Hooma Project Design Document

▼ 1. Project Overview

Hooma is a web-based application designed to facilitate an e-commerce platform for buying and selling furniture. It resembles the functionality of IKEA's website, providing a seamless experience for users to browse, purchase, and manage furniture items. The platform includes two main user roles:

- **Buyers**: Can browse products, add items to their cart, and complete purchases.
- **Sellers/Inventory Managers**: Can manage inventory by creating, reading, updating, and deleting (CRUD) furniture items.

▼ 2. Functional Requirements

- General
  1. Register new user.
  2. Log in securely to main dashboard using credentials (username, email, password).
  3. Reset password if forgotten.
  4. Log out and destroying session.
- **Customer (Buyer)**
  1. Browse furniture items with filters (e.g category, price range, color)
  2. View detailed product information (description, price, dimensions, images).
  3. Add/remove items to/from the shopping cart.
  4. Proceed to checkout and complete payment via integrated payment gateways.
  5. View order history and track orders.

6. Update user profile information (e.g., email, address, phone number, password)

- **Seller**
  1. Manage inventory using CRUD operations:
     - **Create**: Add new furniture items with details (name, description, price, stock quantity, images).
     - **Read**: View all items in inventory with filtering and sorting options.
     - **Update**: Modify existing item details (e.g., price, stock, description).
     - **Delete**: Remove items from inventory.
  2. View sales reports and analytics.

▼ 3. Non Functional Requirement

1. **Performance**: The application should handle up to 1,000 concurrent users without significant lag.
2. **Security**: Implement secure authentication (e.g., JWT or OAuth), data encryption, and protection against common vulnerabilities (e.g., SQL injection, XSS).
3. **Scalability**: The system should scale horizontally to accommodate increased traffic.
4. **Usability**: Intuitive and responsive UI/UX design for both buyers and sellers.

▼ 4. Key Features

4.1 Buyer-Side Features

1. **Product Catalog**:
   - Display furniture items in a grid view.
   - Include filters for categories (e.g., chairs, tables), price ranges, and colors.
   - Search bar for quick access to specific items.
2. **Shopping Cart**:
   - View added items.

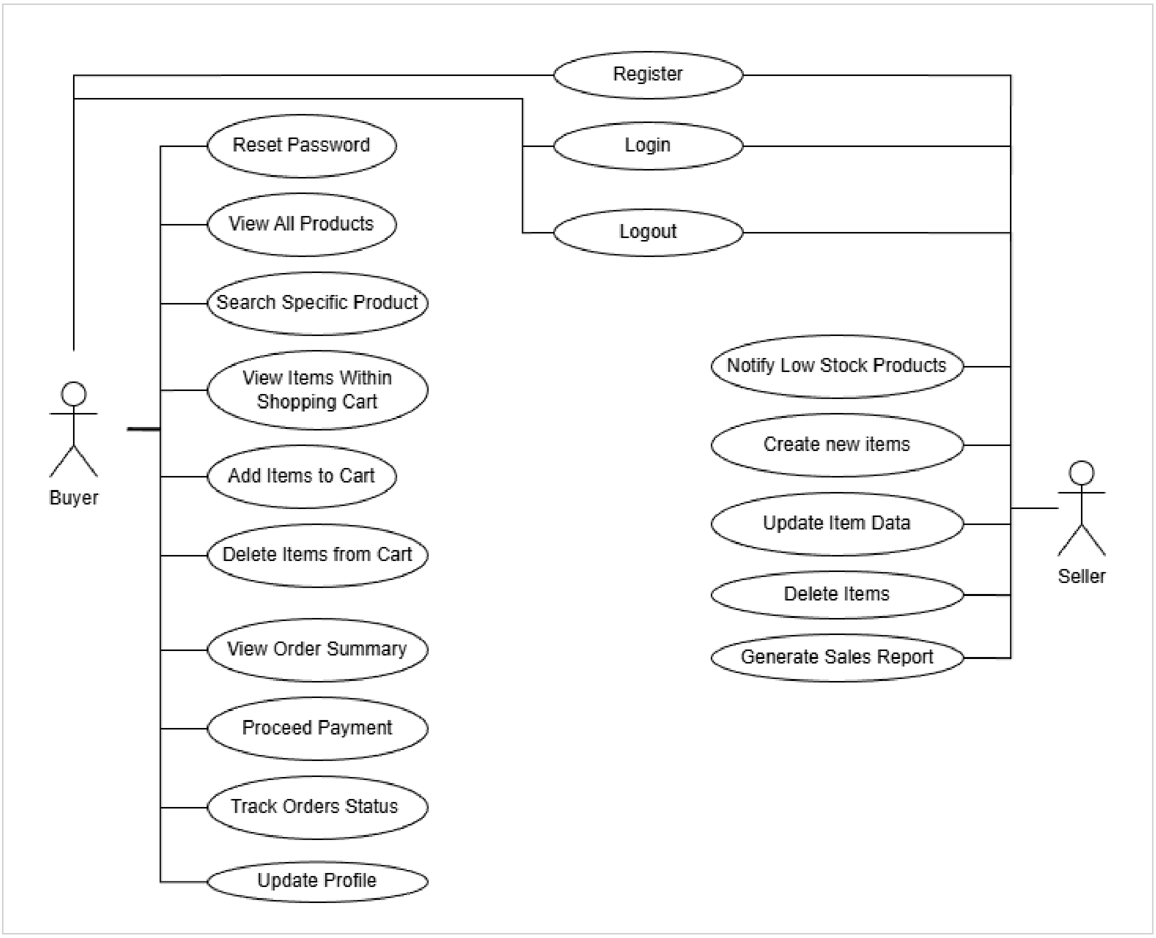
- Add/remove items dynamically.
- Display total price, including taxes.

3. **Checkout Process**:
   - Secure payment gateway integration.
   - Payment method options (e.g., card, QR, transfer)
   - Order summary (inc. shipping cost) before finalizing the purchase.
4. **Order Tracking**:
   - View past and current orders with status updates (e.g., "Processing", "Shipped", “Received”).
5. **Profile Management:**
   - Update email, password, address, phone number, profile picture.
   - View, add, update and delete shipping addresses.

4.2 Seller-Side Features

1. **Admin Dashboard**:
   - Centralized interface for managing inventory.
   - Real-time notifications for low stock levels.
2. **CRUD Operations**:
   - Create: Add new furniture items with multiple image uploads.
   - Read: View all items in a paginated table with sorting and filtering.
   - Update: Edit item details and upload new images.
   - Delete: Remove items with confirmation prompts.
3. **Sales Analytics**:
   - Generate reports on sales trends, popular products, and revenue.

▼ 5. Use Case Diagram and Scenarios



Hooma Use Case Diagram

▼ 6. System Architecture

6.1 High-Level Architecture

The application follows a **three-tier architecture**:

1. **Frontend**: Handles user interaction and displays data.
2. **Backend**: Manages business logic, processes requests, and interacts with the database.
3. **Database**: Stores all persistent data (e.g., user accounts, product inventory, orders).

6.2 Technology Stack

- **Frontend**:
    - Framework: React.js

    - Styling: Tailwind CSS
    - State Management: Redux or Context API (for React)
    - Responsive Design: Mobile-first approach
- **Backend**:
    - Framework: Node.js with Express.js
    - Authentication: JSON Web Tokens (JWT) or OAuth2
    - APIs: RESTful API
- **Database**:
    - Type: NoSQL (MongoDB)
- **Hosting**:
    - Frontend: Netlify or Vercel
    - Backend: AWS EC2, Heroku, or Google Cloud Platform
    - Database: MongoDB Atlas
- **Payment Integration**:
    - QRIS (Indonesian payment gateway
    - Stripe or PayPal for processing payments.

▼ 7. Database Schema

1. Key Collections

Primary collections:

1. **Users**:
    - Stores information about buyers and sellers.
2. **Products**:
    - Contains details about furniture items available for purchase.
3. **Orders**:
    - Tracks orders placed by buyers.
4. **Categories** (Optional):
    - Represents product categories (e.g., chairs, tables) if needed for hierarchical organization.

2. Schema Design

- **Users Collection**

```
{
  "_id": ObjectId("..."), // Unique identifier for th
e user
  "email": "user@example.com", // User’s email addres
s
  "password_hash": "hashed_password", // Hashed passw
ord
  "role": "buyer", // Role: "buyer" or "seller"
  "name": "John Doe",
  "address": {
    "street": "123 Main St",
    "city": "New York",
    "state": "NY",
    "zip": "10001"
  }
}
```

- **Products Collection**

```
{
  "_id": ObjectId("..."), // Unique identifier for th
e product
  "name": "Modern Sofa", // Name of the product
  "description": "A comfortable and stylish sofa.",
// Detailed description
  "price": 499.99, // Price in USD
  "stock_quantity": 50, // Available stock
  "category": "sofas", // Category (e.g., sofas, chai
rs)
  "images": ["url1.jpg", "url2.jpg"], // URLs of prod
uct images
  "dimensions": { // Dimensions of the product
    "width": 80,
    "height": 35,
    "depth": 30
  },
  "material": "Leather", // Material used
  "color": "Black", // Color of the product
  "created_at": ISODate("2026-01-22T10:00:00Z"), // T
imestamp when the product was added
  "updated_at": ISODate("2026-01-22T12:00:00Z") // Ti
mestamp when the product was last updated
}
```

- **Orders Collection**

```
{
  "_id": ObjectId("..."), // Unique identifier for th
e order
  "user_id": ObjectId("..."), // Reference to the use
r who placed the order
  "items": [
    {
      "product_id": ObjectId("..."), // Reference to
the product
      "name": "Modern Sofa", // Product name (denorma
lized for convenience)
      "quantity": 2, // Quantity purchased
      "price": 499.99 // Price at the time of purchas
e
    }
  ],
  "total_price": 999.98, // Total cost of the order
  "status": "Pending", // Order status (e.g., Pendin
g, Shipped, Delivered)
  "payment_method": "Credit Card", // Payment method
used
  "shipping_address": { // Shipping address
    "street": "456 Elm St",
    "city": "Los Angeles",
    "state": "CA",
    "zip": "90001"
  },
  "created_at": ISODate("2026-01-22T14:00:00Z") // Ti
mestamp when the order was placed
}
```

- **Categories Collection (Optional)**

  To manage categories hierarchically or store additional metadata.

  ```
  {
    "_id": ObjectId("..."), // Unique identifier for th
  e category
    "name": "Sofas", // Category name
    "parent_category_id": ObjectId("..."), // Optional:
  Parent category (for subcategories)
    "created_at": ISODate("2026-01-22T10:00:00Z") // Ti
  mestamp when the category was created
  }
  ```

3. Data Modeling Best Practices

- **Embedding**: Use embedding when related data is frequently accessed together and does not change independently. Examples:
  - Product dimensions, images, and descriptions within the `products` collection.
  - Shipping address within the `orders` collection.
- **Referencing**: Use referencing when data is shared across multiple documents or changes independently. Examples:
  - Linking `user_id` in the `orders` collection to the `users` collection.
  - Linking `product_id` in the `orders.items` array to the `products` collection.
- Denormalize data selectively to reduce the number of queries required. For example:
  - Include the product name and price in the `orders.items` array to display order history without querying the `products` collection.

- Indexes improve query performance but consume storage space. Plan your indexes based on common query patterns:
    - `products.category`: For filtering products by category.
    - `products.price`: For sorting products by price.
    - `orders.user_id`: For retrieving a user's orders.

4. Example Queries

- **Add a New Product**

```
db.products.insertOne({
  name: "Dining Table",
  description: "A spacious dining table for family gatherings.",
  price: 799.99,
  stock_quantity: 20,
  category: "tables",
  images: ["table1.jpg", "table2.jpg"],
  dimensions: { width: 120, height: 30, depth: 40 },
  material: "Wood",
  color: "Brown",
  created_at: new Date(),
  updated_at: new Date()
});
```

- **Retrieve Products in a Category**

```
db.products.find({ category: "sofas" }).sort({ price: 1 });
```

- **Update Stock Quantity**

```
db.products.updateOne(
  { _id: ObjectId("...") },
  { $inc: { stock_quantity: -1 } } // Decrease stock by 1
);
```

- **Fetch Order History for a User**

  ```
  db.orders.find({ user_id: ObjectId("...") }).sort({ c
  reated_at: -1 });
  ```

▼ 8. Implementation Plan

8.1 Development Phases

1. **Phase 1: Core Backend Development**
   - Set up the backend server.
   - Implement authentication and authorization.
   - Create APIs for CRUD operations on products and orders.
2. **Phase 2: Frontend Development**
   - Build the buyer-side interface (product catalog, cart, checkout).
   - Develop the seller-side admin dashboard.
3. **Phase 3: Integration**
   - Connect frontend to backend APIs.
   - Test payment gateway integration.
4. **Phase 4: Testing and Deployment**
   - Conduct unit, integration, and end-to-end testing.
   - Deploy to production environment.

8.2 Timeline

- Phase 1: 2 weeks
- Phase 2: 3 weeks
- Phase 3: 2 weeks
- Phase 4: 1 week