Planning the Technical Foundation

1. Define Technical Requirements

Technologies and Tools

Frontend Framework: Next.js for server-side rendering and a seamless user experience.

Backend Development: RESTful APIs for handling data operations and business logic.

Content Management: Sanity for managing product listings and content dynamically.

Authentication: Implement user authentication using Clerk for secure login and registration.

Payment Processing: Stripe to handle secure payment transactions.

Deployment: Deploying application on Vercel for efficient hosting and continuous deployment.

System Components Overview

Frontend Requirements

The frontend will be developed using Next.js to ensure fast performance and server-side rendering. The structure and components of the frontend will include:

Pages

Homepage

- Displays featured furniture products and categories.
- Highlights ongoing promotions or new arrivals.
- Includes a navbar for quick navigation.

Product Listing Page

- Showcases a grid of furniture products with sorting and filtering options (e.g., price, category).

Product Details Page

- Displays detailed information about a specific product, including: High-quality images.
- Description, material details.
- Pricing and stock availability.
- Add-to-cart button.

Cart Page

- Lists all selected items for purchase.
- Allows users to update quantities or remove items.
- Displays subtotal and tax calculations.

Checkout Page

- Collects user details, such as shipping address and payment information.

Backend Requirements: Sanity CMS

Sanity will be used as the backend for managing dynamic content. Key features include:

Content Structure

Product Schema

Fields: Name, Description, Price, Category, Images, Stock Quantity.

Categories for organizing products (e.g., Tables, Chairs, Sofas).

Promotion Schema

Fields: Title, Description, Discount Percentage, Validity Dates.

Used for managing sales or seasonal offers.

Dynamic Updates

- Sanity will allow non-developers to update content directly, such as adding new products or editing existing ones.
- Real-time updates will reflect instantly on the frontend via GROQ.

Third-Party APIs

Third-party APIs will be integrated to provide essential functionality beyond the core system. These include:

Payment Gateway: Stripe

- Handles secure transactions for credit/debit cards and other payment methods.
- Provides webhook-based updates for payment status.

Shipment Tracking: ShipEngine

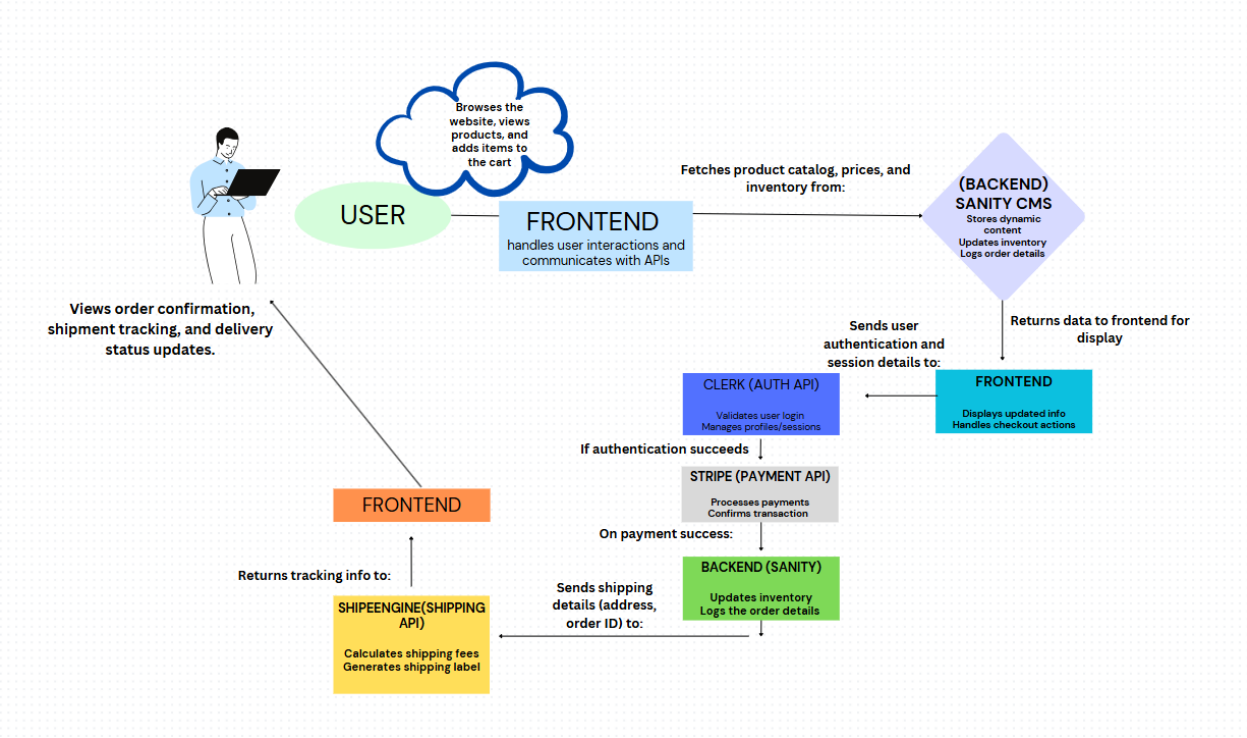
- Offers real-time tracking of deliveries.
- Automatically calculates shipping costs based on product weight and destination.

Authentication: Clerk

- Enables user login via email/password or third-party providers (Google, Facebook).

- Manages session tokens for secure access to user-specific features.

2. System Architecture Plan



Data Flow Process

1. User Interaction

- **User Browses Website**
  The user browses the e-commerce website, views product catalogs, and adds items to the cart.

2. Frontend Operations

- **Handles User Interactions**
  The frontend processes user interactions like browsing, cart management, and checkout.

- **Communicates with APIs**
  Fetches product data, inventory, and prices from the backend CMS (Sanity).

3. Backend (Sanity CMS)

- **Provides Dynamic Content**
  The backend (Sanity CMS) manages and delivers product catalogs, inventory updates, and logs order details.

- **Sends Data to Frontend**
  Returns the required data (products, prices, availability) for display and user interaction.

4. Authentication (Clerk API)

- **Validates User Login**
  Clerk API verifies user login credentials and manages session details.

- **Sends Session Details**
  If authentication succeeds, the session details are sent back to the frontend.

5. Payment Processing (Stripe API)

- **Processes Payments**
  Stripe API handles payment processing and transaction confirmation.

- **Payment Success**
  On payment success, the transaction details are sent to the backend.

6. Backend Updates

- **Updates Inventory and Logs Orders**
  Sanity CMS updates product inventory and logs order details after receiving payment confirmation.

7. Shipping and Tracking (ShipEngine API)

- **Calculates Shipping Fees**
  The ShipEngine API calculates shipping fees based on the user’s address.

- **Generates Shipping Labels**
  It creates shipping labels and provides tracking information.

8. Frontend Displays Updates

- **Order Confirmation and Shipment Tracking**
  The frontend displays order confirmation, shipment tracking details, and delivery status to the user.

Key Workflows

- ## User Registration:
  - Step 1: The user submits registration details via the frontend.
  - Step 2: The frontend sends the data to the backend (Sanity CMS or custom backend).
  - Step 3: Backend validates and stores the user data, then returns a success response.
  - Step 4: The frontend confirms registration to the user.
- ## Product Browsing:
  - Step 1: The user selects a category or searches for a product.
  - Step 2: Frontend sends a request to Sanity CMS with the search or filter criteria.
  - Step 3: Sanity CMS responds with the relevant product data.
  - Step 4: The frontend renders the data for the user.
- ## Order Placement:
  - Step 1: The user adds products to the cart and proceeds to checkout.
  - Step 2: Frontend collects order details and payment information.
  - Step 3: Payment information is sent to Stripe via API.
  - Step 4: Stripe processes the payment and sends a confirmation or error response.
  - Step 5: If payment is successful, the frontend sends the order details to the backend for storage and order confirmation.
- ## Shipment Tracking:
  - Step 1: The user selects an order to track.

- Step 2: Frontend sends a request to the ShipEngine API using the order's tracking ID.
- Step 3: ShipEngine API responds with real-time tracking data.
- Step 4: The frontend displays tracking information to the user.

3. Plan API Requirements

API Requirements:

1. Authentication APIs

Endpoint: /api/auth/register

Method: POST

Description: Registers a new user.

Request Body:

```
{
  "email": "user@example.com",
  "password": "securePassword123"
}
```

Response Example:

```
{
  "message": "User registered successfully.",
  "userId": "123456"
}
```

Endpoint: /api/auth/login

Method: POST

Description: Logs in an existing user.

Request Body:

```
{
  "email": "user@example.com",
  "password": "securePassword123"
}
```

Response Example:

```
{
  "message": "Login successful.",
  "token": "jwt-token-example"
}
```

2. Product APIs

Endpoint: /api/products

Method: GET

Description: Fetches a list of products.

Response Example:

```
[
  {
    "id": "1",
    "name": "Wooden Chair",
    "price": 150,
    "category": "Furniture",
    "stock": 20
```

```
  },
  {
    "id": "2",
    "name": "Oak Table",
    "price": 300,
    "category": "Furniture",
    "stock": 10
  }
]
```


Endpoint: /api/products/:id

Method: GET

Description: Fetches details of a single product by ID.

Response Example:


```
{
  "id": "1",
  "name": "Wooden Chair",
  "price": 150,
  "description": "A sturdy wooden chair made of oak.",
  "category": "Furniture",
  "stock": 20,
  "images": ["url-to-image"]
}
```

3. Cart APIs

Endpoint: /api/cart

Method: POST

Description: Adds an item to the user's cart.

Request Body:

```
{
  "userId": "123456",
  "productId": "1",
  "quantity": 2
}
```

Response Example:

```
{
  "message": "Item added to cart.",
  "cartId": "7890"
}
```

Endpoint: /api/cart/:userId

Method: GET

Description: Retrieves the current user's cart.

Response Example:

```
{
  "userId": "123456",
  "items": [
```

```
    {
      "productId": "1",
      "name": "Wooden Chair",
      "price": 150,
      "quantity": 2
    }
  ]
}
```


4. Order APIs

Endpoint: /api/orders

Method: POST

Description: Creates a new order for a user.

Request Body:


```
{
  "userId": "123456",
  "cartId": "7890",
  "paymentStatus": "success"
}
```


Response Example:


```
{
  "message": "Order placed successfully.",
  "orderId": "56789"
}
```

Endpoint: /api/orders/:orderId

Method: GET

Description: Retrieves details of a specific order.

Response Example:


```
{
  "orderId": "56789",
  "userId": "123456",
  "items": [
    {
      "productId": "1",
      "name": "Wooden Chair",
      "price": 150,
      "quantity": 2
    }
  ],
  "totalPrice": 300,
  "status": "Shipped"
}
```


5. Payment APIs

Endpoint: /api/payment

Method: POST

Description: Processes a payment using Stripe.

Request Body:

```
{
  "userId": "123456",
  "amount": 300,
  "paymentMethod": "card"
}
```

Response Example:

```
{
  "message": "Payment successful.",
  "transactionId": "tx_123abc"
}
```

6. Shipment APIs

Endpoint: /api/shipment

Method: POST

Description: Creates a shipment order.

Request Body:

```
{
  "orderId": "56789",
  "address": "123 Street, City, Country"
}
```

Response Example:

```
{
  "message": "Shipment created successfully.",
```

```
  "trackingId": "track_123abc"
}
```


Endpoint: /api/shipment/:trackingId

Method: GET

Description: Retrieves shipment tracking details.

Response Example:


```
{
  "trackingId": "track_123abc",
  "status": "In Transit",
  "estimatedDelivery": "2025-01-25"
}
```


SANITY DATA SCHEMA

```
export default {
  name: 'ecommerce',
  type: 'document',
  title: 'Ecommerce Data',
  fields: [
```

```
// User Schema
defineType({
  name: 'user',
  type: 'document',
  title: 'User',
  fields: [
    defineField({ name: 'email', type: 'string', title: 'Email' }),
    defineField({ name: 'password', type: 'string', title: 'Password', hidden: true }),
  ],
}),

// Product Schema
defineType({
  name: 'product',
  type: 'document',
  title: 'Product',
  fields: [
    defineField({ name: 'name', type: 'string', title: 'Name' }),
    defineField({ name: 'price', type: 'number', title: 'Price' }),
    defineField({ name: 'category', type: 'string', title: 'Category' }),
    defineField({ name: 'stock', type: 'number', title: 'Stock' }),
    defineField({ name: 'description', type: 'text', title: 'Description' }),
    defineArrayMember({
      name: 'images',
```

```
      type: 'array',
      title: 'Images',
      of: [{ type: 'url' }],
    }),
  ],
}),

                    // Cart Schema
defineType({
  name: 'cart',
  type: 'document',
  title: 'Cart',
  fields: [
    defineField({ name: 'userId', type: 'reference', to: [{ type: 'user' }], title: 'User ID' }),
    defineArrayMember({
      name: 'items',
      type: 'array',
      title: 'Cart Items',
      of: [
        defineType({
          name: 'cartItem',
          type: 'object',
          fields: [
            { name: 'productId', type: 'reference', to: [{ type: 'product' }], title: 'Product ID' },
```

```
          { name: 'quantity', type: 'number', title: 'Quantity' },
        ],
      }),
    ],
  }),
],
}),


                    // Order Schema
defineType({
  name: 'order',
  type: 'document',
  title: 'Order',
  fields: [
    defineField({ name: 'userId', type: 'reference', to: [{ type: 'user' }], title: 'User ID' }),
    defineField({ name: 'cartId', type: 'reference', to: [{ type: 'cart' }], title: 'Cart ID' }),
    defineArrayMember({
      name: 'items',
      type: 'array',
      title: 'Order Items',
      of: [
        defineType({
          name: 'orderItem',
          type: 'object',
```

```
        fields: [
          { name: 'productId', type: 'reference', to: [{ type: 'product' }], title: 'Product ID' },
          { name: 'name', type: 'string', title: 'Product Name' },
          { name: 'price', type: 'number', title: 'Price' },
          { name: 'quantity', type: 'number', title: 'Quantity' },
        ],
      }),
    ],
  }),
  defineField({ name: 'totalPrice', type: 'number', title: 'Total Price' }),
  defineField({ name: 'status', type: 'string', title: 'Order Status' }),
 ],
}),


                    // Payment Schema
defineType({
 name: 'payment',
 type: 'document',
 title: 'Payment',
 fields: [
  defineField({ name: 'userId', type: 'reference', to: [{ type: 'user' }], title: 'User ID' }),
  defineField({ name: 'amount', type: 'number', title: 'Amount' }),
  defineField({ name: 'paymentMethod', type: 'string', title: 'Payment Method' }),
  defineField({ name: 'transactionId', type: 'string', title: 'Transaction ID' }),
```

```
    ],
  }),


                // Shipment Schema
  defineType({
    name: 'shipment',
    type: 'document',
    title: 'Shipment',
    fields: [
      defineField({ name: 'orderId', type: 'reference', to: [{ type: 'order' }], title: 'Order ID' }),
      defineField({ name: 'address', type: 'text', title: 'Shipping Address' }),
      defineField({ name: 'trackingId', type: 'string', title: 'Tracking ID' }),
      defineField({ name: 'status', type: 'string', title: 'Shipment Status' }),
      defineField({ name: 'estimatedDelivery', type: 'datetime', title: 'Estimated Delivery' }),
    ],
  }),
 ],
};
```

4. Security Measures

Authentication & Authorization:

- Implement role-based access control (RBAC).
- Secure sessions with JWT and short-lived tokens.

Data Protection:

- Use HTTPS for encrypted communication.
- Encrypt sensitive data (e.g., passwords using bcrypt).

API Security:

- Validate all inputs to prevent SQL injection and XSS.
- Use rate limiting to prevent abuse.

5. Deployment Strategy

Continuous Integration/Deployment (CI/CD):

GitHub Actions for automating build, test, and deploy pipelines.

Environment Management:

Use .env files for environment-specific variables.

Hosting:

Deploy to Vercel with custom domain configuration.

Conclusion:

This document provides a detailed overview of the technical foundation and design of the Furniture E-Commerce System, outlining the system's architecture, components, and functionalities. The goal is to create a dynamic and efficient e-commerce platform where users can browse, purchase, and track their furniture orders. This documentation will serve as a reference for developers, stakeholders, and team members to understand how the system is structured and functions.