Marketplace Technical Foundation - LibasG

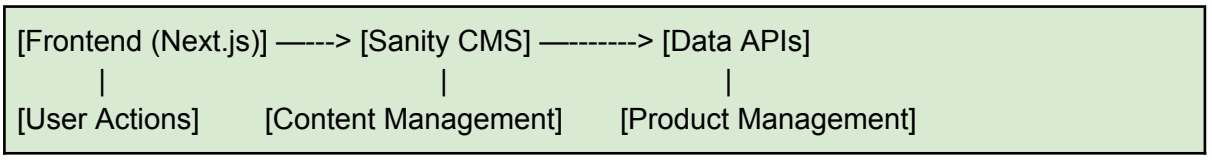
---

1. System Architecture

Overview:

The system architecture is designed to ensure scalability, simplicity, and user-centric functionality for the clothing store marketplace 'LibasG'.

Architecture Diagram:



Components:

1. **Frontend (Next.js):**
    - Handles the user interface (UI) and interactions.
    - Implements responsive designs for seamless use across devices.
2. **Backend (Sanity CMS):**
    - Manages product data, user profiles, and order details.
    - Provides APIs to the front end for data fetching.
3. **APIs:**
    - Facilitates communication between the front end and back end.
    - Supports operations like fetching product listings and managing orders.
4. **Database:**
    - Data is managed within Sanity CMS using defined schemas.
    - Stores product, user, and order information.

---

2. Data Flow and Workflows

Data Flow:

1. **Frontend to Backend:**
    - User actions (e.g., searching, adding to cart) trigger API calls to fetch or modify data.
2. **Backend to Database:**
    - Sanity CMS processes API requests and interacts with the database.
3. **Response:**
    - API sends structured data (JSON format) back to the front end for rendering.

Key Workflows:

1. **Product Browsing:**
    - The user selects a category or searches for products.
    - API fetches filtered product data from Sanity CMS.
    - The front end renders the product listing page dynamically.
2. **Order Placement:**
    - A user adds products to the cart and proceeds to checkout.
    - Checkout details are sent to Sanity CMS, which stores the order.
    - Order confirmation is displayed to the user.
3. **Admin Workflow:**
    - Admin adds or updates product information via Sanity CMS.
    - Changes are immediately available via the APIs.

---

3. API Specifications

Endpoints:

Endpoint	Method	Purpose	Response Example
`/products`	GET	Fetch all products	`{ "id": 1, "name": "Shirt", "price": 1500 }`
`/products/:id`	GET	Fetch details of a single product	`{ "id": 1, "name": "Shirt", "sizes": ["M"] }`
`/categories`	GET	Fetch product categories	`{ "id": 1, "name": "Men" }`
`/users`	GET	Fetch all user profiles	`{ "id": 1, "name": "Ahmed", "email": "test@example.com" }`
`/users/:id`	GET	Fetch details of a single user	`{ "id": 1, "name": "Ahmed", "address": "123 Street" }`
`/cart`	POST	Add product to cart	`{ "status": "success", "cartId": "123" }`
`/orders`	POST	Place a new order	`{ "orderId": "456", "status": "confirmed" }`
`/orders/:id`	GET	Fetch details of an existing order	`{ "orderId": "456", "status": "shipped" }`

Key Features:

- Supports CRUD operations for products, categories, and orders.
- Returns JSON responses to ensure compatibility with frontend.
- Includes status codes for success and error handling.

---

4. Sanity CMS Schema

Product Schema:

```
export default {
    name: 'product',
    type: 'document',
    fields: [
      { name: '_id', type: 'string', title: 'Product ID' },
      { name: 'name', type: 'string', title: 'Product Name' },
      { name: 'price', type: 'number', title: 'Price' },
      { name: 'sizes', type: 'array', of: [{ type: 'string' }], title: 'Available Sizes' },
      { name: 'colors', type: 'array', of: [{ type: 'string' }], title: 'Available Colors' },
      { name: 'category', type: 'string', title: 'Product Category' },
      { name: 'imageURL', type: 'image', title: 'Product Image' },
      { name: 'description', type: 'text', title: 'Description' },
      { name: 'discountPercent', type: 'number', title: 'Discount Percent' },
      { name: 'isNew', type: 'boolean', title: 'Is New' }
    ]
  };
```

Category Schema:

```
export default {
    name: 'category',
    type: 'document',
    fields: [
      { name: 'category', type: 'string', title: 'Category Name' },
      { name: 'name', type: ‘string’, title: 'Product Name' }
    ]
  };
```

User Schema:

```
export default {
  name: 'user',
  type: 'document',
  fields: [
    { name: 'name', type: 'string', title: 'User Name' },
    { name: 'email', type: 'string', title: 'Email Address' },
    { name: 'phone', type: 'string', title: 'Phone Number' },
    { name: 'address', type: 'text', title: 'Address' }
  ]
};
```

Order Schema:

```
export default {
  name: 'order',
  type: 'document',
  fields: [
    { name: 'orderId', type: 'string', title: 'Order ID' },
    { name: 'product', type: 'reference', to: [{ type: 'product' }], title: 'Product' },
    { name: 'user', type: 'reference', to: [{ type: 'user' }], title: 'User' },
    { name: 'status', type: 'string', title: 'Status' },
    { name: 'timestamp', type: 'datetime', title: 'Order Date' }
  ]
};
```

---

5. Security and Performance

1. **Environment Variables:**
    - Use `.env` files to store API keys securely.

Example:

```
NEXT_PUBLIC_API_URL=https://api.example.com
```

2. **Error Handling:**
    - Implement error logging for debugging.
    - Display user-friendly error messages.
3. **Optimization Techniques:**
    - Enable caching for frequently accessed APIs.
    - Minimize large image sizes for faster load times.
