Technical Document Of ‘E - Commerce’ Market Place

- In this document we will define the technical aspects required for a E-Commerce martekplace

1. Frontend Requirments :

- **Technologies used :** Next.js , React.js , Tailwand Css , Shadcn Ui , Aos Animations
- **User-Friendly Interface :** The E-Commerce website will feature a clean and interactive UI. making it easy for user to browser the products and view product detials easily
- **Responsive Design :** The website will be designed to provide a seamless experience across mobile, tablets and desktop devices. This will ensure that users can browse and shop comfortably from any device.
- **Essential Pages :**

1. **Home Page :** The landing page showcasing featrued products , Categories , Deals , and a quick access to the main areas of site
2. **Product Listing Page :** A page that displays all available products with filtering options e.g by category , price
3. **Product Details Page :** A detailed page for each product, with descriptions, images, pricing, reviews, and the option to add the item to the cart.

5. **Cart Page :** A page that lists all products added to the shopping cart, with the option to modify quantities like to Add , and remove the items

6. **Checkout Page :** A page where consumer provides its delivery address which furthure used as a billing address , Payment method , shipping fee , subtotal and a completed order button

7. **Order Confirmation Page :** A page that confirms the purchase and show the order details with the option to to cancel , edit and re-order if user cancel and a continue shopping button

2. **Backend Requirments :**

- **Sanity (CMS) :** Sanity CMS will act as the backend for managing all the dynamic content of our marketplace, such as :

1. **Product Data :** We will create a schema for product data and stored a detailed Product info i.e , name , id , price , sizes , cateogry , dimensions , colors

2. **Consumer Data :** A schema for Consumer Profile , Address , contact Info etc

3. **Order Records :** A schema where we save the details fo consumer orders , i.e consumer details , product details and shipping details

Brief Summary Of Backend :

1. In the backend, we manage the product data. All products throughout the website will have their data fetched dynamically based on their category.

2. The buying process begins when a user adds a product to the cart. The cart will display a subtotal of the product prices, their quantities, and provide add and remove buttons to adjust the product quantity.

3. After finalizing the cart, the user proceeds to the checkout page, where they provide the required information such as their address and contact details. On the same checkout page, a summary will show the subtotal, shipping fee, and the total amount. Payment options like Cash on Delivery (COD) and card payments will also be available.

4. Once the user clicks "Complete Order", all the information, including the products, user details, and total price, is saved into the Sanity schema named Order Record. This schema stores the details of the completed order. Additionally, the system fetches this order data to generate an order summary or receipt for the user after they finalize their purchase.

3. Third-Party API’s :

For our E-commerce market place we use third party API’s i.e ShipEngine for (Shipping tracking) & for payment we use (Strip)

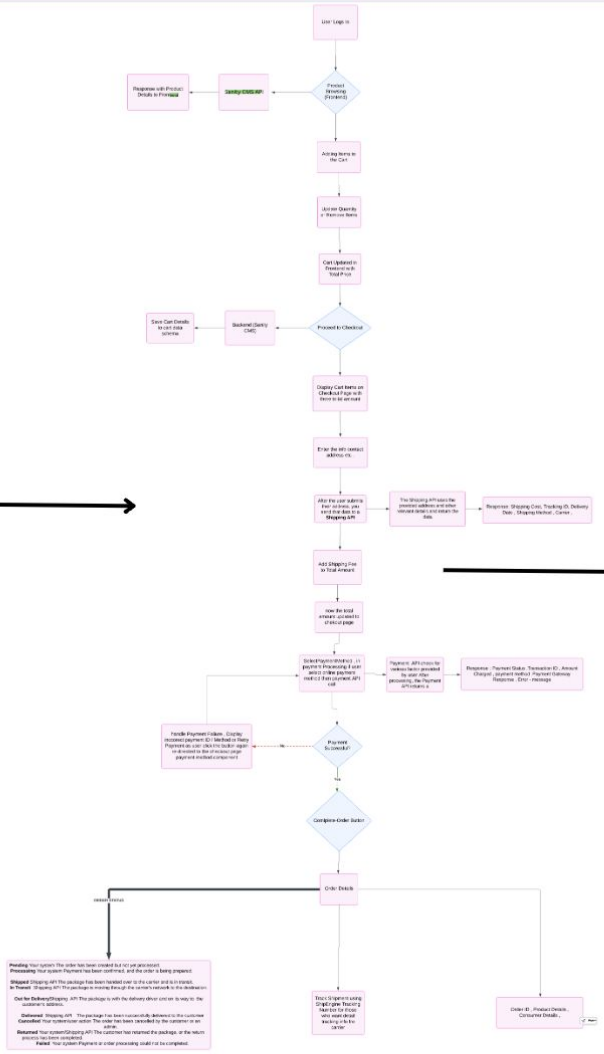
- **Shipping tracking API :**

Provide real-time shipment tracking details of the order that dispatch

- **Payment Gateway API :**

Handle secure payments during checkout (credit card, debit card, PayPal).

System Architecture


User Logs in
Product Browsing (Frontend)
Sanity CMS API
Response with Product Details to Frontend
Adding Items to the Cart
Update Quantity or Remove Items
Cart Updated in Frontend with Total Price
Proceed to Checkout
Backend (Sanity CMS)
Save Cart Details to cart data schema
Display Cart Items on Checkout Page with there total amount
Enter the info contact address etc.
After the user submits their address, you send that data to a Shipping API
The Shipping API uses the provided address and other relevant details and return the data.
Response: Shipping Cost, Tracking ID, Delivery Date , Shipping Method , Carrier ,
Add Shipping Fee to Total Amount
now the total amount updated to checkout page
SelectPaymentMethod , in payment Processing if user select online payment method then payment API call
Payment API check for various factor provided by user After processing , the Payment API returns a
Response : Payment Status , Transaction ID , Amount Charged , payment method , Payment Gateway Response , Error - message
Payment Successful?
No
Handle Payment Failure , Display incorrect payment ID / Method or Retry Payment as user click the button again re-directed to the checkout page payment method component
Yes
Complete-Order Button
Order Details
order status
Pending Your system The order has been created but not yet processed.
Processing Your system Payment has been confirmed, and the order is being prepared.
Shipped Shipping API The package has been handed over to the carrier and is in transit.
In Transit Shipping API The package is moving through the carrier's network to the destination.
Out for Delivery Shipping API The package is with the delivery driver and on its way to the customer's address.
Delivered Shipping API The package has been successfully delivered to the customer
Cancelled Your system/user action The order has been cancelled by the customer or an admin.
Returned Your system/Shipping API The customer has returned the package, or the return process has been completed.
Failed Your system Payment or order processing could not be completed.
Track Shipment using ShipEngine Tracking Number for those who want detail tracking info the carrier
Order ID , Product Details , Consumer Details ,

Description of the work flow on next page

Description of the work flow

1. **User Login :**

- The user logs into their account on the website.
- After login, they can proceed to browse products.

2. **Product Browsing (Frontend) :**

- The frontend (Next.js) sends a request to the Sanity CMS API to fetch the product data.
- The user can browse different product categories and view product details.
- Sanity CMS responds with the product details to display on the front end.

3. **Adding Items to the Cart :**

- The user selects items to add to their cart.
- They have the option to update the quantity of items or remove items from the cart.
- The cart is updated in the frontend, showing the total price of the items.

4. **Proceed to Checkout :**

- When the user clicks the checkout button, the cart data is sent to the backend (Sanity CMS) to save the details (products, quantities, etc.).
- The checkout page displays the cart items with prices and allows the user to enter shipping information (address, contact details).

5. **Shipping Calculation :**

- Based on the user’s shipping address, the shipping fee is calculated via the ShipEngine API.
- The response from ShipEngine includes the shipping cost, tracking ID, and estimated delivery date.
- The shipping fee is added to the total amount, and the updated total is displayed to the user on the checkout page.

6. **Payment Processing :**

- After reviewing the order, the user selects a payment method (e.g., Credit Card, PayPal).
- The payment details are sent to the Payment Gateway API (e.g., Stripe or PayPal) for processing.
- The payment status is returned (successful or failed) along with the transaction ID.

7. **Order Confirmation :**

- Once the payment is successful, the order confirmation page is shown.

The page displays :

- Order Details (Product , total Price etc , order-status) , consumer Info (Address , contact info etc) , Shipping Details (carrier, tracking number, estimated delivery date).
- The user can also track the shipment using the tracking number provided by ShipEngine.

API Requirments and Work Flow :

1. PRODUCTS :

- **Work Flow :**

1. Fetch Products from External API

- Call the external product API (GET /external-products).
- Map the data to match Sanity's schema.
- Push the products into Sanity

2. Sanity as a Backend for Product Details

- Use the slug field in Sanity to fetch product details for the frontend.

- **Endpoints :**

1. Fetch External Products

- Endpoint: /external-products
- Method: GET
- Description: Fetch products from an external service
- Response Example:

[{ "id": 1, "name": "Product A", "price": 100, "category": "Electronics" }]

Array of mutiple objects of products

Further we will send this data to the sanity and as we using sanity as an backend then we fetch the product data from the sanity plus we fetch the data based on the slug field provided in the sanity for each product details page

2. **Payment**

1. Initiate Payment (POST Request)

- Payment is processed through an external Payment Gateway API (e.g., Stripe, PayPal, Razorpay).
- You send order details and payment information provided by the user to the payment gateway API Once the payment gateway processes the payment, it immediately returns a response to your system, indicating whether the transaction was successful or failed. This happens within a few seconds.
- Method: POST
- Endpoint Name: /payment

- **Request Payload Example:**
- So here we send the payment details to payment gateway API for processing the payment once the payment process it immediately returns a response indicating whether the transaction was successful or failed

```
{
"orderId": "12345",
"amount": 120.00,
"currency": "USD",
"paymentMethod": "CreditCard",
"cardDetails": {
"cardNumber": "4111111111111111",
"expiry": "12/25",
"cvv": "123"
}
}
```

If Payment Successfull

```
{
"paymentId": "pay-54321",
"orderId": "12345",
"PaymentStatus": "Success",
"amount": 120.00,
"currency": "USD",
"transactionTime": "2025-01-
20T12:00:00Z"
}
```

If Payment Rejected

```
{
"paymentId": "pay-54321",
"orderId": "12345",
"PaymentStatus": "Failed",
"error": "Insufficient
funds", "transactionTime":
"2025-01-20T12:00:00Z" }
```

- Next Steps Based on Response: If the payment is successful, you can proceed to the next step (e.g., initiate shipment). If the payment fails, you might notify the user and ask them to retry or choose another payment method.

2. In Case of (COD)

- No payment POST request is made upfront.
- The shipment POST request is made first with paymentStatus: "COD" or "Pending", meaning payment is collected later.
- Once the shipment is delivered and the customer pays, you can update the payment method

After the payment work don then Shipment part comes in to Play

3. **Shipment :**

1. **Create Shipment (POST Request)**

- You send the order details, shipping information, and item details via a POST request to /shipment to create a shipment.
- Endpoint Name: /shipment
- Method: POST
- Description: This endpoint is used to send shipment details for processing. It initiates the shipment process and returns an ID and tracking number.
- **Request Payload Example:**

```
{
"orderId": "12345",
"shippingAddress": {
"name": "John Doe",
"address": "123 Main St, City, Country",
"zipCode": "12345", (Optional)
"phone": "+1234567890"
},
"items": [
{
"productId": "6789",
"name": "Product A",
"quantity": 1,
"price": 100
}
],
"shippingMethod": "Express",
"paymentStatus": "Paid for online | COD for offline"
}
```

- Here we send the shipment details for processing so as it initiates the shipment and returns a immediate response

```
{
"shipmentId": "shp-98765",
"orderId": "12345",
"status": "Pending | In Progess",
"carrier": "FedEx",
"shipmentCost": 12.50,
"labelUrl": null
}
```

Now from the respoonse we use this shipmentId , or any related data to make GET Request to fetch the real-time status and details of the shipment

2. **Create Shipment (GET Request) :**

- After shipment initiation, you can fetch the real-time status of the shipment by sending a GET request to /shipment/status/{shipmentId}, where {shipmentId} is the unique identifier returned from the POST request.

- Endpoint Name: /shipment/status/{shipmentId}   i.e   GET /shipment/status/shp-98765
- Method: GET
- Description: This endpoint allows you to fetch the current status of a shipment using the unique shipmentId. It provides updates such as the trackingID , current status, location, and estimated delivery time.

```
{
"shipmentId": "shp-98765",
"orderId": "12345",
"status": "In Transit",
"trackingNumber":"123ABC456",
"estimatedDeliveryTime": "2025-01-25T18:00:00",
"currentLocation": "City A"
}
```

4. **Order Confirmation page**

- As the user click complete order button we will send the whole order details data to the Sanity Scheme named Order to generate the order later on we fetch this data from sanity to show on our confirm order page
- Endpoint Name: /Order
- Method: POST
- **Example Data Format to Send to Sanity:**

```json
{
"orderId": "12345",
"customer": {
"name": "John Doe",
"email": "john.doe@example.com",
"shippingAddress": {
"address": "123 Main St, City, Country",
"zipCode": "12345",
"phone": "+1234567890"
}
},
"products": [
{
"productId": "6789",
"name": "Product A",
"quantity": 1,
"price": 100.00
}
],
"payment": {
"paymentMethod": "CreditCard",
"status": "Paid",
"amount": 120.00,
"currency": "USD",
"transactionTime": "2025-01-20T12:00:00"
},
"shipment": {
"shipmentId": "shp-98765",
"status": "In Progress",
"trackingNumber": "123ABC456",
"estimatedDeliveryTime": "2025-01-25T18:00:00"
},
"orderStatus": "Completed",
"orderDate": "2025-01-20T12:30:00"
}
```

- later on we fetch this data to show on our Order Confirmation page

SCHEMAS

Product Schema Example :

```
 export default {
 name: 'product',
  title: 'Product',
type: 'document',
fields: [

{ name: 'productId', title: 'Product ID', type: 'string' },
{ name: 'name', title: 'Product Name', type: 'string' },
{ name: 'price', title: 'Price', type: 'number' },
{ name: 'tags', title: 'Tags', type: 'array', of: [{ type: 'string' }] },
{ name: 'sizes', title: 'Sizes', type: 'array', of: [{ type: 'string' }] },
{ name: 'color', title: 'Color Options', type: 'array', of: [{ type: 'string' }] },
{ name: 'dimension', title: 'Dimensions', type: 'object',
  fields: [
  { name: 'length', title: 'Length', type: 'number' },
  { name: 'width', title: 'Width', type: 'number' },
  { name: 'height', title: 'Height', type: 'number' },
  { name: 'unit', title: 'Unit', type: 'string', options: { list: ['inches', 'cm', 'mm'] } }
  ]
 },

{ name: 'stock', title: 'Stock Quantity', type: 'number' },
},
```

2. Consumer Schema Example :

```
  export default {
 name: 'consumer',
  title: 'Consumer',
  type: 'document',
 fields: [

 { name: 'consumerId', title: 'Consumer ID', type: 'string' },
 { name: 'fullName', title: 'Full Name', type: 'string' },
 { name: 'email', title: 'Email Address', type: 'string' },
 { name: 'phone', title: 'Phone Number', type: 'string' },
 { name: 'street', title: 'Street', type: 'string' },
 { name: 'city', title: 'City', type: 'string' },
 { name: 'state', title: 'State', type: 'string' },
 { name: 'country', title: 'Country', type: 'string' },
 { name: 'zipCode', title: 'Zip Code', type: 'string' },
 { name: 'paymentMethod', title: 'Payment Method', type: 'string' },
{ name: 'orderHistory', title: 'Order History', type: 'array', of: [{ type: 'reference', to: [{ type: 'order' }] }] }

 ]

}
```

2. Order Schema Example :

```
  export default {
    name: 'order',
     title: 'Order',
 type: 'document',
fields: [
{ name: 'orderId', title: 'Order ID', type: 'string' },
{ name: 'orderStatus', title: 'Order Status', type: 'string' },
{ name: 'consumerDetails', title: 'Consumer Details', type: 'reference', to: [{ type: 'consumer' }] },
{ name: 'productDetails', title: 'Product Details', type: 'array', of: [{ type: 'reference', to: [{ type: 'product' }] }] },
},
```