Easy Trader Joe’s College Meals

Final Report

CSC 240

Claudia Porto

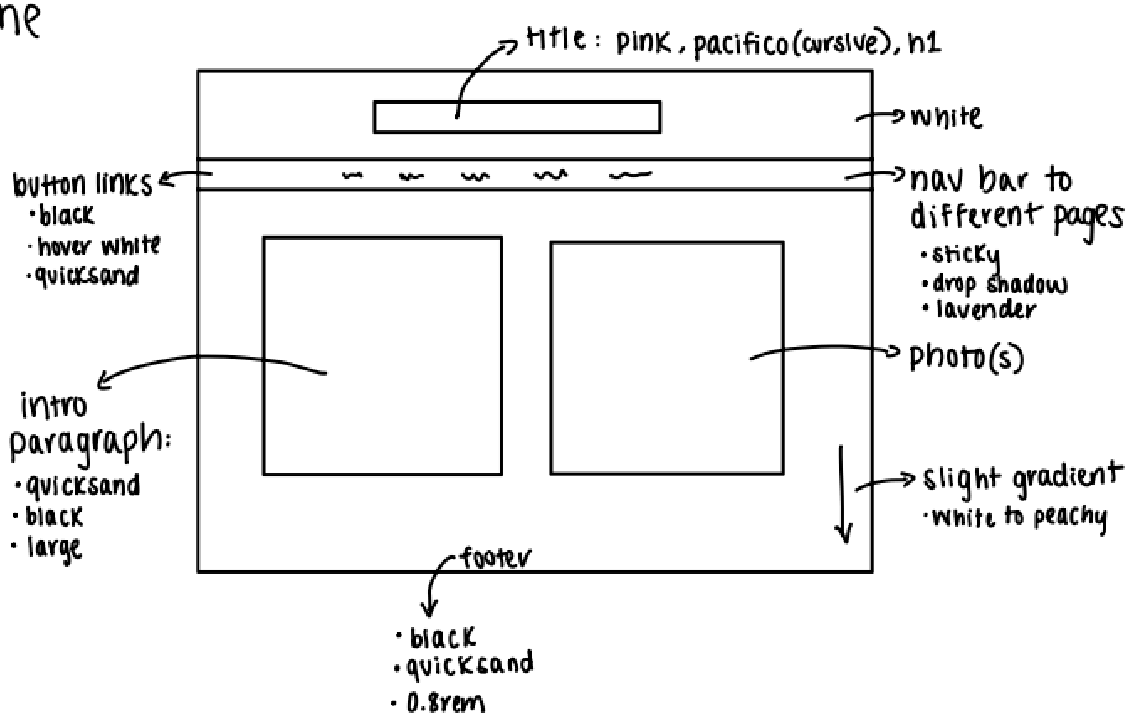
6/6/2025

The purpose of “Easy Trader Joe’s College Meals” is to make healthy, affordable eating feel doable for busy college students. As a student myself, I know how tiring it can be to come home after a long day and still have to figure out what to eat. This site was created out of that exact feeling, the constant “what’s for dinner?” dilemma, and a love for Trader Joe’s as a place that always has something quick, budget-friendly, and fun to cook with.

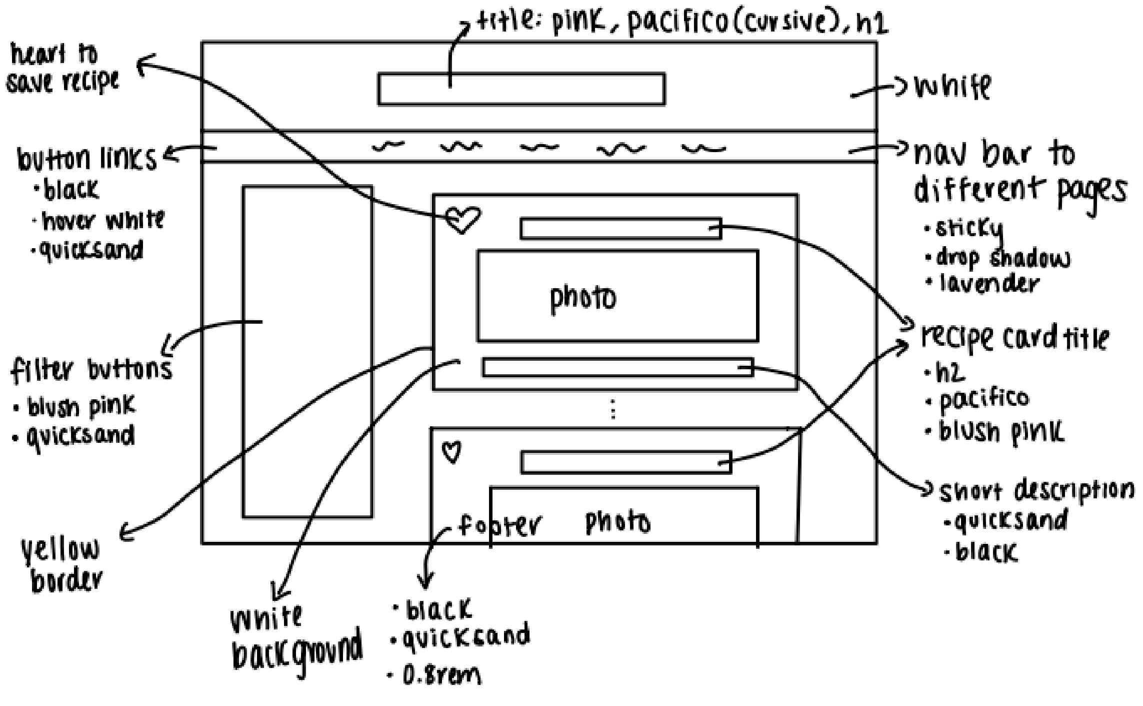
The goal isn’t to turn anyone into a chef, but to help students with limited time, money, or cooking experience find easy meals they’ll actually want to make. Whether you’re cooking in a dorm or off-campus, the site offers simple, low-pressure meal ideas and supports students who may be cooking for the first time. By focusing on convenience and variety, it encourages students to eat better without overthinking it.

The site features several pages that work together to help college students find easy, affordable meals. The Homepage introduces the overall purpose of the site in a friendly, welcoming tone, while the About Page shares the personal story behind the project to help users connect with its purpose. On the Recipes Page, students can browse a curated list of dorm-friendly meals, using filters and a search bar to sort by ingredients, meal type, or dietary preferences or needs. Logged-in users can save favorites on the My Recipes page for quick access. The Shopping Cart generates editable grocery lists from selected recipes, making meal planning more convenient. The Contact Form allows for feedback and questions, and full user authentication ensures secure login and access to personalized features across devices. Together, these elements create a simple, student-centered experience.

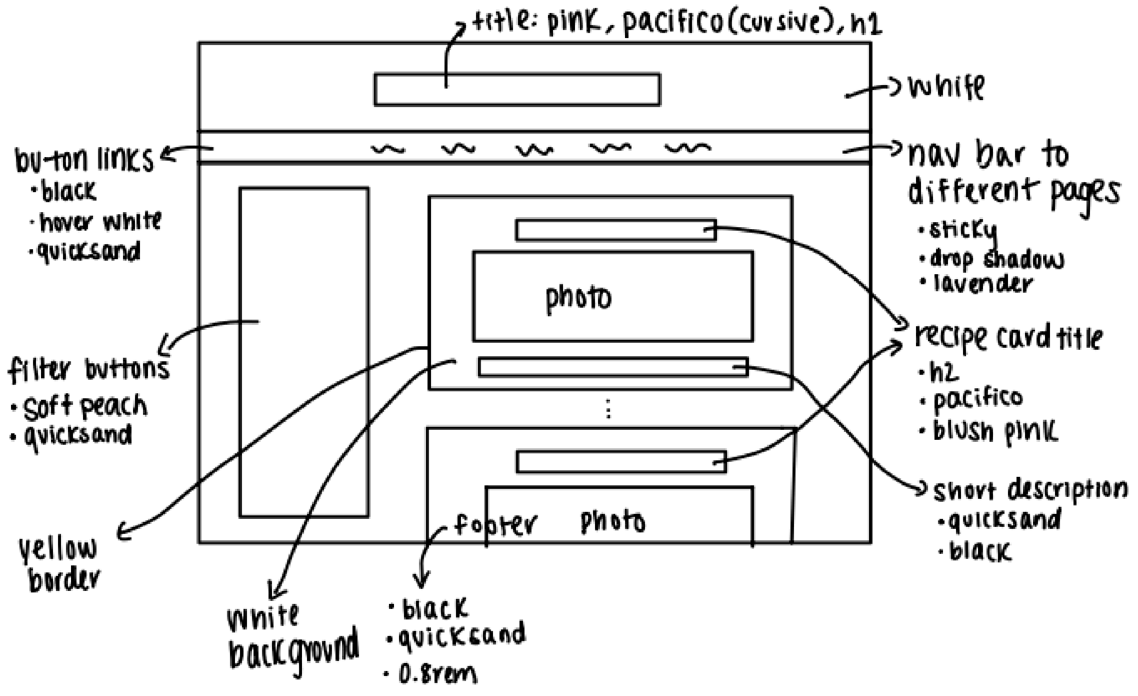
Home



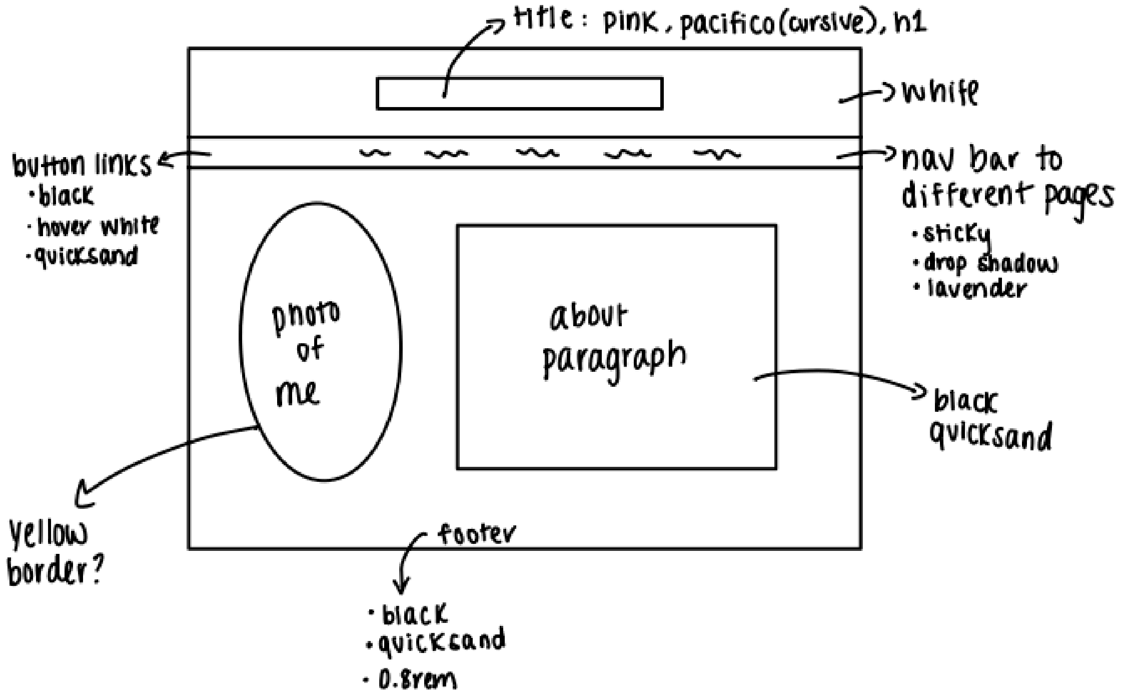
Recipes



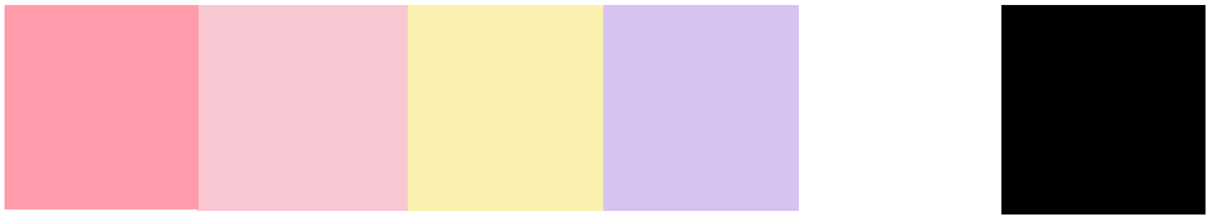
My Recipes



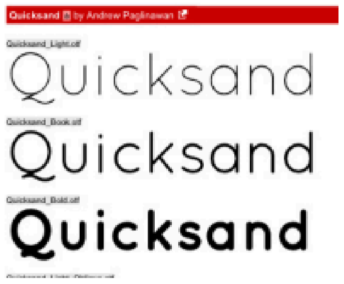
About

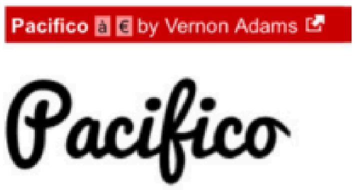


Color Palette



fonts:





The planning phase began with identifying features to help college students find quick, healthy, and affordable meals. I sketched wireframes for four core pages (Home, Recipes, My Recipes, and About), each with a clear purpose: to introduce the purpose of the site, display recipe options with filters, allow students to save favorites, and share the story behind the project, including a contact form. During development, I implemented a fixed vertical sidebar on the Recipes page to keep the search bar and category filters accessible while scrolling. Accessibility and usability were top priorities, with semantic HTML, alternative text for images, and legible fonts integrated throughout. Although the initial plan focused on recipe saving, I added a Shopping Cart feature to let students build and manage grocery lists. I also implemented full user authentication, allowing users to register, log in, and access saved data across devices.

To keep the styling clean and maintainable, I used ID selectors for unique elements and class selectors for reusable components like .recipe-card, .category-buttons, and .about-content. The layout relies on CSS Flexbox and Grid for consistent spacing across the screen. Sticky and fixed positioning keeps important elements like the header and #main-title visible while scrolling. Active navigation links highlight with white text, giving users clear visual feedback as they move through the pages of the site.

The visual design emphasizes a soft, welcoming aesthetic with rounded corners, pastel accents, and subtle hover effects to make interactions feel light and friendly. On the Recipes page, a fixed vertical sidebar keeps category filters and the search bar accessible while scrolling. Selected filter buttons turn lavender with white text and a blush-pink border, making the current view easy to identify. Each recipe card includes an emoji tag that signals the recipe's category, adding personality and improving scanability.

To support confident user interaction, the interface includes brief confirmation messages when recipes are saved or ingredients are added to the cart. Alerts and prompts appear if users try to duplicate or delete items, preventing accidental changes.

The pastel color palette, defined as CSS variables in the :root selector, includes butter yellow, blush pink, lavender, soft peach, and white, ensuring a cohesive and easily updatable theme. Typography combines the playful 'Pacifico' font for headers with the clean, rounded 'Quicksand' for body text. Accessibility remains a priority, with semantic HTML tags, descriptive alternative text, and keyboard navigation included to accommodate a wide range of users.

This website was developed using the MERN stack, a popular full-stack JavaScript framework consisting of MongoDB, Express.js, React, and Node.js, though React is not yet implemented in this version of the website. The current frontend uses HTML, CSS, and JavaScript, but it is structured in a way that would allow easy migration to React as the website grows.

The backend, built with Node.js and Express.js, handles server logic, routing, and communication with the database. For data storage, the project uses MongoDB Atlas, a cloud-hosted NoSQL database that supports flexible and scalable storage of users, recipes, shopping lists, and contact form submissions. The backend interacts with MongoDB through the native MongoDB Node.js driver to perform all CRUD operations.

Development tools included npm for managing dependencies and running the backend, and Postman for testing API endpoints and verifying request/response behavior. The MongoDB Atlas dashboard was used throughout development to monitor collections and manage real-time data. Together, these technologies support future expansion into a full MERN-based architecture.

Folder/File Structure:

```
/Trader Joes Recipes - 4/
│
├── views/
│   ├── index.html
│   ├── about.html
│   ├── recipes.html
│   ├── myrecipes.html
│   └── shoppingcart.html
├── backend/
│   ├── server.js
│   ├── package.json
│   ├── package-lock.json
│   ├── .env
│   ├── node_modules/
│   ├── controllers/
│   │   ├── userController.js
│   │   ├── recipeController.js
│   │   ├── contactFormController.js
│   │   └── shoppingListController.js
│   ├── routes/
│   │   ├── userRoutes.js
│   │   ├── recipeRoutes.js
│   │   ├── contactRoutes.js
│   │   └── shoppingListRoutes.js
│   └── middleware/
│       └── authMiddleware.js
├── Images/
└── README.md
└── auth.js
└── cart.js
└── config.js
└── contact.js
└── recipes.js
└── style.css
```

The project is structured to clearly separate frontend and backend logic, keeping the codebase clean, modular, and easy to navigate. At the root of the /Trader Joes Recipes - 4/ directory are all frontend files, backend folders, and configuration files.

All HTML pages are located in /views/ folder. These include index.html (homepage), about.html(background), recipes.html (meals), myrecipes.html (saved favorites), and shoppingcart.html (grocery list). These are styled using style.css and powered by modular JavaScript files: recipes.js handles recipe filtering and saving, cart.js manages the shopping cart, auth.js handles login and registration, contact.js handles the contact form, and config.js provide supporting functionality. Static image assets are stored in the /Images/ folder, and README.md provides documentation and setup instructions.

The /backend/ folder contains all server-side logic. Server.js launches the Express server and sets up route handling. Configuration settings are securely stored in the .env file. Inside /controllers/, each file defines the logic for handling specific types of data: userController.js, recipeController.js, contactFormController.js, and shoppingListController.js. These are connected via route files in the /routes/ folder via files like userRoutes.js, recipeRoutes.js, contactRoutes.js, and shoppingListRoutes.js. The /middleware/ folder included authMiddleware.js, which verifies user tokens and secures protected routes. Project dependencies are managed through package.json and stored in the node_modules/ folder.

The frontend and backend communicate through a structured API architecture. When users perform an action, such as submitting the contact form, saving a recipe, or updating their shopping list, JavaScript sends HTTP requests to the appropriate backend endpoints. For example, submitting the contact form triggers a POST request to /api/contact, which is routed through contactRoutes.js and handled by contactFormController.js, storing the message in MongoDB Atlas. The same process applies to recipes and shopping list interactions using routes like /api/recipes or /api/shopping-list.

All API responses return JSON, allowing the frontend to update dynamically without reloading the page. With user authentication in place, protected routes require a valid token, which is verified using authMiddleware.js, to ensure personalized and secure data access. API endpoints were tested using Postman to validate functionality and reliability.

The following are some code snippets that connect the MongoDB to the frontend.

MongoDB Connection:

```
async function startServer() {
  try {
    await client.connect();
    console.log("Connected to MongoDB Atlas");

    db = client.db('PortoCSC240');

    // Attach db + ObjectId to every request
    app.use((req, res, next) => {
      req.db = db;
      req.db.bson = { ObjectId };
      next();
    });

    // Routes
    app.use('/api/recipes', recipeRoutes);
    app.use('/api/users', userRoutes);
    app.use('/api/shopping', shoppingListRoutes);
    app.use('/api/contact', contactRoutes);

    app.listen(PORT, () => {
      console.log(`Server running at http://localhost:${PORT}`);
    });

  } catch (err) {
    console.error("MongoDB connection error:", err);
    process.exit(1);
  }
}
```

Sample API Route:

```
const express = require('express');

const router = express.Router();

const { submitForm } = require('../controllers/contactFormController');


router.post('/', submitForm);

module.exports = router;
```

Frontend Fetch Example:

```
if (isValid) {
      fetch("http://localhost:5000/api/contact", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          name: name.value.trim(),
          email: email.value.trim(),
          message: message.value.trim()
        })
      })
      .then(response => response.json())
      .then(data => {
        alert(data.message || "Message sent!");
        form.reset();
      })
      .catch(error => {
        console.error("Error sending message:", error);
        alert("There was a problem submitting your message.");
      });
    }
```

Manual testing was conducted across all main pages (Home, Recipes, My Recipes, and Shopping Cart) to check navigation, filters, and search functionality. The contact form was tested for input validation, ensuring proper error messages appeared for empty or incorrectly formatted fields and that confirmation alerts were shown upon successful submission.

Backend and API testing was performed using Postman to verify endpoints like /register, /login, /favorites, and /contact with both valid and invalid data. This ensured the server correctly handled requests and returned appropriate responses. Additional test cases included user login and registration, saving/removing favorites, generating shopping lists, and submitting forms.

MongoDB Atlas was also tested to confirm that submitted data appeared in the correct collections. This validated that recipe, user, and contact information were being stored and retrieved correctly from the database.

One of the main issues I faced was setting up the login system using authentication middleware with JSON Web Tokens (JWT). While the initial structure was in place, getting the middleware to properly verify tokens and protect user routes was challenging. Another issue was ensuring mobile responsiveness, specifically with recipe cards and images not resizing correctly on smaller screens. This remains part of future work. I also encountered troubles early on when connecting to MongoDB Atlas, particularly handling authentication credentials and asynchronous calls. These issues were eventually resolved by modularizing the backend, which made the logic more manageable and easier to debug.

Overall, the CSS styling and site structure came together smoothly. I was able to create a clean, intuitive layout with a cohesive color palette and responsive interactions. Features like the contact form, category filters, shopping cart, and favorites all worked as intended. One of the most rewarding aspects of the project was building a working backend database that could store and retrieve real user data. It gave the website a real-world depth and a sense of purpose.

Throughout this process, I learned the importance of planning before coding. Sketching layouts and mapping out functionality in advance helped clarify my overall goals and saved time during the development phase. I also learned the value of keeping code modular, organizing logic into controllers, routes, and middleware for better scalability and easier debugging. Another useful skill was using browser tools like “Ctrl + F” to quickly find and style HTML elements in both the code and the Document Object Model.

This project gave me the opportunity to build a full-stack web application that feels both personal and useful. While there’s still room for improvement, such as making the site fully mobile-responsive, enhancing contrast for accessibility, adding a “forgot password” option to the user login, and expanding the recipe collection, I’ve built a strong foundation. Looking ahead, I’d love to allow users to submit their own recipes, connect the app to the official Trader Joe’s website or API for real-time ingredient data, and further improve accessibility. Overall, this project taught me how to bring together design, development, and user needs into a cohesive and meaningful experience.

References

https://developer.mozilla.org/en-US/

https://expressjs.com/

https://fonts.google.com/

https://github.com/auth0/node-jsonwebtoken

https://github.com/dcodeIO/bcrypt.js/

https://github.com/motdotla/dotenv

https://learning.postman.com/

https://mongodb.github.io/node-mongodb-native/

https://nodejs.org/docs/latest/api/

https://restfulapi.net/

https://www.mongodb.com/docs/atlas/

https://www.mongodb.com/docs/drivers/node/current/security/authentication/

https://www.w3schools.com/

https://www.w3schools.com/nodejs/nodejs_mongodb.asp

https://youtu.be/RcxdF3Lzoac?si=l8eWjRp_dngliQsH

https://youtu.be/sx3Lf2EaEEQ?si=B6oLFTC3vt4isP6Y