Day 2: Planning the Technical Foundation

Step 1: Technical Requirements:

1. **Code Editor:** Vs Code Editor

2. **Frontend Requirement:**

   Next JS:

   I Am Using Next JS as a frontend Reasons for choosing next.js are given below:

   a. User Friendly: It has a user-friendly interface.
   b. I have good hand practice with it.
   c. I will make my website scalable and dynamic using it.

3. **Backend Requirement:**

   Sanity as Headless CMS:

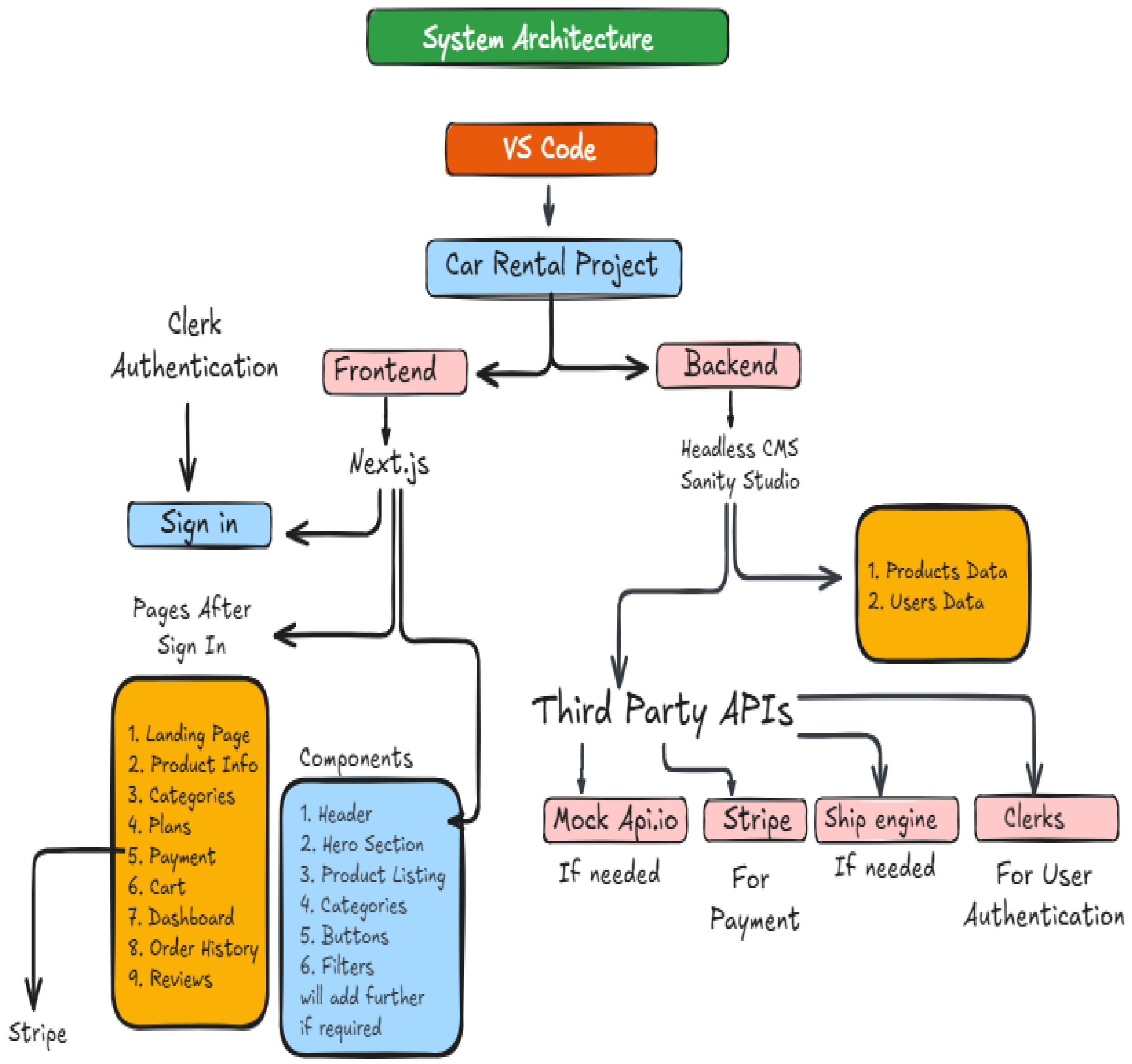
   I am using Sanity as the backend of my project because it is a powerful tool that helps me manage product data, customer details, and order records.

4. **Third Party APIs:**

   a. **Clerk Authentication**
      I will use clerk Authentication to validate the user's sign-in / sign-up.
   b. **Mock Api.io:**
      If I need to fetch data from APIs into sanity, I will use Mock Api.io because I practice creating Mock APIs on it.
   c. **Stripe:**
      I will use Stripe to manage payment gateways.
   d. **Ship Engine:**
      I will use ship engine if I implement tracking in my project.
      As my project is a car rental site related to rental e-commerce I want to add tracking of cars from warehouse to client but it will be done later.

Step 2: System Architecture Design:

Below mentioned Chart shows the System architecture designs that I will implement in my project.



Workflow:

1. User Registration:

- **Frontend (Next.js)** → User fills out the registration form.
- **Sanity CMS** → User Data (e.g. name, email, password) will be stored in Sanity)
- **Confirmation** → A confirmation Email will be sent to the user.

2. Product Browsing:

- **Frontend (Next.js)** → User browses and explores products & their categories.
- **Product Data API (Sanity)** → Fetches product data (e.g., names, images, descriptions, prices) from Sanity CMS.
- **Frontend (Next.js)** → Displays product data dynamically on the website.

3. Order Placement:

- **Frontend (Next.js)** → User adds product to the cart and proceeds to checkout.
- **Frontend (Next.js)** → Order details (Items, quantities, user information) are sent to Sanity CMS.
- **Sanity CMS** → Stored the order details in the Database.
- **Payment Gateway** → Securely processes the payment and confirms the transaction.

4. Shipment Tracking

- **Sanity CMS** → Updates the order with shipping details (e.g., tracking number, carrier)
- **Third-party APIs (Shipment Tracking API)** → Fetching real-time shipment status
- **Frontend (Next.js)** → Displays shipment status (e.g., "In Transit", "Delivered") to the user.

Step 3: API Requirement:

Endpoint Name	Method	Description
/products	GET	Fetch all available products from Sanity
/orders	POST	Create a new order in Sanity CMS
/payment	POST	Process payment for an order
/shipment	GET	Track the shipment status for an order

➢ Endpoint Name/product:

Method: GET
Description: Fetch all available products from Sanity
Response Example

```
{
    “id”:1,
    “name”: “Honda Civic”,
    “price”: “8000”,
    “color”: “Black”,
    “category”: “sports”

 }
```

➢ Endpoint Name /order

Method: POST
Description: Create a new order in Sanity CMS
Payload:

```
{
“productID”: “prod_01”,
“quantity”: “2”,
“totalAmount”: 32000,
“rental_duration”: “2 days”,
“orderDate”:  “01-12-2024”
}
```

Response Example

```
{“status”: “success”,
“message”: “order placed
   successfully”,
“order”:{
“_id”: “order_01”,
“productID”: “prod_01”,
“quantity”: “2”,
“totalAmount”: 32000,
“rental_duration”: “2 days”,
“orderDate”:  “01-12-2024”}
```

➢ Endpoint Name /payment

Method: POST
Description: Process payment for an order
Payload:

```
{
   “order”: {

   “_id”: “order_01”,
   “totalAmount”: 32000
},
“paymentMethod”: “Credit Card”,
“status”: “Completed”,
“transactionID”: “tx_001”,
“totalAmount”: 32000,
“paymentDate”: “01-12-2024”
}
```

Response Example

```
{“paymentstatus”: “success”,
  “transactionID”: “tx_001”,

   “message”: “Payment has been
successfully processed”}
```

➢ Endpoint Name /shipment

Method: POST
Description: Track the shipment status for an order
Response Example:

```
{
   “_id”: “ship_001”,
   “order”: {
            “_id”: “order_001”,
            “totalAmount” 32000
   },
   “trackingNumber”: “track_001”
   “status”: “In Transit”,
   “estimatedDelivery”: “04-12-2024” //depends on the date chosen for it
```

Step 4: Detail About Diagram

The diagram represents the **system architecture** of a **Car Rental Project**. It outlines the relationship between different components, technologies, and third-party services in the project. Here's a detailed breakdown:

Top-Level Overview

- **System Architecture**: The overarching structure of the application.
- **VS Code**: The development environment where the project is being built.

Main Components

1. **Car Rental Project**
    - Divided into **Frontend** and **Backend**.

Frontend

- Built using **Next.js**.
- Handles the **Sign-in** process with the help of Clerks and the pages available after sign-in:
    - **Pages After Sign-in**:
        1. Landing Page
        2. Product Info
        3. Categories
        4. Plans
        5. Payment
        6. Cart
        7. Dashboard
        8. Order History
        9. Reviews

- Also has some fixed (Header/footer) component and reuseable component like (Buttons/filters/products)

    - **Components**:
        - Header
        - Hero Section
        - Product Listing
        - Categories
        - Buttons (additional ones may be added as needed).
        - Filters

Backend

- Uses a **Headless CMS (Sanity Studio)** to manage data:
  - **Products Data**
  - **Users Data**

Third-Party APIs

1. **Mock API.io**: For testing purposes (if needed).
2. **Stripe**: For payment integration.
3. **Ship Engine**: For shipping-related services (if needed).
4. **Clerk**: For user authentication.

User Flow

1. **Clerk Authentication**:
   - Users sign in via Clerk.
   - Access is granted to frontend pages and features after authentication.
2. **Data Flow**:
   - Backend communicates with the **Headless CMS** to fetch/manage data (Products and Users).
   - Third-party APIs are utilized as required for payment, authentication, and shipping.