**Module 0: Project Setup & Basic Structure**

**0.1 Backend Setup (Node.js with MySQL)**

* **Objective: Initialize a Node.js project with Express.js, set up MySQL, and configure the basic structure.**
* **Actions:** 
  + **Initialize Node.js Project:** 
    - **Create a new directory: mkdir online-trading-system && cd online-trading-system.**
    - **Initialize Node.js: npm init -y.**
  + **Install dependencies: npm install express jsonwebtoken bcryptjs cors dotenv morgan slugify axios nodemailer. (Note: mongoose is replaced by Prisma/Sequelize for MySQL).**
  + **Configure Environment Variables:** 
    - **Create a .env file in the root directory.**
    - **Add DATABASE\_URL (MySQL connection string), JWT\_SECRET, PORT (e.g., 5000), and SENDGRID\_API\_KEY for email notifications.**
    - **Create a .env file in the server directory with DATABASE\_URL.**
  + **Set Up MySQL Connection:** 
    - **The project uses Prisma ORM for MySQL connection.**
    - **Run npx prisma migrate dev in the server directory to initialize the database schema.**
  + **Folder Structure:** 
    - **Based on File structure.docx and prompt.docx: server/ containing controllers, models, routes, middlewares, data, utils/db.ts (for Prisma client initialization if not auto-generated), and public (for static files like product images).**
  + **Express Configuration:** 
    - **In app.js (or server.js equivalent in server directory), configure Express with middleware: cors(), express.json(), express.urlencoded({ extended: true }), morgan('dev').**
    - **Serve static files: app.use(express.static('public')).**
  + **Error Handling Middlewares:** 
    - **Create middlewares/notFound.js to handle 404 errors.**
    - **Create middlewares/errorHandler.js for general error handling.**
    - **Use both in app.js (or server.js).**

**0.2 Frontend Setup (Next.js)**

* **Objective: Set up a Next.js project, configure dependencies, and establish the basic structure.**
* **Actions:** 
  + **Initialize Next.js Project:** 
    - **Run: npm create next-app in the root directory (resulting in app/ folder, as per File structure.docx).**
    - **Navigate to the project root directory.**
  + **Install dependencies: npm install bootstrap react-axios zustand react-toastify react-icons axios. (Note: Next.js uses file-based routing, replacing react-router-dom).**
  + **Folder Structure:** 
    - **app/ (Next.js App Router)** 
      * **(dashboard)/ (Admin dashboard routes)**
      * **\_zustand/ (Zustand state management)**
      * **actions/ (Server actions)**
      * **api/ (API routes)**
      * **cart/ (Shopping cart page)**
      * **checkout/ (Checkout page)**
      * **login/ (Login page)**
      * **product/ (Product details page)**
      * **register/ (Registration page)**
      * **search/ (Search page)**
      * **shop/ (Shop/Products page)**
      * **wishlist/ (Wishlist page)**
      * **layout.tsx (Root layout)**
      * **page.tsx (Home page)**
      * **globals.css (Global styles)**
    - **components/ (React Components)** 
      * **Admin/ (Admin components)** 
        + **AdminOrders.tsx**
        + **DashboardProductTable.tsx**
        + **DashboardSidebar.tsx**
      * **Product/ (Product components)** 
        + **ProductItem.tsx**
        + **ProductsSection.tsx**
        + **ProductTabs.tsx**
        + **SingleProductRating.tsx**
      * **Cart/ (Cart components)** 
        + **CartElement.tsx**
        + **QuantityInputCart.tsx**
      * **Layout/ (Layout components)** 
        + **Header.tsx**
        + **Footer.tsx**
        + **SectionTitle.tsx**
      * **UI/ (UI components)** 
        + **CustomButton.tsx**
        + **Filters.tsx**
        + **SearchInput.tsx**
    - **prisma/ (Prisma ORM)** 
      * **schema.prisma (Database schema)**
    - **public/ (Static files)** 
      * **images/**
      * **icons/**
    - **lib/ (Utility functions)** 
      * **utils.ts**
    - **server/ (Server-side code)** 
      * **prisma/**
    - **utils/ (Utility functions)** 
      * **db.ts (Database utilities)**
    - **Configuration files: package.json, next.config.mjs, tailwind.config.ts, tsconfig.json, .env.**
  + **CSS and Notifications:** 
    - **In app/layout.tsx or similar root layout file, import bootstrap/dist/css/bootstrap.min.css and react-toastify/dist/ReactToastify.css.**
    - **Configure ToastContainer in app/layout.tsx (or equivalent root component).**
  + **Routing:** 
    - **Set up Next.js's file-based routing within the app/ directory.**
  + **Layouts:** 
    - **The app/layout.tsx serves as the root layout.**
    - **Specific layouts for customer, admin, and employee would be handled by nested layouts within the app directory.**

**Module 1: Data Seeding & Models**

**1.1 Data Seeding**

* **Objective: Create seed data for products and users, and implement a seeding script.**
* **Actions:** 
  + **Product Seed Data:** 
    - **Create data/seed\_product\_data.json with sample products across categories (e.g., electronics, fashion).**
    - **Include fields based on Product interface: id, slug, title, price, rating, description, mainImage, manufacturer, categoryId, inStock.**
  + **User Seed Data:** 
    - **Create data/seed\_user\_data.json with sample users for roles: customer, employee, admin.**
    - **Include fields based on User interface: id, email, password, role. (Additional fields like fullName, phoneNumber from prompt.docx should also be included).**
  + **Seeding Script:** 
    - **Use server/utills/insertDemoData.js as specified in README.md to import data, hash passwords, and clear collections/tables.**

**1.2 Models (Prisma Schema)**

* **Objective: Define database schemas for core entities: User, Product, Order, Review using Prisma ORM.**
* **Actions:** 
  + **Prisma Schema Definition (prisma/schema.prisma):** 
    - **User Model:**

**model User {**

**id String @id @default(uuid())**

**email String @unique**

**password String**

**role String // e.g., "customer", "employee", "admin"**

**// Additional fields like fullName, phoneNumber, isActive, preferences, loyaltyPoints, createdAt, updatedAt**

**orders Order[]**

**reviews Review[]**

**}**

* + - **Product Model:**

**model Product {**

**id String @id @default(uuid())**

**slug String @unique**

**title String**

**price Float**

**rating Float? // ratingsAverage**

**description String?**

**mainImage String**

**manufacturer String**

**categoryId String**

**category Category @relation(fields: [categoryId], references: [id])**

**inStock Int @default(0) // stockCount**

**// Additional fields like imageUris, isAvailable, tags, numReviews, createdAt, updatedAt**

**orders OrderItem[]**

**reviews Review[]**

**otherImages OtherImages[]**

**}**

* + - **Order Model:**

**model Order {**

**id String @id @default(uuid())**

**adress String**

**apartment String?**

**company String?**

**dateTime DateTime**

**email String**

**lastname String**

**name String**

**phone String**

**postalCode String**

**status OrderStatus @default(processing)**

**city String**

**country String**

**orderNotice String?**

**total Float**

**// Additional fields like customer (relation to User), items (relation to OrderItem), taxRate, taxAmount, orderType, paymentMethod, paymentStatus, createdAt, updatedAt**

**orderItems OrderItem[]**

**customer User @relation(fields: [customerId], references: [id])**

**customerId String**

**}**

**enum OrderStatus {**

**processing**

**canceled**

**delivered**

**}**

* + - **Review Model:**

**model Review {**

**id String @id @default(uuid())**

**userId String**

**productId String**

**rating Float**

**comment String?**

**user User @relation(fields: [userId], references: [id])**

**product Product @relation(fields: [productId], references: [id])**

**createdAt DateTime @default(now())**

**updatedAt DateTime @updatedAt**

**}**

* + - **Category Model:**

**model Category {**

**id String @id @default(uuid())**

**name String**

**products Product[]**

**}**

* + - **OtherImages Model:**

**Đoạn mã**

**model OtherImages {**

**imageID Int @id @default(autoincrement())**

**productID String**

**image String**

**product Product @relation(fields: [productID], references: [id])**

**}**

* + - **WishlistItem Model:**

**model WishListItem {**

**id String @id @default(uuid())**

**userId String**

**productId String**

**user User @relation(fields: [userId], references: [id])**

**product Product @relation(fields: [productId], references: [id])**

**}**

* + - **OrderItem Model (for items in Order):**

**model OrderItem {**

**id String @id @default(uuid())**

**orderId String**

**productId String**

**quantity Int**

**priceAtOrder Float**

**notes String?**

**order Order @relation(fields: [orderId], references: [id])**

**product Product @relation(fields: [productId], references: [id])**

**}**

* + **Logic for password hashing and rating updates: Implement these as separate utility functions or service methods that interact with Prisma, as Prisma itself doesn't have pre-save hooks like Mongoose.**

**Module 2: Authentication & Authorization**

**2.1 Backend**

* **Objective: Implement user authentication and role-based authorization.**
* **Actions:** 
  + **Auth Routes (server/routes/authRoutes.js):** 
    - **POST /api/auth/register: Register new user.**
    - **POST /api/auth/login: Login and return JWT.**
    - **GET /api/auth/me: Get user info (protected).**
  + **Auth Controller (server/controllers/authController.js):** 
    - **Handle registration with password hashing using bcryptjs.**
    - **Handle login with JWT generation using jsonwebtoken.**
    - **Fetch user data for /me.**
  + **Auth Middleware (server/middlewares/auth.js):** 
    - **protect: Verify JWT and attach user to req.user.**
    - **authorize: Check req.user.role against allowed roles.**

**2.2 Frontend**

* **Objective: Implement authentication UI and state management.**
* **Actions:** 
  + **Login Page (app/login/page.tsx):** 
    - **Bootstrap-based form with email and password fields.**
    - **Save JWT and user info in Zustand store (app/\_zustand/authStore.ts) and localStorage.**
  + **Register Page (app/register/page.tsx):** 
    - **Bootstrap-based form with fields: full name, username, email, password, confirm password, phone number.**
    - **Client-side validation, API call, loading state, toast notifications using react-toastify.**
  + **Auth State Management (app/\_zustand/authStore.ts):** 
    - **Zustand store for user, token, isAuthenticated, isLoading, error.**
    - **Actions: login, register, logout, loadUserFromToken.**
  + **Protected Routes:** 
    - **Next.js's middleware or custom components (e.g., in components/common/Protected.tsx) to protect routes based on authentication status and user roles.**

**Module 3: Product & Order Management**

**3.1 Backend**

* **Objective: Implement APIs for product and order management.**
* **Actions:** 
  + **Product Routes (server/routes/productRoutes.js):** 
    - **GET /api/products: List products with filtering and pagination.**
    - **POST /api/products: Create product (admin/employee).**
    - **PUT /api/products/:id: Update product (admin/employee).**
    - **DELETE /api/products/:id: Delete product (admin).**
  + **Order Routes (server/routes/orderRoutes.js):** 
    - **POST /api/orders: Create order (customer).**
    - **GET /api/orders: List orders (employee/admin).**
    - **GET /api/orders/my-orders: List customer’s orders.**
    - **PUT /api/orders/:id/status: Update order status (employee/admin).**
  + **Review Routes (server/routes/reviewRoutes.js):** 
    - **POST /api/products/:productId/reviews: Create review (customer).**
    - **GET /api/products/:productId/reviews: Get reviews.**
    - **PUT /api/reviews/:reviewId: Update review (customer/admin).**
    - **DELETE /api/reviews/:reviewId: Delete review (customer/admin).**

**3.2 Frontend**

* **Objective: Implement product Browse, cart, and order functionalities.**
* **Actions:** 
  + **Home Page (app/page.tsx):** 
    - **Display product categories, search bar (using components/UI/SearchInput.tsx), and featured products in a Bootstrap grid.**
  + **Shop/Products Page (app/shop/page.tsx):** 
    - **Display a list of products, with filtering options (using components/UI/Filters.tsx) and pagination.**
  + **Product Details Page (app/product/[productSlug]/page.tsx):** 
    - **Show product details: images (carousel), description, price, reviews (using components/Product/SingleProductRating.tsx), “Add to Cart” button.**
  + **Cart Page (app/cart/page.tsx):** 
    - **Display cart items (using components/Cart/CartElement.tsx), quantity controls (using components/Cart/QuantityInputCart.tsx), total, and “Proceed to Checkout” button.**
  + **Order Tracking Page (e.g., app/customer/orders/page.tsx):** 
    - **List customer’s orders with status updates (e.g., pending, shipped, delivered).**

**Module 4: Admin & Employee Functions**

**4.1 Backend**

* **Objective: Implement APIs for user management, product moderation, and order processing.**
* **Actions:** 
  + **User Management Routes (server/routes/userRoutes.js):** 
    - **GET /api/users: List users (admin).**
    - **PUT /api/users/:id: Update user (admin).**
    - **DELETE /api/users/:id: Delete user (admin).**
  + **Product Moderation Routes:** 
    - **Extend server/routes/productRoutes.js with moderation endpoints (approve/reject).**
  + **Order Management Routes:** 
    - **Extend server/routes/orderRoutes.js with status updates and dispute handling.**

**4.2 Frontend**

* **Objective: Implement admin and employee dashboards.**
* **Actions:** 
  + **Admin Dashboard Layout (app/(dashboard)/layout.tsx):** 
    - **Includes components/Admin/DashboardSidebar.tsx.**
  + **User Management Page (app/(dashboard)/users/page.tsx):** 
    - **Bootstrap table with user details, filters, and actions (edit, delete, deactivate).**
  + **Product Management Page (app/(dashboard)/products/page.tsx):** 
    - **Table with product details, moderation status, and actions (potentially using components/Admin/DashboardProductTable.tsx).**
  + **Order Management Page (app/(dashboard)/orders/page.tsx):** 
    - **Table with order details, status updates, and dispute resolution tools (potentially using components/Admin/AdminOrders.tsx).**

**Module 5: Payment & Notifications**

**5.1 Backend**

* **Objective: Integrate payment gateways and notification systems.**
* **Actions:** 
  + **Payment Integration:** 
    - **Integrate VNPay or Stripe using their SDKs.**
    - **Create POST /api/payments to process payments.**
  + **Notification System:** 
    - **Use nodemailer with SendGrid for email notifications.**
    - **Create server/utils/sendNotification.js for order and account updates.**

**5.2 Frontend**

* **Objective: Implement payment and notification UI.**
* **Actions:** 
  + **Checkout Page (app/checkout/page.tsx):** 
    - **Form for shipping details (adress, apartment, company, email, lastname, name, phone, postalCode, city, country, orderNotice) and payment method selection.**
    - **Redirect to payment gateway and handle callback.**
  + **Notification Display:** 
    - **Use react-toastify for in-app notifications.**
    - **Display order status updates on relevant pages (e.g., Order Tracking Page).**