Project Mission

Develop and deploy a website that enables users to browse and buy groceries, and admins to manage the inventory & pricing of said groceries and the staff of their organisation in a secure, robust, intuitive, aesthetic and convenient fashion.

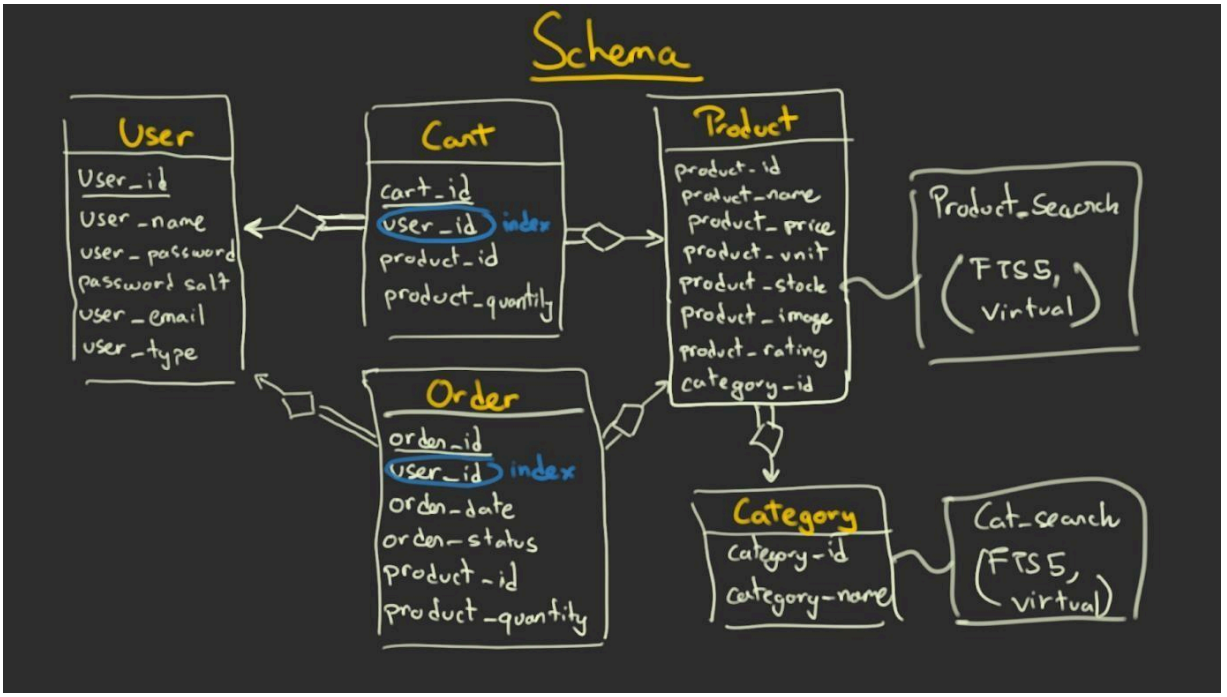
PS: In the end, Aesthetics took a hit, but the core functionality was met.

Technologies Used

A SQLite3 Database was used with the Flask-SQLAlchemy python package to make up the model. Flask-RESTful was used to make APIs that furthered the separation of model and controller code.
Flask was used to host the webpage, and all pages were templated with Jinja2. CSS was designed with Tailwind classes. This accounts for the controller and view code as well.

In addition, the following python packages were also used: Flask-login and Flask-WTF for security, (eg: prevention of CSRF attacks, salted password hashing) matplotlib for plotting graphs of sales presented to admins, and the Tailwind CSS Framework for aesthetic design.

DB Schema Design



The datatypes and the constraints have not been shown here for the sake of brevity, as the most natural datatype was chosen. There are, however, a few features of the schema that are worthy of mention.

1. The user_password column is a non nullable BINARY(32), a hash. The salt is stored separately as a string of 16 characters in the password_salt column. The reason for this is simply efficiency: The default output of the hashing functions in Flask-login redundant mention the hashing type and other details in a long string, which need not be stored in the database. It is assumed as a fact in the encryption and decryption code.

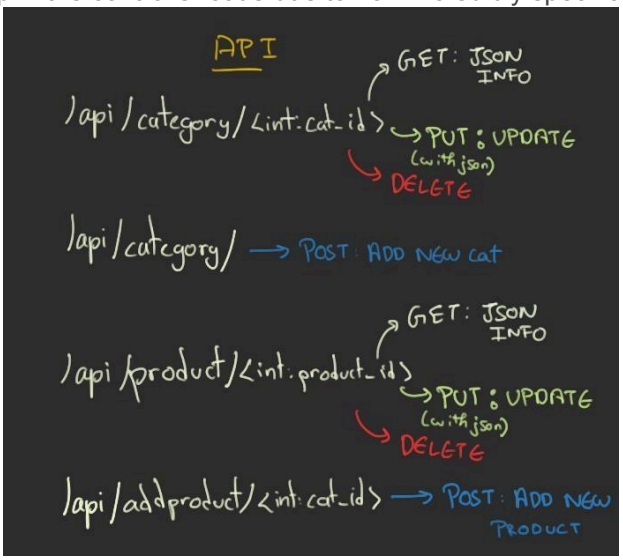
2. The user_id is indexed in both the order and the cart tables as the queries of fetching all of a given user's orders or all of their cart items is extremely common.

3. The cart table has some rather awkward naming, as each entry in the table holds exactly one product for one user. This means that each user will have multiple entries in the table, one for each product in their cart. Intuition asks that a separate box be made for each user, but such a thing is more conveniently deployed in a noSQL scenario, where a non-atomic cart attribute can be used in the user table itself.

4. There are triggers associated with the category and product columns that make corresponding updates to their Full Text Search tables.

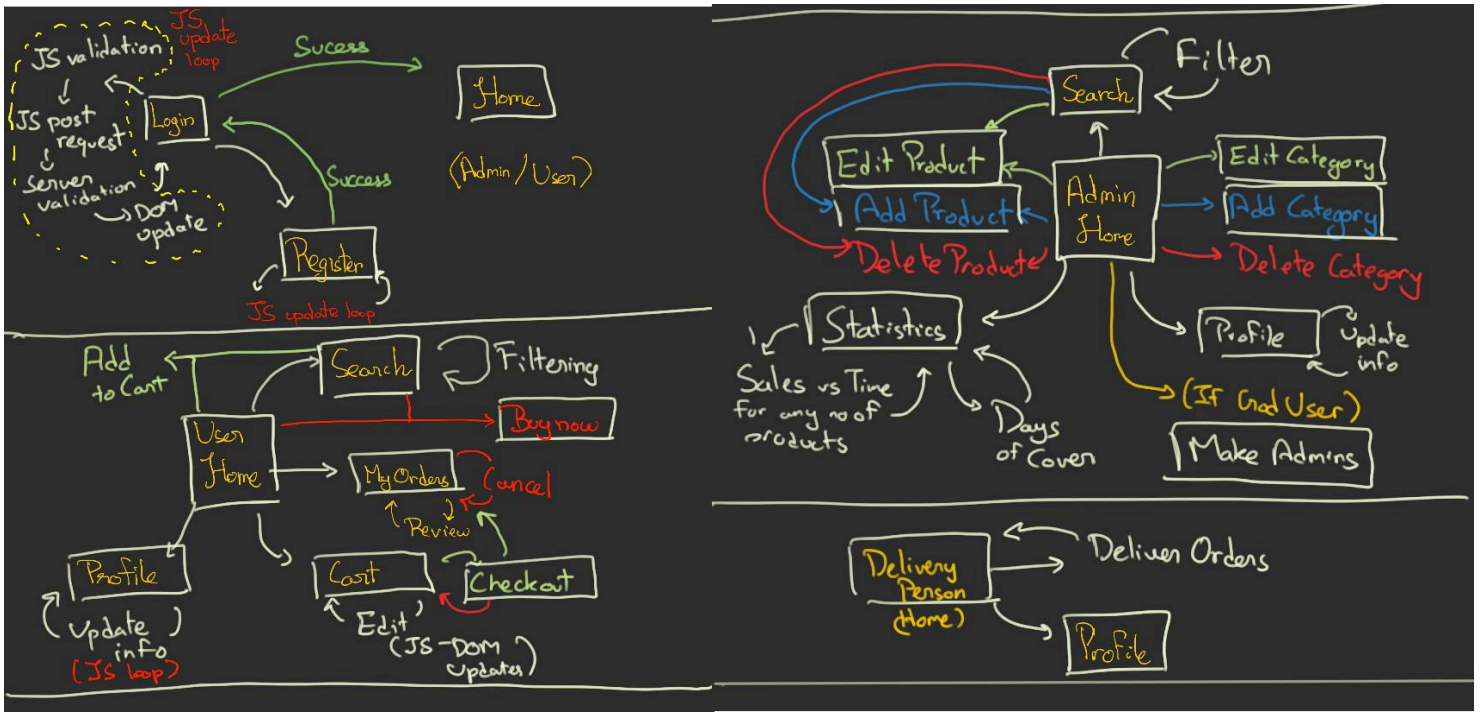
API Design

Products and Categories have a CRUD API. Although this implemented some convenient separation between model and controller code, it was inevitable that a lot of model interaction ended up in the controller code due to how incredibly specific each function's needs were.



PS: this was done only because of explicit project requirements in my course project; I didn't find it particularly useful in this app, although it is very true that such abstraction can be invaluable in larger projects.

Architecture and Features



The above shows the control flow of the webpage. All boxed names are individual webpages. Using this as a rough guideline, here the some of the noteworthy features in the page:

1. The user can purchase individual items directly from the homepage/ search results. They can also add items to their cart from both these pages.

2. Searching allows for filtering by Category, Rating, and price. The results can be sorted by price, ascending or descending, by rating or be shown in the default “Featured” order, which promotes better selling and better rated products.

3. The Admin can perform CRUD on products as well as categories from their homepage/search results.

4. Admins can view statistics about the sales of all products. They can view time series graphs of sales, and predicted number of days before stock runs out for each product.

5. Users with the authorisation of level “God” can change the user type of users other than themselves. They are responsible for the granting of admin authority to certain users.

6. All users have a profile page in which they can update their password and other details. The passwords in the webpage are salted and hashed, and are safe from exposure, even in the event of a breach.

7. Most pages, wherever possible, use JavaScript to send requests to the server and update the page dynamically, making for a significantly better user experience.

The following 2 are unimplemented features that would complete the rating system in the project but were left undone as they were tangential to the main project statements.

1. Users designated as delivery personnel are assigned some orders to deliver. They can close this by marking them as delivered. The user has the ability to mark it delivered from their end as well, or report the delivery as missing.

2. Users can review the orders they have received successfully with 1 to 5 stars.

PS: The search bar filters behave abnormally in some circumstances; I am just too lazy to patch it. Also, all products have 0 stars atm, so you won’t be able to test those filters without changing those attributes in the db via some editor/sql queries.