PROJECT

Tech Stack: React.js, Node.js, Express.js, PostgreSQL, Redis, Prism

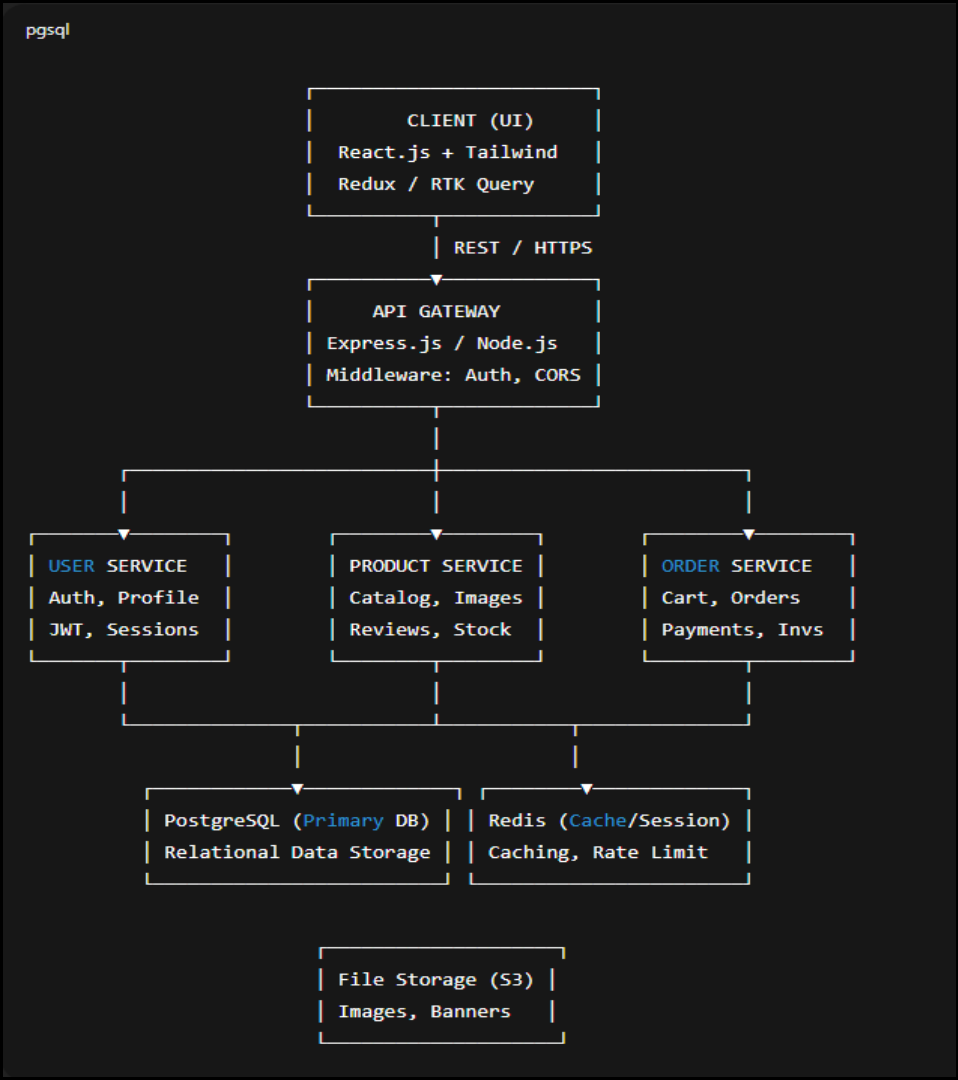
Backend Folder Structure:

```
/backend
  ├── /prisma
  │    ├── schema.prisma
  │    └── migrations/
  ├── /src
  │    ├── routes/
  │    ├── controllers/
  │    ├── services/
  │    └── db.js
  ├── package.json
  └── .env
```

🧩 1. SYSTEM OVERVIEW

🎯 Project Goal

Develop a **full-featured e-commerce platform** that allows users to browse products, manage carts, place orders, and handle payments securely, with an admin dashboard to manage inventory, users, and sales reports.



👤 User Stories for `CUSTOMER` Role

🛒 Account & Authentication

- As a **customer**, I want to **register an account** with my email and password so that I can access the shop.
- As a **customer**, I want to **log in securely** so that I can manage my orders and cart.

🛍️ Browsing & Shopping

- As a **customer**, I want to **browse products by category** so that I can find items I'm interested in.
- As a **customer**, I want to **view product details**, including price, description, stock, and images, so I can decide what to buy.

🛒 Cart Management

- As a **customer**, I want to **add products to my cart** so that I can purchase them later.
- As a **customer**, I want to **update the quantity** of items in my cart so that it reflects what I want to buy.
- As a **customer**, I want to **remove items from my cart** so I can manage my planned purchases.

💳 Order & Payment

- As a **customer**, I want to **place an order** with the items in my cart so that I can complete my purchase.
- As a **customer**, I want to **pay for my order** using a payment provider so that I can finalize the transaction.
- As a **customer**, I want to **see the status of my order** (e.g., placed, shipped, delivered) so that I know when to expect it.
- As a **customer**, I want to **view my order history** so that I can track past purchases.

⭐ Reviews

- As a **customer**, I want to **leave a review** for a product I've purchased so I can share my experience.
- As a **customer**, I want to **rate a product** so that others can benefit from my opinion.

🦍 User Stories for `ADMIN` Role

👨‍💼 User & Role Management

- As an **admin**, I want to **view all users** so that I can manage the user base.
- As an **admin**, I want to **change user roles** so I can promote or demote users (e.g., admin, customer).

📦 Product & Category Management

- As an **admin**, I want to **add new products** with details like name, price, stock, and images so that they appear in the store.
- As an **admin**, I want to **edit existing products** so I can update their information.
- As an **admin**, I want to **delete products** that are no longer for sale.

- As an **admin**, I want to **create and manage product categories**, including parent-child relationships, so that the store stays organized.

📦 Order & Payment Monitoring

- As an **admin**, I want to **view all orders** so I can manage the fulfillment process.
- As an **admin**, I want to **update the status of orders** (e.g., mark as shipped or delivered).
- As an **admin**, I want to **view payment statuses** (pending, paid, failed) so that I can verify transactions.

🔍 Review Moderation

- As an **admin**, I want to **view all product reviews** so I can monitor customer feedback.
- As an **admin**, I want to **delete inappropriate reviews** so the site maintains quality content.

🔁 System Stories (Technical)

These are system-oriented stories useful for backend/API devs.

🗂️ Schema & Data Integrity

- As a **developer**, I want the system to **enforce foreign key constraints** so that data integrity is maintained (e.g., product_id in `OrderItem` must exist in `Product`).
- As a **developer**, I want to **store images as JSON arrays** in products so I can easily retrieve them in APIs.
- As a **developer**, I want the cart structure to support **JSON-based cart items** for quick access and modification.

📝 Summary by Feature

Feature	Users Involved	Key Tables Involved
Authentication	Customer	`User`
Product Browsing	Customer/Admin	`Product`, `Category`, `Review`
Cart	Customer	`Cart`
Orders	Customer/Admin	`Order`, `OrderItem`, `Payment`
Reviews	Customer/Admin	`Review`
Admin Controls	Admin	All

🗃️ 3. DATABASE DESIGN (PostgreSQL)

🧩 Core Tables

1. users

Column	Type	Description
id	UUID (PK)	Unique user ID
name	VARCHAR	Full name
email	VARCHAR (unique)	User email
password_hash	VARCHAR	Encrypted password
role	ENUM('customer','admin')	Access level
created_at	TIMESTAMP	Creation time
updated_at	TIMESTAMP	Update time

2. products

Column	Type	Description
id	UUID (PK)	Product ID
name	VARCHAR	Product name
description	TEXT	Product details
price	DECIMAL	Price
stock	INT	Quantity available
category_id	UUID (FK)	Linked category
images	JSONB	Array of image URLs
created_at	TIMESTAMP	Creation time
updated_at	TIMESTAMP	Update time

3. categories

Column	Type	Description
id	UUID (PK)	Category ID
name	VARCHAR	Category name

Column	Type	Description
parent_id	UUID (nullable, FK)	For subcategories

4. orders

Column	Type	Description
id	UUID (PK)	Order ID
user_id	UUID (FK)	User who ordered
total_amount	DECIMAL	Total price
payment_status	ENUM('pending','paid','failed','refunded')	Status
order_status	ENUM('placed','shipped','delivered','cancelled')	Status
created_at	TIMESTAMP	Order time
updated_at	TIMESTAMP	Update time

5. order_items

Column	Type	Description
id	UUID (PK)	Item ID
order_id	UUID (FK)	Linked order
product_id	UUID (FK)	Linked product
quantity	INT	Number ordered
price	DECIMAL	Item price at time

6. cart

Column	Type	Description
id	UUID (PK)	Cart ID
user_id	UUID (FK)	Linked user
items	JSONB	Product list (productId, qty, price)

7. reviews

Column	Type	Description
id	UUID (PK)	Review ID

Column	Type	Description
product_id	UUID (FK)	Product
user_id	UUID (FK)	Reviewer
rating	INT (1–5)	Rating
comment	TEXT	Review text
created_at	TIMESTAMP	Creation time

8. payments

Column	Type	Description
id	UUID (PK)	Payment ID
order_id	UUID (FK)	Linked order
amount	DECIMAL	Amount paid
provider	VARCHAR	Payment gateway
status	ENUM('success','failed','pending')	Payment status

⚡ 4. REDIS USAGE

- **Session Store**: Maintain logged-in sessions or JWT blacklists.
- **Cache**: Cache frequently accessed products, categories, and cart data.
- **Rate Limiting**: Protect APIs from brute-force attacks.
- **Queue Support (optional)**: For background jobs (email, notifications).

📜 5. SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

5.1 Introduction

The system is an e-commerce web application that allows customers to buy products and admins to manage inventory and sales.

5.2 Scope

- End-user: Browse, add to cart, checkout, pay, view order history.
- Admin: Manage products, categories, users, and orders.
- System: Handle authentication, inventory, order lifecycle, and analytics.

5.3 Functional Overview

1. **User Management**

- Register/Login/Logout
- Profile update
- Password reset

2. **Product Management**
   - CRUD operations (admin)
   - Category management
   - Product search/filter/sort
3. **Cart and Checkout**
   - Add/Remove/Update cart items
   - Address management
   - Order placement and tracking
4. **Payment Integration**
   - Integration with Razorpay / Stripe
   - Transaction verification
5. **Admin Dashboard**
   - Manage orders, customers, and inventory
   - View reports and analytics
6. **Performance Features**
   - Caching with Redis
   - Image storage (AWS S3 / Cloudinary)

⚙️ 6. FUNCTIONAL REQUIREMENTS SPECIFICATION (FRS)

ID	Function	Description	Actor
F1	Register	User signs up with email & password	Customer
F2	Login	Authenticates user & generates JWT	Customer/Admin
F3	View Products	Browse and filter products	Customer
F4	Add to Cart	Add item to cart	Customer
F5	Checkout	Create order and initiate payment	Customer
F6	Payment	Complete payment via gateway	Customer
F7	Order History	View past orders	Customer
F8	Product CRUD	Add, edit, delete products	Admin
F9	Order Management	Update order status	Admin
F10	Reports	Generate sales summary	Admin

🔒 7. NON-FUNCTIONAL REQUIREMENTS

Type	Requirement
Performance	Page load < 2s; API response < 500ms
Security	JWT auth, bcrypt password hashing, HTTPS
Availability	99.9% uptime target
Scalability	Horizontally scalable Node.js and Redis clusters
Maintainability	Modular service-based structure
Usability	Mobile responsive UI
Reliability	Transaction-safe (ACID) order placement

🧠 8. SYSTEM COMPONENTS DETAIL

Frontend (React.js)

- React Router DOM for navigation
- Redux Toolkit for state management
- Axios for API calls
- TailwindCSS / ShadCN for styling
- JWT stored in HttpOnly cookie

Backend (Node.js + Express)

- REST API layer with modular routes
- Authentication middleware
- Sequelize / Prisma ORM for PostgreSQL
- Winston / Morgan for logging
- Redis for caching

Deployment

- **Frontend** → Vercel / Netlify
- **Backend** → Render / AWS EC2 / Railway
- **Database** → PostgreSQL (AWS RDS)
- **Cache** → Redis Cloud / Upstash
- **Storage** → AWS S3 / Cloudinary

📜 9. API STRUCTURE (Example)

Endpoint	Method	Description
/api/auth/register	POST	Create new account
/api/auth/login	POST	Authenticate user
/api/products	GET	Get all products
/api/products/:id	GET	Get product details
/api/cart	GET/POST/DELETE	Manage cart
/api/orders	POST	Create order
/api/orders/:id	GET	Get order details
/api/admin/products	POST	Add product (admin)
/api/admin/orders	GET	List all orders (admin)

🧮 10. FUTURE EXTENSIONS

- Microservices for scaling (User, Product, Order)
- GraphQL API gateway
- Elasticsearch for product search
- Notification service (email/SMS)
- Recommendation engine (AI-based)

🎯 FRONTEND STRUCTURE OVERVIEW

Your frontend (built with **React.js + TailwindCSS + React Router + Redux Toolkit**) will have **two main sections**:

1. **Customer-facing (Storefront)**
2. **Admin Dashboard**

🛍️ 1. CUSTOMER-FACING PAGES (STORE)

🔹 1. Home Page (`/`)

Purpose: Landing page for users.
Contains:

- Hero banner (promotional)
- Featured categories & products
- Offers or best sellers
- Search bar

Learning Concepts: layout, dynamic data fetching, responsive design, lazy loading.

🔹 2. Product Listing Page (`/products` or `/category/:categoryId`)

Purpose: Display all products or filtered by category.
Contains:

- Product grid/list
- Filters (price, rating, category)
- Pagination or infinite scroll

Learning Concepts: query params, API filters, pagination, debounced search.

🔹 3. Product Details Page (`/product/:id`)

Purpose: Show a single product with details.
Contains:

- Product images, description, specs
- Add to Cart button
- Reviews section
- Related products

Learning Concepts: dynamic routes, local state, conditional rendering, related queries.

🔹 4. Cart Page (`/cart`)

Purpose: Manage cart items before checkout.
Contains:

- List of items
- Quantity change, remove
- Subtotal and total
- Proceed to checkout

Learning Concepts: Redux state management, derived state (subtotal), persistence.

🔹 5. Checkout Page (`/checkout`)

Purpose: Capture user address and payment.
Contains:

- Address form
- Order summary
- Payment (Razorpay / Stripe integration)

Learning Concepts: forms, validation, API calls, handling payment responses.

🔹 6. Order Confirmation Page (`/order/success/:id`)

Purpose: Display success message after purchase.
Contains:

- Order summary
- Tracking link
- "Continue shopping" button

Learning Concepts: route params, async data fetch after payment.

🔹 7. Login / Register (`/login`, `/register`)

Purpose: Authentication entry points.
Contains:

- Email/password input
- Redirect after success

Learning Concepts: JWT login, protected routes, conditional rendering (logged-in user).

🔹 8. Profile Page (`/profile`)

Purpose: User dashboard to manage personal info.
Contains:

- Profile info
- Order history
- Password change

Learning Concepts: Auth guard, secured routes, nested routing.

🔹 9. Order Details Page (`/orders/:id`)

Purpose: View single order info.
Contains:

- Items list
- Status (placed/shipped/delivered)
- Invoice download

Learning Concepts: nested API calls, protected routes.

🔹 10. 404 / Error Pages (`/404`, `/error`)

Purpose: Handle invalid routes or API errors gracefully.

👨‍💼 2. ADMIN DASHBOARD PAGES (Protected)

🔸 1. Admin Login (`/admin/login`)

Purpose: Separate admin authentication.

🔸 2. Dashboard Overview (`/admin/dashboard`)

Purpose: Display quick stats — total users, orders, revenue, products.

🔸 3. Product Management (`/admin/products`)

Purpose: CRUD interface for products.

- Add/Edit/Delete product
- Upload images

Learning Concepts: CRUD operations, forms with images (multipart), admin routes.

🔸 4. Category Management (`/admin/categories`)

Purpose: Manage categories and subcategories.

🔸 5. Order Management (`/admin/orders`)

Purpose: View and update order status (placed → shipped → delivered).

🔸 6. User Management (`/admin/users`)

Purpose: View users, deactivate accounts.

🔸 7. Reports & Analytics (`/admin/reports`)

Purpose: Display sales charts, revenue by category, etc.

Learning Concepts: Chart libraries, data visualization.

🧱 Total Frontend Pages (Approx. 16–18)

Section	Pages
Customer	10
Admin	7
Common (error, login, etc.)	1–2
Total	**≈18 Pages**