Hackathon Day 2: Planning the Technical Foundation

MyMarketPlace “FashionShop”

1. Technical Requirements For My E-**commerce** Website

1. Frontend Requirements

The frontend will provide a **user-friendly interface** to ensure a seamless shopping experience. The key features include:

User Interface & Experience

1. Intuitive and visually appealing UI for browsing products.
2. Responsive design to support **mobile and desktop** users.
3. Fast loading speeds and optimized performance.

Essential Pages

1. **Home Page:** Displays featured products, promotions, and categories.
2. **Product Listing Page:** Lists all available products with sorting and filtering options.
3. **Product Details Page:** Shows product images, descriptions, prices, reviews, and an **Add to Cart** button.
4. **Cart Page:** Displays selected products with a summary and total cost.
5. **Checkout Page:** Includes user details, payment options, and order summary.
6. **Order Confirmation Page:** Confirms the successful purchase with order details and tracking link.

2. Backend Requirements (Sanity CMS)

The backend will be powered by **Sanity CMS**, which will serve as the content and data management system.

Sanity CMS Features

- **Product Management:**
  - Create, update, and delete product listings.
  - Manage inventory levels and product attributes (e.g., size, color, category).
- **Customer Data Handling:**
  - Store user profiles, addresses, and past orders securely.
- **Order Management:**
  - Track order status (pending, shipped, delivered, etc.).
  - Maintain order history for customers.

Schema Design in Sanity CMS

- **Product Schema:** Includes name, price, description, images, category, stock status, and tags.
- **User Schema:** Stores user ID, name, email, address, and past purchases.
- **Order Schema:** Contains order details, payment status, shipment tracking, and timestamps.

3. Third-Party API Integrations

To enhance functionality, the website will integrate **third-party APIs** for seamless operations.

Essential API Integrations

- **Payment Gateway API:** Secure transactions via Stripe, PayPal, or Razorpay.
- **Shipment Tracking API:** Real-time tracking updates from shipping providers.
- **Email Notification API:** Order confirmations and updates sent via SMTP or SendGrid.
- **Tax & Currency API:** Automatic calculation of taxes and currency conversion for international users.

4. Security & Performance Considerations

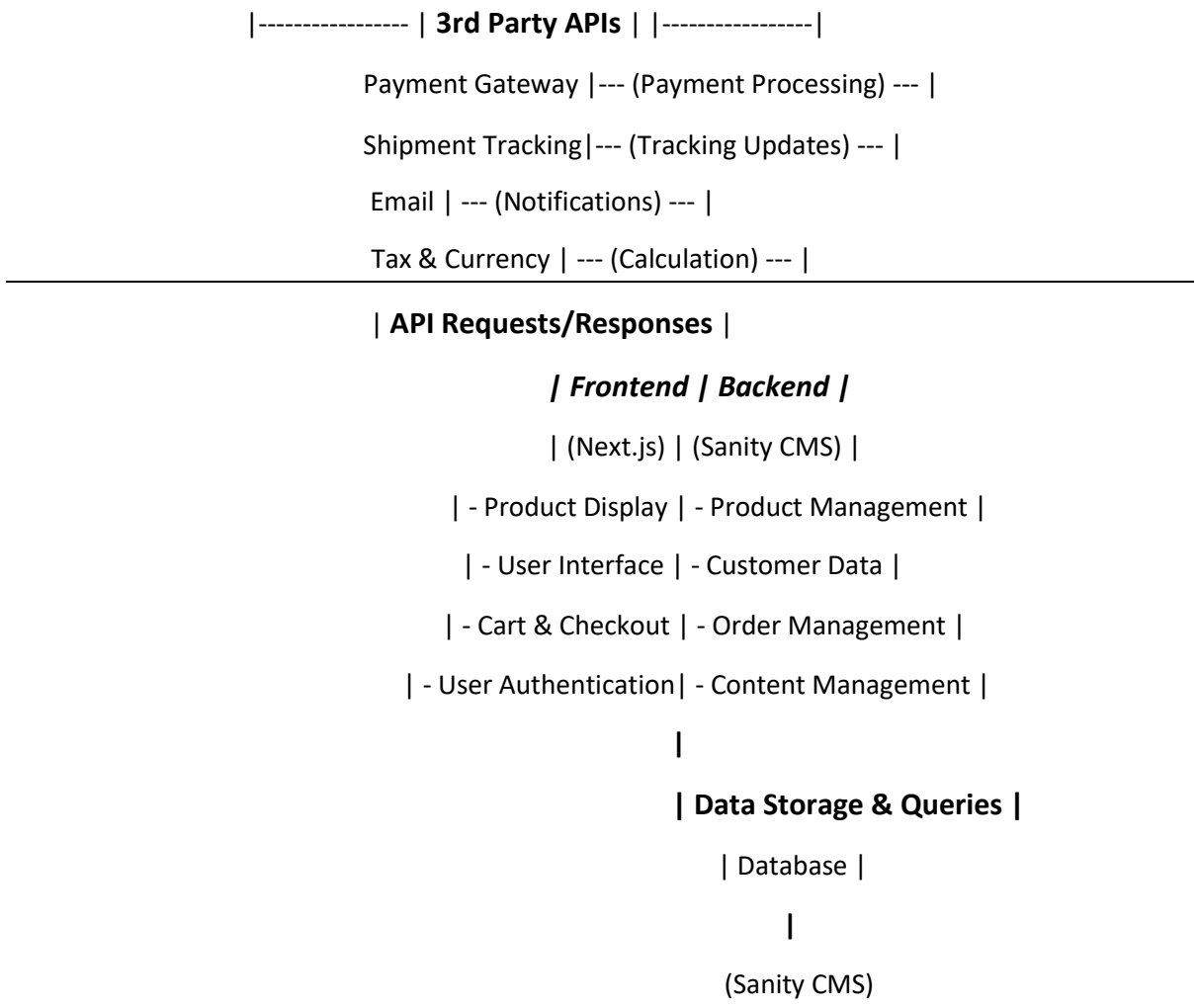
- **Authentication & Authorization:** Implement user authentication (JWT or OAuth).
- **Data Protection:** Secure user data with encryption and best practices.

- **Optimized Performance:** Use caching, lazy loading, and CDN for faster content delivery.

5. Deployment & Hosting

- **Hosting:** Use **Vercel** or **Netlify** for frontend hosting.
- **Sanity CMS:** Cloud-based and scalable backend.
- **Database:** Sanity CMS acts as a NoSQL database.
- **CI/CD Pipeline:** Automate deployment and updates

2. Design System Architecture



| Deployment |

Hosting (Vercel/Netlify)

3. Detailed Workflows

3.1 User Registration:

1. User fills out registration form on the Frontend (Next.js).
2. Frontend sends a request to the Sanity CMS API to create a new user document.
3. Sanity CMS stores the user data in the database.
4. Sanity CMS (or a separate service triggered by Sanity) sends a confirmation email to the user via the Email API.
5. Frontend displays a success message to the user.

3.2 Product Browsing:

1. User navigates to a product category page on the Frontend.
2. Frontend sends a GROQ query to the Sanity CMS API to fetch product data for that category.
3. Sanity CMS retrieves the product data from the database.
4. Sanity CMS returns the product data to the Frontend.
5. Frontend displays the products on the page.

3.3 Adding to Cart:

1. User clicks "Add to Cart" on a Product Details Page.
2. Frontend updates the user's cart data (typically stored in local storage or a cookie).
3. (Optional) Frontend can send an update to Sanity CMS to persist the cart data for logged-in users.

3.4 Order Placement:

1. User proceeds to checkout on the Frontend.
2. Frontend collects order details (shipping address, payment information, etc.).
3. Frontend sends the order details to the Sanity CMS API to create a new order document.
4. Sanity CMS stores the order data in the database.
5. Frontend sends payment information to the Payment Gateway API.
6. Payment Gateway processes the payment.
7. Payment Gateway returns a success/failure response to the Frontend.
8. Frontend updates the order status in Sanity CMS based on the payment response.
9. Sanity CMS (or a separate service) sends an order confirmation email to the user via the Email API.
10. Frontend displays an order confirmation message to the user.

3.5 Shipment Tracking:

1. User views their order details on the Frontend.
2. Frontend retrieves the shipment tracking number from the order data in Sanity CMS.
3. Frontend sends a request to the Shipment Tracking API to get the latest tracking information.
4. Shipment Tracking API returns tracking updates to the Frontend.
5. Frontend displays the tracking updates to the user.

3.6 Inventory Updates:

1. When an order is placed, Sanity CMS triggers a function (e.g., a webhook or serverless function) or uses a listener.
2. The function updates the stock levels of the purchased products in the Sanity CMS database.

4. Security Considerations (Expanded)

- **Authentication:** Implement JWT (JSON Web Tokens) or OAuth for user authentication.
- **Authorization:** Use role-based access control to restrict access to sensitive data and functionalities.
- **Data Validation:** Validate all user inputs on the Frontend and Backend to prevent injection attacks.
- **HTTPS:** Use HTTPS for all communication between the client and server.
- **Regular Security Audits:** Conduct regular security audits and penetration testing to identify and address vulnerabilities.

5. Performance Optimization (Expanded)

- **Caching:** Implement caching at various levels (CDN, browser, server) to reduce latency.
- **Image Optimization:** Optimize images for web performance (size, format).
- **Code Minification:** Minify CSS and JavaScript files to reduce file sizes.
- **Lazy Loading:** Implement lazy loading for images and other resources to improve initial page load time.
- **CDN:** Use a Content Delivery Network (CDN) to distribute static assets globally.

3. Plan API Requirements


```
{
 "apiEndpoints": [
  {
   "name": "Product Data API",
   "endpoint": "/products",
   "method": "GET",
```

```
  "description": "Fetch all available products from Sanity CMS",
  "requestPayload": "N/A",
  "response": {
   "type": "array",
   "description": "Array of product objects",
   "example": [
    {
     "id": 1,
     "name": "Product A",
     "price": 100,
     "stock": 10,
     "image": "url_to_image.jpg",
     // ... other product details
    },
    // ... more products
   ],
   "errorExample": {
    "status": 500,
    "message": "Internal Server Error"
   },
   "pagination": {
    "totalItems": 100,
    "totalPages": 10,
    "currentPage": 1,
    "pageSize": 10
   }
  }
 },
 {
  "name": "Order API",
  "endpoint": "/orders",
  "method": "POST",
  "description": "Create a new order",
  "requestPayload": {
   "type": "object",
   "description": "Order details",
   "example": {
    "customerInfo": {
     "name": "John Doe",
     "email": "john.doe@example.com",
     // ... other customer info
    },
    "productDetails": [
     {
```

```
          "productId": 1,
          "quantity": 2
        },
        // ... other products
      ],
      "paymentStatus": "pending",
      // ... other order details
    }
  },
  "response": {
    "type": "object",
    "description": "Order confirmation or error message",
    "example": {
      "orderId": "ORDER123",
      "message": "Order created successfully"
    },
    "errorExample": {
      "status": 400,
      "message": "Invalid request data"
    }
  }
},
{
  "name": "Shipment API",
  "endpoint": "/shipment",
  "method": "GET",
  "description": "Track order status via a third-party API",
  "requestPayload": "N/A",
  "response": {
    "type": "object",
    "description": "Shipment information",
    "example": {
      "shipmentId": "SHIP456",
      "orderId": "ORDER123",
      "status": "Shipped",
      "expectedDeliveryDate": "2024-11-15"
    },
    "errorExample": {
      "status": 404,
      "message": "Shipment not found"
    }
  }
},
{
```

```
    "name": "Rental Management API",
    "endpoint": "/rental-duration",  // Corrected endpoint name
    "method": "POST",
    "description": "Add rental details for a specific product",
    "requestPayload": {
     "type": "object",
     "description": "Rental details",
     "example": {
      "productId": 456,
      "duration": "7 days",
      "deposit": 500
     }
    },
    "response": {
     "type": "object",
     "description": "Confirmation or validation error",
     "example": {
      "message": "Rental details added successfully"
     },
     "ErrorExample": {
      "status": 400,
      "message": "Invalid rental duration"
     }
    }
   }
  ]
}
```