

Assignment_ID: assignment_category_0005

Restaurant Management Website

About the Company:

We are a well-known restaurant brand that is dedicated to providing our customers with exceptional food experiences. We not only offer delicious meals prepared by top chefs, but we also combine a rich cultural and sensory experience. Our goal is to surpass customers' expectations with a dining experience that goes beyond just food.

From beautifully designed interiors to carefully chosen menus that showcase the best of local and international flavors, we strive to offer more than just a meal. We aim to create lasting memories for our guests by providing excellent service in a sophisticated and lively setting.

Job Overview:

We are seeking a skilled MERN Stack Developer to develop a full-stack Restaurant Management website. This person will be crucial in redesigning our platform, using MongoDB, Express.js, React.js, and Node.js to create a user-friendly and engaging experience for our customers and staff. The candidate will be responsible for developing features that enhance the restaurant's online presence, improve customer interaction, and streamline internal management processes. You are required to create a website where a user can do the **following tasks:-**

- 👉 See All Food Items
- 👉 See Single item
- 👉 Add A Food item
- 👉 Delete An Item
- 👉 Modify An Item

Make sure your website design is unique. Visit [ThemeForest](#), [Dribble](#), [Behance](#), etc. to get some ideas. You can explore component libraries other than DaisyUI. Remember, a unique project will add more value to your portfolio.

🚩: 0 [If we make any changes we will mention them here]

Key Rules

1. Commits & readme

- 📌 Minimum 18 meaningful git commits on the client side.
- 📌 Minimum 10 meaningful commits on the server side.
- 📌 Add a meaningful readme.md file with the name of your website and a live site URL. Include a minimum of five bullet points to feature your website. Write your selected category's name here.

2. Make HomePage responsive

- 📌 Make the Homepage fully responsive (mobile, tablet, and desktop)
- 📌 if possible make the whole website responsive.

3. Fix your Reload & Error Issue

- 📌 If you reload the **protected/private** routes (after login), it will not redirect the user to the login page.
- 📌 If you reload the website, it will not show the site not found. (Netlify & surge).

4. Environment Variables

- 📌 Use the Environment variable to hide the Firebase config keys and MongoDB credentials.

5. Website Naming

- 📌 Give your website a name. The website title will be changed according to your route/page. Suppose your website name is PHero. Then, on the 'login' router/page, your website title will be 'PHero | Login'.

💡 hints: explore the [react-helmet](#) npm package

6. Lorem ipsum :

📌 Don't use any Lorem ipsum text; you can not use the default alert to show any error or success message.

Main Requirements

1. Focus on making the website visually appealing. Ensure that
 - Color contrast is pleasing to the eye
 - The website does not have a gobindo design
 - The website has proper alignment and space
 - If needed, customize the design of any component you are taking from any component library. (For example, you are using Daisy UI & have taken a card component from Daisy, if needed, customize the styling of the card to make it reasonable rather than just copy & paste it.)

📝 Note: Your website can not be related to your previous assignments' layout/design or any practice project shown in the course modules or our conceptual sessions. Ex: You can't copy any design or similar functionality/ layout of

- **Career hub**
 - **Dragon news**
 - **Coffee store**
 - **Car Doctor**
 - Any of your previous assignments or conceptual session projects.
- If any similarities are found, you will get **zero(0)** as a penalty.
2. Make sure to keep the **navbar** and **footer** on all the pages except on the **404 page**. Create a reasonable and meaningful footer. (including website **logo**, **name**, **copyright**, some contact information, social media links, address, etc.)

3. For all of your **CRUD** operations and **login/register**, you have to show a **toast/alert** (*do not use browser alert*).

Navbar

Your website should have a navbar with the following information:

1. **Website name/logo**
2. **Home,**
3. **All Foods,**
4. **Gallery,**
5. **Conditional login/logout**
6. **My Profile.**

Note: **My profile** picture on the navbar is conditional based on **login**. If the user is logged in, the navbar will show the **profile picture and logout button**; otherwise, it will show the **Login button**.

Homepage

👉 **Banner section:** A slider/banner/ a meaningful section. Inside the banner, there will be a Heading Title, a Short Description, and a button that will redirect the user to the **All Foods** page.

👉 **Top Foods section:** Show 6 top-selling Food Items depending on the number of purchases of a food item: **(see single food page to know about purchase count)**

including the following pieces of information:

- ❖ Food Name
- ❖ Food Image
- ❖ Food Category
- ❖ Price
- ❖ Details Button

✳️ On clicking the **details button** will navigate to the Single Food Page.

👉 **see all button**: Below the 6 cards, there will be a see all button that will redirect the user to the **All Foods** page.

👉 **Extra section**: Add **2(Two)** relevant and attractive sections except the **nav**, **banner**, **footer**, and **Top Food section**.

All Food Page 🥑

👉 **Page Title**: The background and the center of the section will have the page name. [demo](#)

👉 **Food Cards**: Here you need to show all the food items stored in your database.

👉 **Search Functionality**: on the top of the food cards section you have to implement a search functionality based on the food name. You can implement a search option using the **backend**.

👉 **Each Card**: Each card Item will have the following information:

- Food Name
- Food Image
- Food Category
- Price
- Quantity (**see challenging part** 👉)
- **Details Button**


✳️ On clicking the **details button** will navigate to the Single Food Page.

Single Food Page 🍕

Show details information about a single food item which will include the following:

- Food Name
- Food Image

- Food Category
- Price
- Made By (Who added the food)
- Food Origin (Country)
- A short description of the food item (for example: ingredients, making procedure, etc.)
- **Purchase button**

 **Note:** clicking the **purchase button** will redirect users to the food **purchase page**. This page will be a **Private Route**. if the user is not logged in then it will redirect him/her to the login page and after successful login the user will redirect to the food purchase page.


👉 **Count purchase number:** Count the number of orders every time a user purchases food. You can use the MongoDB [\\$inc\(\)](#) operator for this. By default, the count will be zero(0).

Example: while creating a new product store the count property and the value will be 0.

Food purchase Page

This page will have a form containing the following information:

- Food Name
- Price
- Quantity
- Buyer Name (**Read-only**. This will be picked from the logged-in user's information)
- Buyer Email (**Read-only**. This will be picked from the logged-in user's information)
- Buying Date (current time user purchases the product. You can use **Date.now()**)
- A button to **purchase** the food item.

 **Note:** on clicking the **purchase button** the information will be stored in the database. Also, On a successful order, you have to show a toast/alert (do not use browser alert).

Gallery Page


Page Title: The background and center of the page should display the title prominently. [demo](#)

Gallery Section: Show all images added by the user in a gallery section. When a user hover over the image, an overlay will be visible on the image with the user's name and feedback.

 For ideas visit this but don't copy this design [link](#)

Add Button: Include an "Add" button on the page for user interaction. When the "Add" button is clicked, a modal should open. Within the modal, include a form where the user can submit their experience along with an image. This form should include fields for:

- User's name. (**read-only**)
- Feedback or experience description.
- Image url.

 Note: if the user is not logged in the modal won't be open. It will redirect the user to the login page. After successfully logging in it will redirect the user to the gallery page.


Login and registration systems


Registration Page

Create a Registration page will have the Email/Password form having the following fields:

- Name

- Email
- Photo URL
- Password



 Note: Make sure you add the user information in a MongoDB collection after successful registration and use them as per your need.

 *Do not enforce the email verification method, as it will inconvenience the examiner. If you want, you can add email verification after receiving the assignment result.*

Login Page

When a user clicks on the login button, they will be redirected to the login page which has the following:

- Email/Password
- A Social Login System
- A link that will help the user toggle the login and registration page

 Both Registration and Login pages, display relevant error () messages when necessary.

My profile

On clicking on my profile (*it can be implemented using a dropdown when clicked on the user profile image.*) there will be three routes:

- 1. My added food items**
- 2. Add a food item**
- 3. My ordered food items**

My added food items Page 🍒

In this route, you have to show all food items added by the currently logged-in user (💡 hints: You have to filter data based on the user's email). You can show all the food items in tabular/card. Each row/card will have:

- Some Food info (example: food img, name, price, etc)
- an **update** button/icon.

📝 clicking the **update button/icon** will redirect to the update page or open a modal. There will be a form that has the product info and an update button. When clicking the update button the product info will be updated. Don't let other users delete your added food items.

Add A food item Page 🍉

This page will have a form having the following fields:

1. Food Name
2. Food Image
3. Food Category
4. quantity
5. Price
6. Add By (name & email: this info added from currently logged-in user.)
7. Food Origin (Country)
8. A short description of the food item (ingredients, making procedure, etc.)
9. Add Item Button

📝 on clicking the **add button** the information will be stored in the database. on successfully adding a food item, you have to show a toast/alert (*do not use browser alert*)

My purchase Page 🍉

In this route, show all food items ordered by the logged-in user. You have to filter the data based on the logged-in user email. You can show all the food items on the table/card. Each row/card will have:

1. Some Food info (example: food img, name, price, added time, food owner, etc)
2. a delete button/icon that will help the user delete the ordered item from the list

📌 Buying Date: You do not need to create a field for taking date as input. You'll generate the buying date using the JS Date method and set it in your purchase info object before inserting it in the database.

💡 **hints:** you can use JS method: `Date.now()`

404 Page ❌

Create a 404 page. Add any interesting jpg/gif on the 404 page. Do not add header & footer on this page. Just add a jpg/gif & a Back to Home button. The Back to Home button will redirect the user to the home page.

Challenge Requirements

Single Food Quantity 💰

📌 You have to implement a feature on the food purchase page where the feature will be: if the available quantity of a food item is **zero** then you have to show a message to the buyer that he/she can not buy that item because that **item is not available**. Also, the purchase button will be disabled for the user.

📌 Also the buyers can't buy more than the available quantity. (assume that a food item has 20 quantities then a user won't be able to buy more than 20 quantities).

📌 Don't let the user purchase his/her own added food items.

Authorize your server routes with the JWT token. 🗝️

Upon login, you will create a JWT token and store it on the client side. You will send the token with the call and verify the user. Implementing 401 and 403 is optional. Ensure you have implemented the JWT token, create a token, and store it on the client side for both email/password-based authentication and social login. You must implement JWT on your private routes.

Optional (But Highly Recommended)

Implement any two tasks from the following optional list:

- 👉 Add a spinner when the data is in a loading state. You can add a gif/jpg, use any package, or customize it using CSS.
- 👉 Explore and implement any of the animations from the Framer Motion.
- 👉 Explore and implement Tanstack query mutations in your api.
- 👉 Add multiple filtering systems on all food items pages. The filtering system should be implemented on the server side. Think about MongoDB \$and, \$or operators.
- 👉 **pagination**: on all Food pages => At the bottom of the cards section you need to create a pagination. First, you have to load 9 cards of data from the database and then you will show the others based on the page number. Implement backend pagination.
- 👉 **Gallery section**: On this page, you can implement infinite image scrolling. Initially, there will be 12 card images. After scrolling down the rest of the image will be shown. Also, add simple animation when the image loads.

What to submit

- ★ Your assignment category ID/variant.
- ★ generated client-side GitHub repository
- ★ generated server-side GitHub repository
- ★ Generated live site link.

Some Guidelines:

1. Save time on the website idea. Just spend 15–20 minutes deciding, find a sample website, and start working on it.
2. Do not waste much time finding the right image. You can always start with a simple idea. Make the website and then add different images.
3. Divide the whole project into several parts and work on one task simultaneously.
4. Stay calm, think before coding, and work sequentially. You will make it.
5. Be strategic about the electricity issue.
6. use chatGPT to generate JSON data.