Web Store API Documentation

Table of Contents

Overview

This is a RESTful API for a Web Store application that sells clothing products. The API supports product management, order processing, user authentication, advanced search capabilities, and comprehensive reporting features.

Technology Stack:

- Node.js with Express.js
- PostgreSQL Database
- JWT Authentication

- bcrypt for password hashing

Authentication

Most endpoints require authentication using JWT (JSON Web Tokens).

How to Authenticate:

1. Register or login to obtain a JWT token
2. Include the token in the Authorization header for subsequent requests

```
Authorization: Bearer <your-jwt-token>
```

User Roles

The system supports three user roles with different access levels:

Role	Description	Permissions
admin	Full system access	All operations including user management
advanced_user	Extended access	Product management, order viewing, report generation
simple_user	Basic access	Limited product management only

Base URL

```
http://localhost:3000/api
```

Authentication Endpoints

1. Register User

Endpoint: `POST /auth/register`

Description: Register a new user in the system.

Authentication: Not required

Request Body:

```
json
```

```
{
  "username": "johndoe",
  "email": "john@example.com",
  "password": "securepass123",
  "role": "simple_user",
  "first_name": "John",
  "last_name": "Doe"
}
```

Validation Rules:

- `username`: minimum 3 characters
- `email`: valid email format
- `password`: minimum 6 characters
- `role`: optional, one of [admin, advanced_user, simple_user], defaults to simple_user

Success Response (201):

```
json

{
  "message": "User registered successfully",
  "user": {
    "id": 1,
    "username": "johndoe",
    "email": "john@example.com",
    "role": "simple_user",
    "first_name": "John",
    "last_name": "Doe",
    "created_at": "2024-01-15T10:30:00.000Z"
  }
}
```

Error Response (400):

```
json

{
  "error": "User with this email or username already exists."
}
```

---

2. Login

Endpoint: `POST /auth/login`

Description: Authenticate a user and receive a JWT token.

Authentication: Not required

Request Body:

```
json

{
  "email": "john@example.com",
  "password": "securepass123"
}
```

Success Response (200):

```
json

{
  "message": "Login successful",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user": {
    "id": 1,
    "username": "johndoe",
    "email": "john@example.com",
    "role": "simple_user",
    "first_name": "John",
    "last_name": "Doe"
  }
}
```

Error Response (401):

```
json

{
  "error": "Invalid credentials"
}
```

---

3. Get Profile

Endpoint: `GET /auth/profile`

Description: Get the authenticated user's profile.

Authentication: Required

Success Response (200):

```json
{
  "user": {
    "id": 1,
    "username": "johndoe",
    "email": "john@example.com",
    "role": "simple_user",
    "first_name": "John",
    "last_name": "Doe",
    "is_active": true,
    "created_at": "2024-01-15T10:30:00.000Z",
    "last_login": "2024-01-15T14:20:00.000Z"
  }
}
```

---

4. Change Password

Endpoint: `POST /auth/change-password`

Description: Change the authenticated user's password.

Authentication: Required

Request Body:

```json
{
  "oldPassword": "securepass123",
  "newPassword": "newsecurepass456"
}
```

Validation Rules:

- `newPassword`: minimum 6 characters

Success Response (200):

```json
```

```
{
  "message": "Password changed successfully"
}
```

Error Response (401):

```
json

{
  "error": "Current password is incorrect"
}
```

Product Endpoints

1. Get All Products

Endpoint: `GET /products`

Description: Retrieve a paginated list of all active products.

Authentication: Not required

Query Parameters:

- `page` (optional, default: 1): Page number
- `limit` (optional, default: 20): Items per page

Example Request:

```
GET /api/products?page=1&limit=20
```

Success Response (200):

```
json
```

```
{
  "products": [
    {
      "id": 1,
      "name": "Nike Running Shoes",
      "description": "Comfortable running shoes",
      "price": "89.99",
      "discount_percentage": "10.00",
      "discounted_price": "80.99",
      "initial_quantity": 100,
      "gender_id": 1,
      "category_id": 4,
      "brand_id": 1,
      "size_id": 3,
      "color_id": 1,
      "is_active": true,
      "created_at": "2024-01-15T10:00:00.000Z",
      "updated_at": "2024-01-15T10:00:00.000Z",
      "gender_name": "Men",
      "category_name": "Shoes",
      "brand_name": "Nike",
      "size_name": "M",
      "color_name": "Red",
      "sold_quantity": "15",
      "current_quantity": "85"
    }
  ],
  "pagination": {
    "page": 1,
    "limit": 20,
    "total": 45,
    "totalPages": 3
  }
}
```


---

2. Get Product by ID

Endpoint: `GET /products/:id`

Description: Retrieve detailed information about a specific product.

Authentication: Not required

Example Request:

```
GET /api/products/1
```

Success Response (200):

```
json

{
  "product": {
    "id": 1,
    "name": "Nike Running Shoes",
    "description": "Comfortable running shoes",
    "price": "89.99",
    "discount_percentage": "10.00",
    "discounted_price": "80.99",
    "initial_quantity": 100,
    "gender_id": 1,
    "category_id": 4,
    "brand_id": 1,
    "size_id": 3,
    "color_id": 1,
    "is_active": true,
    "created_at": "2024-01-15T10:00:00.000Z",
    "updated_at": "2024-01-15T10:00:00.000Z",
    "gender_name": "Men",
    "category_name": "Shoes",
    "brand_name": "Nike",
    "size_name": "M",
    "color_name": "Red",
    "sold_quantity": "15",
    "current_quantity": "85"
  }
}
```

Error Response (404):

```
json

{
  "error": "Product not found"
}
```

---

3. Advanced Product Search

Endpoint: `GET /products/search`

Description: Search and filter products by multiple criteria.

Authentication: Not required

Query Parameters:

- `gender` (optional): Filter by gender (e.g., Men, Women, Children)
- `category` (optional): Filter by category name
- `brand` (optional): Filter by brand name
- `price_min` (optional): Minimum price
- `price_max` (optional): Maximum price
- `size` (optional): Filter by size
- `color` (optional): Filter by color
- `availability` (optional): Filter by stock status (in_stock, out_of_stock)
- `page` (optional, default: 1): Page number
- `limit` (optional, default: 20): Items per page

Example Request:

```
GET /api/products/search?
gender=men&category=jackets&price_min=50&price_max=200&brand=nike&size=L&color=black&availability=in_stock
```

Success Response (200):

```
json
```

```
{
  "products": [
    {
      "id": 5,
      "name": "Nike Winter Jacket",
      "description": "Warm winter jacket",
      "price": "149.99",
      "discount_percentage": "0.00",
      "discounted_price": "149.99",
      "gender_name": "Men",
      "category_name": "Jackets",
      "brand_name": "Nike",
      "size_name": "L",
      "color_name": "Black",
      "current_quantity": "30"
    }
  ],
  "filters": {
    "gender": "men",
    "category": "jackets",
    "brand": "nike",
    "price_min": "50",
    "price_max": "200",
    "size": "L",
    "color": "black",
    "availability": "in_stock"
  },
  "pagination": {
    "page": 1,
    "limit": 20,
    "total": 1,
    "totalPages": 1
  }
}
```


---

4. Get Product Quantity (Real-Time)

Endpoint: `GET /products/:id/quantity`

Description: Get the current available quantity of a product in real-time by calculating initial quantity minus sold quantity from confirmed orders.

Authentication: Not required

Example Request:

```
GET /api/products/1/quantity
```

Success Response (200):

```json
{
  "product_id": 1,
  "name": "Nike Running Shoes",
  "initial_quantity": 100,
  "sold_quantity": 15,
  "current_quantity": 85
}
```

Note: The current_quantity is calculated dynamically as:

```
current_quantity = initial_quantity - sold_quantity (from confirmed/processing/shipped/delivered orders)
```

---

5. Create Product

Endpoint: `POST /products`

Description: Create a new product.

Authentication: Required (admin, advanced_user, simple_user)

Request Body:

```json
{
  "name": "Adidas T-Shirt",
  "description": "Comfortable cotton t-shirt",
  "price": 29.99,
  "discount_percentage": 0,
  "initial_quantity": 50,
  "gender_id": 1,
  "category_id": 1,
  "brand_id": 2,
  "size_id": 3,
  "color_id": 5
}
```

Validation Rules:

- `name`: required
- `price`: required, must be >= 0
- `discount_percentage`: optional, must be between 0-100
- `initial_quantity`: required, must be >= 0

Success Response (201):

```json
{
  "message": "Product created successfully",
  "product": {
    "id": 10,
    "name": "Adidas T-Shirt",
    "description": "Comfortable cotton t-shirt",
    "price": "29.99",
    "discount_percentage": "0.00",
    "discounted_price": "29.99",
    "initial_quantity": 50,
    "gender_id": 1,
    "category_id": 1,
    "brand_id": 2,
    "size_id": 3,
    "color_id": 5,
    "is_active": true,
    "created_at": "2024-01-15T15:00:00.000Z",
    "updated_at": "2024-01-15T15:00:00.000Z"
  }
}
```

---

6. Update Product

Endpoint: `PUT /products/:id`

Description: Update an existing product.

Authentication: Required (admin, advanced_user, simple_user)

Request Body: (all fields optional)

```json
```

```
{
  "name": "Updated Product Name",
  "description": "Updated description",
  "price": 35.99,
  "discount_percentage": 15,
  "initial_quantity": 75,
  "is_active": true
}
```

Success Response (200):

json

```
{
  "message": "Product updated successfully",
  "product": {
    "id": 10,
    "name": "Updated Product Name",
    "price": "35.99",
    "discount_percentage": "15.00",
    "discounted_price": "30.59",
    "updated_at": "2024-01-15T16:00:00.000Z"
  }
}
```

---

7. Delete Product (Soft Delete)

Endpoint: `DELETE /products/:id`

Description: Soft delete a product by setting is_active to false.

Authentication: Required (admin only)

Success Response (200):

json

```
{
  "message": "Product deleted successfully"
}
```

---

8. Apply Discount

Endpoint: `PATCH /products/:id/discount`

Description: Apply a discount percentage to a product.

Authentication: Required (admin, advanced_user)

Request Body:

```
json

{
  "discount_percentage": 20
}
```

Validation Rules:

- `discount_percentage`: required, must be between 0-100

Success Response (200):

```
json

{
  "message": "Discount applied successfully",
  "product": {
    "id": 1,
    "name": "Nike Running Shoes",
    "price": "89.99",
    "discount_percentage": "20.00",
    "discounted_price": "71.99",
    "updated_at": "2024-01-15T16:30:00.000Z"
  }
}
```

Note: This endpoint also creates an entry in the discount_history table.

---

9. Update Stock

Endpoint: `PATCH /products/:id/stock`

Description: Update product stock quantity.

Authentication: Required (admin, advanced_user)

Request Body:

```
json
```

```
{
  "quantity": 50,
  "operation": "add"
}
```

Parameters:

- `quantity`: required, must be >= 0
- `operation`: optional, one of [set, add, subtract], default: set
  - `set`: Set stock to the exact quantity
  - `add`: Add quantity to current stock
  - `subtract`: Subtract quantity from current stock (cannot go below 0)

Success Response (200):

```
json

{
  "message": "Stock updated successfully",
  "product": {
    "id": 1,
    "name": "Nike Running Shoes",
    "initial_quantity": 150,
    "updated_at": "2024-01-15T17:00:00.000Z"
  }
}
```

Category Endpoints

1. Get All Categories

Endpoint: `GET /categories`

Description: Retrieve all categories.

Authentication: Not required

Success Response (200):

```
json
```

```
{
  "categorys": [
    {
      "id": 1,
      "name": "Shirts",
      "description": "Various types of shirts",
      "created_at": "2024-01-15T10:00:00.000Z",
      "updated_at": "2024-01-15T10:00:00.000Z"
    },
    {
      "id": 2,
      "name": "Pants",
      "description": "Trousers and jeans",
      "created_at": "2024-01-15T10:00:00.000Z",
      "updated_at": "2024-01-15T10:00:00.000Z"
    }
  ],
  "count": 2
}
```


---

2. Get Category by ID

Endpoint: `GET /categories/:id`

Description: Retrieve a specific category.

Authentication: Not required

Success Response (200):

json


```
{
  "category": {
    "id": 1,
    "name": "Shirts",
    "description": "Various types of shirts",
    "created_at": "2024-01-15T10:00:00.000Z",
    "updated_at": "2024-01-15T10:00:00.000Z"
  }
}
```


---

3. Create Category

Endpoint: `POST /categories`

Description: Create a new category.

Authentication: Required (admin, advanced_user)

Request Body:

```json
{
  "name": "Accessories",
  "description": "Various accessories"
}
```

Success Response (201):

```json
{
  "message": "category created successfully",
  "category": {
    "id": 6,
    "name": "Accessories",
    "description": "Various accessories",
    "created_at": "2024-01-15T10:00:00.000Z"
  }
}
```

---

4. Update Category

Endpoint: `PUT /categories/:id`

Description: Update a category.

Authentication: Required (admin, advanced_user)

Request Body:

```json
{
  "name": "Updated Category Name",
  "description": "Updated description"
}
```

Success Response (200):

```
json

{
  "message": "category updated successfully",
  "category": {
    "id": 6,
    "name": "Updated Category Name",
    "description": "Updated description",
    "updated_at": "2024-01-15T11:00:00.000Z"
  }
}
```

---

5. Delete Category

Endpoint: `DELETE /categories/:id`

Description: Delete a category.

Authentication: Required (admin only)

Success Response (200):

```
json

{
  "message": "category deleted successfully"
}
```

Error Response (400):

```
json

{
  "error": "Cannot delete category because it is referenced by other records"
}
```

Brand Endpoints

All brand endpoints follow the same pattern as categories.

Base Path: `/brands`

Endpoints:

- `GET /brands` - Get all brands
- `GET /brands/:id` - Get brand by ID
- `POST /brands` - Create brand (admin, advanced_user)
- `PUT /brands/:id` - Update brand (admin, advanced_user)
- `DELETE /brands/:id` - Delete brand (admin)

Request/Response Format: Same as categories

Size Endpoints

Base Path: `/sizes`

Endpoints:

- `GET /sizes` - Get all sizes
- `GET /sizes/:id` - Get size by ID
- `POST /sizes` - Create size (admin, advanced_user)
- `PUT /sizes/:id` - Update size (admin, advanced_user)
- `DELETE /sizes/:id` - Delete size (admin)

Create/Update Request Body:

```json
{
  "name": "XXL",
  "sort_order": 7
}
```

Color Endpoints

Base Path: `/colors`

Endpoints:

- `GET /colors` - Get all colors
- `GET /colors/:id` - Get color by ID
- `POST /colors` - Create color (admin, advanced_user)

- `PUT /colors/:id` - Update color (admin, advanced_user)
- `DELETE /colors/:id` - Delete color (admin)

Create/Update Request Body:

```json
{
  "name": "Navy Blue",
  "hex_code": "#000080"
}
```

Note: hex_code must match pattern: #XXXXXX (6 hex digits)

Gender Endpoints

Base Path: `/genders`

Endpoints:

- `GET /genders` - Get all genders
- `GET /genders/:id` - Get gender by ID
- `POST /genders` - Create gender (admin, advanced_user)
- `PUT /genders/:id` - Update gender (admin, advanced_user)
- `DELETE /genders/:id` - Delete gender (admin)

Default Values: Men, Women, Children, Unisex

Client Endpoints

1. Get All Clients

Endpoint: `GET /clients`

Description: Retrieve a paginated list of all clients.

Authentication: Required (admin, advanced_user)

Query Parameters:

- `page` (optional, default: 1): Page number
- `limit` (optional, default: 20): Items per page

Success Response (200):

json


```
{
  "clients": [
    {
      "id": 1,
      "first_name": "Jane",
      "last_name": "Smith",
      "email": "jane.smith@example.com",
      "phone": "+1234567890",
      "address": "123 Main St",
      "city": "New York",
      "country": "USA",
      "postal_code": "10001",
      "created_at": "2024-01-15T10:00:00.000Z",
      "updated_at": "2024-01-15T10:00:00.000Z"
    }
  ],
  "pagination": {
    "page": 1,
    "limit": 20,
    "total": 35,
    "totalPages": 2
  }
}
```


---

2. Get Client by ID

Endpoint: `GET /clients/:id`

Description: Retrieve a specific client's information.

Authentication: Required (admin, advanced_user)

Success Response (200):

json

```
{
  "client": {
    "id": 1,
    "first_name": "Jane",
    "last_name": "Smith",
    "email": "jane.smith@example.com",
    "phone": "+1234567890",
    "address": "123 Main St",
    "city": "New York",
    "country": "USA",
    "postal_code": "10001",
    "created_at": "2024-01-15T10:00:00.000Z",
    "updated_at": "2024-01-15T10:00:00.000Z"
  }
}
```


---

3. Get Client Orders

Endpoint: `GET /clients/:id/orders`

Description: Retrieve all orders for a specific client.

Authentication: Required (admin, advanced_user)

Success Response (200):

```
json
```

```
{
  "client_id": "1",
  "orders": [
    {
      "id": 5,
      "client_id": 1,
      "order_number": "ORD-1705320000000-1234",
      "status": "delivered",
      "total_amount": "159.98",
      "notes": "Leave at front door",
      "created_at": "2024-01-15T10:00:00.000Z",
      "updated_at": "2024-01-15T12:00:00.000Z",
      "confirmed_at": "2024-01-15T10:30:00.000Z",
      "shipped_at": "2024-01-15T11:00:00.000Z",
      "delivered_at": "2024-01-15T12:00:00.000Z",
      "items_count": "3"
    }
  ],
  "count": 1
}
```


---

4. Create Client

Endpoint: POST /clients

Description: Create a new client.

Authentication: Required (admin, advanced_user)

Request Body:

json


```
{
  "first_name": "Jane",
  "last_name": "Smith",
  "email": "jane.smith@example.com",
  "phone": "+1234567890",
  "address": "123 Main St",
  "city": "New York",
  "country": "USA",
  "postal_code": "10001"
}
```

Validation Rules:

- `first_name`: required
- `last_name`: required
- `email`: required, valid email format
- Other fields are optional

Success Response (201):

```json
{
  "message": "Client created successfully",
  "client": {
    "id": 2,
    "first_name": "Jane",
    "last_name": "Smith",
    "email": "jane.smith@example.com",
    "phone": "+1234567890",
    "address": "123 Main St",
    "city": "New York",
    "country": "USA",
    "postal_code": "10001",
    "created_at": "2024-01-15T10:00:00.000Z",
    "updated_at": "2024-01-15T10:00:00.000Z"
  }
}
```

Error Response (400):

```json
{
  "error": "Client with this email already exists"
}
```

---

5. Update Client

Endpoint: `PUT /clients/:id`

Description: Update an existing client.

Authentication: Required (admin, advanced_user)

Request Body: (all fields optional)

```json
{
  "first_name": "Jane",
  "last_name": "Smith",
  "email": "jane.newemail@example.com",
  "phone": "+1234567890",
  "address": "456 New St",
  "city": "Boston",
  "country": "USA",
  "postal_code": "02101"
}
```

Success Response (200):

```json
{
  "message": "Client updated successfully",
  "client": {
    "id": 1,
    "first_name": "Jane",
    "last_name": "Smith",
    "email": "jane.newemail@example.com",
    "updated_at": "2024-01-15T11:00:00.000Z"
  }
}
```

---

6. Delete Client

Endpoint: `DELETE /clients/:id`

Description: Delete a client.

Authentication: Required (admin only)

Success Response (200):

```json
{
  "message": "Client deleted successfully"
}
```

Error Response (400):

```
json

{
  "error": "Cannot delete client because they have associated orders"
}
```

Order Endpoints

1. Get All Orders

Endpoint: `GET /orders`

Description: Retrieve a paginated list of all orders.

Authentication: Required (admin, advanced_user)

Query Parameters:

- `page` (optional, default: 1): Page number
- `limit` (optional, default: 20): Items per page
- `status` (optional): Filter by order status

Example Request:

```
GET /api/orders?page=1&limit=20&status=pending
```

Success Response (200):

```
json
```

```
{
  "orders": [
    {
      "id": 1,
      "client_id": 1,
      "order_number": "ORD-1705320000000-1234",
      "status": "pending",
      "total_amount": "159.98",
      "notes": "Please deliver after 5 PM",
      "created_at": "2024-01-15T10:00:00.000Z",
      "updated_at": "2024-01-15T10:00:00.000Z",
      "confirmed_at": null,
      "shipped_at": null,
      "delivered_at": null,
      "first_name": "Jane",
      "last_name": "Smith",
      "email": "jane.smith@example.com",
      "items_count": "2"
    }
  ],
  "pagination": {
    "page": 1,
    "limit": 20,
    "total": 45,
    "totalPages": 3
  }
}
```


---

2. Get Order by ID

Endpoint: `GET /orders/:id`

Description: Retrieve detailed information about a specific order including all order items.

Authentication: Required (admin, advanced_user)

Success Response (200):

```
json
```

```
{
  "order": {
    "id": 1,
    "client_id": 1,
    "order_number": "ORD-1705320000000-1234",
    "status": "confirmed",
    "total_amount": "159.98",
    "notes": "Please deliver after 5 PM",
    "created_at": "2024-01-15T10:00:00.000Z",
    "updated_at": "2024-01-15T10:30:00.000Z",
    "confirmed_at": "2024-01-15T10:30:00.000Z",
    "shipped_at": null,
    "delivered_at": null,
    "first_name": "Jane",
    "last_name": "Smith",
    "email": "jane.smith@example.com",
    "phone": "+1234567890",
    "address": "123 Main St",
    "city": "New York",
    "country": "USA",
    "items": [
      {
        "id": 1,
        "order_id": 1,
        "product_id": 1,
        "product_name": "Nike Running Shoes",
        "quantity": 2,
        "unit_price": "79.99",
        "total_price": "159.98",
        "created_at": "2024-01-15T10:00:00.000Z",
        "current_product_name": "Nike Running Shoes",
        "product_description": "Comfortable running shoes"
      }
    ]
  }
}
```


---

3. Create Order

Endpoint: POST /orders

Description: Create a new order with multiple items.

Authentication: Required (admin, advanced_user)

Request Body:

json

```
{
  "client_id": 1,
  "items": [
    {
      "product_id": 1,
      "quantity": 2
    },
    {
      "product_id": 5,
      "quantity": 1
    }
  ],
  "notes": "Please deliver after 5 PM"
}
```

Validation Rules:

- `client_id`: required, must be a valid client ID
- `items`: required, array with at least 1 item
- `items[].product_id`: required, must be >= 1
- `items[].quantity`: required, must be >= 1
- `notes`: optional

Business Logic:

- Validates client exists
- Validates all products exist
- Checks sufficient stock for each product
- Calculates total amount using discounted prices
- Uses database transaction to ensure atomicity

Success Response (201):

json

```
{
  "message": "Order created successfully",
  "order": {
    "id": 10,
    "client_id": 1,
    "order_number": "ORD-1705320000000-5678",
    "status": "pending",
    "total_amount": "209.98",
    "notes": "Please deliver after 5 PM",
    "created_at": "2024-01-15T14:00:00.000Z",
    "first_name": "Jane",
    "last_name": "Smith",
    "email": "jane.smith@example.com",
    "items": [
      {
        "id": 15,
        "order_id": 10,
        "product_id": 1,
        "product_name": "Nike Running Shoes",
        "quantity": 2,
        "unit_price": "79.99",
        "total_price": "159.98"
      },
      {
        "id": 16,
        "order_id": 10,
        "product_id": 5,
        "product_name": "Nike Winter Jacket",
        "quantity": 1,
        "unit_price": "50.00",
        "total_price": "50.00"
      }
    ]
  }
}
```


Error Responses:

404 - Client Not Found:

```
json
```

```
{
  "error": "Client not found"
}
```

404 - Product Not Found:

```
json

{
  "error": "Product with ID 99 not found"
}
```

400 - Insufficient Stock:

```
json

{
  "error": "Insufficient stock for product \"Nike Running Shoes\". Available: 5, Requested: 10"
}
```

---

4. Update Order Status

Endpoint: `PATCH /orders/:id/status`

Description: Update the status of an order.

Authentication: Required (admin, advanced_user)

Request Body:

```
json

{
  "status": "confirmed"
}
```

Valid Status Values:

- `pending`
- `confirmed`
- `processing`
- `shipped`
- `delivered`

- `cancelled`

Success Response (200):

```json
{
  "message": "Order status updated successfully",
  "order": {
    "id": 1,
    "status": "confirmed",
    "confirmed_at": "2024-01-15T15:00:00.000Z",
    "updated_at": "2024-01-15T15:00:00.000Z"
  }
}
```

Note: Certain status updates automatically set timestamp fields:

- `confirmed` → sets `confirmed_at`
- `shipped` → sets `shipped_at`
- `delivered` → sets `delivered_at`

Error Response (400):

```json
{
  "error": "Invalid status",
  "validStatuses": ["pending", "confirmed", "processing", "shipped", "delivered", "cancelled"]
}
```

---

5. Cancel Order

Endpoint: `PATCH /orders/:id/cancel`

Description: Cancel an order (shortcut for setting status to cancelled).

Authentication: Required (admin, advanced_user)

Success Response (200):

```json
```

```
{
  "message": "Order cancelled successfully",
  "order": {
    "id": 1,
    "status": "cancelled",
    "updated_at": "2024-01-15T15:30:00.000Z"
  }
}
```


---

6. Get Order Statistics

Endpoint: GET /orders/statistics

Description: Get overall order statistics including counts by status and revenue.

Authentication: Required (admin, advanced_user)

Success Response (200):


```
json

{
  "statistics": {
    "total_orders": "150",
    "pending_orders": "20",
    "confirmed_orders": "30",
    "processing_orders": "25",
    "shipped_orders": "40",
    "delivered_orders": "30",
    "cancelled_orders": "5",
    "total_revenue": "25849.50",
    "average_order_value": "178.27"
  }
}
```


Note: Revenue calculations exclude cancelled orders.

Report Endpoints

1. Get Dashboard

Endpoint: GET /reports/dashboard

Description: Get comprehensive dashboard statistics including orders, products, and clients.

Authentication: Required (admin, advanced_user)

Success Response (200):

json


```
{
  "dashboard": {
    "overall": {
      "total_orders": "150",
      "total_revenue": "25849.50",
      "average_order_value": "178.27",
      "pending_orders": "20",
      "confirmed_orders": "30"
    },
    "today": {
      "orders_today": "5",
      "revenue_today": "489.95"
    },
    "this_month": {
      "orders_this_month": "45",
      "revenue_this_month": "7850.20"
    },
    "products": {
      "total_products": "80",
      "total_stock": "2500",
      "available_stock": "2150",
      "out_of_stock_count": "5"
    },
    "clients": {
      "total_clients": "120"
    }
  }
}
```


---

2. Get Daily Earnings

Endpoint: `GET /reports/earnings/daily`

Description: Get earnings for a specific date or today.

Authentication: Required (admin, advanced_user)

Query Parameters:

- `date` (optional): Date in format YYYY-MM-DD (defaults to today)

Example Request:

```
GET /api/reports/earnings/daily?date=2024-01-15
```

Success Response (200):

```
json

{
  "daily_earnings": {
    "date": "2024-01-15",
    "total_orders": "8",
    "total_earnings": "1250.50",
    "average_order_value": "156.31"
  }
}
```

Response for Date with No Orders:

```
json

{
  "date": "2024-01-20",
  "total_orders": 0,
  "total_earnings": 0,
  "average_order_value": 0
}
```

---

3. Get Monthly Earnings

Endpoint: `GET /reports/earnings/monthly`

Description: Get earnings for a specific month or current month.

Authentication: Required (admin, advanced_user)

Query Parameters:

- `year` (optional): Year (defaults to current year)
- `month` (optional): Month 1-12 (defaults to current month)

Example Request:

```
GET /api/reports/earnings/monthly?year=2024&month=1
```

Success Response (200):

```json
{
  "monthly_earnings": {
    "year": "2024",
    "month": "1",
    "total_orders": "45",
    "total_earnings": "7850.20",
    "average_order_value": "174.45"
  }
}
```

---

4. Get Earnings by Date Range

Endpoint: `GET /reports/earnings/range`

Description: Get earnings breakdown by day for a specific date range.

Authentication: Required (admin, advanced_user)

Query Parameters:

- `start_date` (required): Start date in format YYYY-MM-DD
- `end_date` (required): End date in format YYYY-MM-DD

Example Request:

```
GET /api/reports/earnings/range?start_date=2024-01-01&end_date=2024-01-31
```

Success Response (200):

```json
```

```
{
 "date_range": {
  "start_date": "2024-01-01",
  "end_date": "2024-01-31"
 },
 "summary": {
  "total_orders": "45",
  "total_earnings": "7850.20",
  "average_order_value": "174.45"
 },
 "daily_breakdown": [
  {
   "date": "2024-01-01",
   "orders_count": "3",
   "earnings": "450.50"
  },
  {
   "date": "2024-01-02",
   "orders_count": "5",
   "earnings": "825.00"
  }
 ]
}
```


Error Response (400):


```
json

{
 "error": "start_date and end_date are required"
}
```


---

5. Get Top-Selling Products

Endpoint: `GET /reports/products/top-selling`

Description: Get the top-selling products by quantity sold.

Authentication: Required (admin, advanced_user)

Query Parameters:

- `limit` (optional, default: 10): Number of products to return
- `period` (optional): Filter by time period

- `today`: Today only
- `week`: Last 7 days
- `month`: Last 30 days
- `year`: Last 365 days
- If not specified: All time

Example Request:

```
GET /api/reports/products/top-selling?limit=5&period=month
```

Success Response (200):

json


```
{
  "period": "month",
  "top_selling_products": [
    {
      "id": 1,
      "name": "Nike Running Shoes",
      "price": "89.99",
      "discounted_price": "80.99",
      "category_name": "Shoes",
      "brand_name": "Nike",
      "total_sold": "85",
      "total_revenue": "6884.15",
      "times_ordered": "42"
    },
    {
      "id": 5,
      "name": "Adidas T-Shirt",
      "price": "29.99",
      "discounted_price": "25.49",
      "category_name": "Shirts",
      "brand_name": "Adidas",
      "total_sold": "120",
      "total_revenue": "3058.80",
      "times_ordered": "60"
    }
  ]
}
```

6. Get Product Performance

Endpoint: `GET /reports/products/:product_id/performance`

Description: Get detailed performance metrics for a specific product.

Authentication: Required (admin, advanced_user)

Example Request:

```
GET /api/reports/products/1/performance
```

Success Response (200):

```json
```

```
{
  "product_performance": {
    "id": 1,
    "name": "Nike Running Shoes",
    "price": "89.99",
    "discounted_price": "80.99",
    "initial_quantity": 200,
    "sold_quantity": "85",
    "current_quantity": "115",
    "times_ordered": "42",
    "total_units_sold": "85",
    "total_revenue": "6884.15",
    "average_selling_price": "80.99"
  },
  "monthly_sales": [
    {
      "year": "2024",
      "month": "1",
      "units_sold": "45",
      "revenue": "3644.55"
    },
    {
      "year": "2023",
      "month": "12",
      "units_sold": "40",
      "revenue": "3239.60"
    }
  ]
}
```


Note: monthly_sales shows the last 12 months of data.

---

7. Get Sales by Category

Endpoint: `GET /reports/sales/by-category`

Description: Get sales breakdown by product category.

Authentication: Required (admin, advanced_user)

Query Parameters:

- `period` (optional): Filter by time period (today, week, month, year)

Example Request:

```
GET /api/reports/sales/by-category?period=month
```

Success Response (200):

json


```
{
  "period": "month",
  "sales_by_category": [
    {
      "id": 4,
      "category_name": "Shoes",
      "orders_count": "65",
      "units_sold": "120",
      "total_revenue": "9850.50"
    },
    {
      "id": 1,
      "category_name": "Shirts",
      "orders_count": "85",
      "units_sold": "150",
      "total_revenue": "4250.75"
    },
    {
      "id": 3,
      "category_name": "Jackets",
      "orders_count": "40",
      "units_sold": "55",
      "total_revenue": "7845.00"
    }
  ]
}
```


---

8. Get Sales by Brand

Endpoint: `GET /reports/sales/by-brand`

Description: Get sales breakdown by product brand.

Authentication: Required (admin, advanced_user)

Query Parameters:

- `period` (optional): Filter by time period (today, week, month, year)

Example Request:

```
GET /api/reports/sales/by-brand?period=week
```

Success Response (200):

json


```
{
  "period": "week",
  "sales_by_brand": [
    {
      "id": 1,
      "brand_name": "Nike",
      "orders_count": "45",
      "units_sold": "85",
      "total_revenue": "8450.50"
    },
    {
      "id": 2,
      "brand_name": "Adidas",
      "orders_count": "60",
      "units_sold": "120",
      "total_revenue": "5850.25"
    },
    {
      "id": 3,
      "brand_name": "Zara",
      "orders_count": "30",
      "units_sold": "55",
      "total_revenue": "3245.80"
    }
  ]
}
```

User Management Endpoints

1. Get All Users

Endpoint: `GET /users`

Description: Get a list of all users.

Authentication: Required (admin only)

Success Response (200):

```json
{
  "users": [
    {
      "id": 1,
      "username": "admin_user",
      "email": "admin@example.com",
      "role": "admin",
      "first_name": "Admin",
      "last_name": "User",
      "is_active": true,
      "created_at": "2024-01-01T10:00:00.000Z",
      "last_login": "2024-01-15T09:00:00.000Z"
    },
    {
      "id": 2,
      "username": "john_doe",
      "email": "john@example.com",
      "role": "advanced_user",
      "first_name": "John",
      "last_name": "Doe",
      "is_active": true,
      "created_at": "2024-01-10T14:00:00.000Z",
      "last_login": "2024-01-15T08:30:00.000Z"
    }
  ]
}
```

---

2. Get User by ID

Endpoint: `GET /users/:id`

Description: Get details of a specific user.

Authentication: Required (any authenticated user can view)

Success Response (200):

```json
```

```
{
  "user": {
    "id": 2,
    "username": "john_doe",
    "email": "john@example.com",
    "role": "advanced_user",
    "first_name": "John",
    "last_name": "Doe",
    "is_active": true,
    "created_at": "2024-01-10T14:00:00.000Z",
    "last_login": "2024-01-15T08:30:00.000Z"
  }
}
```


---

3. Get Users by Role

Endpoint: `GET /users/role/:role`

Description: Get all users with a specific role.

Authentication: Required (admin only)

Valid Roles: admin, advanced_user, simple_user

Example Request:

```
GET /api/users/role/advanced_user
```

Success Response (200):

```
json
```

```
{
  "role": "advanced_user",
  "users": [
    {
      "id": 2,
      "username": "john_doe",
      "email": "john@example.com",
      "role": "advanced_user",
      "first_name": "John",
      "last_name": "Doe",
      "is_active": true,
      "created_at": "2024-01-10T14:00:00.000Z",
      "last_login": "2024-01-15T08:30:00.000Z"
    }
  ],
  "count": 1
}
```


Error Response (400):


```
json

{
  "error": "Invalid role",
  "validRoles": ["admin", "advanced_user", "simple_user"]
}
```


---

4. Update User

Endpoint: `PUT /users/:id`

Description: Update user information.

Authentication: Required

- Users can update their own account
- Admins can update any account

Request Body: (all fields optional)


```
json
```

```
{
  "username": "new_username",
  "email": "newemail@example.com",
  "role": "advanced_user",
  "first_name": "John",
  "last_name": "Smith",
  "is_active": true
}
```

Notes:

- Non-admin users cannot change `role` or `is_active` fields
- Admins can change all fields for any user

Success Response (200):

```
json

{
  "message": "User updated successfully",
  "user": {
    "id": 2,
    "username": "new_username",
    "email": "newemail@example.com",
    "role": "advanced_user",
    "first_name": "John",
    "last_name": "Smith",
    "is_active": true,
    "updated_at": "2024-01-15T16:00:00.000Z"
  }
}
```

Error Response (403):

```
json

{
  "error": "You can only update your own account"
}
```

---

5. Update User Password

Endpoint: `PATCH /users/:id/password`

Description: Update a user's password.

Authentication: Required

- Users can update their own password
- Admins can update any user's password

Request Body:

```
json

{
  "newPassword": "newsecurepass456"
}
```

Validation Rules:

- `newPassword`: minimum 6 characters

Success Response (200):

```
json

{
  "message": "Password updated successfully"
}
```

---

6. Deactivate User

Endpoint: `PATCH /users/:id/deactivate`

Description: Deactivate a user account.

Authentication: Required (admin only)

Note: Users cannot deactivate their own account.

Success Response (200):

```
json

{
  "message": "User deactivated successfully"
}
```

Error Response (400):

```
json
```

```
{
  "error": "You cannot deactivate your own account"
}
```

---

7. Activate User

Endpoint: `PATCH /users/:id/activate`

Description: Activate a previously deactivated user account.

Authentication: Required (admin only)

Success Response (200):

```
json

{
  "message": "User activated successfully"
}
```

---

8. Delete User

Endpoint: `DELETE /users/:id`

Description: Permanently delete a user account.

Authentication: Required (admin only)

Note: Users cannot delete their own account.

Success Response (200):

```
json

{
  "message": "User deleted successfully"
}
```

Error Response (400):

```
json

{
  "error": "You cannot delete your own account"
}
```

Error Handling

Standard Error Response Format

All error responses follow this format:

```json
{
  "error": "Error message describing what went wrong",
  "details": "Additional technical details (in development mode)"
}
```

HTTP Status Codes

Code	Meaning	Description
200	OK	Request successful
201	Created	Resource created successfully
400	Bad Request	Invalid request data or validation error
401	Unauthorized	Missing or invalid authentication token
403	Forbidden	User doesn't have permission for this action
404	Not Found	Requested resource doesn't exist
500	Internal Server Error	Server error occurred

Common Error Scenarios

1. Validation Errors

Status Code: 400

```json
{
  "error": "Validation failed",
  "details": [
    {
      "msg": "Invalid email address",
      "param": "email",
      "location": "body"
    }
  ]
}
```

2. Authentication Errors

Status Code: 401

```json
{
  "error": "Access denied. No token provided."
}
```

```json
{
  "error": "Invalid or expired token."
}
```

3. Authorization Errors

Status Code: 403

```json
{
  "error": "Access denied. Insufficient permissions.",
  "requiredRoles": ["admin", "advanced_user"],
  "userRole": "simple_user"
}
```

4. Resource Not Found

Status Code: 404

```json
{
  "error": "Product not found"
}
```

```json
{
  "error": "Route not found",
  "path": "/api/invalid-endpoint"
}
```

5. Database Constraint Violations

Status Code: 400

```json
{
  "error": "Cannot delete category because it is referenced by other records"
}
```

```json
{
  "error": "User with this email or username already exists."
}
```

Setup and Installation

Prerequisites

- Node.js (v14 or higher)
- PostgreSQL (v12 or higher)
- npm or yarn

Installation Steps

1. Clone the repository

```bash
git clone <repository-url>
cd webstore-backend
```

2. Install dependencies

```bash
npm install
```

3. Configure environment variables

Create a `.env` file in the root directory:

```properties
```

```
# Server Configuration
PORT=3000
NODE_ENV=development

# Database Configuration
DB_HOST=localhost
DB_PORT=5432
DB_NAME=webstore_db
DB_USER=postgres
DB_PASSWORD=your_password

# JWT Configuration
JWT_SECRET=your_jwt_secret_key_here
JWT_EXPIRES_IN=24h

# CORS Configuration
CORS_ORIGIN=http://localhost:4200
```

4. Create the database

bash

```
psql -U postgres
CREATE DATABASE webstore_db;
\q
```

5. Run the database schema

bash

```
psql -U postgres -d webstore_db -f Schema.sql
```

6. Start the server

Development mode:

bash

```
npm run dev
```

Production mode:

bash

```
npm start
```

The server will start on `http://localhost:3000`

Testing the API

Using cURL

Register a user:

```bash
curl -X POST http://localhost:3000/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{
    "username": "testuser",
    "email": "test@example.com",
    "password": "password123",
    "first_name": "Test",
    "last_name": "User"
  }'
```

Login:

```bash
curl -X POST http://localhost:3000/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "email": "test@example.com",
    "password": "password123"
  }'
```

Get products (with authentication):

```bash
curl -X GET http://localhost:3000/api/products \
  -H "Authorization: Bearer YOUR_JWT_TOKEN"
```

Using Postman

1. Import the API endpoints into Postman
2. Create an environment variable for the base URL: `http://localhost:3000/api`

3. Create an environment variable for the JWT token after login
4. Use `{{baseUrl}}` and `{{token}}` in your requests

Key Features Summary

1. Advanced Product Search

- Filter by multiple criteria simultaneously
- Support for price ranges
- Real-time stock availability filtering

2. Real-Time Inventory Tracking

- Dynamic quantity calculation
- Automatic stock reduction on order confirmation
- Clear visibility of sold vs. available quantities

3. Role-Based Access Control

- Three distinct user roles
- Granular permissions per endpoint
- Secure JWT-based authentication

4. Comprehensive Reporting

- Daily, monthly, and date range earnings
- Top-selling products analysis
- Category and brand performance metrics
- Product-specific performance tracking

5. Order Management

- Multi-item order support
- Status tracking with timestamps
- Transaction-based order creation
- Stock validation before order placement

6. Discount Management

- Product-level discount application
- Automatic discounted price calculation

- Historical tracking of price changes

Database Schema Overview

Main Tables

- **users**: User accounts and authentication
- **clients**: Customer information
- **products**: Product catalog
- **orders**: Order headers
- **order_items**: Order line items
- **categories, brands, sizes, colors, genders**: Reference tables
- **discount_history**: Price change tracking

Key Views

- **product_quantities**: Real-time stock calculation view

Relationships

- Products reference categories, brands, sizes, colors, genders
- Orders reference clients
- Order items reference orders and products
- Discount history references products and users

Notes for Frontend Integration

1. **Authentication Flow:**
   - Store JWT token after successful login
   - Include token in Authorization header for all protected requests
   - Handle token expiration (24-hour validity)
2. **Pagination:**
   - Use `page` and `limit` query parameters
   - Response includes pagination metadata
3. **Error Handling:**
   - Always check HTTP status codes

- Display user-friendly error messages
- Handle validation errors with field-specific feedback

4. **Real-Time Updates:**
   - Product quantities update automatically based on order status
   - Refresh product list after creating orders

5. **Role-Based UI:**
   - Hide/disable features based on user role
   - Admin: Full access
   - Advanced User: Products, orders, reports
   - Simple User: Limited product management

Contact and Support

For questions or issues, please contact the development team or open an issue in the repository.

API Version: 1.0.0
Last Updated: January 2024