, the user is introduced to the sign up/login page. I used a template online for this (linked in my readme) and modified it to 1. Accept my forms for the backend and 2. To change the theme I wanted. In main.rs, the function index() runs this page and uses HomePage.html.tera from templates.

**Signing up:**

For the **signup** portion, the user has to submit an email and a password. This is necessary for the rocket\_auth “users” backend. In Homepage.html.tera, upon clicking sign up button, the form invokes POST towards /signup. Going back to main.rs, post /signup invokes the method post\_signup, which uses the form + auth’s library to create a new user. Afterwards, it calls a get /signup, which invokes the function sign\_up\_page(). This loads the NewUser html file, alongside passing the user information in json format.

In the **NewUser** page, the user is asked to input their current inventory of food. For simplification purposes, my website only focuses on the following foods: apples, bananas, watermelons, broccoli, oranges, and grapes. In NewUser.html, all food counts are required, in number format, and cannot be put below zero. Furthermore, the form follows the struct “Submission” in main.rs, which contains apple\_count, banana\_count, etc, in u32 format. After submission of the form, the site invokes POST towards /newUserSubmit. In main.rs, this calls the method formsub(), which simply states the form submitted and redirects the /db, my dashboard page (discussed below).

**Logging in:**

If the user already has an account, they can just insert their email and password. In homepage.html, this calls POST to /login. Like above, post /login calls post\_login in main.rs, which uses the form and auth library to authenticate said user. Afterwards, it redirects the user to /db, the dashboard page.

**Dashboard:**

After either logging in, or a new user inserts their inventory count, a user is taken to /db, the dashboard page that takes Dashboard.html.tera. From the function get\_login in main.rs, this also takes the following data: the user info, the website inventory (currently it is a static mut, ideally it instead pulled from backend), and user stock (also currently a static mut called u\_stock, should pull from backend ideally). If the user is logged in, then at the top it shows the following: user email, user inventory count (e.g. 0 apples, 0 bananas, etc. via u\_stock), an edit inventory button and a Logout button. As the name suggests, edit inventory takes the user to a page where they can edit their existing inventory counts.

If Logout is clicked, then it invokes /logout, which in main.rs calls the logout function. In said function, it simply logs the user out via rocket\_auth library and then returns the user to the home sign up page.

The main purpose of this page is to show the user how much stock is currently present on the website regarding each food. With this, there are six images of the food accompanied with text showcasing their count. Said count is called from the website\_inv static object, although ideally I would grab this information from the backend. All images can be clicked on, in which the user would then be able to trade their existing inventory for the type of food they clicked on.

**The Trading Pages:**

From the dashboard page, a user is redirected to a trading page. In said page, it consists of a text description, an image of the item to be gained from the trade, and a form that accepts how much of said food you want alongside the fruit you will be trading. For an example, if the user clicks on the image of an apple in the dashboard and then submits “5” and “Banana” in the form, then this means the user will gain 5 apples from the trade all while trading away 5 bananas from their personal inventory. After a successful trade, u\_stock is updated, as well as the website\_inv (using previous example, website inventory will lose 5 apples and gain 5 bananas).

All six foods have their own trading pages sharing the same format/display, with minor changes to each food’s image and text description. For each food they also have their own struct for the form and a function that updates the u\_stock and website\_inv. For an example, the food Apple has the following in main.rs: AppleForm struct, appleform POST function that modifies the inventory/stock counts, and an async GET apple function to load the page.

After a user makes a trade via pressing the trade button, they are redirected to a page that states their trade was successful. Alongside this, in the background in main.rs, the inventory counts are modified as previously mentioned.

**Acknowledged problems/room for improvement:**

1. Currently in the backend, the only thing that is properly used and modified is the table of existing users. As mentioned multiple times in this paper, the website inventory and user inventory is currently maintained in a static mut. What this means is that that inventory isn’t saved when the program stops running. Alongside this, it also means that I have to call unsafe{}, as I am calling and editing a static mut. If my project were perfect, it would have every instance where unsafe{} is used and replace the static mut with a connection to my tables in the backend.

A black background with white text

Description automatically generated with low confidence

With limited knowledge of rocket and limited resources available, I found the task of connecting my manually created tables to my main function to be difficult. As I found and shared in the Discord, some guides in the internet are outdated, and others are missing information (such as what crates are utilized, what is imported, and what is installed). For reference, below is the schema for my manually created tables.

A computer screen shot of white text

Description automatically generated with low confidence

1. I should enforce that a user cannot make a trade that exceeds the current website inventory count. E.g. If a user wants 10 apples but the website only has 9, it will redirect to an error page.
2. Modify existing error pages. E.g. If a user makes a password less than 8 characters, it will redirect to an error page that doesn’t match the theme of the website.
3. All other minor changes/suggestions added to my README