Complete API Documentation

Introduction

This documentation covers the API endpoints for products, reviews, and user authentication. All endpoints follow standardized request and response formats.

Common Information

Base URLs

- Products: `/api/v1/products`
- Reviews: `/api/v1/reviews`
- Users: `/api/v1/user`
- address: `/api/v1/address`
- cart: `/api/v1/cart`
- order: `/api/v1/orders`
- category: `/api/v1/category`

Authentication

Authentication is handled via JWT tokens that can be provided through:

- `Authorization` header
- HTTP-only cookies

Standard Error Response

All endpoints return standardized error responses:

```
{
  "statusCode": 400,
  "data": {},
  "message": "Error message",
  "success": false
}
```

Product API

Get All Products

Retrieves a paginated list of all available products.

URL: `/api/v1/products/`
Method: `GET`
Auth Required: No

Query Parameters:

- `page` : Page number for pagination (default: 1)
- `limit` : Number of products per page (default: 10)

Success Response:

```
{
  "success": true,
  "products": [
    {
      "_id": "60d21b4667d0d8992e610c85",
      "name": "Product Name",
      "description": "Product description",
      "brand": "Brand Name",
      "price": 99.99,
      "countInStock": 50,
      "category": {
        "_id": "60d21b4667d0d8992e610c80",
        "name": "Category Name"
      },
      "images": [
        "https://cloudinary.com/image1.jpg",
        "https://cloudinary.com/image2.jpg"
      ],
      "slug": "product-name"
    }
  ],
  "currentPage": 1,
  "totalPages": 5,
  "totalProducts": 42
}
```


Notes:

- Only returns products where `isAvailable` is true
- Results are sorted by creation date (newest first)

Get Products by Category

Retrieves all products belonging to a specific category.

URL: `/api/v1/products/category/:categoryId`
Method: `GET`
Auth Required: No

URL Parameters:

- `categoryId` : ID of the category to filter products by

Query Parameters:

- `page` : Page number for pagination (default: 1)
- `limit` : Number of products per page (default: 10)

Success Response:

```
{
  "statusCode": 200,
  "data": {
    "products": [
      {
        "_id": "60d21b4667d0d8992e610c85",
        "name": "Product Name",
        "description": "Product description",
        "brand": "Brand Name",
        "price": 99.99,
        "countInStock": 50,
        "category": {
          "_id": "60d21b4667d0d8992e610c80",
          "name": "Category Name"
        },
        "images": [
          "https://cloudinary.com/image1.jpg",
          "https://cloudinary.com/image2.jpg"
        ],
        "slug": "product-name"
      }
    ],
    "currentPage": 1,
    "totalPages": 3,
    "totalProducts": 25
  },
  "message": "Products fetched successfully",
  "success": true
}
```


Notes:

- Only returns products where `isAvailable` is true
- Returns 404 if no products are found in the category

Get Product by ID

Retrieves detailed information about a specific product.

URL: `/api/v1/products/:productId`
Method: `GET`
Auth Required: No

URL Parameters:

- `productId`: ID of the product to retrieve

Success Response:


```
{
  "statusCode": 200,
  "data": {
    "_id": "60d21b4667d0d8992e610c85",
    "name": "Product Name",
    "description": "Product description",
    "brand": "Brand Name",
    "price": 99.99,
    "countInStock": 50,
    "category": ["60d21b4667d0d8992e610c80"],
    "images": [
      "https://cloudinary.com/image1.jpg",
      "https://cloudinary.com/image2.jpg"
    ],
    "slug": "product-name"
  },
  "message": "Product fetched successfully",
  "success": true
}
```

Notes:

- Only returns products where `isAvailable` is true

Create Product

Creates a new product.

URL: `/api/v1/products/`
Method: `POST`
Auth Required: Yes (Admin only)

Request Body:

```
{
  "name": "New Product",
  "description": "Product description",
  "brand": "Brand Name",
  "price": 99.99,
  "countInStock": 50,
  "category": ["60d21b4667d0d8992e610c80"]
}
```

File Upload:

- Field name: `images`
- Maximum: 5 images
- Format: Form-data with fields and files

Success Response:

```
{
  "statusCode": 201,
  "data": {
    "_id": "60d21b4667d0d8992e610c85",
    "name": "New Product",
    "description": "Product description",
    "brand": "Brand Name",
    "price": 99.99,
    "countInStock": 50,
    "category": ["60d21b4667d0d8992e610c80"],
    "images": [
      "https://cloudinary.com/image1.jpg",
      "https://cloudinary.com/image2.jpg"
    ],
    "slug": "new-product",
    "isAvailable": true
  },
  "message": "Product created successfully.",
  "success": true
}
```

Notes:

- Product name must be unique (generates a unique slug)
- At least one image is required
- Maximum of 5 images allowed
- All categories must be valid

Update Product

Updates an existing product.

URL: `/api/v1/products/:productId`
Method: `PUT`
Auth Required: Yes (Admin only)

URL Parameters:

- `productId`: ID of the product to update

Request Body:

```
{
  "name": "Updated Product Name",
  "description": "Updated description",
  "brand": "Updated Brand",
  "price": 109.99,
  "countInStock": 45,
  "category": ["60d21b4667d0d8992e610c80", "60d21b4667d0d8992e610c81"],
  "existingImages": ["https://cloudinary.com/image1.jpg"]
}
```

File Upload:

- Field name: `images` (optional)
- Maximum: 5 images total (existing + new)
- Format: Form-data with fields and files

Success Response:

```
{
  "statusCode": 200,
  "data": {
    "_id": "60d21b4667d0d8992e610c85",
    "name": "Updated Product Name",
    "description": "Updated description",
    "brand": "Updated Brand",
    "price": 109.99,
    "countInStock": 45,
    "category": ["60d21b4667d0d8992e610c80", "60d21b4667d0d8992e610c81"],
    "images": [
      "https://cloudinary.com/image1.jpg",
      "https://cloudinary.com/newimage.jpg"
    ],
    "slug": "updated-product-name",
    "isAvailable": true
  },
  "message": "Product updated successfully.",
  "success": true
}
```

Notes:

- All fields are optional - only the provided fields will be updated
- If updating name, it must be unique
- Maximum of 5 images allowed in total
- Existing images not included in the `existingImages` array will be removed

Update Product Stock

Updates just the stock quantity of a product.

URL: `/api/v1/products/:productId/stock`
Method: `PUT`
Auth Required: Yes (Admin only)

URL Parameters:

- `productId`: ID of the product to update stock

Request Body:

```
{
  "countInStock": 75
}
```

Success Response:

```
{
  "statusCode": 200,
  "data": {
    "_id": "60d21b4667d0d8992e610c85",
    "name": "Product Name",
    "countInStock": 75
    /* other product fields */
  },
  "message": "Product stock updated successfully",
  "success": true
}
```

Notes:

- Only updates the `countInStock` field
- Stock count must be a non-negative number

Delete Product

Soft-deletes a product by marking it as unavailable.

URL: `/api/v1/products/:productId`
Method: `DELETE`
Auth Required: Yes (Admin only)

URL Parameters:

- `productId`: ID of the product to delete

Success Response:

```
{
  "statusCode": 200,
  "data": {
    "_id": "60d21b4667d0d8992e610c85",
    "name": "Product Name",
    /* other product fields */
    "isAvailable": false
  },
  "message": "Product deleted successfully",
  "success": true
}
```

Notes:

- Product is not actually removed from the database
- Sets `isAvailable` to false
- Also deletes all reviews associated with this product

Review API

Create Review

Creates a new review for a product.

URL: `/api/v1/reviews/`
Method: `POST`
Auth Required: Yes

Request Body:

```
{
  "productId": "60d21b4667d0d8992e610c85",
  "rating": 4.5,
  "comment": "Great product, highly recommended!"
}
```

Success Response:

```
{
  "statusCode": 201,
  "data": {
    "_id": "60d21b4667d0d8992e610c86",
    "product": "60d21b4667d0d8992e610c85",
    "user": "60d21b4667d0d8992e610c87",
    "rating": 4.5,
    "comment": "Great product, highly recommended!"
  },
  "message": "Review created",
  "success": true
}
```


Notes:

- Users can only submit one review per product
- Product rating is automatically updated based on the average of all reviews

Update Review

Updates an existing review.

URL: `/api/v1/reviews/:reviewId`
Method: `PUT`
Auth Required: Yes

URL Parameters:

- `reviewId` : ID of the review to update

Request Body:

```
{
  "rating": 5,
  "comment": "Even better than I initially thought!"
}
```

Success Response:


```
{
  "statusCode": 200,
  "data": {
    "_id": "60d21b4667d0d8992e610c86",
    "product": "60d21b4667d0d8992e610c85",
    "user": "60d21b4667d0d8992e610c87",
    "rating": 5,
    "comment": "Even better than I initially thought!"
  },
  "message": "Review updated",
  "success": true
}
```


Notes:

- Users can only update their own reviews
- Product rating is automatically recalculated after update

Delete Review

Deletes a review.

URL: `/api/v1/reviews/:reviewId`
Method: `DELETE`
Auth Required: Yes

URL Parameters:

- `reviewId` : ID of the review to delete

Success Response:

```
{
  "statusCode": 200,
  "data": {
    "_id": "60d21b4667d0d8992e610c86",
    "product": "60d21b4667d0d8992e610c85",
    "user": "60d21b4667d0d8992e610c87",
    "rating": 5,
    "comment": "Even better than I initially thought!"
  },
  "message": "Review deleted",
  "success": true
}
```


Notes:

- Users can only delete their own reviews

Get Reviews by Product

Retrieves all reviews for a specific product.

URL: `/api/v1/reviews/product/:productId`
Method: `GET`
Auth Required: Yes

URL Parameters:

- `productId`: ID of the product to get reviews for

Success Response:


```
{
  "statusCode": 200,
  "data": [
    {
      "_id": "60d21b4667d0d8992e610c86",
      "product": "60d21b4667d0d8992e610c85",
      "user": {
        "_id": "60d21b4667d0d8992e610c87",
        "name": "John Doe"
      },
      "rating": 5,
      "comment": "Great product, highly recommended!"
    },
    {
      "_id": "60d21b4667d0d8992e610c88",
      "product": "60d21b4667d0d8992e610c85",
      "user": {
        "_id": "60d21b4667d0d8992e610c89",
        "name": "Jane Smith"
      },
      "rating": 4,
      "comment": "Good quality but a bit expensive."
    }
  ],
  "message": "Reviews fetched",
  "success": true
}
```


Notes:

- Returns an array of reviews with populated user information (name only)

Get Reviews by User

Retrieves all reviews made by a specific user.

URL: `/api/v1/reviews/user/:userId`
Method: `GET`
Auth Required: Yes

URL Parameters:

- `userId` : ID of the user to get reviews for

Success Response:

```
{
  "statusCode": 200,
  "data": [
    {
      "_id": "60d21b4667d0d8992e610c86",
      "user": "60d21b4667d0d8992e610c87",
      "product": {
        "_id": "60d21b4667d0d8992e610c85",
        "name": "Wireless Headphones"
      },
      "rating": 5,
      "comment": "Great product, highly recommended!"
    },
    {
      "_id": "60d21b4667d0d8992e610c90",
      "user": "60d21b4667d0d8992e610c87",
      "product": {
        "_id": "60d21b4667d0d8992e610c91",
        "name": "Smartphone Case"
      },
      "rating": 3,
      "comment": "Average quality, does the job."
    }
  ],
  "message": "Reviews fetched",
  "success": true
}
```

Notes:

- Returns an array of reviews with populated product information (name only)

Get Review by ID

Retrieves a specific review by its ID.

URL: `/api/v1/reviews/:reviewId`
Method: `GET`
Auth Required: Yes

URL Parameters:

- `reviewId` : ID of the review to retrieve

Success Response:

```
{
  "statusCode": 200,
  "data": {
    "_id": "60d21b4667d0d8992e610c86",
    "user": "60d21b4667d0d8992e610c87",
    "product": {
      "_id": "60d21b4667d0d8992e610c85",
      "name": "Wireless Headphones"
    },
    "rating": 5,
    "comment": "Great product, highly recommended!"
  },
  "message": "Review fetched",
  "success": true
}
```

Notes:

- Returns a single review with populated product information (name only)

User Authentication API

Public Endpoints

Register User

Creates a new user account and sends email verification.

URL: `/api/v1/user/register`
Method: `POST`
Auth Required: No

Request Body:

```
{
  "fullName": "John Doe",
  "email": "john@example.com",
  "mobile": "1234567890",
  "password": "securePassword123"
}
```

Success Response:

```
{
  "statusCode": 201,
  "data": {
    "fullName": "John Doe",
    "email": "john@example.com",
    "mobile": "1234567890",
    "status": "Inactive",
    "_id": "60d21b4667d0d8992e610c85"
  },
  "message": "User registered. Please verify your email address.",
  "success": true
}
```

Login User

Authenticates a user and returns tokens.

URL: `/api/v1/user/login`
Method: `POST`
Auth Required: No

Request Body:

```
{
  "email": "john@example.com",
  "password": "securePassword123"
}
```

OR

```
{
  "mobile": "1234567890",
  "password": "securePassword123"
}
```

Success Response:

```
{
  "statusCode": 200,
  "data": {
    "user": {
      "_id": "60d21b4667d0d8992e610c85",
      "fullName": "John Doe",
      "email": "john@example.com",
      "mobile": "1234567890",
      "status": "Active"
    },
    "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  },
  "message": "User logged in successfully",
  "success": true
}
```


Notes:

- Sets HTTP-only cookies for `accessToken` and `refreshToken`
- If email is not verified, returns 403 and sends a new verification email

Verify Email

Verifies a user's email address using the token sent during registration.

URL: `/api/v1/user/verify-email/:token`
Method: `GET`
Auth Required: No

URL Parameters:

- `token`: Email verification token

Success Response:


```
{
  "statusCode": 200,
  "data": null,
  "message": "Email verified successfully",
  "success": true
}
```


Notes:

- If token is expired, sends a new verification email automatically

Forgot Password

Sends a password reset link to the user's email.

URL: `/api/v1/user/forget-password`
Method: `POST`
Auth Required: No

Request Body:


```
{
  "email": "john@example.com"
}
```


Success Response:


```
{
  "statusCode": 200,
  "data": {},
  "message": "If the email is valid, a password reset link has been sent.",
  "success": true
}
```


Notes:

- Always returns success response even if email doesn't exist (security)

Reset Password

Resets user's password using a token received by email.

URL: `/api/v1/user/reset-password/:token`
Method: `POST`
Auth Required: No

URL Parameters:

- `token` : Password reset token

Request Body:

```
{
  "password": "newSecurePassword123",
  "confirmPassword": "newSecurePassword123"
}
```

Success Response:

```
{
  "statusCode": 200,
  "data": {},
  "message": "Password reset successfully",
  "success": true
}
```

Refresh Token

Generates a new access token using a refresh token.

URL: `/api/v1/user/refresh-token`
Method: `POST`
Auth Required: No

Request Body:

```
{
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

Success Response:

```
{
  "statusCode": 200,
  "data": {
    "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  },
  "message": "Access token refreshed successfully.",
  "success": true
}
```

Notes:

- Refresh token can also be provided via cookies

Protected User Endpoints

Get Current User

Returns the current authenticated user's details.

URL: `/api/v1/user/current-user`
Method: `GET`
Auth Required: Yes

Success Response:

```
{
  "statusCode": 200,
  "data": {
    "_id": "60d21b4667d0d8992e610c85",
    "fullName": "John Doe",
    "email": "john@example.com",
    "mobile": "1234567890",
    "status": "Active"
  },
  "message": "User details fetched successfully",
  "success": true
}
```


Update Account Details

Updates the user's profile information.

URL: `/api/v1/user/update-account`
Method: `PUT`
Auth Required: Yes

Request Body:


```
{
  "fullName": "John Smith",
  "email": "john.smith@example.com",
  "mobile": "9876543210"
}
```


Success Response:


```
{
  "statusCode": 200,
  "data": {
    "user": {
      "_id": "60d21b4667d0d8992e610c85",
      "fullName": "John Smith",
      "email": "john.smith@example.com",
      "mobile": "9876543210",
      "status": "Active"
    }
  },
  "message": "account details updated successfully",
  "success": true
}
```


Notes:

- At least one field is required

Change Password

Changes the user's password.

URL: `/api/v1/user/change-password`
Method: `POST`
Auth Required: Yes

Request Body:


```
{
  "currentPassword": "currentSecurePassword123",
  "newPassword": "newSecurePassword123",
  "confirmNewPassword": "newSecurePassword123"
}
```


Success Response:

```
{
  "statusCode": 200,
  "data": {},
  "message": "Password changed successfully",
  "success": true
}
```

Logout User

Logs out the current user by invalidating tokens.

URL: `/api/v1/user/logout`
Method: `POST`
Auth Required: Yes

Success Response:

```
{
  "statusCode": 200,
  "data": {},
  "message": "User logged Out",
  "success": true
}
```

Create Address

Creates a new address for the authenticated user.

URL: `/`
Method: `POST`
Auth Required: Yes (JWT)

Request Body:

```
{
  "addressLine1": "123 Main Street",
  "city": "New York",
  "state": "NY",
  "pincode": "10001",
  "country": "USA",
  "mobile": "1234567890"
}
```

Success Response:

```
{
  "statusCode": 201,
  "data": {
    "addressLine1": "123 Main Street",
    "city": "New York",
    "state": "NY",
    "pincode": "10001",
    "country": "USA",
    "mobile": "1234567890"
  },
  "message": "Address created successfully",
  "success": true
}
```

Get All User Addresses

Retrieves all addresses for the authenticated user.

URL: `/`
Method: `GET`
Auth Required: Yes (JWT)

Success Response:

```
{
  "statusCode": 200,
  "data": [
    {
      "addressLine1": "123 Main Street",
      "city": "New York",
      "state": "NY",
      "pincode": "10001",
      "country": "USA",
      "mobile": "1234567890"
    },
    {
      "addressLine1": "456 Park Avenue",
      "city": "Boston",
      "state": "MA",
      "pincode": "02108",
      "country": "USA",
      "mobile": "9876543210"
    }
  ],
  "message": "Addresses retrieved successfully",
  "success": true
}
```

Note:

- If no addresses are found, the message will be "No addresses found"

Get Address by ID

Retrieves a specific address by its ID.

URL: `/:addressId`
Method: `GET`
Auth Required: Yes (JWT)

URL Parameters:

- `addressId`: Address ID

Success Response:

```
{
  "statusCode": 200,
  "data": {
    "addressLine1": "123 Main Street",
    "city": "New York",
    "state": "NY",
    "pincode": "10001",
    "country": "USA",
    "mobile": "1234567890"
  },
  "message": "Address retrieved successfully",
  "success": true
}
```

Error Response:

```
{
  "statusCode": 404,
  "data": {},
  "message": "Address not found",
  "success": false
}
```

Update Address

Updates an existing address by its ID.

URL: `/:addressId`
Method: `PUT`
Auth Required: Yes (JWT)

URL Parameters:

- `addressId`: Address ID

Request Body:

```
{
  "addressLine1": "123 Main Street, Apt 4B",
  "city": "New York",
  "state": "NY",
  "pincode": "10001",
  "country": "USA",
  "mobile": "1234567890"
}
```

Success Response:

```
{
  "statusCode": 200,
  "data": {
    "addressLine1": "123 Main Street, Apt 4B",
    "city": "New York",
    "state": "NY",
    "pincode": "10001",
    "country": "USA",
    "mobile": "1234567890"
  },
  "message": "Address updated successfully",
  "success": true
}
```

Error Response:

```
{
  "statusCode": 404,
  "data": {},
  "message": "Address not found or unauthorized",
  "success": false
}
```

Delete Address

Deletes an address by its ID.

URL: `/:addressId`
Method: `DELETE`
Auth Required: Yes (JWT)

URL Parameters:

- `addressId`: Address ID

Success Response:

```
{
  "statusCode": 200,
  "data": "Address deleted successfully",
  "message": "Address deleted successfully",
  "success": true
}
```

Error Response:

```
{
  "statusCode": 404,
  "data": {},
  "message": "Address not found",
  "success": false
}
```

Validation

All endpoints use specific validators:

- `createAddressValidator` for POST
- `getAddressByIdValidator` for GET /:addressId
- `updateAddressValidator` for PUT
- `deleteAddressValidator` for DELETE

Category API

Base URL

```
/api/v1/category
```

Endpoints

Get All Categories

Retrieves all categories.

URL: `/`
Method: `GET`
Auth Required: No

Success Response:

```
{
  "statusCode": 200,
  "data": [
    {
      "_id": "60d21b4667d0d8992e610c85",
      "name": "Electronics",
      "slug": "electronics",
      "description": "Electronic items and gadgets"
    },
    {
      "_id": "60d21b4667d0d8992e610c86",
      "name": "Clothing",
      "slug": "clothing",
      "description": "Fashion items"
    }
  ],
  "message": "Categories fetched successfully",
  "success": true
}
```

Create Category

Creates a new category. Requires authentication.

URL: `/`
Method: `POST`
Auth Required: Yes (JWT)

Request Body:

```
{
  "name": "Electronics",
  "description": "Electronic items and gadgets"
}
```

Success Response:

```
{
  "statusCode": 201,
  "data": {
    "_id": "60d21b4667d0d8992e610c85",
    "name": "Electronics",
    "slug": "electronics",
    "description": "Electronic items and gadgets"
  },
  "message": "Category created successfully",
  "success": true
}
```

Update Category

Updates an existing category by slug. Requires authentication.

URL: `/:slug`
Method: `PUT`
Auth Required: Yes (JWT)

URL Parameters:

- `slug` : Category slug

Request Body:

```
{
  "name": "Electronics and Gadgets",
  "description": "Updated description for electronic items"
}
```

Success Response:

```
{
  "statusCode": 200,
  "data": {
    "_id": "60d21b4667d0d8992e610c85",
    "name": "Electronics and Gadgets",
    "slug": "electronics-and-gadgets",
    "description": "Updated description for electronic items"
  },
  "message": "Category updated successfully",
  "success": true
}
```

Delete Category

Deletes a category by slug. Requires admin authentication.

URL: `/:slug`

Method: `DELETE`
Auth Required: Yes (JWT + Admin)

URL Parameters:

- `slug`: Category slug

Success Response:

```
{
  "statusCode": 200,
  "data": {},
  "message": "Category deleted successfully",
  "success": true
}
```

Authentication Note

- The `verifyJWT` middleware is applied to POST and PUT endpoints
- The `isAdmin` middleware is additionally applied to the DELETE endpoint
- Validation is performed using the appropriate validators:
  - `createCategoryValidator` for POST
  - `updateCategoryValidator` for PUT
  - `deleteCategoryValidator` for DELETE

Order API

Base URL

```
/api/v1/orders
```

Authentication

All order endpoints require JWT authentication via the `Authorization` header or cookies.

Error Responses

All endpoints return standardized error responses:

```
{
  "statusCode": 400,
  "data": {},
  "message": "Error message",
  "success": false
}
```

Endpoints

Create Payment Intent

Creates a payment intent with Stripe for order processing.

URL: `/payment-intent`
Method: `POST`
Auth Required: Yes

Request Body:

```
{
  "addressId": "60d21b4667d0d8992e610c85"
}
```

Success Response:

```
{
  "statusCode": 200,
  "data": {
    "clientSecret": "pi_3NrTM1SIKMhYjLOX1hUxRxVo_secret_jQmrJWyxzZFOV5tGQv14m6i1E",
    "paymentIntentId": "pi_3NrTM1SIKMhYjLOX1hUxRxVo",
    "totalAmount": 249.97
  },
  "message": "Payment Intent created",
  "success": true
}
```


Notes:

- Validates that address belongs to the authenticated user
- Checks if cart is empty
- Verifies sufficient stock for all products
- Calculates total amount based on current product prices
- Uses Stripe to create a payment intent in INR currency

Create Order

Creates a new order after successful payment.

URL: `/`
Method: `POST`
Auth Required: Yes

Request Body:


```
{
  "paymentIntentId": "pi_3NrTM1SIKMhYjLOX1hUxRxVo",
  "addressId": "60d21b4667d0d8992e610c85"
}
```


Success Response:


```
{
  "statusCode": 201,
  "data": {
    "_id": "60d21b4667d0d8992e610c86",
    "userId": "60d21b4667d0d8992e610c87",
    "orderId": "ORD-a1b2c3d4",
    "products": [
      {
        "productId": "60d21b4667d0d8992e610c88",
        "quantity": 2,
        "priceAtPurchase": 99.99
      },
      {
        "productId": "60d21b4667d0d8992e610c89",
        "quantity": 1,
        "priceAtPurchase": 49.99
      }
    ],
    "paymentId": "pi_3NrTM1SIKMhYjLOX1hUxRxVo",
    "paymentStatus": "Completed",
    "deliveryAddress": "60d21b4667d0d8992e610c85",
    "totalAmount": 249.97,
    "orderStatus": "Processing",
    "createdAt": "2023-09-15T12:00:00.000Z",
    "updatedAt": "2023-09-15T12:00:00.000Z"
  },
  "message": "Order placed successfully",
  "success": true
}
```

Cart API Documentation

Overview

The Cart API allows users to manage their shopping carts by adding products, retrieving cart contents, removing specific items, and clearing the entire cart.

Base URL

```
/api/v1/cart
```

Authentication

All cart endpoints require authentication using JWT. Include the authentication token in the request header:

```
Authorization: Bearer YOUR_JWT_TOKEN
```

Endpoints

Add Product to Cart

Adds a product to the user's cart or increases the quantity if the product is already in the cart.

- **URL**: `/add`
- **Method**: `POST`
- **Auth Required**: Yes
- **Request Body**:

```
{
  "productId": "string",
  "quantity": "number" // Optional, defaults to 1
}
```

- **Success Response**:
  - **Code**: 201 Created (New cart item) or 200 OK (Updated existing item)
  - **Content Example**:

```
{
  "statusCode": 201,
  "data": {
    "_id": "cartItemId",
    "userId": "userId",
    "productId": "productId",
    "quantity": 1
  },
  "message": "Product added to cart"
}
```

- **Error Responses**:
  - **Code**: 404 Not Found
    - **Content**: `{ "statusCode": 404, "message": "Product not found." }`
  - **Code**: 400 Bad Request
    - **Content**: `{ "statusCode": 400, "message": "Insufficient stock" }`
- **Notes**:
  - If stock is insufficient, an email alert will be sent to the administrator.

Get Cart

Retrieves the user's current cart with product details and total amount.

- **URL**: `/`
- **Method**: `GET`
- **Auth Required**: Yes
- **Success Response**:
  - **Code**: 200 OK

- **Content Example**:

```
{
  "statusCode": 200,
  "data": {
    "cartItems": [
      {
        "_id": "cartItemId",
        "userId": "userId",
        "productId": {
          "_id": "productId",
          "name": "Product Name",
          "price": 99.99,
          "images": ["image1.jpg", "image2.jpg"],
          "countInStock": 10
        },
        "quantity": 2
      }
    ],
    "totalAmount": 199.98
  },
  "message": "Cart fetched"
}
```

Remove Item from Cart

Removes a specific item from the user's cart.

- **URL**: `/:cartItemId`
- **Method**: `DELETE`
- **Auth Required**: Yes
- **URL Parameters**: `cartItemId=[string]`
- **Success Response**:
  - **Code**: 200 OK
  - **Content Example**:

```
{
  "statusCode": 200,
  "data": null,
  "message": "Item removed from cart"
}
```

- **Error Response**:
  - **Code**: 404 Not Found
    - **Content**: `{ "statusCode": 404, "message": "Cart item not found." }`

Clear Cart

Removes all items from the user's cart.

- **URL**: `/`
- **Method**: `DELETE`
- **Auth Required**: Yes
- **Success Response**:
  - **Code**: 200 OK
  - **Content Example**:

```
{
  "statusCode": 200,
  "data": null,
  "message": "Cart cleared successfully"
}
```

Error Handling

The API uses standard HTTP status codes to indicate the success or failure of requests. Common error codes include:

- `400` - Bad Request (invalid input)
- `401` - Unauthorized (authentication failed)

- `404` - Not Found (resource doesn't exist)
- `500` - Internal Server Error

Error responses follow this format:

```
{
  "statusCode": 400,
  "message": "Error message"
}
```

Implementation Notes

Notes:

- Clears the access and refresh token cookies
- Verifies payment intent status with Stripe
- Attempts to confirm payment if not already successful
- Creates a unique order ID
- Updates product inventory (reduces stock)
- Adds order to user's order history
- Clears the user's cart
- Sends order confirmation email to the user
- Uses MongoDB transaction to ensure all operations succeed or fail together
- The cart system checks product availability before adding items.
- When a user attempts to add more products than are available in stock, an automatic email alert is sent to the admin.
- Cart items are linked to products with population of essential product details when fetching the cart.
- The API automatically calculates the total amount based on product prices and quantities.