Task DAY 2

Nazia Imran
16-jan-2025

HEKTO

THE TECHNICAL FOUNDATION for

Transitioning to Marketplace

1.System Architecture Overview

Here's a high-level system architecture diagram and description for my Next.js project based on the given routes and components. This architecture assumes modular design, clear separation of concerns, and scalability.

System Architecture Description

1. Application Structure

- **Pages Directory (`src/app`)**
  Handles routing and page-level logic. Each page in the directory corresponds to a specific route in my application. Dynamic routing is supported for detail views (e.g., `blog/[singleblog]` and `product/[productdetail]`).

- **Components Directory (`src/components`)**
  Contains reusable UI components, such as cards, sections, headers, and footers. These components are used to build pages.

2.Architecture Layers

- **Frontend Presentation Layer**
  - Built with React and Next.js.
  - Handles UI/UX with reusable components from the `src/components` directory.

    - Ensures responsive and dynamic rendering via server-side rendering (SSR) or static site generation (SSG) provided by Next.js.

- **Routing Layer**
    - Defined under `src/app`.
    - Manages navigation and maps URLs to specific pages. Dynamic routes like `product/[productdetail]` and `blog/[singleblog]` handle detail views for products and blog entries, respectively.

- **Component Layer**
    - A collection of reusable, modular, and atomic components such as `CategoryCard`, `Footer`, and `Navbar`.
    - Organized by feature or purpose to maintain scalability and readability.

- **State Management Layer**
    - If applicable (not explicitly stated in directory), a global state management tool (like Redux, Context API, or Zustand) can handle shared states like cart data or user authentication.

- **Backend or API Layer** (Integration)
    - Though part of my current directory, backend API integration can be assumed for dynamic data like product details, blogs, and cart items. API calls would typically be handled via `getServerSideProps` or `getStaticProps`.

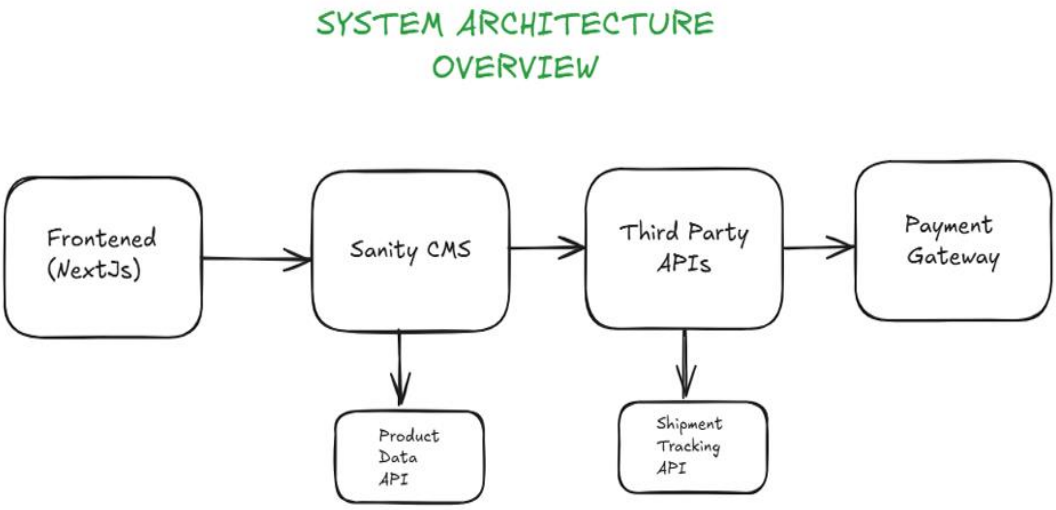


Diagram: Here is a high-level system architecture diagram:

Detailed Folder Overview

1. **`src/app`** (Routing and Pages)
    - **Static Routes**: `/about`, `/billing`, `/blog`, `/cart`, `/completed`, `/contact`, `/faq`, `/login`, `/shopgrid`.
    - **Dynamic Routes**: `/blog/[singleblog]`, `/product/[productdetail]`.
2. **`src/components`** (Reusable Components)
    - **Cards**: `CategoryCard`, `LeatestCard`, `TrendingCard`, etc.
    - **Sections**: `FeatureSection`, `FilterSection`, `Hero`, etc.
    - **Shared Elements**: `Navbar`, `Footer`, `Logo`,

2.Sanity CMS io:

1.Acts as the backend to store and manage product, customer, and order data.

2.Provides APIs to fetch and update data dynamically.

[API/Database]

|

+--> Fetched in pages via SSR or SSG.

|

+--> Passed as props to components for rendering.

Workflow:

1. **User Interaction**:
   - User navigates to `/shopgrid` or `/product` to browse products.

2. **Data Fetching**:
   - **Sanity API** fetches product categories and data (title, price, stock, images).

3. **Frontend Display**:
   - Products are displayed using reusable components like `Category` and `FilterSection`.
   - Users can filter products by category or search parameters.

---

3. Order Placement

Files Involved:

- `src/app/cart/page.tsx`
- `src/app/billing/page.tsx`
- `src/components/shopcard.tsx`
- `src/components/Footer.tsx`
- `Sanity CMS`

Workflow:

1. **User Interaction**:
   - User adds products to the cart from `/product/[productdetail]` or `/shopgrid`.
   - Items are displayed on the `/cart` page via the `shopcard` component.

2. **Checkout Process**:
   - User proceeds to `/billing` to complete the order.

3. **Data Handling**:
   - Order details (user info, product IDs, total amount) are saved in **Sanity CMS**.
   - Payment Gateway processes the transaction.

---

4. Shipment Tracking

Files Involved:

- `src/app/completed/page.tsx`
- `src/components/Header.tsx`
- **Third-Party API**

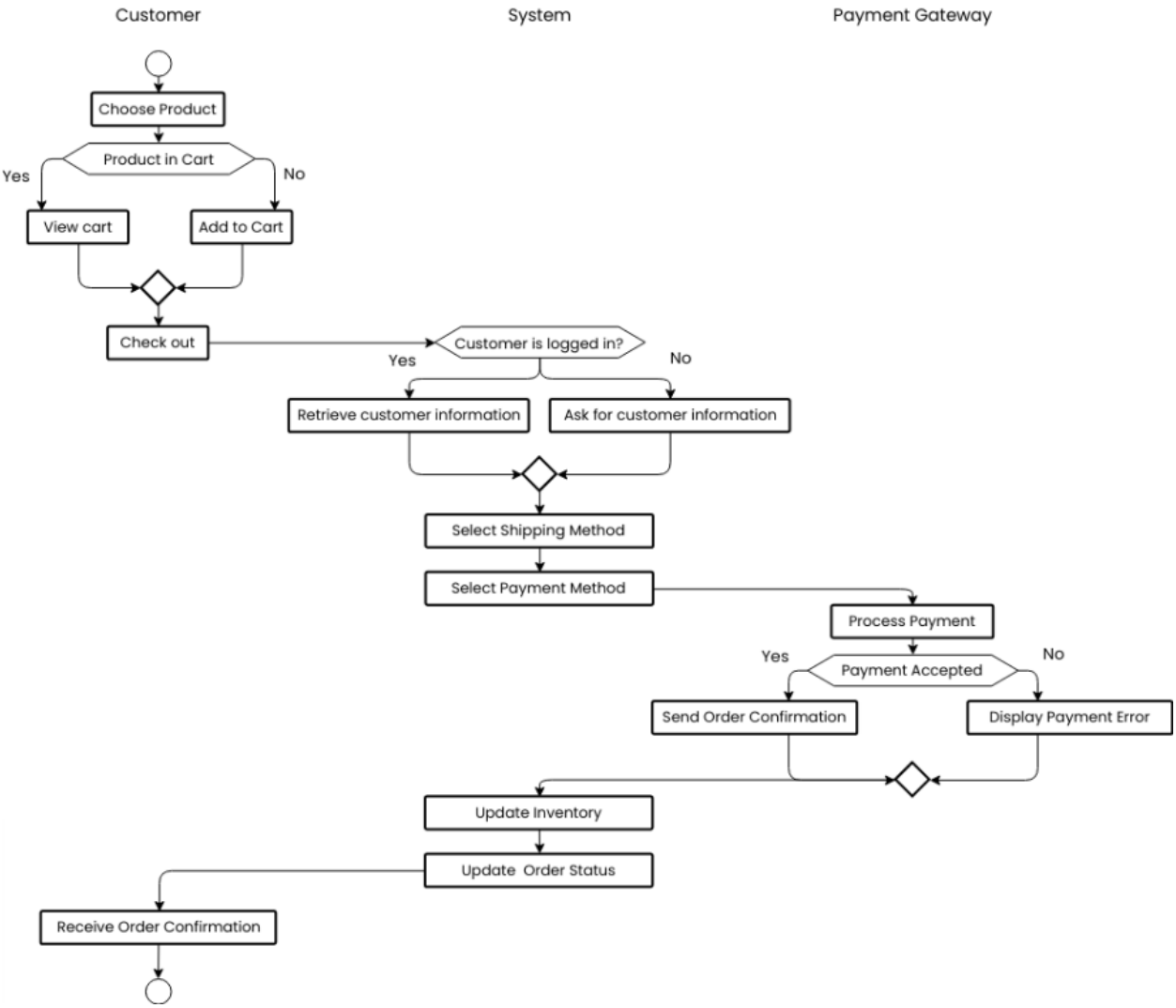
Workflow:

1. **User Interaction**:
   - After placing an order, users can check their shipment status on `/completed`.

2. **Data Fetching**:
   - **Third-Party API** fetches real-time shipment updates (e.g., status, delivery time).

3. **Frontend Display**:
    - Shipment details are shown dynamically on `/completed`.
    - Users can view updates like "Order Dispatched" or "Delivered."

Diagram Work Flow:



From Placement Order To Delivery

5.API Endpoints

Authentication

1. `POST /api/users/register` – Register a new user.
2. `POST /api/users/login` – Authenticate user and return a JWT token.

Categories

1. `GET /api/categories` – Retrieve all product categories.
2. `GET /api/categories/{id}/products` – Retrieve products under a specific category.

Products

1. `GET /api/products/{id}` – Retrieve detailed information about a specific product.
2. `GET /api/products` – Retrieve all products with filtering and pagination options.

Cart

1. `POST /api/cart/add` – Add a product to the user's cart.
2. `GET /api/cart` – Retrieve the current user's cart details.

Orders

1. `POST /api/orders/checkout` – Process checkout and create an order.
2. `GET /api/orders/{id}` – Retrieve the details of a specific order.

Payment

1. `POST /api/orders/checkout` – Process payment and checkout.
2. `GET /api/orders/{id}` – Retrieve the details of a specific payment.

Shipment

1. `POST /api/orders/shipment` – Process shipment and create an order.
2. `GET /api/orders/{id}` – Retrieve the details of a specific shipment.

Comments

1. `GET /api/comments` – Retrieve all comments posts.
2. `GET /api/comments/{id}` – Retrieve details of a specific comments post.

FAQ

1. `GET /api/faqs` – Retrieve frequently asked questions for user support.