Name: Usra Naz

Roll No: 00144665
Day/Time: Friday- 09:00AM -12:00PM

Marketplace Technical Foundation - [Clothing E-Commerce]

Technical Requirements for Clothing eCommerce Website

This document outlines the technical requirements for developing a clothing eCommerce marketplace. It includes detailed descriptions of the frontend, backend, and integration with third-party services to ensure a seamless and scalable platform.

1. Frontend Requirements

The frontend will serve as the user-facing part of the marketplace, designed to provide an intuitive and engaging shopping experience.

1.1 User-Friendly Interface

- The design will be clean and simple, making it easy for users to navigate and find products.
- Products will be categorized clearly into sections such as Men’s Clothing, Women’s Clothing, and Kids’ Clothing.

1.2 Responsive Design

- The website will adapt seamlessly to different screen sizes, ensuring compatibility with both mobile and desktop devices.

1.3 Essential Pages

The website will consist of the following core pages:

1. **Home Page**:
    - **Purpose**: The first point of interaction for users; highlights the website's features and offerings.
    - **Key Sections**:
        - **Hero Banner**: Display featured collections, discounts, or promotions.
        - **New Arrivals**: Showcase the latest products.
        - **Bestsellers**: Highlight popular items.
        - **Categories**: Quick links to Men's, Women's, Kids' clothing.
        - **Footer**: Links to policies, contact information, and social media.
2. **Product Listing Page**:
    - **Purpose**: Display all products in a specific category or based on user filters.
    - **Key Features**:
        - **Product Grid/List**: Show products with images, names, and prices.
        - **Filters**: Allow users to filter by size, color, price range, and brand.
        - **Sort Options**: Sort products by price, popularity, or newest arrivals.
        - **Pagination or Infinite Scrolling**: Load more products as users scroll.
3. **Product Details Page**:
    - **Purpose**: Provide detailed information about a selected product.
    - **Key Features**:
        - **Product Images**: Zoom-in functionality for clear views.

        - **Description**: Detailed information about the product (fabric, care instructions, etc.).
        - **Price**: Show the current price and any discounts.
        - **Size and Color Options**: Allow users to select size and color.
        - **Stock Availability**: Indicate whether the item is in stock.
        - **Add to Cart Button**: Add the item to the cart with selected options.
        - **Reviews and Ratings**: Show user reviews and ratings for the product.

4. **Cart Page**:
    - **Purpose**: Summarize items the user has added to their cart for purchase.
    - **Key Features**:
        - **Product List**: Show products with selected size, color, price, and quantity.
        - **Total Cost**: Display the subtotal, taxes, and final cost.
        - **Edit Options**: Allow users to update quantities or remove items.
        - **Proceed to Checkout Button**: Directs users to the checkout page.
5. **Checkout Page**:
    - **Purpose**: Collect user details for order processing.
    - **Key Sections**:
        - **Shipping Details**: Collect user name, address, phone number, and email.
        - **Payment Information**: Display payment options (credit card, PayPal, etc.).
        - **Order Summary**: Summarize items being purchased.
        - **Place Order Button**: Complete the transaction and send order details to the backend.
6. **Order Confirmation Page**:
    - **Purpose**: Confirm the successful placement of an order.
    - **Key Features**:
        - **Order Summary**: List purchased items with quantities and price.
        - **Tracking ID**: Display shipment tracking information (if available).
        - **Thank You Message**: Acknowledge the customer for their purchase.
        - **Next Steps**: Links to track the order or continue shopping.
7. **User Registration Page**:
    - **Purpose**: Allow new users to create an account.
    - **Key Features**:
        - **Input Fields**: Name, email, password, and phone number.
        - **Sign Up Button**: Save user details in the backend.
8. **Login Page**:
    - **Purpose**: Enable existing users to log in.
    - **Key Features**:
        - **Input Fields**: Email and password.

        - **Forgot Password Link**: Help users recover access to their account.

9. **User Profile Page**:

    - **Purpose**: Allow users to view and manage their personal details.
    - **Key Sections**:
        - **Personal Information**: Name, email, phone number, and address.
        - **Order History**: List of previous orders with statuses.
        - **Wishlist**: Show saved products for later purchase.

10. **Wishlist Page**:

    - **Purpose**: Display products that users have saved for future consideration.
    - **Key Features**:
        - **Product List**: Show saved items with prices and stock status.
        - **Move to Cart Option**: Allow users to add wishlist items to their cart.

11. **Search Results Page**:

    - **Purpose**: Display products matching the user's search query.
    - **Key Features**:
        - **Search Bar**: Input field for user queries.
        - **Product Grid/List**: Show results matching the search terms.

12. **About Us Page**:

    - **Purpose**: Share information about your brand and story.
    - **Key Features**:
        - **Mission and Vision**: Explain your goals and values.
        - **Brand Story**: Describe how your business started.
        - **Contact Information**: Provide email, phone, and address details.

13. **Contact Us Page**:

    - **Purpose**: Allow users to reach out with inquiries or issues.
    - **Key Features**:
        - **Contact Form**: Fields for name, email, subject, and message.
        - **Business Details**: Email, phone number, and physical address.
        - **Social Media Links**: Direct users to your social media platforms.

14. **FAQ Page**:

    - **Purpose**: Address common customer queries.
    - **Key Sections**:
        - **Shipping**: Delivery timelines and fees.
        - **Returns**: Process for returning items.
        - **Payments**: Accepted payment methods and refund policies.

15. **Privacy Policy and Terms of Service Page**:

    - **Purpose**: Inform users about your policies.

    - **Key Sections**:
        - **Data Collection**: Explain how user data is collected and used.
        - **Terms of Use**: Guidelines for using the website.

Summary of Pages

- Home Page
- Product Listing Page
- Product Details Page
- Cart Page
- Checkout Page
- Order Confirmation Page
- User Registration Page
- Login Page
- User Profile Page
- Wishlist Page
- Search Results Page
- About Us Page
- Contact Us Page
- FAQ Page
- Privacy Policy and Terms of Service Page

2. Sanity CMS as Backend

Sanity CMS will act as the backbone for managing all dynamic content and data related to the website.

2.1 Product Data Management

- Products will be managed with fields such as:
    - Name, Description, Price
    - Available Sizes and Colors
    - Stock Levels
    - Product Images

2.2 Customer Data Management

- Maintain user profiles, including order history and personal details.

2.3 Order Records

- Store information about all orders placed, such as:
    - Product details (name, quantity, price).
    - Customer details (name, address, contact information).
    - Payment status and shipping details.

3. Third-Party APIs

To enhance functionality and automate backend tasks, the platform will integrate with the following APIs:

3.1 Shipment Tracking API

- Provides real-time updates on shipment status.
- Example APIs: **Shippo**, **EasyPost**.
- **Use Case**:
  - Display live shipment tracking information on the Order Confirmation Page.

3.2 Payment Gateway API

- Securely handle payment processing.
- Example APIs: **Stripe**, **PayPal**, **Razorpay**.
- **Use Case**:
  - Process transactions during checkout and return a confirmation status.

3.3 Optional APIs

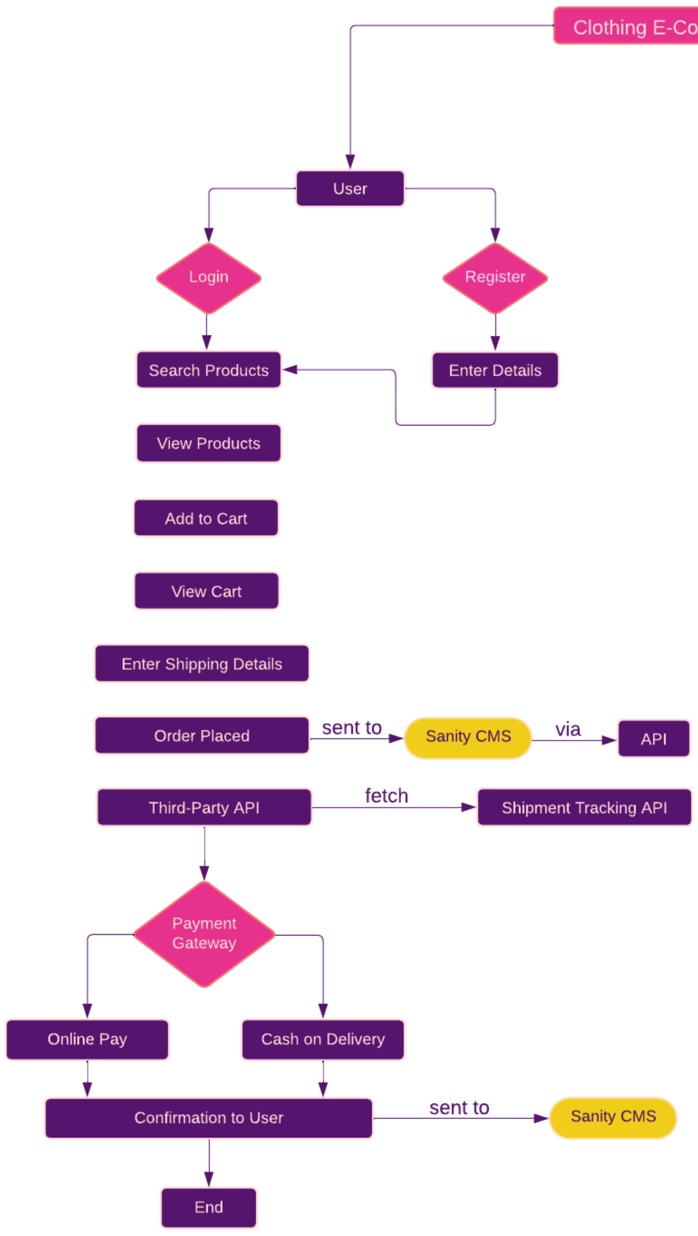
- **SMS/Email Notifications**: Notify customers about order confirmations, shipping updates, etc.
  - Example APIs: **Twilio**, **SendGrid**.

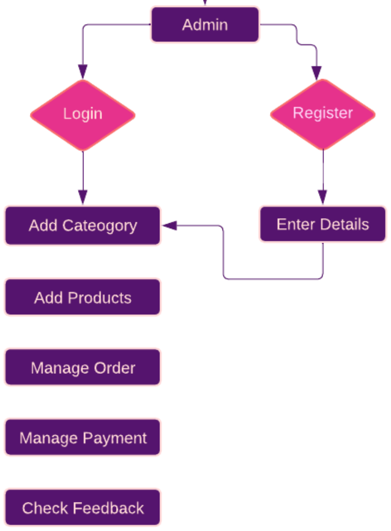
How These Requirements Align with Business Goals

1. **User-Friendly Design**:
    - A clean interface and responsive design attract and retain customers, ensuring a pleasant shopping experience.
2. **Sanity CMS**:
    - Centralized data management simplifies inventory updates and ensures scalability as the business grows.
3. **Third-Party APIs**:
    - Automates essential tasks like payment processing and shipment tracking, improving operational efficiency and customer satisfaction.

This technical plan forms the foundation for a robust eCommerce platform that meets customer needs while supporting business scalability.

Key Workflow





System Architecture Overview

Frontend (Next.js) Interaction with Sanity CMS

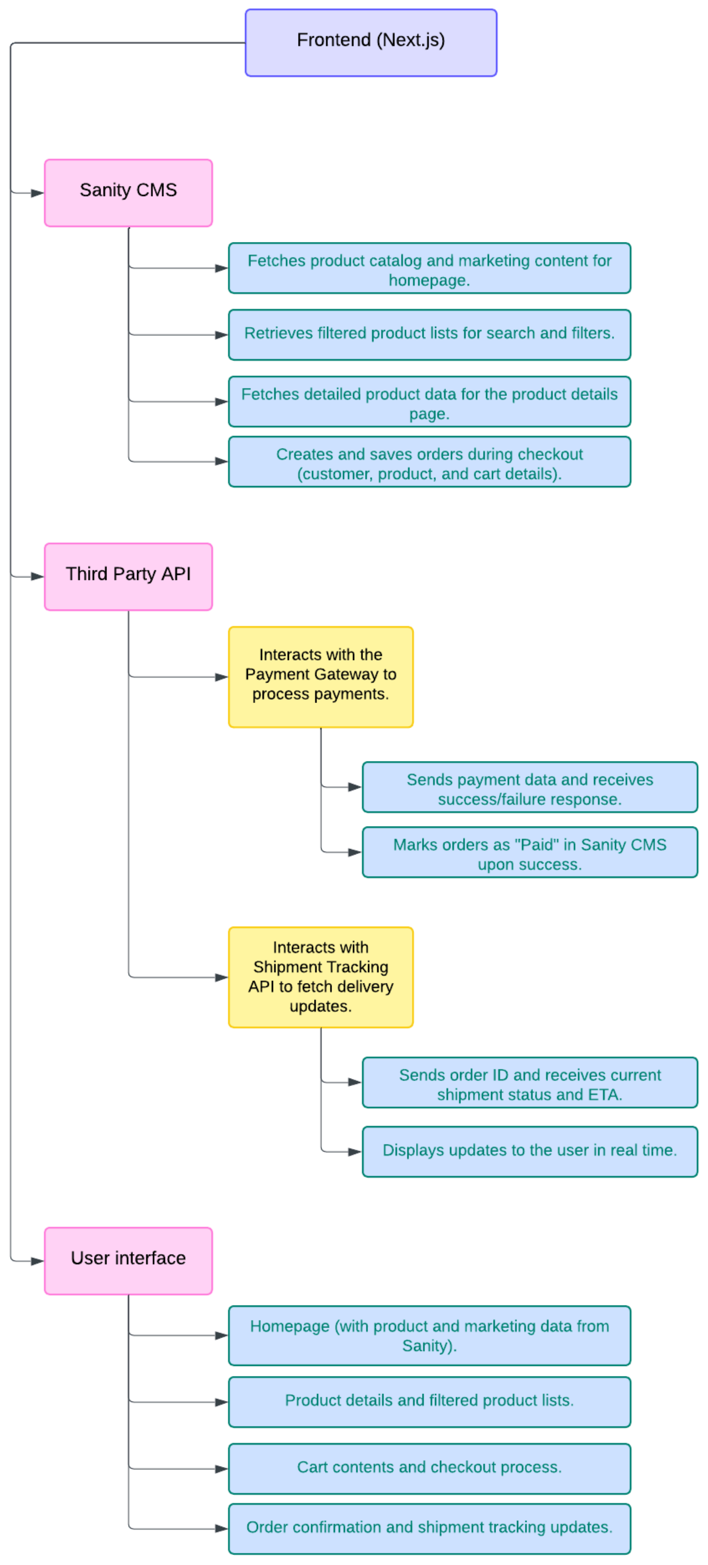
1. **Home Page/Product List Loading:**
    - User visits the website.
    - **Frontend Request:** Sends a GET request to Sanity CMS to fetch:
        - Product catalog (ID, name, image, price, stock status).
        - Promotions, banners, and featured products.
    - **Sanity CMS Response:** Returns structured data (JSON) for products and homepage content.
    - **Frontend Action:** Dynamically renders the homepage with product information and marketing content.
2. **Search/Filter Products:**
    - User interacts with search or filter options.
    - **Frontend Request:** Sends a query to Sanity CMS, including:
        - Keywords (search term).
        - Selected filter criteria (e.g., category, price range).
    - **Sanity CMS Response:** Returns a refined list of products matching the criteria.
    - **Frontend Action:** Updates the UI to display filtered products.
3. **Product Details Page:**
    - User clicks on a product to view details.
    - **Frontend Request:** Sends a GET request to Sanity CMS with the product ID.
    - **Sanity CMS Response:** Returns detailed product information:
        - Name, description, price, stock, reviews, and images.
    - **Frontend Action:** Renders the product details page with all the fetched information.
4. **Cart Management:**
    - User adds/removes products from the cart.
    - **Frontend Action:** Updates the cart state locally (e.g., in Redux or localStorage).
    - Optionally syncs the cart state with Sanity CMS for logged-in users.
5. **Checkout Process:**
    - User initiates checkout.
    - **Frontend Action:**
        - Collects user information (e.g., name, email, shipping address).
        - Gathers cart details (product IDs, quantities, total price).

    - **Frontend Request:** Sends a POST request to Sanity CMS to create a new order.
    - **Sanity CMS Response:**
        - Saves order data.
        - Returns an order confirmation ID.

Frontend Interaction with Third-Party APIs

1. **Payment Gateway:**
    - During checkout, user provides payment details.
    - **Frontend Request:**
        - Sends payment data (amount, card details, user info) to the third-party payment gateway.
    - **Payment Gateway Response:**
        - Returns payment status (success or failure).
        - If successful:
            - Returns a transaction ID and confirmation receipt.
            - **Frontend Action:** Updates the UI with a success message.
            - **Sanity CMS Request:** Marks the order as "Paid."
        - If failed:
            - Returns an error message.
            - **Frontend Action:** Prompts the user to retry payment.
2. **Shipment Tracking:**
    - After order placement, shipment status becomes important.
    - **Frontend Request:** Sends the order ID to the third-party shipment tracking API.
    - **Shipment Tracking API Response:**
        - Provides real-time shipment updates:
            - Current status (e.g., "Dispatched," "In Transit").
            - Expected delivery date.
            - Tracking URL for further details.
        - **Frontend Action:** Displays shipment updates on the user dashboard or order history page.
3. **Real-Time Updates:**
    - The frontend periodically polls the third-party APIs for updates or uses WebSockets for real-time communication (if supported by APIs).
    - **Frontend Action:** Reflects real-time changes in:
        - Payment status (e.g., awaiting payment confirmation).
        - Shipment tracking (e.g., updated ETA).

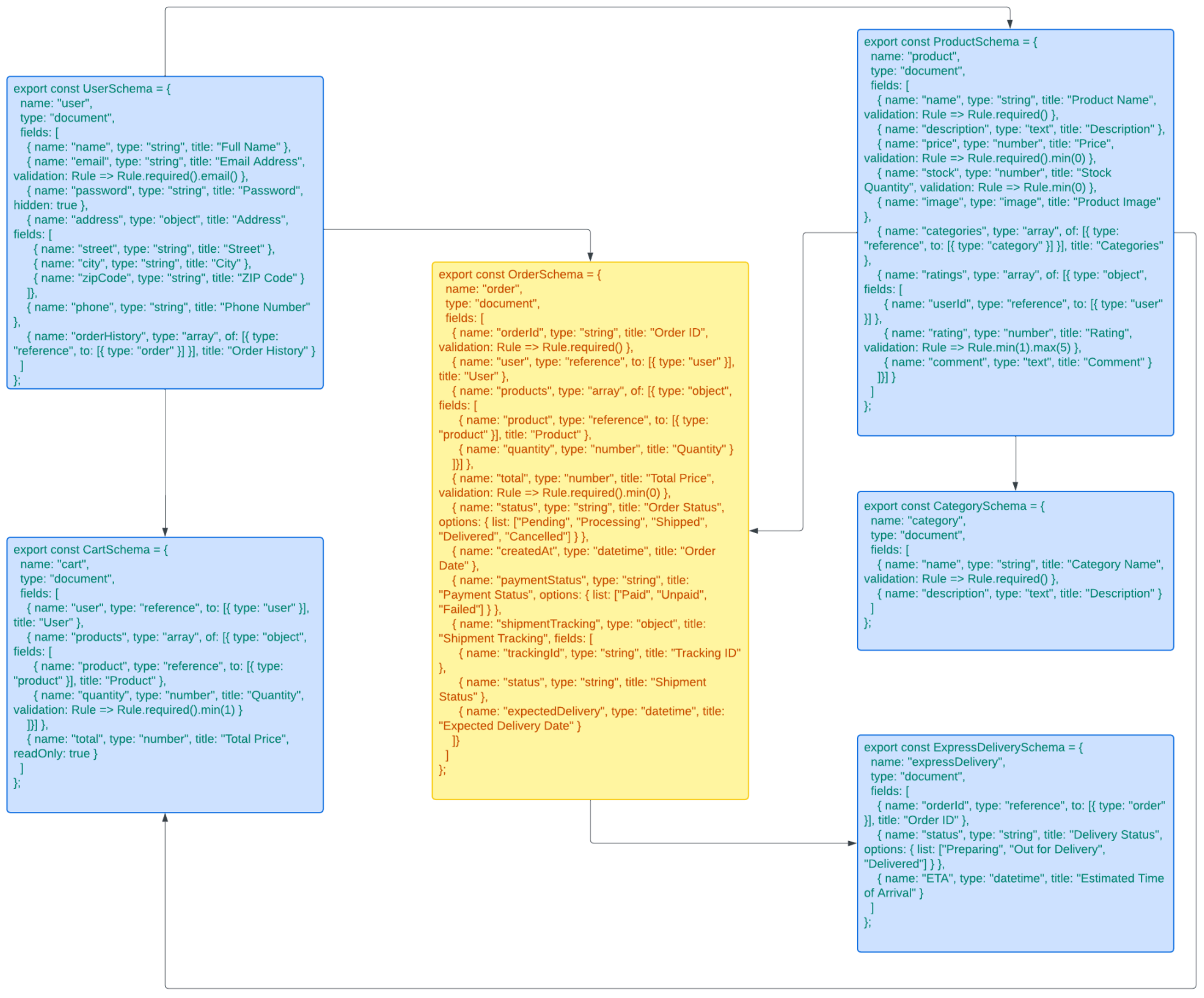
System Architecture Diagram


Frontend (Next.js)
Sanity CMS
Fetches product catalog and marketing content for homepage.
Retrieves filtered product lists for search and filters.
Fetches detailed product data for the product details page.
Creates and saves orders during checkout (customer, product, and cart details).
Third Party API
Interacts with the Payment Gateway to process payments.
Sends payment data and receives success/failure response.
Marks orders as "Paid" in Sanity CMS upon success.
Interacts with Shipment Tracking API to fetch delivery updates.
Sends order ID and receives current shipment status and ETA.
Displays updates to the user in real time.
User interface
Homepage (with product and marketing data from Sanity).
Product details and filtered product lists.
Cart contents and checkout process.
Order confirmation and shipment tracking updates.

API Endpoint

Endpoint	Method	Purpose	Payload	Response Example
/products	GET	Fetch all product details for display	None	[{"id": 1, "name": "T-shirt", "price": 500, "stock": 10, "image": "url"}]
products/{id}	GET	Fetch details of a specific product	None	{"id": 1, "name": "T-shirt", "price": 500, "description": "Cotton T-shirt", "stock": 10}
/products/search	GET	Search products with keywords	{ "query": "shirt", "filters": {"price": "low-to-high"} }	[{"id": 1, "name": "T-shirt", "price": 500}]
/cart	POST	Add a product to the cart	{ "userId": 123, "productId": 1, "quantity": 2 }	{"cartId": 456, "status": "Product added to cart"}
/cart	GET	Get all products currently in the cart	{ "userId": 123 }	{ "cartId": 456, "products": [{"id": 1, "name": "T-shirt", "quantity": 2}]}
/cart/{id}	DELETE	Remove a product from the cart	{ "userId": 123, "productId": 1 }	{"status": "Product removed from cart"}
/checkout	POST	Place an order	{ "userId": 123, "cartId": 456, "paymentMethod": "Card" }	{"orderId": 789, "status": "Order placed", "paymentStatus": "Successful"}
/order/{id}	GET	Fetch details of a specific order	{ "userId": 123 }	{"orderId": 789, "status": "Processing", "items": [{"id": 1, "name": "T-shirt", "quantity": 2}]}
/shipment/{id}	GET	Track the shipment status of an order	None	{"shipmentId": 101, "orderId": 789, "status": "In Transit", "ETA": "2 days"}
/categories	GET	Fetch all product categories	None	{"categories": ["Men", "Women", "Kids"]}
/categories/{id}	GET	Fetch products belonging to a specific category	None	{"category": "Men", "products": [{"id": 1, "name": "T-shirt", "price": 500}]}
/user/register	POST	Register a new user	{ "name": "John Doe", "email": "john@example.com", "password": "password123" }	{"userId": 123, "status": "Registration successful"}
/user/login	POST	Authenticate an existing user and provide a token	{ "email": "john@example.com", "password": "password123" }	{"userId": 123, "token": "jwt-token", "status": "Login successful"}
/express-delivery-status	GET	Fetch real-time delivery updates for express orders	{ "orderId": 123 }	{"orderId": 123, "status": "Out for Delivery", "ETA": "15 minutes"}
/payment/process	POST	Process payment for an order	{ "orderId": 789, "amount": 2000, "method": "Card" }	{"paymentId": 202, "status": "Payment Successful"}
/profile	GET	Fetch user profile details	{ "userId": 123 }	{ "userId": 123, "name": "John Doe", "email": "john@example.com", "orderHistory": [...] }
/profile/update	PUT	Update user profile details	{ "userId": 123, "name": "John Updated" }	{"status": "Profile updated successfully"}

Schema



```
export const UserSchema = {
  name: "user",
  type: "document",
  fields: [
    { name: "name", type: "string", title: "Full Name" },
    { name: "email", type: "string", title: "Email Address",
validation: Rule => Rule.required().email() },
    { name: "password", type: "string", title: "Password",
hidden: true },
    { name: "address", type: "object", title: "Address",
fields: [
      { name: "street", type: "string", title: "Street" },
      { name: "city", type: "string", title: "City" },
      { name: "zipCode", type: "string", title: "ZIP Code" }
    ]},
    { name: "phone", type: "string", title: "Phone Number"
},
    { name: "orderHistory", type: "array", of: [{ type:
"reference", to: [{ type: "order" }] }], title: "Order History" }
  ]
};
```

```
export const CartSchema = {
  name: "cart",
  type: "document",
  fields: [
    { name: "user", type: "reference", to: [{ type: "user" }],
title: "User" },
    { name: "products", type: "array", of: [{ type: "object",
fields: [
      { name: "product", type: "reference", to: [{ type:
"product" }], title: "Product" },
      { name: "quantity", type: "number", title: "Quantity",
validation: Rule => Rule.required().min(1) }
    ]}] },
    { name: "total", type: "number", title: "Total Price",
readOnly: true }
  ]
};
```

```
export const OrderSchema = {
  name: "order",
  type: "document",
  fields: [
    { name: "orderId", type: "string", title: "Order ID",
validation: Rule => Rule.required() },
    { name: "user", type: "reference", to: [{ type: "user" }],
title: "User" },
    { name: "products", type: "array", of: [{ type: "object",
fields: [
      { name: "product", type: "reference", to: [{ type:
"product" }], title: "Product" },
      { name: "quantity", type: "number", title: "Quantity" }
    ]}] },
    { name: "total", type: "number", title: "Total Price",
validation: Rule => Rule.required().min(0) },
    { name: "status", type: "string", title: "Order Status",
options: { list: ["Pending", "Processing", "Shipped",
"Delivered", "Cancelled"] } },
    { name: "createdAt", type: "datetime", title: "Order
Date" },
    { name: "paymentStatus", type: "string", title:
"Payment Status", options: { list: ["Paid", "Unpaid",
"Failed"] } },
    { name: "shipmentTracking", type: "object", title:
"Shipment Tracking", fields: [
      { name: "trackingId", type: "string", title: "Tracking ID"
},
      { name: "status", type: "string", title: "Shipment
Status" },
      { name: "expectedDelivery", type: "datetime", title:
"Expected Delivery Date" }
    ]}
  ]
};
```

```
export const ProductSchema = {
  name: "product",
  type: "document",
  fields: [
    { name: "name", type: "string", title: "Product Name",
validation: Rule => Rule.required() },
    { name: "description", type: "text", title: "Description" },
    { name: "price", type: "number", title: "Price",
validation: Rule => Rule.required().min(0) },
    { name: "stock", type: "number", title: "Stock
Quantity", validation: Rule => Rule.min(0) },
    { name: "image", type: "image", title: "Product Image"
},
    { name: "categories", type: "array", of: [{ type:
"reference", to: [{ type: "category" }] }], title: "Categories"
},
    { name: "ratings", type: "array", of: [{ type: "object",
fields: [
      { name: "userId", type: "reference", to: [{ type: "user"
}] },
      { name: "rating", type: "number", title: "Rating",
validation: Rule => Rule.min(1).max(5) },
      { name: "comment", type: "text", title: "Comment" }
    ]}] }
  ]
};
```

```
export const CategorySchema = {
  name: "category",
  type: "document",
  fields: [
    { name: "name", type: "string", title: "Category Name",
validation: Rule => Rule.required() },
    { name: "description", type: "text", title: "Description" }
  ]
};
```

```
export const ExpressDeliverySchema = {
  name: "expressDelivery",
  type: "document",
  fields: [
    { name: "orderId", type: "reference", to: [{ type: "order"
}], title: "Order ID" },
    { name: "status", type: "string", title: "Delivery Status",
options: { list: ["Preparing", "Out for Delivery",
"Delivered"] } },
    { name: "ETA", type: "datetime", title: "Estimated Time
of Arrival" }
  ]
};
```