Day:02
Prepared By Arisha

Fashion E-Commerce System Architecture & Implementation

1. Sanity CMS Integration

Objective

Use Sanity CMS to manage products and orders dynamically.

Steps to Set Up Sanity CMS

1. Install Sanity CLI

Run the following commands to install and initialize Sanity CMS:

```
npm install -g @sanity/cli
sanity init
```

2. Define Product Schema

Create a new file, product.js, and define the schema:

```
export default {
  name: "product",
  title: "Product",
  type: "document",
  fields: [
    { name: "title", type: "string", title: "Title" },
    { name: "price", type: "number", title: "Price" },
    { name: "image", type: "image", title: "Image" },
    { name: "description", type: "text", title: "Description" },
    { name: "category", type: "string", title: "Category" },
    { name: "size", type: "array", of: [{ type: "string" }], title: "Sizes" }
  ]
};
```

3. Define Order Schema

Create another schema file, order.js, to manage orders:

```
export default {
  name: 'order',
  title: 'Order',
  type: 'document',
  fields: [
    { name: 'orderNumber', type: 'string', title: 'Order Number' },
    { name: 'customer', type: 'object', title: 'Customer Info', fields: [
      { name: 'firstName', type: 'string', title: 'First Name' },
      { name: 'lastName', type: 'string', title: 'Last Name' },
      { name: 'email', type: 'string', title: 'Email' },
      { name: 'phone', type: 'string', title: 'Phone' }
    ]},
    { name: 'products', type: 'array', of: [{ type: 'reference', to: [{ type: 'product' }] }], title: 'Products'
},
    { name: 'totalAmount', type: 'number', title: 'Total Amount' },
    { name: 'orderStatus', type: 'string', title: 'Order Status' },
    { name: 'createdAt', type: 'datetime', title: 'Order Date' }
  ]
};
```

4. Add Schemas to schema.js

Ensure that both schemas are included in schema.js:

```
import product from './product';
import order from './order';

export default createSchema({
  name: 'default',
  types: schemaTypes.concat([product, order])
});
```

5. Querying Orders in the Frontend

Fetch order details from Sanity using GROQ:

```
const query = '*[_type == "order"]';
client.fetch(query).then(data => console.log(data));
```

---

2. Deployment & Testing

1. Deploy the Frontend

Run your frontend project locally or deploy it to Vercel or Netlify to test product data fetching.

2. Handling Pagination & Optimization

If you have many products, optimize with pagination using GROQ:

const query = '*[_type == "product"] | order(name asc) [0...10]';

This fetches the first 10 products in alphabetical order.

---

3. Payment API Integration

Objective

Integrate third-party payment gateways like Stripe, Razorpay, and PayPal for secure transactions.

Steps to Integrate Payment Gateway

1. Set Up API Credentials – Obtain API keys from Stripe/Razorpay/PayPal.

2. Implement Payment Processing – Use SDKs to handle payments securely.

3. Verify Transactions – Ensure order confirmation only after successful payment.

---

4. Shipment API Integration

Objective

Use a third-party API (e.g., Shippo, AfterShip) for real-time shipment tracking.

1. Modify Order Schema for Tracking

Update order.js to include shipment details:

```
export default {
  name: 'order',
  title: 'Order',
  type: 'document',
  fields: [
    { name: 'trackingNumber', type: 'string', title: 'Tracking Number' },
    { name: 'carrier', type: 'string', title: 'Carrier', options: { list: [
      { title: 'FedEx', value: 'fedex' },
      { title: 'UPS', value: 'ups' },
      { title: 'DHL', value: 'dhl' }
    ]}},
    { name: 'estimatedDeliveryDate', type: 'datetime', title: 'Estimated Delivery Date' },
    { name: 'trackingURL', type: 'url', title: 'Tracking URL' }
  ]
};
```

2. Fetching Shipment Data via GROQ

Retrieve tracking details in the frontend:

```
const query = '*[_type == "order"]{orderNumber, trackingNumber, carrier, trackingURL,
estimatedDeliveryDate}';
client.fetch(query).then(data => console.log(data));
```

3. Displaying Tracking Information in the UI

```
const FetchOrders = () => {
  const [orders, setOrders] = useState([]);

  useEffect(() => {
```

```
    const query = '*[_type == "order"]{orderNumber, trackingNumber, carrier, trackingURL,
estimatedDeliveryDate}';
    client.fetch(query).then(data => setOrders(data));
  }, []);

  return (
    <div>
      {orders.map(order => (
        <div key={order._id}>
          <h3>Order: {order.orderNumber}</h3>
          <p>Tracking Number: {order.trackingNumber}</p>
          <p>Carrier: {order.carrier}</p>
          <p>Estimated Delivery: {new
Date(order.estimatedDeliveryDate).toLocaleDateString()}</p>
          <p><a href={order.trackingURL} target="_blank" rel="noopener noreferrer">Track your
order</a></p>
        </div>
      ))}
    </div>
  );
};
```

---

5. Advanced Shipment Tracking (Optional)

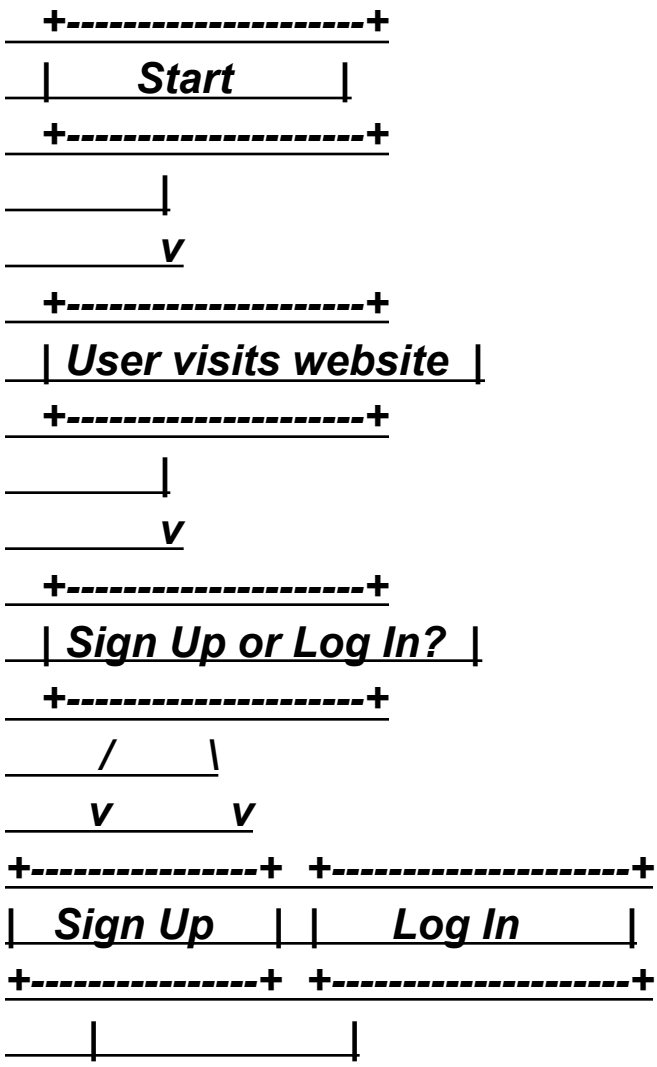
For real-time tracking updates, store tracking history:

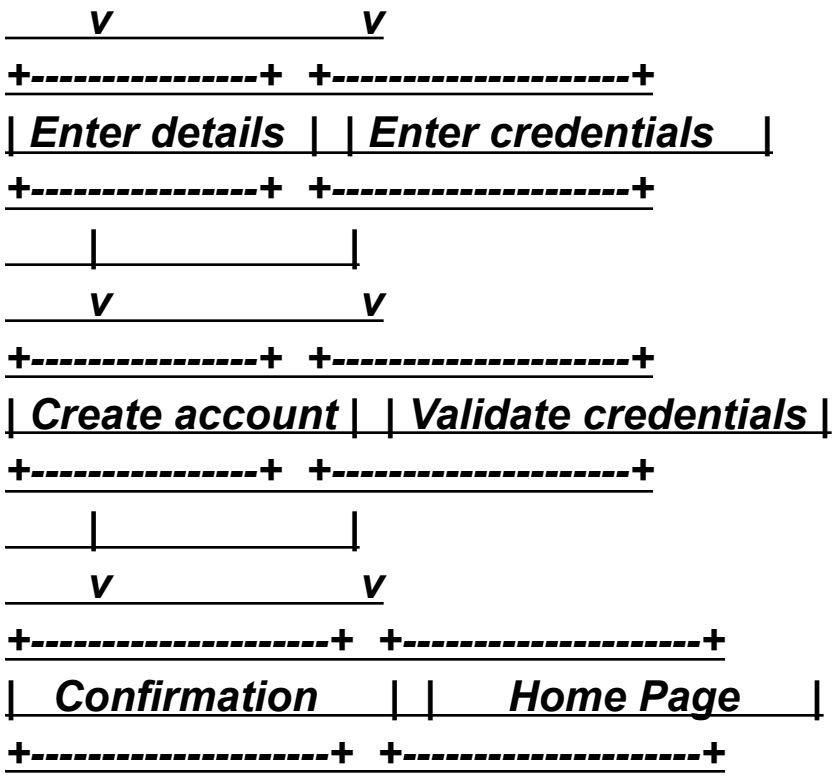
```
{
  name: 'trackingHistory',
  title: 'Tracking History',
  type: 'array',
  of: [{ type: 'object', fields: [
    { name: 'status', type: 'string' },
    { name: 'date', type: 'datetime' },
    { name: 'location', type: 'string' }
  ]}]
}
```

---

6. Final System Architecture

```
User --> [Frontend (Next.js)]
     |
     v
 [Sanity CMS] --> [Product API]
     |
     v
 [Cart API] --> [Payment Gateway (Stripe)]
     |
     v
 [Order API] --> [Shipment API (AfterShip)]
```




Enter details
Enter credentials
Create account
Validate credentials
Confirmation
Home Page