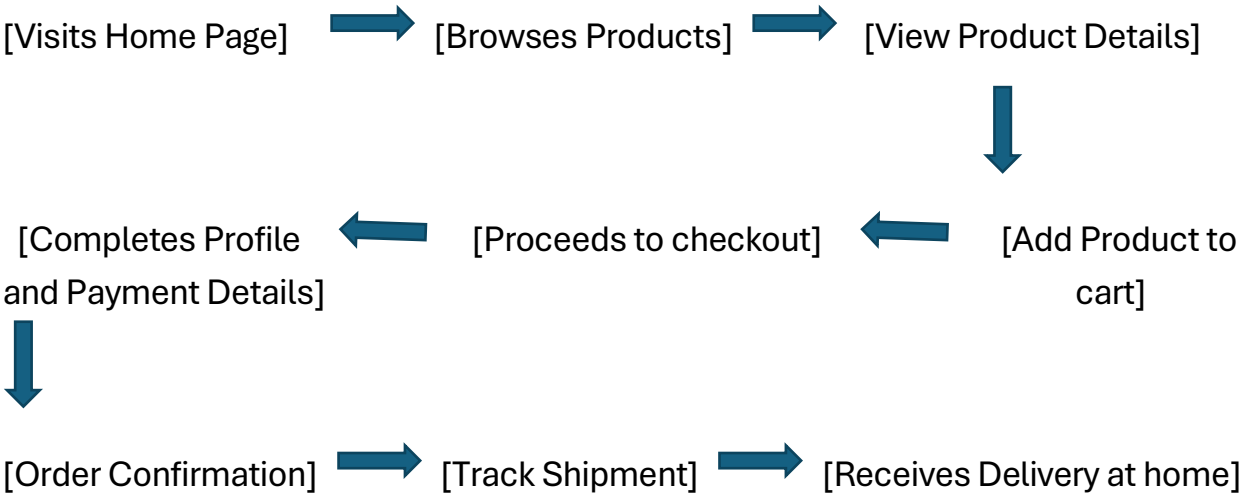
Submission title:

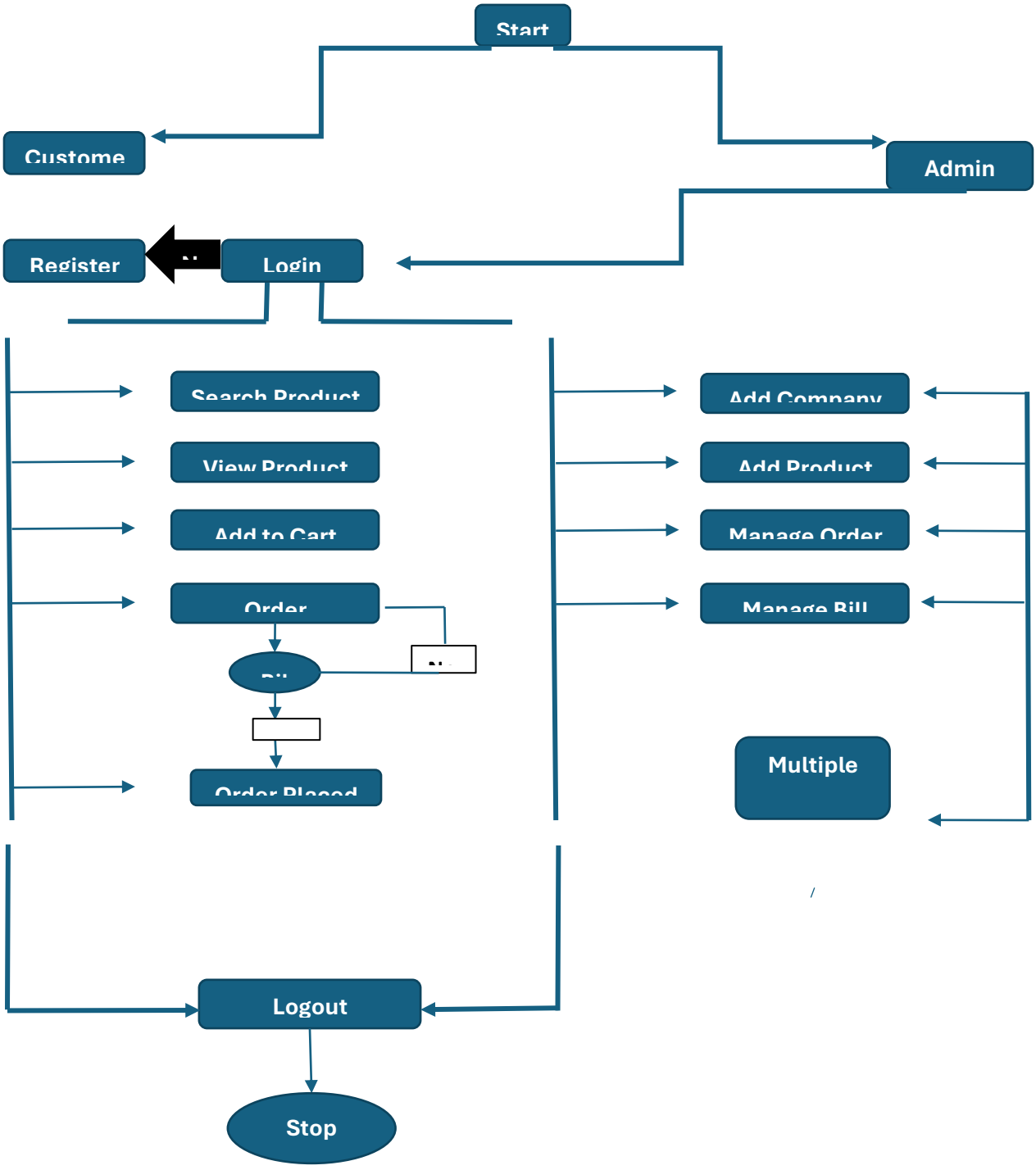
"Marketplace Technical Foundation - [Rida's Ecommerce Project]"

Workflow Diagram Visual Representation:



Frontend Requirements:

Flow Chart by Rida:



APIEndpoints:

Endpoint	Method	Description	Parameters	Response Example
/api/furniture	GET	Fetch all furniture items	None	`{ id: 1, name: "Sofa" }`
/api/furniture/:id	GET	Fetch a single furniture item	id (Path)	`{ id: 1, name: "Sofa" }`
/api/furniture	POST	Add a new furniture item	name, price, category (Body)	`{ success: true, id: 10 }`
/api/furniture/:id	PUT	Update a furniture item	id (Path), name, price (Body)	`{ success: true }`
/api/furniture/:id	DELETE	Delete a furniture item	id (Path)	`{ success: true }`
/api/categories	GET	Fetch all furniture categories	None	`{ categories: ["Chairs", "Tables"] }`

Component Roles:

1. **Frontend with Next.js**: The frontend should be built using **Next.js**, leveraging its capabilities like server-side rendering (SSR), static site generation (SSG), and API routes to ensure fast, SEO-friendly, and scalable web pages.
2. **Sanity CMS Integration**: Use **Sanity CMS** to manage and update content dynamically. Ensure that the CMS is set up with a clear content model, allowing non-developers to easily add and edit content in real time. The CMS should be seamlessly integrated with the Next.js frontend.
3. **Third-Party API Integration**: The application must integrate at least one **third-party API** (e.g., for payment processing, user authentication, external data fetching, or social media feeds). The API should be consumed effectively, displaying its data on the frontend in a user-friendly manner.
4. **Performance and Optimization**: Focus on optimizing the application for fast load times and a smooth user experience. Use Next.js features like lazy loading, code splitting, and image optimization to improve performance.

5. **Documentation:** Provide clear documentation on how the application works, including setup instructions for running the project, integrating the CMS, and interacting with the third-party API.

Key Workflows:

1. User Registration & Authentication

- **Frontend**:
  - Users can register or log in via a form on the frontend (Next.js).
  - A third-party API (e.g., Firebase Auth, Auth0) handles user authentication and registration.
- **Backend**:
  - The API verifies the credentials and creates user profiles in the system.
- **CMS**:
  - Store additional user data (like preferences or past orders) in **Sanity CMS** for personalized content or recommendations.
- **Workflow**:
  - Upon registration or login, a session or JWT token is created and stored on the client side for persistent authentication.

2. Product Browsing

- **Frontend**:
  - Products are displayed dynamically on the frontend, with Next.js fetching data either at build time (SSG) or runtime (SSR).
- **CMS**:
  - Products, categories, and pricing information are managed through **Sanity CMS**. Content creators can update product details without requiring a developer.
- **API**:
  - The frontend fetches product details, images, and availability from the CMS via Sanity's API (using GROQ queries).
- **Workflow**:
  - Users can browse product categories and view individual product pages, with real-time updates if the CMS data changes.

3. Order Placement

- **Frontend**:
  - Users can add products to their cart, review their order, and proceed to checkout.
- **API**:
  - An API call is made to submit the order, and payment processing is handled through a third-party API (e.g., Stripe or PayPal).
  - Once the payment is processed, an order confirmation and a unique order ID are generated.
- **CMS**:
  - Order data (such as order details, items, and customer info) can be stored and managed in the CMS for administrative purposes.
- **Workflow**:
  - After payment, the user receives an email with order details and an estimated shipping date.

4. Shipment Tracking

- **Frontend**:
  - Users can track the status of their orders by entering an order ID or logging into their account.
- **API**:
  - The system integrates with a third-party shipping API (e.g., UPS, FedEx) to fetch real-time shipment tracking information.
- **CMS**:
  - Shipment status (shipped, in transit, delivered) is updated in the CMS, which can be used for admin dashboards.
- **Workflow**:
  - As the order progresses, users can see live updates on their shipment status, with tracking numbers, delivery date estimates, and carrier information.

Sanity Schema Examples:

1. User Schema

```
export default {
  name': 'user',
  type: 'document',
  fields: [
    { name: 'username', type: 'string' },
    { name: 'email', type: 'string' },
    { name: 'orderHistory', type: 'array', of: [{ type: 'reference',
to: [{ type: 'order' }] }] },
  ],
}
```

2. Product Schema

```
export default {
  name: 'product',
  type: 'document',
  fields: [
    { name: 'title', type: 'string' },
    { name: 'description', type: 'text' },
    { name: 'price', type: 'number' },
    { name: 'image', type: 'image' },
    { name: 'stock', type: 'number' },
  ],
}
```

3. Order Schema

```
export default {
  name: 'order',
  type: 'document',
  fields: [
    { name: 'orderId', type: 'string' },
    { name: 'user', type: 'reference', to: [{ type: 'user' }] },
    { name: 'products', type: 'array', of: [{ type: 'reference', to:
```

```
[{ type: 'product' }] }] },
    { name: 'totalPrice', type: 'number' },
    { name: 'paymentStatus', type: 'string', options: { list:
['Pending', 'Completed', 'Failed'] } },
  ],
}
```

4. Shipment Schema

```
export default {
  name: 'shipment',
  type: 'document',
  fields: [
    { name: 'trackingNumber', type: 'string' },
    { name: 'carrier', type: 'string' },
    { name: 'shipmentStatus', type: 'string', options: { list: ['In
Transit', 'Out for Delivery', 'Delivered'] } },
  ],
}
```

5. Checkout Schema (Optional)

```
export default {
  name: 'checkout',
  type: 'document',
  fields: [
    { name: 'user', type: 'reference', to: [{ type: 'user' }] },
    { name: 'cartItems', type: 'array', of: [{ type: 'reference', to:
[{ type: 'product' }] }] },
    { name: 'totalAmount', type: 'number' },
    { name: 'isCheckedOut', type: 'boolean', initialValue: false },
  ],
}
```

Entity Relationship Diagram: