API Requirements

Product Listings

```
GET /products
Description: Retrieves a list of all products.
Response: {
  "products": [
    {
      "id": 1,
      "name": "Sofa Model X",
      "description": "Comfortable sofa",
      "price": 1200,
      "images": ["url1", "url2"],
      "categories": ["sofas", "modern"]
    },
    ...
  ]
}
```

```
GET /categories
Description: Retrieves all product categories.
Response:  {
  "categories": ["sofas", "stools", "modern", "vintage"]
}
```

Shopping Cart

```
GET /cart
Description: Retrieves the current user's cart items.
Request Params: user_id
Response: {
  "cart": [
    {
      "product_id": 1,
      "name": "Sofa Model X",
      "quantity": 1,
      "price": 1200
    },
    ...
  ]
}
```

```
POST /cart/add
Description: Adds a product to the cart.
Request Body: {
  "user_id": 1,
  "product_id": 1,
  "quantity": 1
}
```

```
Response:

{
  "message": "Product added to cart"
}
```

```
POST /cart/add
Description: Adds a product to the cart.
Request Body:{
  "user_id": 1,
  "product_id": 1,
  "quantity": 1
}
```

```
Response:    {
               "message": "Product added to cart"
             }
```

```
POST /cart/remove
Description: Removes a product from the cart.
Request Body: {
  "user_id": 1,
  "product_id": 1
}
Response: {
  "message": "Product removed from cart"
}
```

Order Management

```
POST /order/checkout
Description: Places an order based on cart contents.
Request Body: {
  "user_id": 1,
  "cart": [
    { "product_id": 1, "quantity": 1 },
    { "product_id": 2, "quantity": 2 }
  ],
  "shipping_address": "123 Main St, City, Country"
}
Response: {
  "order_id": 123,
  "total_price": 3000,
  "status": "Order placed successfully"
}
```

```
GET /order/:id
Description: Retrieves details of a specific order by ID.
Request Params: id
Response: {
  "order_id": 123,
  "status": "Shipped",
  "items": [
    { "product_id": 1, "quantity": 1, "price": 1200 }
  ],
  "total_price": 3000
}
```

User Profile

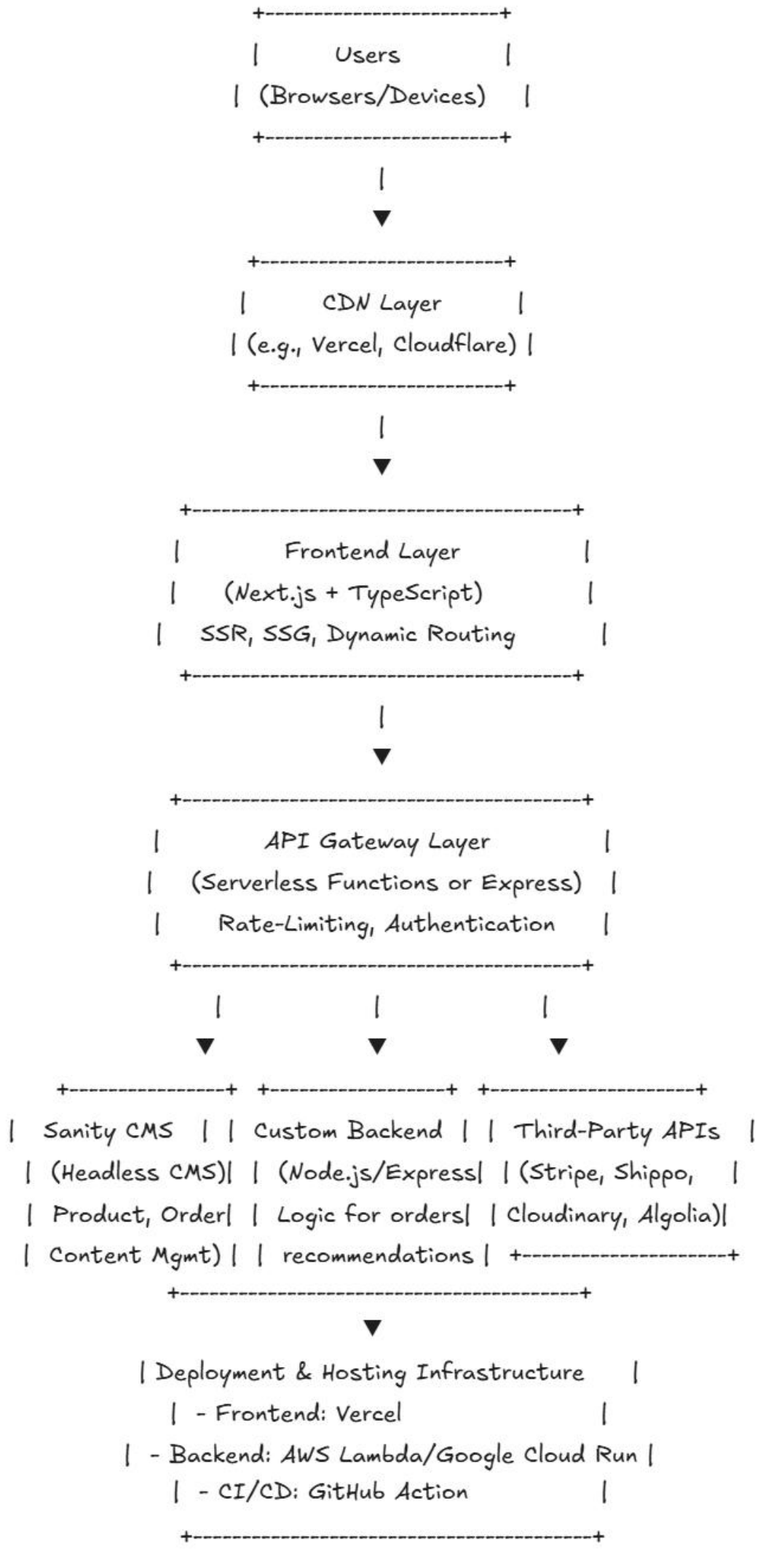
```
GET /user/profile
Description: Retrieves the user's profile information.
Request Params: user_id
Response: {
  "user_id": 1,
  "name": "John Doe",
  "email": "user@example.com",
  "address": "123 Main St"
}
```

```
PUT /user/profile/update
Description: Updates the user's profile details.
Request Body: {
  "user_id": 1,
  "name": "John Doe",
  "email": "new-email@example.com",
  "address": "456 Elm St"
}
```

```
Response: {
  "message": "Profile updated successfully"
}
```

Payment Processing

```
POST /payment/process
Description: Processes payment for an order.
Request Body:{
  "order_id": 123,
  "payment_method": "credit_card",
  "amount": 3000
}
Response:{
  "payment_status": "Successful",
  "transaction_id": "abc123"
}
```


Users
(Browsers/Devices)
CDN Layer
(e.g., Vercel, Cloudflare)
Frontend Layer
(Next.js + TypeScript)
SSR, SSG, Dynamic Routing
API Gateway Layer
(Serverless Functions or Express)
Rate-Limiting, Authentication
Sanity CMS
(Headless CMS)
Product, Order
Content Mgmt)
Custom Backend
(Node.js/Express
Logic for orders
recommendations
Third-Party APIs
(Stripe, Shippo,
Cloudinary, Algolia)
Deployment & Hosting Infrastructure
- Frontend: Vercel
- Backend: AWS Lambda/Google Cloud Run
- CI/CD: GitHub Action

Technical Requirements

Frontend Requirements:

1. Framework and Language:

   Next.js and Typescript

2. For Styling:

   Tailwind Css for responsive designing

3. UI/UX Design

   Intuitive navigation with a clear product catalog layout.

4. State Management:

   Use React Context API for managing global states (e.g., cart, user preferences).

5. Pages Included:
   - Home page
   - Shop
   - Product page
   - Cart
   - FAQ
   - About
   - Contact

Backend Requirements:

1. CMS Configuration:

   Sanity CMs as the content management system for managing product details

2. Integration:

   Connect Sanity CMS to the frontend via Sanity's GROQ queries.

3. Database Management:

   Utilize Sanity Studio for managing assets (e.g., product images ,product details ,stock availabilityetc).

Third Party API's:

1. Payment Gateway:

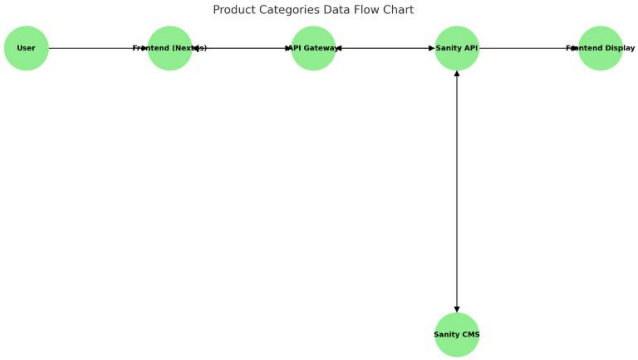
   Integrate with Stripe or PayPal for secure payments and subscriptions (if offering premium plans).

2. Shipping and Logistics:

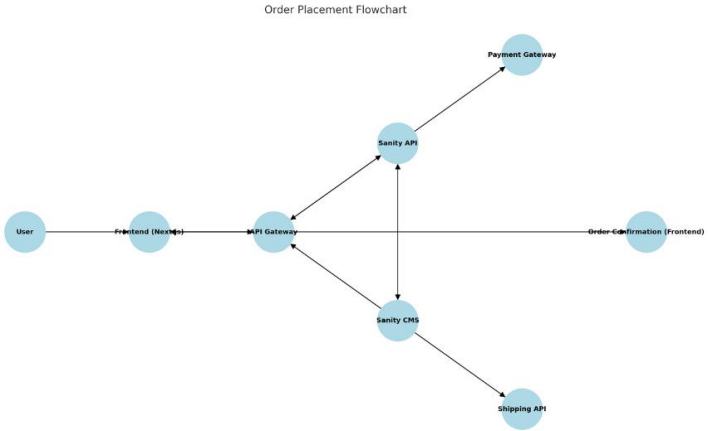
   Use APIs like Shippo, EasyPost, or directly integrate with shipping providers (e.g., UPS, FedEx).

Design System Architecture

User
|
▼
Frontend (Next.js)
|-- Validates Form Data
|-- Sends POST /api/signup
|
▼
API Gateway
|-- Routes to Backend
|
▼
Backend Logic
|-- Validates Data
|-- Hashes Password
|-- Sends Data to Sanity CMS
|-- Triggers Confirmation Email
|
▼
Sanity CMS
|-- Stores User Data
|
▼
Email Service (e.g., SendGrid)
|-- Sends Confirmation Email
|
▼
Frontend
|-- Displays Success Message



User: Interacts with the frontend to browse product categories.
Frontend (Next.js): Sends a request to the API Gateway.
API Gateway: Routes the request to the Sanity API.
Sanity API: Communicates with Sanity CMS to fetch product category data.
Sanity CMS: Retrieves the data and returns it to the Sanity API.
Sanity API: Sends the data back through the API Gateway to the Frontend (Next.js).
Frontend Display: Dynamically renders the fetched product categories to the user.



User: Initiates the order by placing it via the Frontend (Next.js).
Frontend: Sends the order details to the API Gateway.
API Gateway: Routes the request to:
Sanity API, which records the order details in Sanity CMS.
Payment Gateway to process the payment securely.
Payment Gateway: Confirms the payment and sends a response back to the API Gateway.
API Gateway: Contacts the Shipping API to initiate shipment and fetch tracking details.
Shipping API: Returns tracking information to the API Gateway.
API Gateway: Sends updated order and shipment details back to the Frontend.
Frontend: Displays the Order Confirmation page with payment and shipping details.