**User Side Features and Functions:**

User Authentication:

1. User Registration:
   * Allow users to create new accounts by providing necessary information like name, email, and password.
   * Validate user inputs, including email format and password strength.
   * Store user credentials securely in the database (passwords should be hashed).
2. User Login:
   * Provide a login form where users can enter their registered email and password.
   * Authenticate user credentials against stored data in the database.
   * Use sessions or tokens to maintain user authentication across multiple requests.
3. Password Management:
   * Implement features for users to reset their passwords in case of forgotten passwords.
   * Send password reset links or temporary passwords securely via email.
   * Use secure methods for password storage (e.g., hashing with salt) to protect user data.
4. Authentication Middleware:
   * Use middleware to authenticate user requests to protected resources (e.g., user profile, checkout process).
   * Redirect unauthenticated users to the login page or display appropriate error messages.
5. Secure Authentication Practices:
   * Implement measures like rate limiting, CAPTCHA, and account lockout to prevent brute-force attacks.
   * Use HTTPS (SSL/TLS) to encrypt data transmission, especially during authentication.

Account Management:

1. User Profile Management:
   * Allow users to view and update their profile information (e.g., name, address, contact details).
   * Implement forms for users to change passwords or update account settings.
2. Account Verification:
   * Optionally implement email verification for new user accounts to ensure valid email addresses.
   * Send verification emails with unique confirmation links upon registration.
3. Account Deactivation and Closure:
   * Provide functionality for users to deactivate or close their accounts if needed.
   * Handle account deletion securely, ensuring proper data handling and GDPR compliance if applicable.
4. Session Management:
   * Manage user sessions securely to maintain authentication state during a user's visit.
   * Implement session expiration policies to enhance security (e.g., automatic logout after a period of inactivity).
5. Access Control:
   * Define user roles and permissions to control access to specific features or sections of the application.
   * Restrict admin-level functionalities to authorized personnel only.

Register a New User

* Endpoint: POST /auth/register
* Description: Register a new user with the provided details.
* Request Body:
* jsonCopy code
* {
* "name": "John Doe",
* "email": "john.doe@example.com",
* "password": "securepassword"
* }
* Responses:
  + 201 Created: User successfully registered.
  + 400 Bad Request: Invalid request body.
  + 409 Conflict: Email address already exists.

Login User

* Endpoint: POST /auth/login
* Description: Authenticate user and generate an access token.
* Request Body:
* jsonCopy code
* {
* "email": "john.doe@example.com",
* "password": "securepassword"
* }
* Responses:
  + 200 OK:
  + jsonCopy code
  + {
  + "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c"
  + }
  + 401 Unauthorized: Invalid credentials.

Logout User

* Endpoint: POST /auth/logout
* Description: Invalidate the user's access token.
* Request Headers:
* jsonCopy code
* {
* "Authorization": "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c"
* }
* Responses:
  + 200 OK: Successfully logged out.
  + 401 Unauthorized: Invalid or missing access token.

Account Management Endpoints

Get User Profile

* Endpoint: GET /users/me
* Description: Retrieve the authenticated user's profile information.
* Request Headers:
* jsonCopy code
* {
* "Authorization": "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c"
* }
* Responses:
  + 200 OK:
  + jsonCopy code
  + {
  + "id": "123",
  + "name": "John Doe",
  + "email": "john.doe@example.com",
  + "createdAt": "2024-04-18T12:00:00Z"
  + }
  + 401 Unauthorized: Invalid or missing access token.

Update User Profile

* Endpoint: PUT /users/me
* Description: Update the authenticated user's profile information.
* Request Headers:
* jsonCopy code
* {
* "Authorization": "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c"
* }
* Request Body:
* jsonCopy code
* {
* "name": "John Doe",
* "email": "john.doe@example.com"
* }
* Responses:
  + 200 OK: User profile updated successfully.
  + 400 Bad Request: Invalid request body.
  + 401 Unauthorized: Invalid or missing access token.

Change Password

* + Endpoint: PUT /api/users/me/password
  + Description: Allows the authenticated user to change their password by providing the current and new password.

Forgot Password (Reset Password)

* + Endpoint: POST /api/auth/forgot-password
  + Description: Initiates the process to reset a user's password. Sends an email with a password reset link to the user's registered email address.

Reset Password

* + Endpoint: POST /api/auth/reset-password
  + Description: Validates the password reset token and allows the user to set a new password.
* Delete Account
  + Endpoint: DELETE /api/users/me
  + Description: Allows the authenticated user to delete their account permanently.

Error Responses

Common error responses returned by the API:

* 400 Bad Request: Invalid request parameters or missing required fields.
* 401 Unauthorized: Authentication credentials are missing or invalid.
* 403 Forbidden: The authenticated user does not have permission to perform the requested action.
* 404 Not Found: Resource not found (e.g., user profile does not exist).
* 409 Conflict: Resource conflict (e.g., email address already exists).

**Product Browsing and Searching:**

### Product Browsing Functions

1. Product Listing:
   * Displaying a list of products on category pages, featured products sections, and search result pages.
   * Include essential details for each product such as name, image, price, and rating.
   * Implement pagination to manage large product catalogs and improve user experience.
2. Product Categories and Filters:
   * Categorize products into logical categories (e.g., electronics, clothing, books).
   * Provide filter options to allow users to narrow down product listings based on various attributes such as price range, brand, size, color, and availability.
   * Enable multiple selection for certain filters (e.g., selecting multiple categories or brands simultaneously).
3. Sorting Options:
   * Allow users to sort products based on different criteria such as price (low to high or high to low), popularity, new arrivals, and relevance.
4. Product Details Page:
   * Provide a dedicated page for each product with comprehensive details including descriptions, specifications (e.g., brand, color, size, dimensions, weight), customer reviews, and related products.
   * Allow users to view high-resolution images of the product from different angles.
5. Wishlist and Favorites:
   * Enable users to add products to their wishlist or favorites list for future reference or purchase.
6. Recently Viewed Products:
   * Display a list of recently viewed products to help users quickly revisit items of interest.

### Product Searching Functions

1. Keyword Search:
   * Enable full-text search across product names, descriptions, categories, brands, and other relevant attributes.
   * Support advanced search features such as partial matches, synonyms, and spell correction for improved search accuracy.
2. Search Results Page:
   * Display search results in a clear and organized manner, highlighting matching products based on relevance.
   * Implement pagination and sorting options for search results to improve usability.
3. Search Filters and Refinements:
   * Allow users to refine search results further using filters and facets (e.g., category, price range, brand, color, size, ratings).
   * Provide visual cues to indicate active filters and allow users to remove or modify filters dynamically without reloading the page.

### **Product Browsing and Searching API**

#### **List Products**

* Endpoint: GET /api/products
* Description: Retrieve a list of products with optional filtering and pagination.
* Query Parameters:
  + category: Filter products by category (optional).
  + minPrice: Filter products with a minimum price (optional).
  + maxPrice: Filter products with a maximum price (optional).
  + sortBy: Sort products by a specified attribute (e.g., price, name).
  + sortOrder: Specify sort order (asc for ascending, desc for descending).
  + page: Page number for pagination (default: 1).
  + limit: Number of products per page (default: 10).
* Response:
  + 200 OK on success:
  + json
  + Copy code

{

"total": 100,

"products": [

{

"id": "1",

"name": "Product A",

"category": "Electronics",

"price": 499.99,

"description": "Description of Product A",

"imageUrl": "https://example.com/product-a.jpg"

},

{

"id": "2",

"name": "Product B",

"category": "Clothing",

"price": 29.99,

"description": "Description of Product B",

"imageUrl": "https://example.com/product-b.jpg"

},

...

]

}

* + 400 Bad Request if invalid query parameters are provided.

#### **Get Product Details**

* Endpoint: GET /api/products/{productId}
* Description: Retrieve detailed information about a specific product.
* Path Parameters:
  + productId: ID of the product to retrieve.
* Response:
  + 200 OK on success:
  + json
  + Copy code

{

"id": "1",

"name": "Product A",

"category": "Electronics",

"price": 499.99,

"description": "Description of Product A",

"imageUrl": "https://example.com/product-a.jpg",

"specifications": {

"brand": "Brand X",

"color": "Black",

"weight": "1.5 kg"

},

"reviews": [

{

"userId": "123",

"username": "JohnDoe",

"rating": 4,

"comment": "Great product!"

},

...

]

}

* + 404 Not Found if the specified product ID does not exist.

#### **Search Products**

* Endpoint: GET /api/products/search
* Description: Search products based on keywords and optional filters.
* Query Parameters:
  + q: Keywords to search for (required).
  + category: Filter products by category (optional).
  + minPrice: Filter products with a minimum price (optional).
  + maxPrice: Filter products with a maximum price (optional).
  + sortBy: Sort products by a specified attribute (e.g., price, name).
  + sortOrder: Specify sort order (asc for ascending, desc for descending).
  + page: Page number for pagination (default: 1).
  + limit: Number of products per page (default: 10).
* Response:
  + 200 OK on success:
  + json
  + Copy code

{

"total": 50,

"products": [

{

"id": "1",

"name": "Product A",

"category": "Electronics",

"price": 499.99,

"description": "Description of Product A",

"imageUrl": "https://example.com/product-a.jpg"

},

{

"id": "2",

"name": "Product B",

"category": "Clothing",

"price": 29.99,

"description": "Description of Product B",

"imageUrl": "https://example.com/product-b.jpg"

},

...

]

}

* + 400 Bad Request if invalid query parameters are provided.

### **Error Responses**

Common error responses returned by the API:

* 400 Bad Request: Invalid query parameters or missing required fields.
* 404 Not Found: Resource not found (e.g., product not found).
* 500 Internal Server Error: Unexpected server error.

### **Additional Functions and Considerations**

#### **Wishlist and Favorites**

* Endpoint: /api/wishlist/add (POST)
* Description: Add a product to the user's wishlist.
* Request Body:
* json
* Copy code

{

"productId": "1"

}

* Response:
  + 200 OK on success.
  + 404 Not Found if the product ID does not exist.

#### **Recently Viewed Products**

* Endpoint: /api/recently-viewed
* Description: Retrieve a list of recently viewed products for the authenticated user.
* Response Format: Array of product objects similar to product details response.

**Shopping Cart and Checkout:**

### **hopping Cart Functions**

1. Add Product to Cart:
   * Allow users to add products to their shopping cart.
   * Update the quantity of items in the cart, including options for size, color, or other variations.
2. View Cart Contents:
   * Display a summary of the items currently in the user's cart.
   * Show details such as product name, quantity, price, subtotal, and the option to remove items or update quantities.
3. Update Cart:
   * Enable users to update the quantity of items or remove items from their cart.
   * Implement real-time updates to reflect changes in the cart subtotal and total price.
4. Save Cart for Later:
   * Provide an option to save selected items in the cart for future purchase or reference (e.g., wishlist functionality).

### **Checkout Functions**

1. Proceed to Checkout:
   * Initiate the checkout process by moving items from the cart to the checkout page.
   * Display a summary of the selected items, shipping address, and payment details.
2. Guest Checkout vs. Registered User Checkout:
   * Allow both guest checkout (without requiring account creation) and registered user checkout for a seamless experience.
3. Shipping Address and Payment Information:
   * Collect user's shipping address, contact information, and preferred payment method during checkout.
   * Provide validation for shipping address fields (e.g., zip code, state/province).
4. Order Summary and Confirmation:
   * Display a comprehensive order summary before finalizing the purchase.
   * Allow users to review and confirm the order details (items, shipping method, total cost) before proceeding.
5. Order Placement and Confirmation:
   * Process the order and generate an order confirmation with a unique order ID.
   * Send email confirmation to the user with the order details and estimated delivery date.
6. Order History and Tracking:
   * Store order details in the user's account for future reference.
   * Provide order tracking functionality to allow users to monitor the status of their shipments

### **Shopping Cart API**

#### **Add Product to Cart**

* Endpoint: POST /api/cart/add
* Description: Add a product to the user's shopping cart.
* Request Body:
* json
* Copy code

{

"productId": "123",

"quantity": 1,

"variant": {

"size": "Large",

"color": "Red"

}

}

* Response:
  + 200 OK on success:
  + json
  + Copy code

{

"message": "Product added to cart successfully"

}

* + 404 Not Found if the specified product ID does not exist.

#### **View Cart Contents**

* Endpoint: GET /api/cart
* Description: Retrieve the contents of the user's shopping cart.
* Response:
  + 200 OK on success:
  + json
  + Copy code

{

"items": [

{

"productId": "123",

"name": "Product A",

"quantity": 2,

"price": 49.99,

"subtotal": 99.98,

"variant": {

"size": "Large",

"color": "Red"

}

},

{

"productId": "456",

"name": "Product B",

"quantity": 1,

"price": 29.99,

"subtotal": 29.99

}

],

"totalItems": 3,

"totalPrice": 129.97

}

#### **Update Cart Item**

* Endpoint: PUT /api/cart/update/{productId}
* Description: Update the quantity or variant of a product in the user's shopping cart.
* Path Parameters:
  + productId: ID of the product to update.
* Request Body:
* json
* Copy code

{

"quantity": 3,

"variant": {

"size": "Medium",

"color": "Blue"

}

}

* Response:
  + 200 OK on success:
  + json
  + Copy code

{

"message": "Cart item updated successfully"

}

* + 404 Not Found if the specified product ID is not found in the cart.

#### **Remove Item from Cart**

* Endpoint: DELETE /api/cart/remove/{productId}
* Description: Remove a product from the user's shopping cart.
* Path Parameters:
  + productId: ID of the product to remove.
* Response:
  + 200 OK on success:
  + json
  + Copy code

{

"message": "Product removed from cart successfully"

}

* + 404 Not Found if the specified product ID is not found in the cart.

### **Checkout API**

#### **Initiate Checkout**

* Endpoint: POST /api/checkout
* Description: Initiate the checkout process to proceed with order placement.
* Request Body:
* json
* Copy code

{

"items": [

{

"productId": "123",

"quantity": 2,

"variant": {

"size": "Large",

"color": "Red"

}

},

{

"productId": "456",

"quantity": 1

}

],

"shippingAddress": {

"street": "123 Main St",

"city": "Cityville",

"state": "State",

"postalCode": "12345",

"country": "US"

},

"paymentMethod": "credit\_card"

}

* Response:
  + 200 OK on success:
  + json
  + Copy code

{

"orderId": "789",

"message": "Order placed successfully"

}

* + 400 Bad Request if the request is invalid.

#### **Get Order Details**

* Endpoint: GET /api/orders/{orderId}
* Description: Retrieve details of a specific order.
* Path Parameters:
  + orderId: ID of the order to retrieve.
* Response:
  + 200 OK on success:
  + json
  + Copy code

{

"orderId": "789",

"items": [

{

"productId": "123",

"name": "Product A",

"quantity": 2,

"price": 49.99,

"subtotal": 99.98,

"variant": {

"size": "Large",

"color": "Red"

}

},

{

"productId": "456",

"name": "Product B",

"quantity": 1,

"price": 29.99,

"subtotal": 29.99

}

],

"totalPrice": 129.97,

"shippingAddress": {

"street": "123 Main St",

"city": "Cityville",

"state": "State",

"postalCode": "12345",

"country": "US"

},

"paymentMethod": "credit\_card",

"status": "completed"

}

* + 404 Not Found if the specified order ID does not exist.

### **Error Responses**

Common error responses returned by the API:

* 400 Bad Request: Invalid request parameters or missing required fields.
* 404 Not Found: Resource not found (e.g., product not found in the cart, order not found).
* 500 Internal Server Error: Unexpected server error

### **Implementing Tax Calculation in API**

To implement tax calculation within your e-commerce API, consider the following approaches:

#### **Retrieve Tax Rates API**

* Endpoint: GET /api/tax-rates
* Description: Retrieve tax rates applicable to a specific location.
* Query Parameters:
  + country: Country code (e.g., ISO 3166-1 alpha-2 code).
  + state: State/province name or code.
  + postalCode: Postal code.
* Response:
  + 200 OK on success:
  + json
  + Copy code

{

"country": "US",

"state": "CA",

"taxRate": 0.0825

}

* + 404 Not Found if tax rates are not available for the specified location.

#### **Calculate Tax for Order API**

* Endpoint: POST /api/calculate-tax
* Description: Calculate taxes for a given order based on the provided items and shipping address.
* Request Body:
* json
* Copy code

{

"items": [

{

"productId": "123",

"quantity": 2,

"price": 49.99

},

{

"productId": "456",

"quantity": 1,

"price": 29.99

}

],

"shippingAddress": {

"country": "US",

"state": "CA",

"postalCode": "90001"

}

}

* Response:
  + 200 OK on success:
  + json
  + Copy code

{

"subtotal": 129.97,

"taxAmount": 10.73,

"total": 140.70

}

* + 400 Bad Request if the request is invalid or missing required fields.

### **Considerations for Tax Calculation**

* Dynamic Tax Rates: Ensure that tax rates can be updated dynamically based on changes in tax laws or regulations.
* Exemptions and Special Cases: Handle exemptions for certain products or categories that may be tax-exempt or subject to reduced tax rates.
* Integration with External Services: Integrate with third-party tax calculation services or APIs (e.g., taxjar, Avalara) for accurate and up-to-date tax calculations.
* Display of Taxes: Clearly display tax amounts and breakdowns on order summaries and invoices for transparency and compliance.

### 

### **Order Management**

1. Order Creation:
   * Allow customers to place orders through the website or mobile app.
   * Capture order details such as selected products, quantities, shipping address, and preferred payment method.
2. Order Processing:
   * Validate and process incoming orders, including inventory verification to ensure products are available for shipment.
   * Calculate order totals, including product costs, taxes, shipping fees, and discounts.
3. Order Tracking:
   * Provide real-time status updates to customers on their orders (e.g., order received, order shipped, order delivered).
   * Allow customers to track shipments using tracking numbers provided by shipping carriers.
4. Shipping and Fulfillment:
   * Generate shipping labels and packing slips for orders ready to be shipped.
   * Coordinate with shipping carriers (e.g., UPS, FedEx) to arrange pickup and delivery of packages.
5. Order Modifications and Cancellations:
   * Allow customers to modify or cancel orders before they are processed for shipping.
   * Implement business rules for handling order modifications (e.g., restocking fees, order revision deadlines).
6. Returns and Refunds:
   * Facilitate returns and refunds for eligible products based on return policies.
   * Process refunds promptly upon receiving returned items and update order status accordingly.
7. Customer Communication:
   * Send order confirmation emails to customers upon order placement.
   * Notify customers of order status changes, shipment tracking details, and delivery notifications.
8. Reporting and Analytics:
   * Generate reports on order trends, sales performance, and customer behavior for business analysis.
   * Use analytics to optimize inventory management, pricing strategies, and customer satisfaction.

### **Order Management API Documentation**

#### **1. Create Order**

* Endpoint: POST /api/orders
* Description: Create a new order based on customer's cart contents and checkout details.
* Request Body:
* json
* Copy code

{

"customerId": "123",

"items": [

{

"productId": "456",

"quantity": 2

},

{

"productId": "789",

"quantity": 1

}

],

"shippingAddress": {

"street": "123 Main St",

"city": "Cityville",

"state": "State",

"postalCode": "12345",

"country": "US"

},

"paymentMethod": "credit\_card"

}

* Response:
  + 201 Created on success:
  + json
  + Copy code

{

"orderId": "abc123",

"message": "Order created successfully"

}

* + 400 Bad Request if the request is invalid or missing required fields.

#### **2. Get Order Details**

* Endpoint: GET /api/orders/{orderId}
* Description: Retrieve detailed information about a specific order.
* Path Parameters:
  + orderId: ID of the order to retrieve.
* Response:
  + 200 OK on success:
  + json
  + Copy code

{

"orderId": "abc123",

"customerId": "123",

"items": [

{

"productId": "456",

"name": "Product A",

"quantity": 2,

"price": 29.99,

"subtotal": 59.98

},

{

"productId": "789",

"name": "Product B",

"quantity": 1,

"price": 49.99,

"subtotal": 49.99

}

],

"totalPrice": 109.97,

"shippingAddress": {

"street": "123 Main St",

"city": "Cityville",

"state": "State",

"postalCode": "12345",

"country": "US"

},

"paymentMethod": "credit\_card",

"status": "processing"

}

* + 404 Not Found if the specified order ID does not exist.

#### **3. Update Order Status**

* Endpoint: PUT /api/orders/{orderId}/status
* Description: Update the status of a specific order.
* Path Parameters:
  + orderId: ID of the order to update.
* Request Body:
* json
* Copy code

{

"status": "shipped"

}

* Response:
  + 200 OK on success:
  + json
  + Copy code

{

"message": "Order status updated successfully"

}

* + 400 Bad Request if the request is invalid or missing required fields.

#### **4. Cancel Order**

* Endpoint: DELETE /api/orders/{orderId}
* Description: Cancel a specific order.
* Path Parameters:
  + orderId: ID of the order to cancel.
* Response:
  + 200 OK on success:
  + json
  + Copy code

{

"message": "Order canceled successfully"

}

* + 404 Not Found if the specified order ID does not exist.

### **Order History API**

#### **Get Order History for Customer**

* Endpoint: GET /api/orders/history/{customerId}
* Description: Retrieve order history for a specific customer.
* Path Parameters:
  + customerId: ID of the customer whose order history is being retrieved.
* Response:
  + 200 OK on success:
  + json
  + Copy code

{

"customerId": "123",

"orders": [

{

"orderId": "abc123",

"items": [

{

"productId": "456",

"name": "Product A",

"quantity": 2,

"price": 29.99,

"subtotal": 59.98

},

{

"productId": "789",

"name": "Product B",

"quantity": 1,

"price": 49.99,

"subtotal": 49.99

}

],

"totalPrice": 109.97,

"status": "delivered",

"orderDate": "2024-04-15T10:30:00Z"

},

{

"orderId": "def456",

"items": [

{

"productId": "123",

"name": "Product C",

"quantity": 3,

"price": 39.99,

"subtotal": 119.97

}

],

"totalPrice": 119.97,

"status": "completed",

"orderDate": "2024-04-10T15:45:00Z"

}

]

}

* + 404 Not Found if no orders are found for the specified customer ID.

### **Returns and Refunds API**

#### **Initiate Return Request**

* Endpoint: POST /api/orders/returns
* Description: Initiate a return request for one or more items from a specific order.
* Request Body:
* json
* Copy code

{

"orderId": "abc123",

"items": [

{

"productId": "456",

"quantity": 1

}

],

"reason": "Wrong size"

}

* Response:
  + 200 OK on success:
  + json
  + Copy code

{

"returnId": "xyz789",

"message": "Return request submitted successfully"

}

* + 400 Bad Request if the request is invalid or missing required fields.

#### **Process Refund**

* Endpoint: POST /api/orders/refunds
* Description: Process a refund for a returned item and update inventory accordingly.
* Request Body:
* json
* Copy code

{

"returnId": "xyz789",

"refundAmount": 29.99

}

* Response:
  + 200 OK on success:
  + json
  + Copy code

{

"message": "Refund processed successfully"

}

* + 404 Not Found if the specified return ID is not found.

### **Order Tracking API Documentation**

#### **Track Order Shipment**

* Endpoint: GET /api/orders/{orderId}/track
* Description: Retrieve tracking information for a specific order shipment.
* Path Parameters:
  + orderId: ID of the order for which tracking information is requested.
* Response:
  + 200 OK on success:
  + json
  + Copy code

{

"orderId": "abc123",

"status": "shipped",

"trackingNumber": "XYZ123456789",

"carrier": "UPS",

"trackingUrl": "https://www.ups.com/track?tracking\_number=XYZ123456789"

}

* + 404 Not Found if the specified order ID does not exist or if tracking information is not available.

### **Sample Response Explanation**

* orderId: The unique identifier of the order being tracked.
* status: The current status of the shipment (e.g., shipped, out for delivery, delivered).
* trackingNumber: The tracking number assigned by the shipping carrier for this shipment.
* carrier: The name of the shipping carrier handling the shipment (e.g., UPS, FedEx, USPS).
* trackingUrl: A URL link to the shipping carrier's website for tracking the shipment using the provided tracking number.

### **Reporting and Analytics API**

#### **Generate Order Report**

* Endpoint: GET /api/reports/orders
* Description: Generate a report summarizing order-related metrics within a specified date range.
* Query Parameters:
  + startDate: Start date of the report period (optional).
  + endDate: End date of the report period (optional).
* Response:
  + 200 OK on success:
  + json
  + Copy code

{

"startDate": "2024-01-01",

"endDate": "2024-04-30",

"totalOrders": 100,

"totalSales": 5000.00,

"averageOrderValue": 50.00,

"topSellingProducts": [

{

"productId": "123",

"name": "Product A",

"totalQuantitySold": 200,

"totalRevenue": 10000.00

},

{

"productId": "456",

"name": "Product B",

"totalQuantitySold": 150,

"totalRevenue": 7500.00

}

]

}

* + 400 Bad Request if the date range is invalid.

### **Error Handling**

* Handle common HTTP status codes such as 400 Bad Request, 404 Not Found, and 500 Internal Server Error appropriately.
* Provide clear and informative error messages in response payloads for better API usability

**Security and Privacy**

### Data Encryption API

#### **Encrypt Data**

* Endpoint: POST /api/encryption/encrypt
* Description: Encrypt sensitive data using AES-256 encryption.
* Request Body:
* **json**
* **Copy code**

**{**

**"data": "Sensitive information to encrypt"**

**}**

* **Response:**
  + 200 OK **on successful encryption:**
  + **json**
  + **Copy code**

**{**

**"encryptedData": "6nJiZG4NQtbSehqsFLPb4q89Q8e4B5WuCrIveE9CwHk="**

**}**

#### **Decrypt Data**

* Endpoint: POST /api/encryption/decrypt
* Description: Decrypt previously encrypted data using AES-256 decryption.
* Request Body:
* **json**
* **Copy code**

**{**

**"encryptedData": "6nJiZG4NQtbSehqsFLPb4q89Q8e4B5WuCrIveE9CwHk="**

**}**

* **Response:**
  + 200 OK **on successful decryption:**
  + **json**
  + **Copy code**

**{**

**"decryptedData": "Sensitive information to encrypt"**

**}**

### **Privacy Controls API**

#### **User Data Preferences**

* Endpoint: GET /api/users/{userId}/privacy
* Description: Retrieve user's privacy settings and data preferences.
* Path Parameters:
  + userId**: ID of the user whose privacy settings are being retrieved.**
* **Response:**
  + 200 OK **on success:**
  + **json**
  + **Copy code**

**{**

**"userId": "123",**

**"dataPreferences": {**

**"marketingEmails": true,**

**"personalizedAds": false,**

**"dataSharing": {**

**"analytics": true,**

**"thirdParties": false**

**}**

**}**

**}**

#### **Update Data Preferences**

* Endpoint: PUT /api/users/{userId}/privacy
* Description: Update user's privacy settings and data preferences.
* Path Parameters:
  + userId**: ID of the user whose privacy settings are being updated.**
* **Request Body:**
* **json**
* **Copy code**

**{**

**"dataPreferences": {**

**"marketingEmails": false,**

**"personalizedAds": true,**

**"dataSharing": {**

**"analytics": true,**

**"thirdParties": true**

**}**

**}**

**}**

* **Response:**
  + 200 OK **on success:**
  + **json**
  + **Copy code**

**{**

**"message": "User privacy preferences updated successfully"**

**}**

### **Compliance API**

#### **Check Compliance Status**

* Endpoint: GET /api/compliance/status
* Description: Check compliance status with data protection regulations (e.g., GDPR, CCPA).
* Response:

200 OK **on success:**

**json**

**Copy code**

**{**

**"gdprCompliant": true,**

**"ccpaCompliant": false**

**}**

### **Product Management**

### **Product Management API Documentation**

#### **Create a New Product**

* Endpoint: POST /api/products
* Description: Create a new product with specified details.
* Request Body:
* **json**
* **Copy code**
  + - **{**
    - **"name": "Product Name",**
    - **"description": "Product Description",**
    - **"category": "Electronics",**
    - **"price": 499.99,**
    - **"images": ["image1.jpg", "image2.jpg"]**
    - **}**
* **Response:**
  + 201 Created **on success:**
  + **json**
  + **Copy code**
    - **{**
    - **"productId": "123",**
    - **"message": "Product created successfully"**
    - **}**
  + 400 Bad Request **if request body is invalid.**

#### **Get Product Details**

* Endpoint: GET /api/products/{productId}
* Description: Retrieve details of a specific product.
* Path Parameters:
  + productId**: ID of the product to retrieve.**
* **Response:**
  + 200 OK **on success:**
  + **json**
  + **Copy code**
    - **{**
    - **"productId": "123",**
    - **"name": "Product Name",**
    - **"description": "Product Description",**
    - **"category": "Electronics",**
    - **"price": 499.99,**
    - **"images": ["image1.jpg", "image2.jpg"],**
    - **"inventory": {**
    - **"quantity": 100**
    - **}**
    - **}**
  + 404 Not Found **if product with specified ID does not exist.**

#### **Update Product Details**

* Endpoint: PUT /api/products/{productId}
* Description: Update details of a specific product.
* Path Parameters:
  + productId**: ID of the product to update.**
* **Request Body:**
* **json**
* **Copy code**
  + - **{**
    - **"price": 449.99,**
    - **"images": ["updated\_image.jpg"]**
    - **}**
* **Response:**
  + 200 OK **on success:**
  + **json**
  + **Copy code**
    - **{**
    - **"message": "Product updated successfully"**
    - **}**
  + 404 Not Found **if product with specified ID does not exist.**

#### **Delete a Product**

* Endpoint: DELETE /api/products/{productId}
* Description: Delete a specific product.
* Path Parameters:
  + productId**: ID of the product to delete.**
* **Response:**
  + 200 OK **on success:**
  + **json**
  + **Copy code**
    - **{**
    - **"message": "Product deleted successfully"**
    - **}**
  + 404 Not Found **if product with specified ID does not exist.**

#### **Search Products**

* Endpoint: GET /api/products/search
* Description: Search for products based on specified query parameters.
* Query Parameters:
  + q**: Keyword to search for in product names or descriptions.**
  + category**: Filter products by category.**
  + minPrice**: Filter products by minimum price.**
  + maxPrice**: Filter products by maximum price.**
* **Response:**
  + 200 OK **on success:**
  + **json**
  + **Copy code**
    - **{**
    - **"results": [**
    - **{**
    - **"productId": "123",**
    - **"name": "Product Name",**
    - **"description": "Product Description",**
    - **"category": "Electronics",**
    - **"price": 449.99,**
    - **"images": ["image1.jpg"]**
    - **},**
    - **{**
    - **"productId": "456",**
    - **"name": "Another Product",**
    - **"description": "Another Description",**
    - **"category": "Clothing",**
    - **"price": 29.99,**
    - **"images": ["image2.jpg"]**
    - **}**
    - **]**
    - **}**

#### **Manage Product Inventory**

* Endpoint: PUT /api/products/{productId}/inventory
* Description: Update inventory quantity of a specific product.
* Path Parameters:
  + productId**: ID of the product to update inventory for.**
* **Request Body:**
* **json**
* **Copy code**
  + - **{**
    - **"quantity": 100**
    - **}**
* **Response:**
  + 200 OK **on success:**
  + **json**
  + **Copy code**
    - **{**
    - **"message": "Product inventory updated successfully"**
    - **}**
  + 404 Not Found **if product with specified ID does not exist.**

#### **Product Reviews and Ratings**

* Get Product Reviews:
  + Endpoint: GET /api/products/{productId}/reviews
  + Description: Retrieve reviews and ratings for a specific product.
  + Path Parameters:
    - productId**: ID of the product to retrieve reviews for.**
  + **Response:**
    - 200 OK **on success:**
    - **json**
    - **Copy code**
    - **{**
    - **"productId": "123",**
    - **"reviews": [**
    - **{**
    - **"userId": "user123",**
    - **"rating": 5,**
    - **"comment": "Great product!"**
    - **},**
    - **{**
    - **"userId": "user456",**
    - **"rating": 4,**
    - **"comment": "Good quality"**
    - **}**
    - **]**
    - **}**
* **Submit Product Review:**
  + **Endpoint:** POST /api/products/{productId}/reviews
  + Description: Submit a review and rating for a specific product.
  + Path Parameters:
    - productId**: ID of the product to submit review for.**
  + **Request Body:**
  + **json**
  + **Copy code**
    - **{**
    - **"userId": "user789",**
    - **"rating": 3,**
    - **"comment": "Average product"**
    - **}**
  + **Response:**
    - 201 Created **on success:**
    - **json**
    - **Copy code**
    - **{**
    - **"message": "Review submitted successfully"**
    - **}**

#### **Create New Category**

* Endpoint: POST /api/categories
* Description: Create a new product category.
* Request Body:
* **json**
* **Copy code**
  + - **{**
    - **"name": "New Category",**
    - **"parentId": "electronics"**
    - **}**
* **Response:**
  + 201 Created **on success:**
  + **json**
  + **Copy code**
    - **{**
    - **"categoryId": "newCategory",**
    - **"message": "Category created successfully"**
    - **}**

### **Pricing and Discount API**

#### **Apply Discount to Products**

* Endpoint: PUT /api/products/{productId}/discount
* Description: Apply a discount (e.g., percentage) to a specific product.
* Path Parameters:
  + productId**: ID of the product to apply the discount to.**
* **Request Body:**
* **json**
* **Copy code**
  + - **{**
    - **"discountPercentage": 10**
    - **}**
* **Response:**
  + 200 OK **on success:**
  + **json**
  + **Copy code**
    - **{**
    - **"message": "Discount applied successfully"**
    - **}**

#### **Get Product Pricing Information**

* Endpoint: GET /api/products/{productId}/pricing
* Description: Retrieve pricing information for a specific product, including regular price, sale price, and applied discounts.
* Path Parameters:
  + productId**: ID of the product to retrieve pricing information for.**
* **Response:**
  + 200 OK **on success:**
  + **json**
  + **Copy code**
    - **{**
    - **"regularPrice": 499.99,**
    - **"salePrice": 449.99,**
    - **"discountPercentage": 10**
    - **}**

### **Analytics and Reporting API**

#### **Generate Product Sales Report**

* Endpoint: GET /api/reports/product-sales
* Description: Generate a report summarizing product sales within a specified date range.
* Query Parameters:
  + startDate**: Start date of the report period (optional).**
  + endDate**: End date of the report period (optional).**
* **Response:**
  + 200 OK **on success:**
  + **json**
  + **Copy code**
    - **{**
    - **"startDate": "2024-01-01",**
    - **"endDate": "2024-04-30",**
    - **"totalSales": 50000.00,**
    - **"topSellingProducts": [**
    - **{**
    - **"productId": "123",**
    - **"name": "Product A",**
    - **"totalQuantitySold": 200,**
    - **"totalRevenue": 10000.00**
    - **},**
    - **{**
    - **"productId": "456",**
    - **"name": "Product B",**
    - **"totalQuantitySold": 150,**
    - **"totalRevenue": 7500.00**
    - **}**
    - **]**
  + **}**

### **Content Management**

#### **Product Catalog Management**

* Endpoint: POST /api/products
* Description: Create a new product listing.
* Request Body:
* **json**
* **Copy code**

**{**

**"name": "Product Name",**

**"description": "Product Description",**

**"category": "Electronics",**

**"price": 499.99,**

**"images": ["image1.jpg", "image2.jpg"]**

**}**

* **Response:**
  + 201 Created **on success:**
  + **json**
  + **Copy code**

**{**

**"productId": "123",**

**"message": "Product created successfully"**

**}**

#### **Content Publishing and Management**

* Endpoint: POST /api/content/banners
* Description: Publish a new promotional banner.
* Request Body:
* **json**
* **Copy code**

**{**

**"title": "Special Sale!",**

**"image": "banner-image.jpg",**

**"link": "/special-offers"**

**}**

* **Response:**
  + 201 Created **on success:**
  + **json**
  + **Copy code**

**{**

**"bannerId": "banner1",**

**"message": "Banner published successfully"**

**}**

#### **Blog Management**

* Endpoint: POST /api/content/articles
* Description: Create a new blog article.
* Request Body:
* **json**
* **Copy code**

**{**

**"title": "10 Tips for Choosing the Right Smartphone",**

**"content": "Lorem ipsum dolor sit amet, consectetur adipiscing elit...",**

**"author": "John Doe",**

**"tags": ["smartphones", "buying-guide"]**

**}**

* **Response:**
  + 201 Created **on success:**
  + **json**
  + **Copy code**

**{**

**"articleId": "article1",**

**"message": "Article published successfully"**

* + **}**

### **Inventory Management**

1. Inventory Tracking:
   * Real-time Tracking: Monitor inventory levels in real-time to know the quantity and location of each item.
   * Barcode/RFID Systems: Use barcode or RFID technology to accurately track inventory movements and updates.
2. Inventory Optimization:
   * Demand Forecasting: Predict future demand based on historical data to optimize inventory levels and prevent stockouts.
   * Safety Stock: Maintain a safety buffer of inventory to account for unexpected demand fluctuations or delays in supply.
3. Order Management:
   * Reorder Point: Set a reorder point to automatically trigger purchase orders when inventory levels drop below a certain threshold.
   * Order Fulfillment: Streamline order processing and fulfillment workflows to ensure timely delivery of products to customers.
4. Inventory Analysis:
   * ABC Analysis: Classify inventory items into categories (e.g., A, B, C) based on their value and prioritize management efforts accordingly.
   * Inventory Turnover Ratio: Calculate how quickly inventory is sold and replaced to optimize stock levels and cash flow.
5. Stock Control:
   * Batch and Lot Tracking: Manage inventory items based on batch or lot numbers to trace and recall specific products if needed.
   * Serial Number Tracking: Assign unique serial numbers to individual inventory items for traceability and warranty management.
6. Warehouse Management:
   * Optimized Storage: Organize warehouse layout and storage locations to maximize space utilization and facilitate efficient picking.
   * Pick-Pack-Ship Process: Implement streamlined processes for picking, packing, and shipping orders to minimize fulfillment times.
7. Supplier Collaboration:
   * Vendor Management: Establish strong relationships with suppliers and coordinate inventory levels to ensure timely replenishment.
   * Just-in-Time (JIT) Inventory: Work closely with suppliers to implement JIT inventory practices to reduce holding costs and minimize inventory storage.
8. Inventory Reporting:
   * Inventory Valuation: Calculate the value of inventory on hand using different valuation methods (e.g., FIFO, LIFO).
   * Inventory Aging Analysis: Monitor aging inventory to identify slow-moving or obsolete items for liquidation or discounting.

### **API Documentation**

#### **Retrieve Product Inventory**

* Endpoint: GET /api/products/{productId}/inventory
* Description: Retrieve current inventory details for a specific product.
* Path Parameters:
  + productId**: ID of the product to retrieve inventory for.**
* **Response:**
  + 200 OK **on success:**
  + **json**
  + **Copy code**

**{**

**"productId": "123",**

**"inventory": {**

**"quantity": 100,**

**"location": "Warehouse A"**

**}**

**}**

#### **Adjust Product Inventory**

* Endpoint: POST /api/products/{productId}/inventory/adjust
* Description: Adjust inventory quantity for a specific product.
* Path Parameters:
  + productId**: ID of the product to adjust inventory for.**
* **Request Body:**
* **json**
* **Copy code**

**{**

**"adjustmentType": "increase",**

**"quantity": 10,**

**"reason": "Received new shipment"**

**}**

* **Response:**
  + 200 OK **on success:**
  + **json**
  + **Copy code**

**{**

**"message": "Inventory adjusted successfully"**

**}**

#### **Allocate Inventory for Order**

* Endpoint: POST /api/orders/{orderId}/allocate
* Description: Allocate inventory quantities for a specific order to reserve stock.
* Path Parameters:
  + orderId**: ID of the order to allocate inventory for.**
* **Request Body:**
* **json**
* **Copy code**

**{**

**"items": [**

**{**

**"productId": "123",**

**"quantity": 2**

**},**

**{**

**"productId": "456",**

**"quantity": 1**

**}**

**]**

**}**

* **Response:**
  + 200 OK **on success:**
  + **json**
  + **Copy code**

**{**

**"message": "Inventory allocated successfully"**

**}**

#### **Fulfill Order and Update Inventory**

* Endpoint: POST /api/orders/{orderId}/fulfill
* Description: Fulfill an order and update inventory quantities accordingly.
* Path Parameters:
  + orderId**: ID of the order to fulfill.**
* **Request Body:**
* **json**
* **Copy code**

**{**

**"items": [**

**{**

**"productId": "123",**

**"quantity": 2**

**},**

**{**

**"productId": "456",**

**"quantity": 1**

**}**

**]**

**}**

* **Response:**
  + 200 OK **on success:**
  + **json**
  + **Copy code**

**{**

**"message": "Order fulfilled successfully"**

**}**

#### **Retrieve Low Inventory Products**

* Endpoint: GET /api/inventory/low
* Description: Retrieve products with low inventory quantities (below a specified threshold).
* Query Parameters:
  + threshold**: Minimum quantity threshold for low inventory (default: 10).**
* **Response:**
  + 200 OK **on success:**
  + **json**
  + **Copy code**

**{**

**"lowInventoryProducts": [**

**{**

**"productId": "123",**

**"name": "Product A",**

**"quantity": 5,**

**"location": "Warehouse B"**

**},**

**{**

**"productId": "456",**

**"name": "Product B",**

**"quantity": 8,**

**"location": "Warehouse A"**

**}**

**]**

**}**

#### **Track Inventory History**

* Endpoint: GET /api/products/{productId}/inventory/history
* Description: Retrieve inventory history for a specific product.
* Path Parameters:
  + productId**: ID of the product to retrieve inventory history for.**
* **Response:**
  + 200 OK **on success:**
  + **json**
  + **Copy code**

**{**

**"productId": "123",**

**"inventoryHistory": [**

**{**

**"date": "2024-04-01",**

**"quantityChange": 10,**

**"reason": "Received new shipment"**

**},**

**{**

**"date": "2024-04-05",**

**"quantityChange": -3,**

**"reason": "Fulfilled orders"**

**}**

**]**

* + **}**
* Bulk Inventory Updates:
  + Endpoint: POST /api/products/inventory/bulk-update
  + Description: Update inventory quantities for multiple products in bulk.
  + Request Body:
  + **json**
  + **Copy code**

**{**

**"updates": [**

**{**

**"productId": "123",**

**"quantity": 20**

**},**

**{**

**"productId": "456",**

**"quantity": 15**

**}**

**]**

**}**

* + **Response:**
    - 200 OK **on success:**
    - **json**
    - **Copy code**

**{**

**"message": "Bulk inventory update successful"**

**}**

* **Inventory Reconciliation:**
  + **Endpoint:** POST /api/inventory/reconciliation
  + Description: Perform inventory reconciliation to match physical inventory counts with system records.
  + Request Body:
  + **json**
  + **Copy code**

**{**

**"inventoryCounts": [**

**{**

**"productId": "123",**

**"physicalCount": 100**

**},**

**{**

**"productId": "456",**

**"physicalCount": 50**

**}**

**]**

**}**

* + **Response:**
    - 200 OK **on success:**
    - **json**
    - **Copy code**

**{**

**"message": "Inventory reconciliation completed"**

**}**

* **Inventory Alerts and Notifications:**
  + **Endpoint:** GET /api/inventory/alerts
  + Description: Retrieve alerts and notifications related to low inventory levels or stockouts.
  + Response:
    - 200 OK **on success:**
    - **json**
    - **Copy code**

**{**

**"alerts": [**

**{**

**"productId": "123",**

**"name": "Product A",**

**"quantity": 5,**

**"location": "Warehouse B",**

**"alertType": "Low Inventory"**

**},**

**{**

**"productId": "456",**

**"name": "Product B",**

**"quantity": 0,**

**"location": "Warehouse A",**

**"alertType": "Stockout"**

**}**

**]**

**}**

* **Inventory Status Dashboard:**
  + **Endpoint:** GET /api/inventory/dashboard
  + Description: Retrieve a dashboard with real-time inventory status and metrics.
  + Response:
    - 200 OK **on success:**
    - **json**
    - **Copy code**

**{**

**"totalProducts": 100,**

**"totalInventory": 5000,**

**"lowInventoryCount": 10,**

**"stockoutCount": 3,**

**"inventoryByLocation": {**

**"Warehouse A": 3000,**

**"Warehouse B": 2000**

**}**

**}**

* **Inventory Reports:**
  + **Endpoint:** GET /api/reports/inventory
  + Description: Generate inventory reports based on specified criteria (e.g., by product category, by location).
  + Query Parameters:
    - category**: Filter by product category.**
    - location**: Filter by inventory location.**
    - startDate**,** endDate**: Filter by date range.**
  + **Response:**
    - 200 OK **on success:**
    - **json**
    - **Copy code**

**{**

**"reportType": "Inventory Summary",**

**"startDate": "2024-01-01",**

**"endDate": "2024-04-30",**

**"data": [**

**{**

**"category": "Electronics",**

**"totalProducts": 50,**

**"totalInventory": 2500**

**},**

**{**

**"category": "Apparel",**

**"totalProducts": 30,**

**"totalInventory": 1500**

**}**

**]**

* + - **}**