FrontEnd -

The frontend is written in Vite React. It comprises of majorly 4 different pages - Login, Register, Home, RecipeView

Dependencies used -

Mantine

react-toastify

react-icons

crypto-js

General -   
  
I have used mantine as primary UI library which is also consistent across all the webpages. I am able to define colorSchmes with the help of this library and also themes can be set using Mantine-themes. I have also added a theme toogle in the navbar which is able to change the theme for all the webpages. On searches or backend API requests, the frontend gets blurred, notifying that the page is loading. All the pages have navbar which has the following buttons - Home, login, register, logout, ThemeSelector

**Login** -

The login page has a form with email and password fields. This makes a post request to /login backend which verifies the credentials. For encrypting the password I have used crypto-js which has shared encryption key and IV with the backend so that the backend can decrypt the encrypted credentials and verify the user. For error showing I have used react-toastify which tells the user if any error arises. Validation checks are also enabled for this form. On successful login the user details (name, email, token, id) are stored in the local storage which will be used to identify the user. If this is already present in the local storage, it implies that the user is logged in. Any request by the user in the further sections will pass the token as authorisation to check the permissions of the user.

**Register** -

This page has a similar form which requires user to enter first name, last name, email and password. This also has validation checks enabled. On submit, the crypto-js library encrypts the password for secure transmission to the backend, which registers the new user details. On error, react-toastify shows the error message. On success, it leads to login page for the user to enter.

**Home** -

The home page is divided into 4 sections.

1. Search (Default)-   
   This section has a search textbox and a multiselect input. User can either select one at a time to query the recipes. If the user selects textbox, then the text he enters will be matched to return recipes with matching name. If multiple ingredients are selected from the multiselect box, then any recipe which contains any of these will be returned.   
   This section has the form and will show the recipes through a card component which has several features -   
   On favorite pressed - will toggle this recipe as favourite or remove from favourite by backend API request. (Will show the favourite toggle on the frontend, before API request. If something goes wrong will toggle it again and show the error)  
   Show details - will take the user to /recipe page
2. Favourite -  
   This section shows the favourites of the user using the card components. It can take the user to recipe page or toggle favourites. To reduce frequent API requests. This will only once load the favorites. If user removes any recipe from favourites, it makes the API call and removes this recipe from local list (backend will not be contacted to get new favourite list)
3. My Recipes -   
   This allows the user to add a new recipe and also view all the recipes created by the user. The add recipe form can be seen by pressing the Add recipe button. The image is taken in the form of an image URL, others are strings or numbers (time). In addition to the basic card features, this section cards will also have delete button which will delete the selected recipe.
4. Profile -  
   Basic information about the user. Users can also press Logout to sign out of the app.

**Recipe** -

This is the section to view the complete recipe with procedure. If the user is the recipe maker himself then user will also be show 2 options - edit and delete.

Every user will be able to see title, favourite button, Image (default image if none provided), Time, Description, Procedure.

The owner will be able to update the recipe if he clicks on edit and then update. Evey procedure step has a separate text area to edit. They can also add any step after any step or remove any step or change any ingredient or anything else.

Clicking on delete will lead the user to Home page.

Backend

The backend is written in django. It could have been in Javascirpt or FastAPI. Django was chosen as it provides high level functionalities, structure and inbuilt ORM. Only for documenting the various endpoints, I have installed swagger and django-rest-framework (swagger works with that).

The backend has 2 apps - User and Recipe (everything related to recipe)

**User** -

Login and Register endpoints. They have field validations and use crypto library to decrypt the encrypted password sent by the frontend

**Recipe** -

This section is major part of the backend. It has 4 models which are interconnected to each other.

Since this recipe app has many endpoints to avoid confusions I have created 4 different views file instead of 1. These are inside the recipe/views folder and they specify the respective views for the recipe endpoints. These are crud, favourite, ingredients, search

1. Recipe -   
   This is the main recipe model which contains information about the recipe along with recipe image url and information about the user who created the recipe. For local development django could server the static images, so I had set the recipe image to IMAGE FIELD, but that created problems in vercel/render deployment. So switched it to image url. This object has many to many relation with Ingredient table.
2. Ingredient -  
   This table contains ingredients and may be associated with multiple recipes. Whenever any recipe gets updated, these are checked to see if any dangling ingredient are present.
3. Procedure -   
   This table contains procedures or steps of procedure. Each procedure has a step description, the order of this step-in recipe, and is linked to one recipe. This was modelled so that there is a hierarchy in the procedure and we will be able to maintain the sequence or change this if necessary.
4. Favourite -   
   This table contains (user, recipe) pairs to show that a user has favorited a particular recipe.

Endpoint-specific documentation is done on swagger and can be accessed on [Your API (foodz.onrender.com)](https://foodz.onrender.com/redoc/) or [Your API (foodz.onrender.com)](https://foodz.onrender.com/swagger)   
  
**Challenges**

While the development was pretty straightforward. I had to think about the following -

1. UI - Since no particular UI style was defined, I selected a minimal and straightforward design to achieve the desired result.
2. Recipe model to have Image URL instead of ImageFile. This was done as it is better in case of deployment and no cloud storage. (For virtual machines I would have chosen to use Minio as cloud storage and included the link in the image URL as a field).
3. Procedure - To model the hierarchy of steps in the procedure I decided to make it a separate table and reference it with the recipe table.
4. Encryption - Used the same encryption keys to encrypt and decrypt the password, instead of sending a hashed password for security reasons.
5. Dividing the Home view into 4 different independent sections. The Backend is requested whenever the user clicks on the header of the section. This was also done to minimize backend requests as explained in the favorites section.
6. While local development was very fast, there were problems when vercel was used for backend hosting. The requests were taking too long to be answered before which the API timeout was expired. I optimized the endpoints a little bit by making bulk transactions and atomic transactions (As only some part of recipe was uploaded, the procedure was left out). But the problem was with vercel as render is working fine.