My Dorm Store
System Design Document

Frontend CRC

Class name: RegisterForm	
Sub Classes: NA Parent Classes: NA	
Responsibilities: ● Render registration form UI ● Validate user input before submission ● Send registration data to backend API	Collaborators: ● LoginForm ● Authentication

Class name: LoginForm	
Sub Classes: NA Parent Classes: NA	
Responsibilities: ● Render login form UI ● Send login data to backend API ● Display login error messages	Collaborators: ● RegisterForm

Class name: Navbar	
Sub Classes: NA Parent Classes: NA	
Responsibilities: ● Display navigation links ● Search for categories	Collaborators:

Class name: ProductList	
Sub Classes: NA Parent Classes: NA	
Responsibilities: ● Fetch and display product data ● Allow browsing through products	Collaborators: ● ProductCard ● ProductListPage

Class name: Cart	
Sub Classes: NA Parent Classes: NA	
Responsibilities: ● Manage selected products ● Add/remove items	Collaborators: ● CheckoutPage ● ProductList

Class name: CheckoutPage	
Sub Classes: NA Parent Classes: NA	
Responsibilities: ● Show cart summary ● Submit final order	Collaborators: ● Cart

Class name: Contact us/SupportPage	
Sub Classes: NA Parent Classes: NA	
Responsibilities: ● Render static informational content	Collaborators:

Class name: OurStoryPage
Sub Classes: NA Parent Classes: NA

Responsibilities: ● Render the static “Our story” page content ● Display information about the background and blogs of the company	Collaborators:

Backend CRC

Class name: authentication	
Sub Classes: NA Parent Classes: NA	
Responsibilities: ● Handle register/login routes ● Validate credentials ● Return JWT/token or error	Collaborators: ● PostgreSQL ● jsonwebtoken ● bcrypt

System Interactions and Assumptions

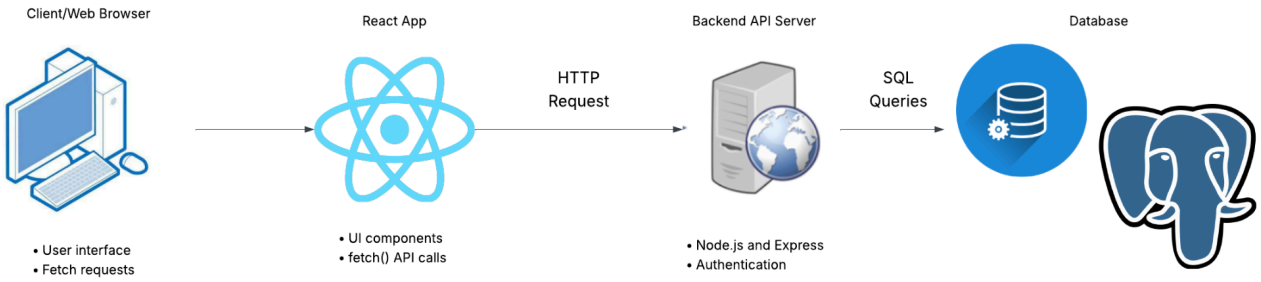
- Operating System Assumptions
    - Development and testing are performed on macOS
    - The system is assumed to be compatible with other Unix-based systems (e.g., Linux) and cross-platform in deployment (i.e., can be run on Windows with Node.js and PostgreSQL installed)
- Database Assumptions
    - The system uses PostgreSQL for data storage
    - Tables include:
        - users: for account credentials (email, password)
        - products: for product listings
        - carts: for user shopping cars
- Compiler/Runtime Assumptions
    - Backend is developed with Node.js using ES module syntax
    - Frontend is built using React and uses JSX/JavaScript

    - The system runs using:
        - npm run dev or npm start for frontend
        - node server.js for backend
- API Interaction
    - Frontend communicates with backend over HTTP using fetch() API:
        - POST /registerUser - to register new users
        - POST /loginUser - to log users in
    - All requests are sent from http://localhost:3000 to backend on http://localhost:5000
- Session/Auth Assumptions
    - JWT is used for generating login tokens
    - credentials: “include” is set on the frontend fetch calls
- Other Assumptions
    - Assumes developer access to:
        - .env file for PostgreSQL credentials
        - A running PostgreSQL instance on localhost:5432
        - Correct folder structure

System Architecture

Architecture Type: Three-Tier Architecture (Client-Server Model)

The system is built on a three-tier architecture, which separates the application into three distinct layers: Frontend (Client), Backend (Server), and Database (Data). We chose this architecture for its strong separation of concerns, which clearly divides the application into presentation, logic, and data layers. This structure improves maintainability, making the code easier to manage, test, and extend as the project evolves. It also improves scalability, as each tier (frontend, backend, and database) can be scaled independently to handle increased load. Moreover, the architecture ensures that sensitive data such as passwords is handled only by the backend server, reducing exposure. Lastly, it promotes reusability, allowing the same backend APIs to support multiple types of clients, such as web and mobile applications in the future.



Components Description

- **Client/Web Browser**
  - HTML/CSS, JavaScript (React)
  - Responsibility:
    - Displays the user interface
    - Sends Http requests to the backend
    - Handles user input for registration, login, browsing, and cart interactions
- **React App**
  - React
  - Responsibility:
    - Provides UI components like Login, Register, Product List, and Cart
    - Sends authenticated requests to the backend
- **Backend API server**
  - Node.js, Express.js
  - Responsibility:
    - Handles Http requests from the frontend
    - Performs validation, hashing (bcrypt), and authentication (JWT)
    - Communicates with PostgreSQL using SQL queries via pg package
- **Database**
  - PostgreSQL
  - Responsibility:
    - Stores user accounts, product details, cart data, etc
    - Executes SQL queries to insert or retrieve user and product info