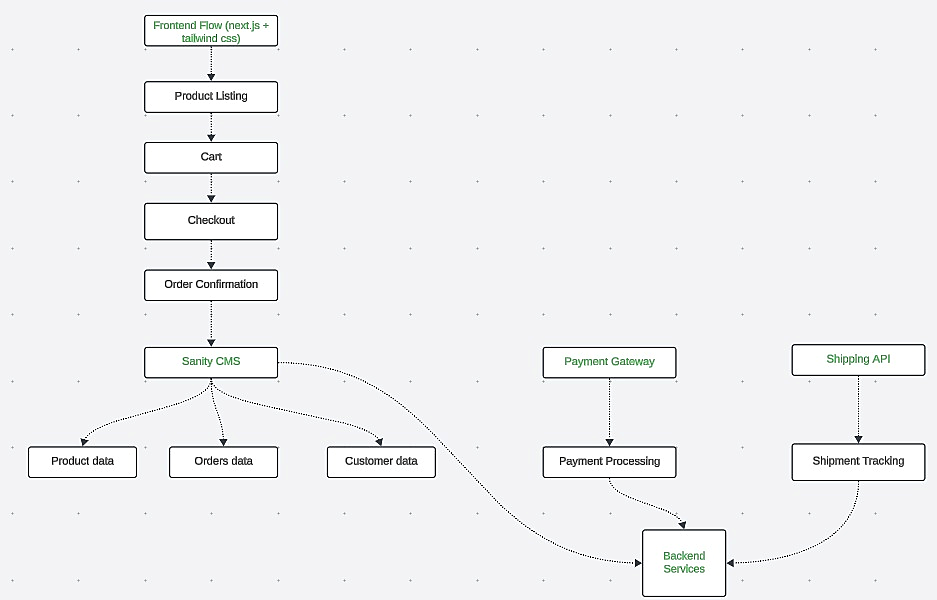
**Marketplace Technical Foundation - ShopVista**

**1) System architecture Overview:**



**Component Roles**

This system architecture represents a typical e-commerce workflow integrating a frontend, content management system (CMS), backend services, and third-party APIs. Here's a brief description of each component and how they interact:

**1. Frontend Flow (Next.js + Tailwind CSS)**

* Displays product listings, handles cart management, and facilitates the checkout process.

**2. Product Listing**

* Role: Displays all available products fetched from the CMS or backend and pulls product data from the Sanity CMS through APIs.

**3. Cart**

* Allows customers to select and manage items they want to purchase and tracks selected products and calculates totals. May communicate with backend services for inventory validation.

**4. Checkout**

* Collects customer information (address, payment details) and confirms the purchase and passes payment information to the Payment Gateway and customer/order details to the backend.

**5. Order Confirmation**

* Confirms successful payment and order placement and updates the backend and CMS with order and customer data.

**6. Sanity CMS**

* Manages and stores product data, orders, and customer data.
  + Supplies product data to the frontend.
  + Stores order and customer data after purchase, fetched from the backend or frontend.

**7. Backend Services**

* Acts as the core processing unit.
  + Processes order information.
  + Communicates with external APIs for payment processing and shipment tracking.
  + Manages inventory and updates the CMS.

**8. Payment Gateway**

* Facilitates secure payment processing.
  + Receives payment data from the frontend during checkout.
  + Confirms successful or failed payments with backend services.

**9. Shipping API**

* Handles shipment tracking and logistics.
  + Receives order data from backend services.
  + Provides tracking details, which can be sent back to the frontend for customer updates.

**2. Key Workflows:**

1.The frontend (built with Next.js and styled with Tailwind CSS) loads and displays the product listing by fetching product data from the **Sanity CMS**.

2.Product details (e.g., name, price, description, stock) are fetched from the CMS and displayed dynamically.

**3.**The frontend updates the cart state locally or optionally sends a request to the **Backend Services** to validate product availability (inventory check).

4.The cart page displays the selected items, quantities, and total price, using locally stored cart data or fetching details from the **Backend Services**.

5.The frontend collects customer details (e.g., name, email, address) and sends the order summary to the **Backend Services**.

6.The **Backend Services** validates the order (e.g., stock, pricing) and generates a unique order ID.

**7.**Payment details are sent from the frontend to the **Payment Gateway**, which processes the transaction and confirms success or failure to the backend.

**8.**The **Backend Services** updates the **Sanity CMS** with the order, customer details, and payment status.

9.The frontend displays the order confirmation page to the user with the order ID and details.

10.The **Backend Services** sends order details to the **Shipping API**, which generates a tracking number and initiates shipment.

11.The **Shipping API** provides shipment updates to the **Backend Services**, which are stored in the **Sanity CMS**.

12.The frontend fetches shipment status from the CMS and displays tracking details to the user.

**3. Category specific instructions:**

Our e-commerce platform is built with a robust architecture that ensures seamless functionality and scalability. The **frontend**, developed using Next.js and styled with Tailwind CSS, provides a responsive and user-friendly interface where customers can browse products, add them to the cart, and proceed to checkout. Product data, including descriptions, pricing, and stock levels, is dynamically fetched from **Sanity CMS**, which also stores orders and customer information.

During checkout, the platform integrates with **Backend Services** to validate inventory, generate order IDs, and handle critical operations like updating the CMS and coordinating with third-party APIs. Payments are securely processed through a **Payment Gateway**, ensuring a smooth transaction experience for customers. Once an order is confirmed, the **Shipping API** is used to generate tracking numbers and manage real-time shipment updates. The platform also keeps customers informed through status updates displayed on the frontend, fetched from the CMS, ensuring a streamlined and engaging shopping experience.

**4. API Endpoints:**A table of information

Description automatically generated with medium confidence

**5. Sanity Schema Example:**

// Product Schema

{

name: 'product',

fields: [

{name: 'productId', type: 'string'},

{name: 'name', type: 'string'},

{name: 'price', type: 'decimal'},

{name: 'stock', type: 'integer'},

{name: 'categoryId', type: 'string'},

{name: 'Tags', type: 'array'},

{name: 'orderId', type: 'reference'}

]

}

// Order Schema

{

name: 'order',

fields: [

{name: 'orderId', type: 'string'},

{name: 'customerId', type: 'reference'},

{name: 'shipmentId', type: 'reference'},

{name: 'status', type: 'string'},

{name: 'timestamp', type: 'date'}

]

}

// Shipment Schema

{

name: 'shipment',

fields: [

{name: 'shipmentId', type: 'string'},

{name: 'orderId', type: 'reference'},

{name: 'zoneId', type: 'string'},

{name: 'status', type: 'string'},

{name: 'deliveryDate', type: 'date'}

]

}

// Customer Schema

{

name: 'customer',

fields: [

{name: 'customerId', type: 'string'},

{name: 'fullName', type: 'string'},

{name: 'ContactInfo', type: 'object'},

{name: 'address', type: 'object'},

{name: 'orderHistory', type: 'array'}

]

}

// Inventory Schema

{

name: 'inventory',

fields: [

{name: 'inventoryId', type: 'string'},

{name: 'productId', type: 'reference'},

{name: 'quantity', type: 'integer'},

{name: 'warehouseLocation', type: 'string'}

]

}

// Categories Schema

{

name: 'categories',

fields: [

{name: 'categoryId', type: 'string'},

{name: 'name', type: 'string'},

{name: 'description', type: 'string'}

]

}

// Seller Schema

{

name: 'sellers',

fields: [

{name: 'sellerId', type: 'string'},

{name: 'zoneId', type: 'reference'},

{name: 'companyName', type: 'string'},

{name: 'phone', type: 'string'},

{name: 'returnPolicy', type: 'object'}

]

}

// DeliveryZones Schema

{

name: 'deliveryZones',

fields: [

{name: 'zoneId', type: 'string'},

{name: 'coverageArea', type: 'array'},

{name: 'assignedDrivers', type: 'array'}

]

}

### Core Entity Relationships

- Products -> Categories (Many-to-One)

- Orders -> Customers (Many-to-One)

- Orders -> Products (Many-to-Many)

- Shipments -> Orders (One-to-One)

- Products -> Inventory (One-to-One)

- Sellers -> DeliveryZones (Many-to-Many)

5. Technical Requirements

### Frontend Requirements

1. Responsive Design

- Mobile-first approach

- Breakpoints: 320px, 768px, 1024px, 1440px

- Progressive web app capabilities

2. Core Pages

- Home page with product listings

- Product details page

- Shopping cart

- Checkout process

- User account management

- Order tracking

### CMS Integration

1. Product Management

- CRUD operations

- Media handling

- Category management

- Inventory tracking

2. Order Processing

- Order status management

- Payment processing

- Shipment tracking

- Return management

### Third-Party Integrations

1. Payment Gateway (Stripe)

- Secure payment processing

- Multiple payment methods

- Refund handling

2. Shipping Integration

- Real-time tracking

- Multiple carrier support

- Delivery zone management

## 6. Security Measures

1. Authentication

- JWT-based authentication

- Role-based access control

- Session management

2. Data Protection

- HTTPS encryption

- Data encryption at rest

- Input validation

- XSS protection

## 7. Performance Optimization

1. Frontend Optimization

- Code splitting

- Lazy loading

- Image optimization

- Caching strategies

2. Backend Optimization

- Database indexing

- Query optimization

- Cache implementation

- Rate limiting