MUHAMMAD FAYYAZ

Technical Requirements

1. Frontend Requirements

- **Framework:** Next.js (for server-side rendering, SEO, and dynamic routing).
- **Styling:** Tailwind CSS (for responsive, modern design).
- **State Management:** React Context API or Zustand for global state handling (e.g., cart state).
- **Deployment:** Vercel (optimized for Next.js).
- **Features:**
  - Homepage: Dynamic product listings, featured wallets.
  - Product Details Page: Detailed descriptions, images, and reviews.
  - Shopping Cart and Checkout.
  - Responsive Design for mobile, tablet, and desktop.

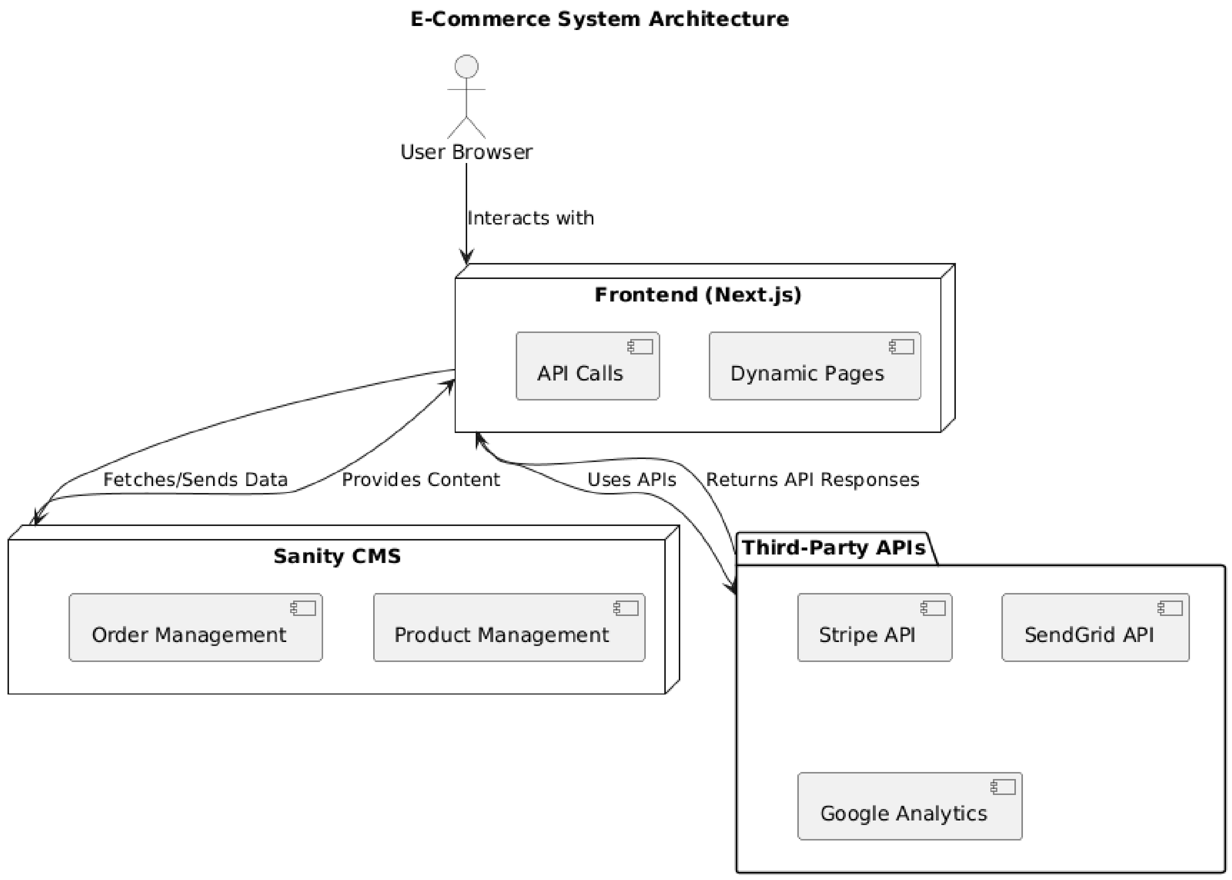
2. Backend Requirements (Sanity CMS)

- **Content Management System:** Sanity.
- **Schema Design:**
  - Products: Title, description, price, category, images, stock quantity.
  - Categories: Name, description.
  - Reviews: Product ID, rating, comment, user.
  - Orders: Order ID, user details, product details, payment status.
- **Integration:** Use Sanity's API to fetch and manage content dynamically.

3. Third-Party APIs

- **Stripe API:** For payment processing.
- **Google Analytics:** For tracking user behavior.
- **SendGrid or Nodemailer:** For sending order confirmation emails.
- **Geolocation API:** For shipping and tax calculations based on location.

System Architecture Diagram:



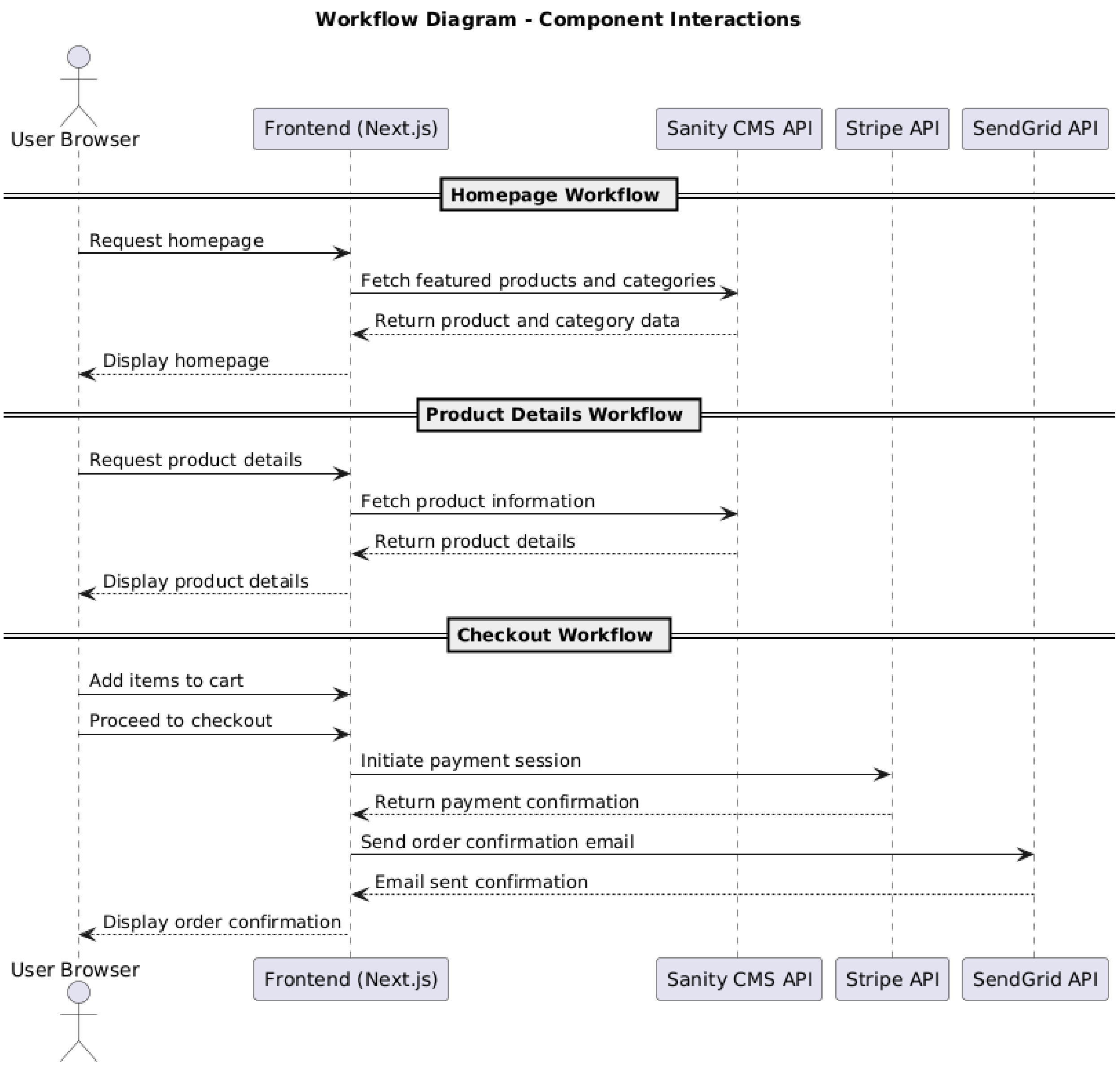
Explanation:

1. **User Browser:**
   - Users access the platform, browse products, view details, and proceed to checkout.
2. **Frontend (Next.js):**
   - Fetches product data from Sanity CMS.
   - Sends user actions (e.g., add to cart) to Third-Party APIs like Stripe and SendGrid.
3. **Sanity CMS:**
   - Stores and serves product, category, and order data to the frontend.
4. **Third-Party APIs:**
   - **Stripe API** handles payments.

- **SendGrid API** sends order confirmations.
- **Google Analytics** tracks user behavior.

Data flows seamlessly between these components to deliver a dynamic user experience.

Workflow:



Workflow Diagram Explanation

1. **Homepage Workflow**
   - User accesses the homepage.
   - Frontend fetches featured products and categories from Sanity CMS.
   - Products and promotions are dynamically displayed.
2. **Product Details Workflow**
   - User clicks on a product.
   - Frontend sends a request to Sanity CMS for detailed product information.
   - Product page displays details, reviews, and similar products.
3. **Checkout Workflow**
   - User adds items to the cart.
   - Cart state is maintained on the frontend.
   - At checkout, frontend integrates with Stripe for payment processing.
   - After payment, an order confirmation email is sent using SendGrid.

Data Schema Design:

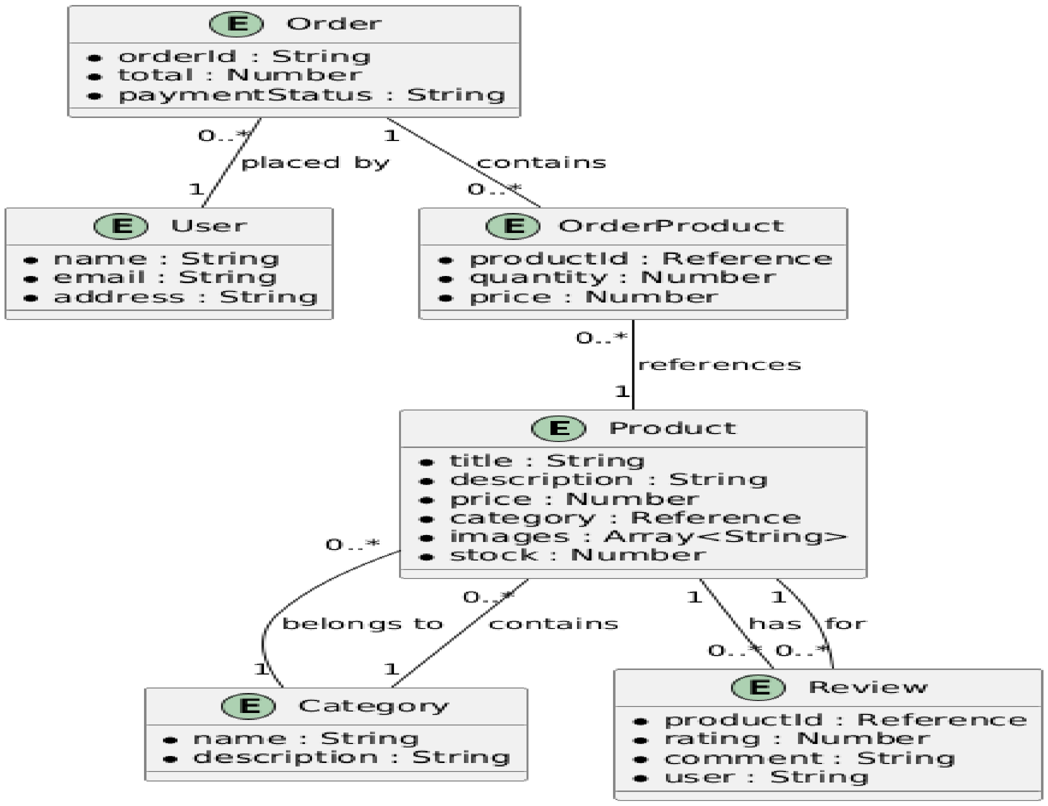
Product Schema { "title": "String", "description": "String", "price": "Number", "category": "Reference", "images": "Array<String>", "stock": "Number" }

Category Schema { "name": "String", "description": "String" }

Review Schema { "productId": "Reference", "rating": "Number", "comment": "String", "user": "String" }

Order Schema { "orderId": "String", "user": { "name": "String", "email": "String", "address": "String" },
"products": [{ "productId": "Reference", "quantity": "Number", "price": "Number" }], "total": "Number", "paymentStatus": "String" }

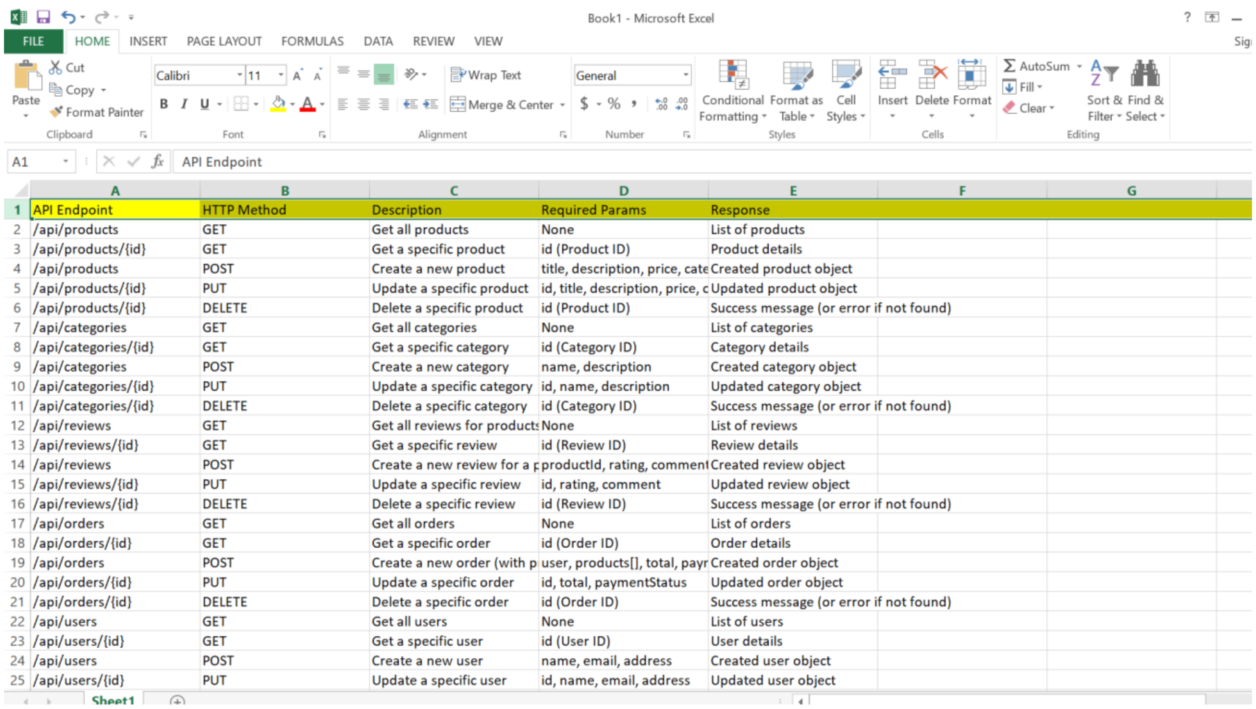
Entity Relationships:



Key Relationship Changes:

1. **Product to Review**: A product can have many reviews (1-to-many), but a review is for one product (many-to-1).
2. **Category to Product**: A category can contain many products (1-to-many), but a product belongs to one category (many-to-1).
3. **Order to OrderProduct**: An order can contain many products, with a quantity and price for each (1-to-many), while an order product references a single product (many-to-1).
4. **Order to User**: An order is placed by one user (many-to-1).

Plan API's Endpoint:



API Endpoint	HTTP Method	Description	Required Params	Response
/api/products	GET	Get all products	None	List of products
/api/products/{id}	GET	Get a specific product	id (Product ID)	Product details
/api/products	POST	Create a new product	title, description, price, cate	Created product object
/api/products/{id}	PUT	Update a specific product	id, title, description, price, c	Updated product object
/api/products/{id}	DELETE	Delete a specific product	id (Product ID)	Success message (or error if not found)
/api/categories	GET	Get all categories	None	List of categories
/api/categories/{id}	GET	Get a specific category	id (Category ID)	Category details
/api/categories	POST	Create a new category	name, description	Created category object
/api/categories/{id}	PUT	Update a specific category	id, name, description	Updated category object
/api/categories/{id}	DELETE	Delete a specific category	id (Category ID)	Success message (or error if not found)
/api/reviews	GET	Get all reviews for products	None	List of reviews
/api/reviews/{id}	GET	Get a specific review	id (Review ID)	Review details
/api/reviews	POST	Create a new review for a p	productId, rating, comment	Created review object
/api/reviews/{id}	PUT	Update a specific review	id, rating, comment	Updated review object
/api/reviews/{id}	DELETE	Delete a specific review	id (Review ID)	Success message (or error if not found)
/api/orders	GET	Get all orders	None	List of orders
/api/orders/{id}	GET	Get a specific order	id (Order ID)	Order details
/api/orders	POST	Create a new order (with p	user, products[], total, payr	Created order object
/api/orders/{id}	PUT	Update a specific order	id, total, paymentStatus	Updated order object
/api/orders/{id}	DELETE	Delete a specific order	id (Order ID)	Success message (or error if not found)
/api/users	GET	Get all users	None	List of users
/api/users/{id}	GET	Get a specific user	id (User ID)	User details
/api/users	POST	Create a new user	name, email, address	Created user object
/api/users/{id}	PUT	Update a specific user	id, name, email, address	Updated user object

Key Notes:

GET: Retrieve data (list or specific item).

POST: Create new data (product, category, review, order, user).

PUT: Update existing data.

DELETE: Remove data

Tools & Technologies (RoadMapping)

- **Frontend**: Next.js, Tailwind CSS, React, Axios
- **Backend**: Sanity/Strapi (CMS), Node.js, Express.js
- **Database**: MongoDB, PostgreSQL, or MySQL
- **Authentication**: JWT, NextAuth.js, or Passport.js
- **Payment Gateway**: Stripe, PayPal
- **Testing**: Jest, Cypress, Mocha
- **Deployment**: Vercel