**📘 Module 1: User Management & Authentication**

### 🎯 Purpose:

To manage all user-related operations including registration, login, role assignment, profile management, and authentication mechanisms for the e-commerce platform. This module forms the backbone of the user experience and access control for different user types (admins, buyers, anonymous users).

### 🧩 Module Components:

1. **User Types**:
   * Anonymous User
   * Registered User (Buyer)
   * Admin (Super Admin, Category Admin)
2. **User Registration & Login**:
   * Supports traditional password-based login using JWT.
   * Allows third-party login via OAuth 2.0 + OpenID Connect (e.g., Google login).
   * Email verification is required post-registration.
   * Optional phone number verification using OTP.
3. **Authentication Methods**:
   * ✅ **Password-Based (JWT)**: For traditional login with hashed password.
   * ✅ **OAuth 2.0 + OpenID Connect**: For seamless third-party login (e.g., Gmail).
4. **Roles & Permissions**:
   * Role-based access control using Roles and UserRoles.
   * Predefined roles: Buyer, Super Admin, Category Admin, etc.
5. **Admin/User Management**:
   * Super Admin can assign Category Admins.
   * Buyer-specific data stored separately in Buyer table.
6. **Profile Management**:
   * Users can update personal info, set primary address, and manage contact info.
7. **Security Features**:
   * Hashed passwords using strong encryption.
   * Token expiration for JWT.
   * Rate-limiting and account lockout policies.

### 🔗 API Endpoints (Suggested):

#### User Auth

* POST /api/auth/register
* POST /api/auth/login (JWT)
* GET /api/auth/oauth/google/callback (OAuth)
* POST /api/auth/logout

#### User Info

* GET /api/user/profile
* PUT /api/user/profile/update

#### Role Management

* GET /api/roles
* POST /api/admin/assign-role

### 🔐 Security Recommendations:

* Implement reCAPTCHA on login/registration
* Use HTTPS across the entire app
* Secure refresh token rotation for JWT
* Use Azure Key Vault or equivalent to store client secrets for OAuth

### 🧭 Flow Summary:

* New user registers via form or Gmail login
* System assigns default role (Buyer)
* Admins manage roles and access
* Each authenticated user can maintain profile, address, cart, etc.
* Anonymous users can browse but not checkout

## 📦 Module 2: Product & Category Management

### 🔍 Module Purpose

This module is responsible for managing the product catalog of the e-commerce application. It includes handling categories, subcategories, products, product images, variants, tags, and the respective admin-level access control. This module ensures that all products are properly categorized, searchable, and manageable based on the roles assigned.

### 👥 Roles Involved

| **Role** | **Access & Permissions** |
| --- | --- |
| **Super Admin** | Full access to manage all categories, products, and assign category-level admins |
| **Category Admin** | CRUD operations for assigned categories and related products |
| **Buyer / Visitor** | Can browse, search, and filter products by categories and tags |

### 🧩 Key Functionalities

1. **Category Management**
   * Create, update, delete categories
   * Assign category admins
   * Add category-level metadata (e.g., display order, icons)
   * Categories can be nested (Subcategory support)
2. **Product Management**
   * Add/Edit/Delete products
   * Associate products with categories
   * Enable/disable products (active/inactive)
3. **Product Images**
   * Upload multiple images for each product
   * Set a primary image
4. **Variants**
   * Manage product variants (e.g., size, color)
   * Track SKU and inventory at variant level
5. **Tags & Filters**
   * Add tags for search optimization (e.g., “New Arrival”, “Popular”)
   * Filter products based on tags, category, price, availability
6. **Admin Restrictions**
   * A category admin can manage only assigned categories and their respective products
   * Super admin has access to all data

### 📌 Tables Used (from SQL schema)

| **No.** | **Table Name** | **Purpose** |
| --- | --- | --- |
| 7. | Categories | Stores main product categories |
| 8. | Subcategories | Allows category nesting for detailed structure |
| 9. | Products | Stores product-level data |
| 10. | ProductImages | Stores multiple images per product |
| 11. | ProductVariants | Defines product variations (size, color, etc.) |
| 12. | ProductTags | Tags to enhance filtering and discovery |
| 13. | Tags | Master list of available tags |
| 14. | CategoryAdmins | Mapping of category-level admin to categories |

### 🔄 Data Flow Summary

1. **Super Admin** creates categories and subcategories.
2. Super Admin assigns a **Category Admin** to a category.
3. **Category Admin** logs in and adds products to assigned categories.
4. While adding a product:
   * Uploads images.
   * Adds variants with SKU and inventory.
   * Tags the product appropriately.
5. Product is available for **Buyers** to browse or search.

### 🧾 API Requirements (Sample)

| **API** | **Method** | **Description** |
| --- | --- | --- |
| /api/categories | GET | Get all categories |
| /api/categories | POST | Add new category (Admin only) |
| /api/products | GET | List products by category, tags |
| /api/products/:id | PUT | Update product (Admin only) |
| /api/products/:id/images | POST | Upload product images |
| /api/products/:id/variants | POST | Add variants |
| /api/tags | GET | Get available tags |

### 🔐 Validation & Rules

* Category names must be unique.
* Tags must be from the master list.
* Variants must have unique SKU values.
* Products without an image or category assignment cannot be made active.

### 🧠 Suggestions for Enhancement

* Add bulk upload feature for products (via CSV/Excel).
* Add soft-delete for products to retain data history.
* Add AI-based tag suggestions on product upload.
* Provide analytics dashboard per category admin (e.g., most viewed products).

## ❤️🛒 Module 3: Wishlist & Cart Management

### 🔍 Module Purpose

This module manages the **personalized Wishlist and Shopping Cart** functionalities for users (both buyers and anonymous). It allows users to **save products for future interest**, **add items to cart**, **adjust quantities**, and **move items between wishlist and cart**. It also ensures that the system maintains **accurate cart counts** throughout the shopping journey, including after order placement.

### 👥 Roles Involved

| **Role** | **Access & Permissions** |
| --- | --- |
| **Anonymous User** | Can add items to cart (session-based), wishlist disabled |
| **Buyer (Logged-in)** | Full access to Wishlist and Cart features |
| **Super Admin** | Can monitor cart statistics (optional, for analytics) |

### 🧩 Key Functionalities

1. **Wishlist Management**
   * Add/remove products to/from wishlist
   * View wishlist anytime
   * Move items from wishlist to cart
2. **Cart Management**
   * Add/update/remove products in cart
   * Update product quantity
   * Move items from cart to wishlist
   * Cart items maintained even across login sessions
3. **Cross-Movement**
   * From wishlist → cart (retains product and quantity)
   * From cart → wishlist (removes from cart, adds to wishlist)
4. **Cart Syncing & Count**
   * Cart count is updated in real time
   * After order placement, cart is cleared and count reset
   * Cart counts are shown in the UI header for better UX
5. **Anonymous Cart**
   * Use cookies or session ID for temporary cart before login
   * Option to merge anonymous cart into user’s cart after login

### 📌 Tables Used (from SQL schema)

| **No.** | **Table Name** | **Purpose** |
| --- | --- | --- |
| 15. | Wishlist | Stores the user’s wishlist master record |
| 16. | WishlistItems | Stores products added to the wishlist |
| 17. | Cart | Stores the user’s cart master record |
| 18. | CartItems | Stores products and quantities in the cart |

### 🔄 Data Flow Summary

1. Logged-in **Buyer** adds products to **Wishlist** or **Cart**.
2. Items in wishlist/cart are stored against user ID.
3. User can **move items** between wishlist and cart anytime.
4. On placing an order:
   * Cart is emptied
   * Cart count is reset
   * Inventory is adjusted (linked to Order Management module)

### 🧾 API Requirements (Sample)

| **API** | **Method** | **Description** |
| --- | --- | --- |
| /api/wishlist | GET | Get current user’s wishlist |
| /api/wishlist | POST | Add product to wishlist |
| /api/wishlist/:id | DELETE | Remove item from wishlist |
| /api/cart | GET | Get current cart |
| /api/cart | POST | Add product to cart |
| /api/cart/:itemId | PUT | Update cart item quantity |
| /api/cart/:itemId | DELETE | Remove product from cart |
| /api/cart/move-to-wishlist | POST | Move cart item to wishlist |
| /api/wishlist/move-to-cart | POST | Move wishlist item to cart |

### 🧠 Business Rules & Validations

* A product can only exist **once** in wishlist/cart per user.
* Quantity must be ≥ 1 and ≤ available inventory.
* Moving an item from cart → wishlist **removes** it from the cart.
* Anonymous cart is stored using cookie/session and merged on login.
* Cart is **emptied** after order is placed.
* Wishlist remains intact even after order placement.

### 🧠 Suggestions for Enhancement

* **Share Wishlist**: Let users share wishlist via link or email.
* **Smart Suggestions**: Show "Recently Wishlisted" or "Also in Cart" suggestions.
* **Save for Later**: Let users move items out of cart to a "save for later" section.
* **Cart Abandonment Tracking**: Track and email users who leave items in their cart.

## 📦 Module 4: Orders & Order Management

### 📘 Module Purpose

This module handles the entire **order lifecycle**, from placing an order from the cart to processing, shipping, delivery, return/refund, and maintaining historical order data. It supports different **order statuses**, links to users, products, payments, and shipping, and is critical to business reporting and customer experience.

### 👥 Roles Involved

| **Role** | **Access & Permissions** |
| --- | --- |
| **Buyer** | Place/view orders, track delivery, request return/refund |
| **Super Admin** | Monitor, manage all orders and analytics |
| **Category Admin** | View orders related to their category's products only |
| **Anonymous User** | Cannot place order; must register/login |

### 🧩 Key Functionalities

1. **Place Order from Cart**
   * Buyer places order from cart
   * Captures address, total price, and payment method
2. **Order Processing**
   * Admin or system updates order status (Pending, Shipped, etc.)
   * Payment linked with payment table
3. **Order Status Management**
   * Track stages: Pending, Confirmed, Packed, Shipped, Delivered, Cancelled, Returned
4. **Multiple Products Per Order**
   * One order can contain multiple products with different quantities
   * Maintained in OrderItems table
5. **Order History**
   * Buyers can view full order history
   * Each order item has individual status for better tracking
6. **Return/Refund**
   * Initiate return request (linked to ReturnRequests table)
   * Refunds tracked (future expansion)

### 📌 Tables Used (from SQL schema)

| **No.** | **Table Name** | **Purpose** |
| --- | --- | --- |
| 19. | Orders | Main order record (user, total, payment) |
| 20. | OrderItems | List of products in each order |
| 21. | OrderStatus | Tracks status updates over time |
| 22. | ShippingDetails | Stores delivery address and shipment info |
| 23. | ReturnRequests | Optional — handles product return flow |

### 🔄 Data Flow Summary

1. User adds product(s) to cart
2. On checkout:
   * Order is created
   * Cart is cleared
   * Items copied to OrderItems
   * Shipping address & payment linked
3. Admin can update status
4. User tracks status from portal
5. Return can be initiated if applicable

### 🧾 API Requirements (Sample)

| **API** | **Method** | **Description** |
| --- | --- | --- |
| /api/orders | POST | Place new order |
| /api/orders/:orderId | GET | Get order details |
| /api/orders/history | GET | Get order history |
| /api/orders/:orderId/status | PUT | Update order status |
| /api/orders/:orderId/return | POST | Request a return |
| /api/shipping/:orderId | GET | Get shipping details |

### 🔐 Business Rules & Validations

* User must be logged in to place an order
* One order can have multiple products (handled in OrderItems)
* Order total is calculated at the time of placement
* Orders once placed **cannot be edited** by the buyer
* Returns are allowed only if the status is Delivered
* Shipping address must be selected from saved addresses

### 🔄 Order Status Lifecycle (Example)

| **Status** | **Description** |
| --- | --- |
| Pending | Order placed, awaiting confirmation |
| Confirmed | Payment confirmed, order ready |
| Packed | Items packed and ready for shipment |
| Shipped | Shipped via courier |
| Delivered | Order delivered to user |
| Cancelled | Cancelled by user or admin |
| Returned | Returned by user and refund issued |

### 🧠 Suggestions for Enhancement

* **Partial Return/Refunds** per item
* **Delivery Tracking** with courier API
* **Admin Notifications** on high-value orders
* **Invoice Generation** as PDF
* **SMS/Email** order updates to user
* **Analytics**: Best-selling products, delivery rates, etc.

## Module 5: Payment & Transaction Management

### 🔹 Module Purpose

This module handles all the logic related to user payments, transaction records, and payment method integrations. It ensures secure, traceable, and flexible payment processing for orders in the e-commerce platform.

### ✅ Functional Requirements

| **ID** | **Requirement Description** |
| --- | --- |
| PM-01 | The system must support multiple payment methods: **Credit Card, Debit Card, UPI (Google Pay, PhonePe, Paytm), and Pay Later (e.g., LazyPay, Simpl)**. |
| PM-02 | Payments should be initiated during order confirmation. |
| PM-03 | Each payment attempt should be tracked in a transaction log (success, failed, pending). |
| PM-04 | Refunds (partial or full) must be supported. |
| PM-05 | Payment status should update the related order's status. |
| PM-06 | All failed payments should be retriable. |
| PM-07 | Users must be able to view their payment history. |
| PM-08 | Sensitive data (e.g., card number, UPI ID) should not be stored directly, only masked or tokenized forms (PCI-DSS compliant). |

### 🔐 Authentication & Security

* **OAuth 2.0 + OpenID Connect** will manage secure access to third-party payment APIs.
* Secure redirection via payment gateways (like Razorpay, PayU, Stripe) will be used.
* Data will be encrypted using SSL.
* No sensitive card/UPI data will be stored — only token references.

### 🧠 Payment Methods Supported

| **Payment Type** | **Description** | **Example Providers** |
| --- | --- | --- |
| Credit/Debit Card | Standard card-based payments | Visa, Mastercard, Rupay |
| UPI | Unified Payments Interface | PhonePe, Google Pay, Paytm |
| Pay Later | Deferred payments or EMIs | Simpl, LazyPay, ZestMoney |
| Net Banking | Direct from user’s bank | Axis, HDFC, SBI, ICICI |

### 🔄 Process Flow (Simplified)

1. **User places order →**
2. **System redirects to selected payment method page**
3. **Third-party gateway handles authentication**
4. **Payment status returned to app (success, fail, pending)**
5. **Order table & transaction table updated accordingly**
6. **Email/SMS confirmation sent to user**

### 🧾 Key Tables Involved

* Payments: Stores each payment attempt.
* Transactions: Tracks each transaction for audit/logging.
* PaymentMethods: List of supported payment types.
* Refunds: Logs any refund activities.

### 🧪 Suggested Tech Stack

* **Gateway**: Razorpay, Stripe, PayU, Cashfree (UPI/Pay Later supported)
* **API Protocol**: REST + OAuth 2.0 tokens
* **Security**: PCI-DSS compliance for any card processing

### 📤 Related APIs

| **Endpoint** | **Method** | **Description** |
| --- | --- | --- |
| /api/payment/initiate | POST | Start a payment for an order |
| /api/payment/callback | POST | Receive gateway response |
| /api/payment/status/{orderId} | GET | Check payment status |
| /api/refund | POST | Trigger a refund request |
| /api/payment/history/{userId} | GET | Fetch past transactions |

### 🧩 Integration Points

* Linked with Orders table to map payments.
* Connected to Users for user-specific transaction history.
* Interacts with external payment APIs (GPay, PhonePe, Card processors).

## 📘 Module 6: Admin & Role Management

### 🔹 Module Purpose

This module manages system roles and permissions for various types of users — especially different categories of administrators such as Super Admins, Category Admins, and Order Managers. It controls access to CRUD operations across the platform and enforces authorization boundaries based on user roles.

### ✅ Functional Requirements

| **ID** | **Requirement Description** |
| --- | --- |
| AR-01 | System must support **role-based access control (RBAC)**. |
| AR-02 | Roles like Super Admin, Category Admin, Order Manager, and Buyer must be predefined. |
| AR-03 | Admins can be assigned one or multiple roles. |
| AR-04 | Super Admin can create/update/delete any role or assign them. |
| AR-05 | Category Admin can only manage assigned categories/products. |
| AR-06 | Order Manager can only view and process orders, refunds, and delivery status. |
| AR-07 | The system must restrict API and UI access based on user roles. |
| AR-08 | System should allow viewing of all users and their roles. |

### 🧠 Types of Roles

| **Role Name** | **Description** | **Access Scope** |
| --- | --- | --- |
| Super Admin | Full system access | All modules |
| Category Admin | Manages categories and products | Product Management |
| Order Manager | Handles orders and refunds | Order/Payment Modules |
| Buyer | End customer | Shopping & Order Placement |
| Anonymous | Not logged in | Limited browsing |

### 🔐 Authentication & Authorization

* Users authenticate via **Password-based JWT login** or **OAuth 2.0 + OpenID Connect (e.g., Google)**.
* Once logged in, **authorization** is managed by role-permission mapping.
* Middleware and API guards will validate roles on each secure route.

### 🔄 Process Flow

1. **User logs in →**
2. **User role fetched from DB →**
3. **Frontend UI loads based on permissions**
4. **API requests checked for authorization**
5. **Only permitted actions can be performed**

### 🧾 Key Tables Involved

| **Table Name** | **Description** |
| --- | --- |
| Users | Stores all registered users |
| Roles | Predefined system roles |
| UserRoles | Many-to-many mapping between users and roles |
| Admins | Specialized table for admin-only logic and types (Super, Category, etc.) |

### ⚙️ Role-Admin Handling

* **Super Admins** can:
  + Manage roles (create, delete, assign)
  + Assign category/product rights to Category Admins
  + Oversee all modules
* **Category Admins**:
  + Linked to specific categories via mapping (e.g., AdminCategoryMapping)
  + Can perform CRUD on assigned products only
* **Buyers**:
  + No write access to product or admin interfaces

### 📤 Related APIs

| **Endpoint** | **Method** | **Description** |
| --- | --- | --- |
| /api/roles | GET/POST/DELETE | Manage role definitions |
| /api/admins | GET/POST/DELETE | Admin-specific operations |
| /api/userroles | GET/POST | Assign roles to users |
| /api/category-admin-mapping | POST/GET | Link category admins to categories |

### 🧩 Integration Points

* **User Management Module**: All users are managed here.
* **Category/Product Module**: Category Admins are restricted to their scope.
* **Order Module**: Only specific roles like Order Managers can process or modify orders.

### 📋 Special Notes

* A single user can hold **multiple roles**.
* Permissions are validated both in **frontend UI routing** and **backend APIs**.
* For maintainability, role definitions should be **seeded** into the database initially.

## 📘 Module 7: Reviews, Ratings & Reporting

### 🔹 Module Purpose

This module allows users to share feedback via ratings and reviews, and enables admins to generate detailed reports on products, categories, and system usage.

### ✅ Functional Requirements

| **ID** | **Requirement Description** |
| --- | --- |
| RR-01 | Buyers can rate and review products they've purchased. |
| RR-02 | Each product rating is linked to the buyer and product. |
| RR-03 | Only users with completed purchases can post reviews. |
| RR-04 | Admins can view and moderate submitted reviews. |
| RR-05 | Review includes rating (1–5 stars), title, message, and optional images. |
| RR-06 | Reporting dashboard shows statistics like most viewed, most purchased, most rated products. |
| RR-07 | Reports can be generated by Super Admins and downloaded in CSV/PDF. |
| RR-08 | Daily/weekly system activity logs should be summarized for analytics. |

### 🧾 Key Tables Involved

| **Table Name** | **Description** |
| --- | --- |
| ProductReviews | Stores buyer reviews and ratings |
| SystemReports | Dynamic or scheduled reports (traffic, sales, views) |
| ProductViews | (Optional) Tracks product page visits |
| OrderHistory | Data source for purchase-based reports |

### 🧠 Logic Flow

1. Buyer completes a purchase
2. Buyer opens product page → **Review section enabled**
3. Review submitted → saved with timestamp
4. Admin dashboard shows reviews with moderation tools
5. Reports generated based on filters (category, timeframe, etc.)

### 🔐 Permissions

| **Role** | **Access** |
| --- | --- |
| Buyer | Submit reviews |
| Super Admin | Generate all reports |
| Category Admin | View reports limited to assigned categories |
| Anonymous | View reviews only |

### 📤 Related APIs

| **Endpoint** | **Method** | **Description** |
| --- | --- | --- |
| /api/reviews | GET/POST/DELETE | Manage product reviews |
| /api/reports/sales | GET | Generate sales report |
| /api/reports/traffic | GET | Track product view/report |
| /api/reports/export | GET | Export report as PDF/CSV |

## 📘 Module 8: Support, Logs & Messaging

### 🔹 Module Purpose

This module manages user communication, logs for debugging, and automated email/SMS notifications for transactional or system events.

### ✅ Functional Requirements

| **ID** | **Requirement Description** |
| --- | --- |
| SL-01 | Users can send messages via a **Contact Us** form. |
| SL-02 | Admins can respond to support requests via dashboard or email. |
| SL-03 | The system logs all critical events and errors. |
| SL-04 | Email and SMS logs are stored for audit and traceability. |
| SL-05 | Email/SMS templates should be reusable and stored in DB. |
| SL-06 | Logs should be filterable by type, date, and user ID. |
| SL-07 | Support messages can be marked as open/resolved/closed. |

### 🧾 Key Tables Involved

| **Table Name** | **Description** |
| --- | --- |
| ContactMessages | Stores messages sent from Contact Us form |
| SystemLogs | Logs exceptions, warnings, login attempts |
| EmailTemplates | Stores subject/body for reusable emails |
| SmsLogs | Stores SMS send history |
| EmailLogs | Logs for sent transactional emails |

### 🧠 Logic Flow

1. **Contact Us**: Anonymous user fills form → message saved
2. Admin replies via backend or email
3. Errors (e.g., failed payments) are logged automatically
4. Emails (order confirmation, password reset) use stored templates
5. Logs are reviewed by Super Admin for debugging and audits

### 🔐 Permissions

| **Role** | **Access** |
| --- | --- |
| Anonymous | Submit contact forms |
| Super Admin | View all logs & messages |
| Category Admin | Limited view of messages/logs related to assigned modules |

### 📤 Related APIs

| **Endpoint** | **Method** | **Description** |
| --- | --- | --- |
| /api/contact-us | POST | Submit support message |
| /api/system-logs | GET | View filtered logs |
| /api/email-templates | GET/POST/PUT | Manage email templates |
| /api/email-logs | GET | View email send history |
| /api/sms-logs | GET | View SMS delivery logs |

### 🔔 Notification Triggers (Email/SMS)

| **Event** | **Channel** | **Triggered For** |
| --- | --- | --- |
| Order Placed | Email & SMS | Buyer |
| Refund Processed | Email | Buyer |
| Admin Role Assigned | Email | Admin |
| Support Reply | Email | User |