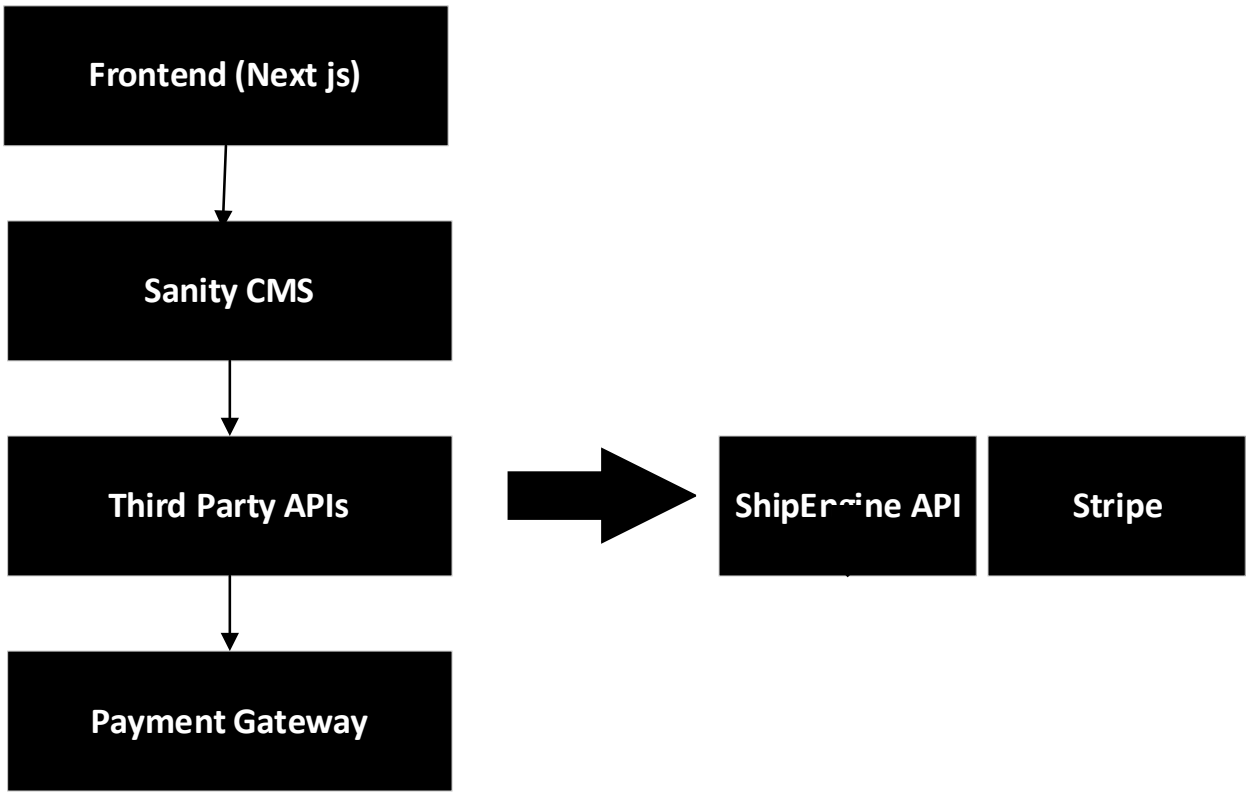
DAY 2: Planning The Technical Foundation

System Architecture Overview:



Key Components:

1. **Frontend (Next js):**
   - Acts as the user interface for the application.
   - Built with Next.js, it allows users to interact with the system, view content, and submit data or requests.

2. **Sanity CMS**:
   - A content management system that manages and stores content dynamically.
   - Provides a backend interface to create, update, and organize content displayed on the frontend.

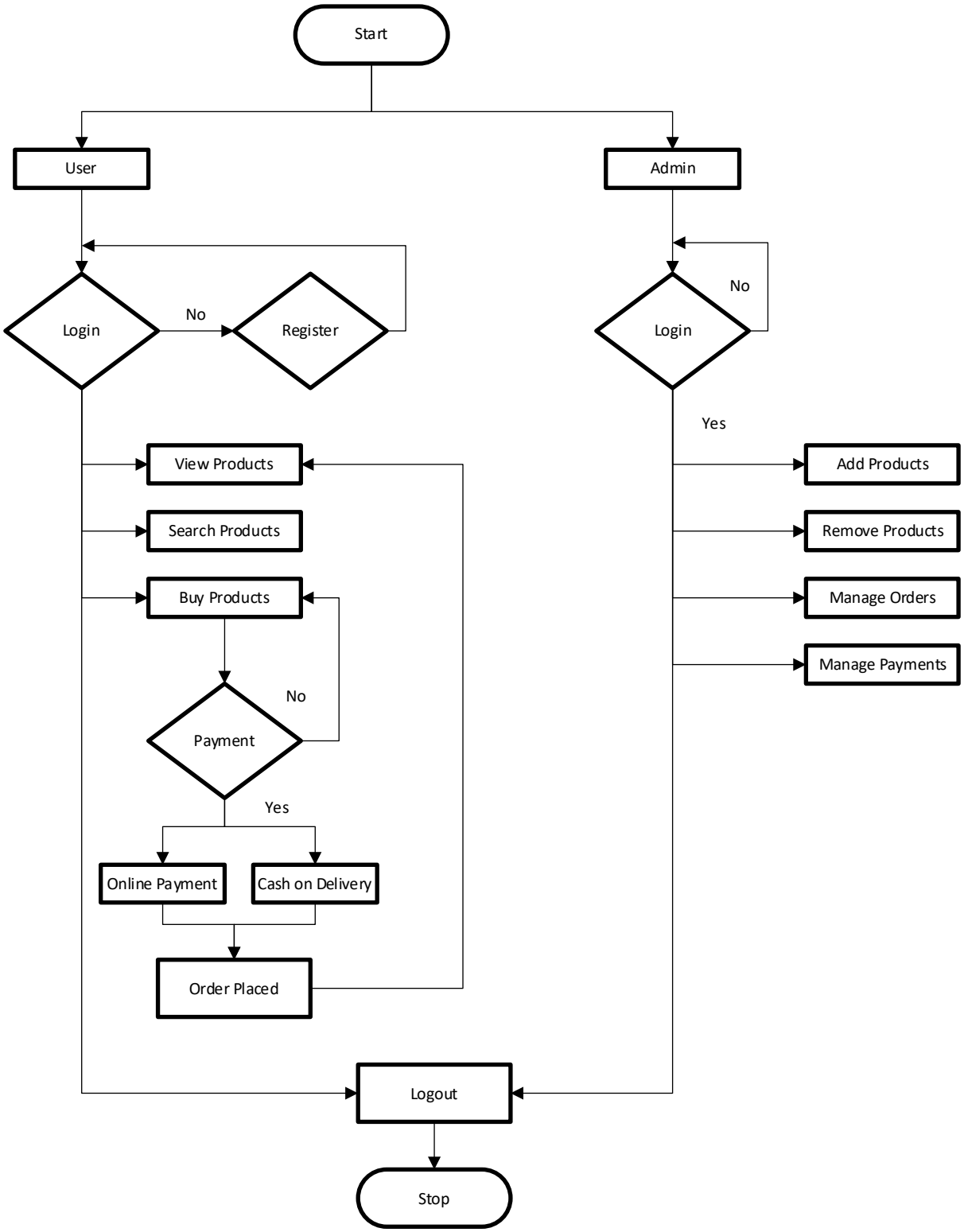
3. **Third-Party APIs**:
   - **ShipEngine API**: Used for managing shipping and logistics, such as calculating shipping rates and tracking shipments.
   - **Stripe**: Handles payment processing, including transactions, subscriptions, and billing.

4. **Payment Gateway**:
   - Facilitates secure payment transactions between the application and payment processors.
   - Ensures sensitive financial data is transmitted securely.

API Requirements:

API Endpoints	Method	Purpose	Request Payload	
/products	GET	Fetches all product details	None	[{ "id": 1, "name": "Product A", "price": 100 }, { "id": 2, "name": "Product B", "price": 150 }]
/products/{id}	GET	Fetches details of a specific product	None	{ "id": 1, "name": "Product A", "price": 100, "stock": 50 }
/products	POST	Adds a new product	{ "name": "Product C", "price": 200, "stock": 30 }	{ "message": "Product added successfully", "id": 3 }
/products/{id}	PUT	Updates an existing product	{ "name": "Product A Updated", "price": 110 }	{ "message": "Product updated successfully" }
/products/{id}	DELETE	Deletes a product	None	{ "message": "Product deleted successfully" }
/categories	GET	Fetches all product categories	None	[{ "id": 1, "name": "Electronics" }, { "id": 2, "name": "Books" }]
/orders	GET	Fetches all orders	None	[{ "orderId": 101, "total": 500, "status": "Pending" }, { "orderId": 102, "total": 300, "status": "Completed" }]
/orders/{id}	GET	Fetches details of a specific order	None	{ "orderId": 101, "items": [{ "productId": 1, "quantity": 2 }], "total": 500, "status": "Pending" }
/orders	POST	Places a new order	{ "items": [{ "productId": 1, "quantity": 2 }, { "productId": 2, "quantity": 1 }] }	{ "message": "Order placed successfully", "orderId": 103 }

Key Workflows:



Sanity Schema Design:

Products Schema

```jsx
export default product = {
  name: "product",
  type: "document",
  title: "Product",
  fields: [
    {
      name: "name",
      type: "string",
      title: "Product Name",
      validation: (Rule) => Rule.required(),
    },
    {
      name: "price",
      type: "number",
      title: "Price",
      validation: (Rule) => Rule.required().min(0),
    },
    {
      name: "stock",
      type: "number",
      title: "Stock",
      validation: (Rule) => Rule.required().min(0),
    },
    {
      name: "category",
      type: "reference",
      to: [{ type: "category" }],
      title: "Category",
    },
    {
      name: "image",
      type: "image",
      title: "Product Image",
    },
    {
      name: "description",
      type: "text",
      title: "Description",
    },
  ],
};
```

Categories Schema

```
const category = {
  name: "category",
  type: "document",
  title: "Category",
  fields: [
    {
      name: "name",
      type: "string",
      title: "Category Name",
      validation: (Rule) => Rule.required(),
    },
    {
      name: "description",
      type: "text",
      title: "Description",
    },
  ],
};

export default category;
```

Orders Schema

```javascript
const order = {
  name: "order",
  type: "document",
  title: "Order",
  fields: [
    {
      name: "orderId",
      type: "string",
      title: "Order ID",
      validation: (Rule) => Rule.required(),
    },
    {
      name: "customer",
      type: "reference",
      to: [{ type: "user" }],
      title: "Customer",
    },
    {
      name: "items",
      type: "array",
      of: [
        {
          type: "object",
          fields: [
            { name: "product", type: "reference", to: [{ type: "product" }] },
            {
              name: "quantity",
              type: "number",
              validation: (Rule) => Rule.required().min(1),
            },
          ],
        },
      ],
      title: "Order Items",
    },
    {
      name: "total",
      type: "number",
      title: "Total Amount",
      validation: (Rule) => Rule.required().min(0),
    },
    {
      name: "status",
      type: "string",
      title: "Order Status",
      options: {
        list: ["Pending", "Completed", "Cancelled"],
      },
      validation: (Rule) => Rule.required(),
    },
  ],
};
export default order;
```

Collaboration Notes:

Feedback: My experience during this marketplace hackathon was incredible, as it provided me with valuable insights into business operations and how businesses work, enhancing both my technical and strategic understanding.