Luxe Ecommerce Project Documentation

Ali Elsawy, Khaled Beheiry, Mohammed Moawad

Contents

1 Written Proposal

1.1 Project Idea

"Luxe Ecommerce" is a modern, full-stack e-commerce web application designed to provide a sophisticated online shopping platform.

1.2 Use Case

The primary goal is to offer a seamless online shopping experience for end-users, enabling them to browse products, manage personal carts and wishlists, place orders securely, and track their shipment status. Concurrently, it provides an administrative interface for staff to manage the product catalog, user accounts, and oversee order fulfillment efficiently. The application leverages a robust tech stack, including Next.js and Prisma, chosen for performance, scalability, and type safety.

2 Software Requirements Specifications (SRS)

2.1 Functional Requirements

- User registration and secure login.
- Password hashing (`bcryptjs`) for enhanced security.
- Role-based access control distinguishing between `USER` and `ADMIN`.
- Comprehensive product browsing with detailed views, including multiple images per product.
- Product categorization for organized browsing.
- Functionality for searching and filtering products.
- Management of a persistent shopping cart (add, remove, update quantity).
- Wishlist feature for saving products for later consideration.
- User address management (add, view, edit, set default).
- Secure order placement process using items from the cart.
- Selection of a shipping address during checkout.
- Order status tracking (`PENDING`, `PROCESSING`, `SHIPPED`, `DELIVERED`, `CANCELLED`).
- User access to personal order history.
- Admin capabilities for managing user accounts (view, potentially edit/delete).
- Admin tools for managing the product catalog (view, add, edit, delete, manage stock, feature products).
- Admin interface for managing orders (view details, update status).
- Admin management of product categories.

2.2 Non-functional Requirements

- **Security**: User passwords must be securely hashed. Authentication managed via `NextAuth.js` using secure tokens (`jsonwebtoken`).
- **Usability**: The application must feature a clear, intuitive, and responsive user interface suitable for various devices (achieved via React, Tailwind CSS, Framer Motion).
- **Maintainability**: The codebase must be well-structured, readable, and type-safe, facilitated by TypeScript and Prisma.
- **Performance**: Efficient database querying (via Prisma) and optimized frontend rendering (leveraging Next.js features like SSR/SSG).

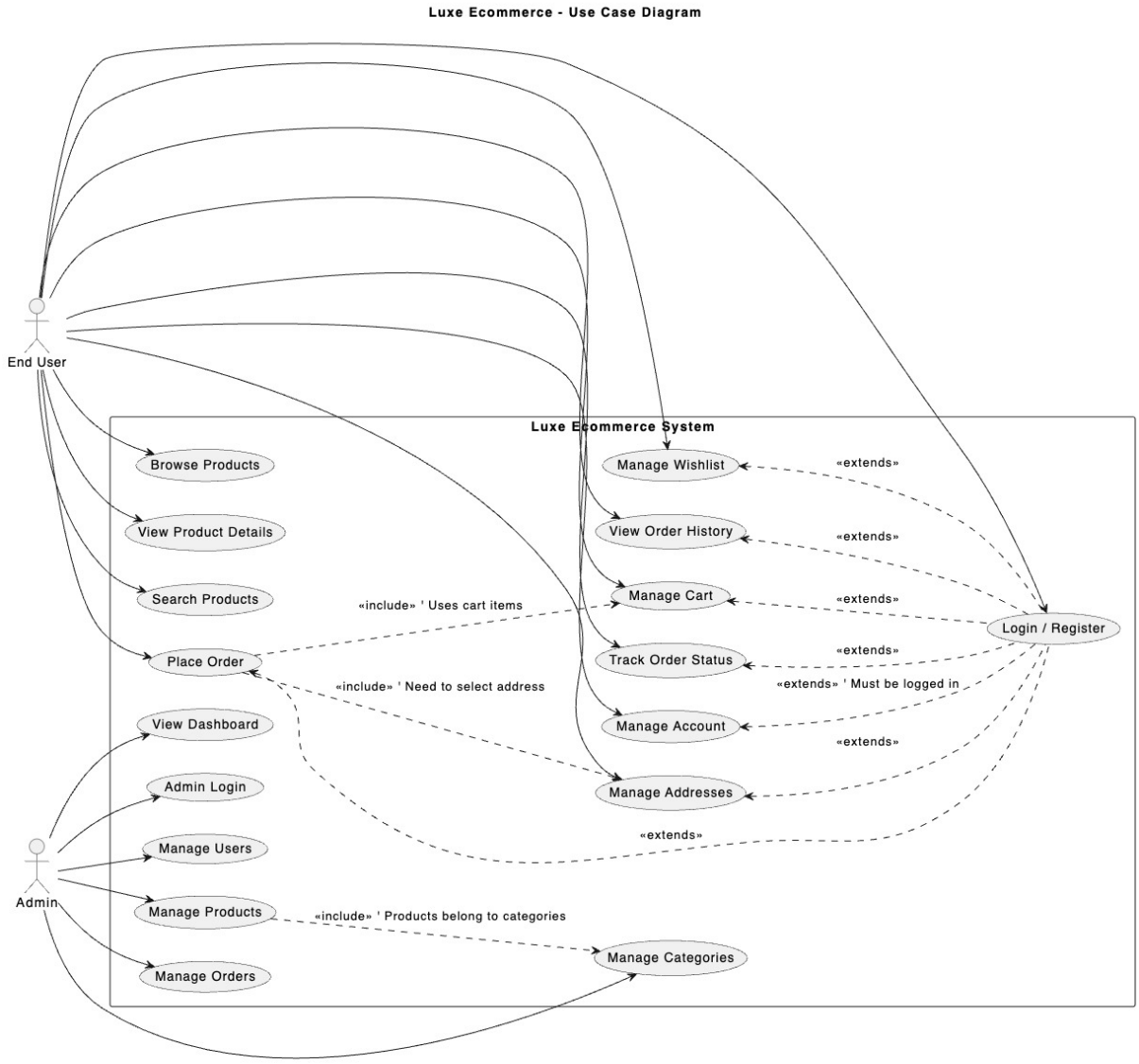
- **Reliability**: Graceful error handling through custom error pages (`_error.tsx`) and `ErrorBoundary` components.
- **Scalability**: The architecture (Next.js API routes, Prisma with a potentially scalable database backend) should support future growth in users and data.

2.3 Stakeholders

- **End Users**: Customers interacting with the storefront.
- **Administrators**: Staff responsible for site management.
- **Developers**: Team responsible for building and maintaining the application.

2.4 Use Cases

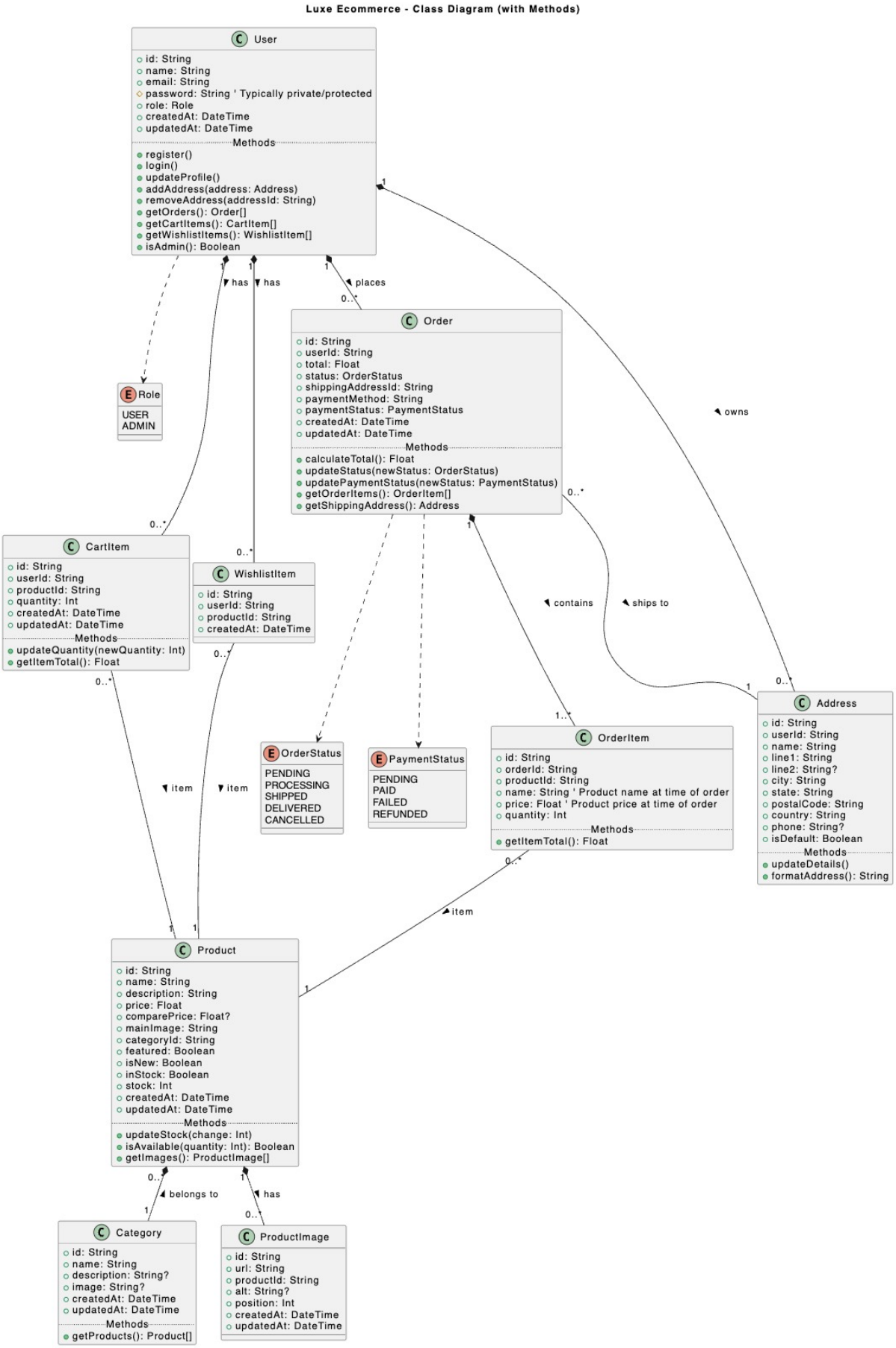
- Register Account
- Log In / Log Out
- Browse Products (All / By Category)
- Search Products
- View Product Details
- Add Product to Cart
- View/Modify Shopping Cart
- Add Product to Wishlist
- View/Modify Wishlist
- Manage User Addresses
- Checkout / Place Order
- View Order History
- Track Order Status
- (Admin) Manage Product Catalog
- (Admin) Manage Categories
- (Admin) Manage User Accounts
- (Admin) Manage Orders



2.5 Class Diagram (Conceptual Description)

A class diagram would visually represent the main entities defined in the Prisma schema (`User`, `Product`, `Order`, `Category`, `CartItem`, `OrderItem`, `Address`, etc.).

- Each class would contain attributes corresponding to the fields defined in the `schema.prisma` file.
- Key methods representing core business logic could be inferred (e.g., `User.addOrder()`, `Product.updateStock()`).
- Relationships between classes (one-to-one, one-to-many, many-to-many) would mirror those defined by the `@relation` attributes in Prisma, indicating associations and multiplicities.
- Enumerated types (`Role`, `OrderStatus`, `PaymentStatus`) would be included as distinct elements or attributes with restricted values.

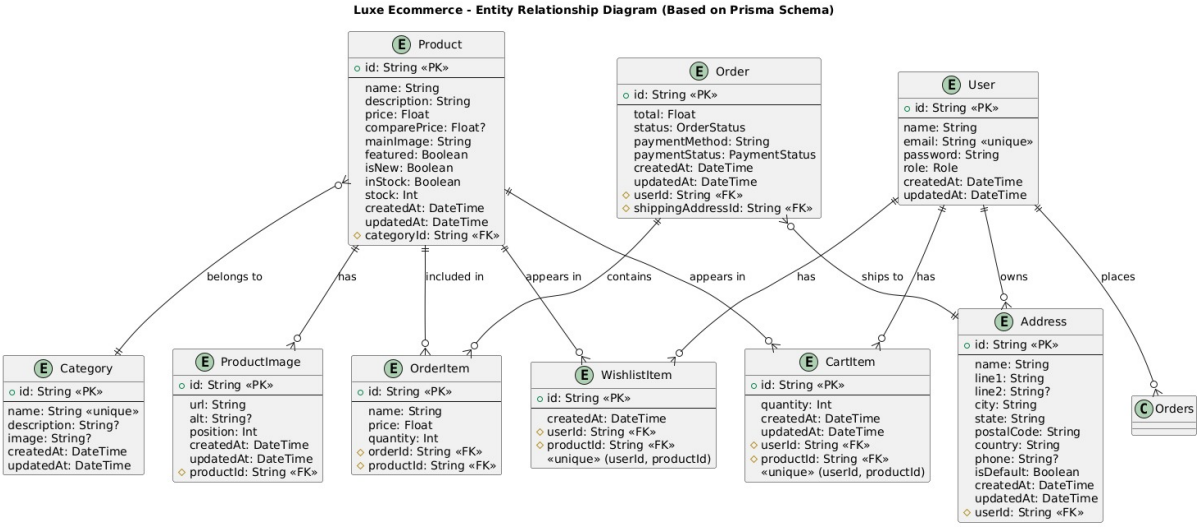

Luxe Ecommerce - Class Diagram (with Methods)
User
id: String
name: String
email: String
password: String ' Typically private/protected
role: Role
createdAt: DateTime
updatedAt: DateTime
Methods
register()
login()
updateProfile()
addAddress(address: Address)
removeAddress(addressId: String)
getOrders(): Order[]
getCartItems(): CartItem[]
getWishlistItems(): WishlistItem[]
isAdmin(): Boolean
Role
USER
ADMIN
Order
id: String
userId: String
total: Float
status: OrderStatus
shippingAddressId: String
paymentMethod: String
paymentStatus: PaymentStatus
createdAt: DateTime
updatedAt: DateTime
Methods
calculateTotal(): Float
updateStatus(newStatus: OrderStatus)
updatePaymentStatus(newStatus: PaymentStatus)
getOrderItems(): OrderItem[]
getShippingAddress(): Address
CartItem
id: String
userId: String
productId: String
quantity: Int
createdAt: DateTime
updatedAt: DateTime
Methods
updateQuantity(newQuantity: Int)
getItemTotal(): Float
WishlistItem
id: String
userId: String
productId: String
createdAt: DateTime
OrderStatus
PENDING
PROCESSING
SHIPPED
DELIVERED
CANCELLED
PaymentStatus
PENDING
PAID
FAILED
REFUNDED
OrderItem
id: String
orderId: String
productId: String
name: String ' Product name at time of order
price: Float ' Product price at time of order
quantity: Int
Methods
getItemTotal(): Float
Address
id: String
userId: String
name: String
line1: String
line2: String?
city: String
state: String
postalCode: String
country: String
phone: String?
isDefault: Boolean
Methods
updateDetails()
formatAddress(): String
Product
id: String
name: String
description: String
price: Float
comparePrice: Float?
mainImage: String
categoryId: String
featured: Boolean
isNew: Boolean
inStock: Boolean
stock: Int
createdAt: DateTime
updatedAt: DateTime
Methods
updateStock(change: Int)
isAvailable(quantity: Int): Boolean
getImages(): ProductImage[]
Category
id: String
name: String
description: String?
image: String?
createdAt: DateTime
updatedAt: DateTime
Methods
getProducts(): Product[]
ProductImage
id: String
url: String
productId: String
alt: String?
position: Int
createdAt: DateTime
updatedAt: DateTime
has
has
places
owns
contains
ships to
item
item
item
belongs to
has

3 ERD & Enhanced ERD (Based on Prisma Schema)

The database structure is defined in `prisma/schema.prisma`. A visual ERD can be generated using tools like dbdiagram.io or Lucidchart based on this schema.

3.1 Entities

- `User`: Customer/Admin information.
- `Product`: Details of items available for sale.
- `Category`: Product groupings.
- `ProductImage`: Additional images for products.
- `CartItem`: Association between a user, a product, and quantity in the cart.
- `WishlistItem`: Association between a user and a product saved for later.
- `Order`: Represents a completed purchase transaction.
- `OrderItem`: Details of a specific product within an order (captures price/name at time of order).
- `Address`: User's shipping/billing addresses.



3.2 Relationships (Primary Examples)

- `User` 1 – * `Order`
- `User` 1 – * `CartItem`
- `User` 1 – * `WishlistItem`
- `User` 1 – * `Address`
- `Category` 1 – * `Product`
- `Product` * – 1 `Category`
- `Product` 1 – * `ProductImage`
- `Product` 1 – * `CartItem`
- `Product` 1 – * `OrderItem`
- `Product` 1 – * `WishlistItem`
- `Order` 1 – * `OrderItem`
- `Order` * – 1 `Address` (Shipping Address)

(Note: Cardinality indicated conceptually; Prisma relations define specific foreign keys).

3.3 Enhanced Features & Constraints

- **Enumerated Types**: `Role`, `OrderStatus`, `PaymentStatus` enforce specific allowed values for respective fields.
- **Unique Constraints**: Applied to fields like `User.email`, `Category.name`, and composite keys like (`userId`, `productId`) in `CartItem` and `WishlistItem`.
- **Referential Actions**: `onDelete: Cascade` used for related entities like `ProductImage`, `CartItem`, ensuring data consistency when a parent record (e.g., User, Product) is deleted.
- **Default Values**: Automatically assigned values for fields like `createdAt`, `User.role`, `Order.status`, etc.
- **Optional Fields**: Nullable fields indicated by `?` (e.g., `Product.comparePrice`).
- **Timestamps**: Automatic `createdAt` and `updatedAt` fields for tracking record modifications.

4 Project Scope, Tools, and Design Decisions

4.1 Project Scope

The project delivers a comprehensive e-commerce platform encompassing the entire customer journey from browsing and discovery to checkout and order tracking. It includes essential user management features (authentication, profiles, addresses) and a full administrative backend for managing products, categories, users, and orders. Key functionalities like cart persistence, wishlists, stock management, and role-based access are central to the scope. Future enhancements outside the current scope might include payment gateway integrations, customer reviews, or advanced analytics.

4.2 Tools & Technologies

- **Framework**: Next.js 14
- **Language**: TypeScript
- **Database/ORM**: SQLite (Development), Prisma Client
- **Authentication**: NextAuth.js v4, `bcryptjs`, `jsonwebtoken`
- **Styling**: Tailwind CSS, PostCSS
- **Frontend Core**: React 18
- **UI/Animation**: Framer Motion, GSAP, react-icons
- **Utilities**: react-intersection-observer, react-use
- **Development Env**: Node.js, `ts-node`, ESLint
- **Potential 3D**: Three.js (if utilized)

4.3 Design Decisions

- **Full-stack TypeScript**: Adopted for end-to-end type safety, improving code reliability and maintainability.
- **Next.js Framework**: Chosen for its integrated features including server-side rendering (SSR), static site generation (SSG), optimized image handling, API routes, and file-based routing, contributing to performance and developer efficiency.
- **Prisma ORM**: Selected for its type-safe database access, straightforward schema definition and migration management, and simplified query construction. SQLite used initially for ease of development setup.
- **Tailwind CSS**: Employed for its utility-first approach, enabling rapid development of consistent and customizable user interfaces.

- **NextAuth.js**: Integrated to handle complex authentication flows securely and simplify session management.
- **Component-Based UI**: Leveraging React's component model to build a modular, reusable, and maintainable frontend codebase.
- **API Routes**: Utilizing Next.js API routes for creating backend endpoints, keeping frontend and backend logic colocated within the same project structure where appropriate.
- **Centralized Types**: Relying on Prisma-generated types and potentially a dedicated `/types` directory for consistency across the application.
- **Robust Error Handling**: Implementing custom error pages (`_error.tsx`) and `ErrorBoundary` components to enhance user experience during unexpected issues.