ProductAPI Endpoint Documentation

February 8, 2026

1 Original Code

The following is the original `ProductAPI` endpoint.

```
/**
 * Product API endpoints
 */
const productRouter = express.Router();

// Get all products with filtering and pagination
productRouter.get('/', async (req, res) => {
  try {
    const {
      category,
      minPrice,
      maxPrice,
      sort = 'createdAt',
      order = 'desc',
      page = 1,
      limit = 20,
      inStock
    } = req.query;

    // Build filter
    const filter = {};

    if (category) {
      filter.category = category;
    }

    if (minPrice !== undefined || maxPrice !== undefined) {
      filter.price = {};
      if (minPrice !== undefined) filter.price.$gte = parseFloat(minPrice);
      if (maxPrice !== undefined) filter.price.$lte = parseFloat(maxPrice);
    }

    if (inStock === 'true') {
      filter.stockQuantity = { $gt: 0 };
    }

    // Calculate pagination
    const skip = (parseInt(page) - 1) * parseInt(limit);

    // Determine sort order
    const sortOptions = {};
    sortOptions[sort] = order === 'asc' ? 1 : -1;

    // Execute query
    const products = await ProductModel.find(filter)
      .sort(sortOptions)
```

```
      .skip(skip)
      .limit(parseInt(limit));

    // Get total count for pagination
    const totalProducts = await ProductModel.countDocuments(filter);

    return res.status(200).json({
      products,
      pagination: {
        total: totalProducts,
        page: parseInt(page),
        limit: parseInt(limit),
        pages: Math.ceil(totalProducts / parseInt(limit))
      }
    });
  } catch (error) {
    console.error('Error fetching products:', error);
    return res.status(500).json({
      error: 'Server error',
      message: 'Failed to fetch products'
    });
  }
});

// Get product by ID
productRouter.get('/:productId', async (req, res) => {
  try {
    const { productId } = req.params;

    const product = await ProductModel.findById(productId);

    if (!product) {
      return res.status(404).json({
        error: 'Not found',
        message: 'Product not found'
      });
    }

    return res.status(200).json(product);
  } catch (error) {
    console.error('Error fetching product:', error);

    // Check if error is invalid ObjectId format
    if (error.name === 'CastError') {
      return res.status(400).json({
        error: 'Invalid ID',
        message: 'Invalid product ID format'
      });
    }

    return res.status(500).json({
      error: 'Server error',
      message: 'Failed to fetch product'
    });
  }
});

module.exports = productRouter;
```

2 Product API Documentation

This documentation covers the endpoints for managing and retrieving product information.

2.1 1. List Products API

Get a list of products with flexible filtering, sorting.

Endpoint: `GET /api/products`

Authentication: No authentication required for this endpoint.

Query Parameters

Parameter	Type	Required	Description
category	String	No	Filter products by category
minPrice	Number	No	Filter products with price $\geq$ value
maxPrice	Number	No	Filter products with price $\leq$ value
sort	String	No	Field to sort by (default: 'createdAt')
order	String	No	Sort order: 'asc' or 'desc' (default: 'desc')
page	Number	No	Page number for pagination (default: 1)
limit	Number	No	Number of items per page (default: 20)
inStock	Boolean	No	When 'true', only show products with stock $>$ 0

Table 1: List Products Query Parameters

Response

Success Response (Code: 200 OK)

```
{
  "products": [
    {
      "_id": "61fa9bcf5c130b2e6d675432",
      "name": "Wireless Headphones",
      "description": "High-quality wireless headphones with noise cancellation"
          ,
      "price": 89.99,
      "category": "electronics",
      "stockQuantity": 45,
      "createdAt": "2023-02-01T15:32:47Z",
      "updatedAt": "2023-03-15T09:21:08Z"
    }
  ],
  "pagination": {
    "total": 42,
    "page": 1,
    "limit": 20,
    "pages": 3
  }
}
```

Error Response (Code: 500 Internal Server Error)

```
{
  "error": "Server error",
  "message": "Failed to fetch products"
}
```

2.2 2. Get Product by ID

Retrieve detailed information for a specific product using its unique identifier.

Endpoint: `GET /api/products/:productId`

Authentication: No authentication required for this endpoint.

Path Parameters

Parameter	Type	Required	Description
productId	String	Yes	The unique MongoDB ObjectId of the product

Table 2: Get Product Path Parameters

Response

Success Response (Code: 200 OK)

```
{
  "_id": "61fa9bcf5c130b2e6d675432",
  "name": "Wireless Headphones",
  "description": "High-quality wireless headphones with noise cancellation",
  "price": 89.99,
  "category": "electronics",
  "stockQuantity": 45,
  "createdAt": "2023-02-01T15:32:47Z",
  "updatedAt": "2023-03-15T09:21:08Z"
}
```

Error Responses

Code: 400 Bad Request (Occurs when the productId provided is not a valid MongoDB ObjectId format).

```
{
  "error": "Invalid ID",
  "message": "Invalid product ID format"
}
```

Code: 404 Not Found (Occurs when the ID is valid but no product exists with that ID).

```
{
  "error": "Not found",
  "message": "Product not found"
}
```

2.3 Examples

Example 1: Filtering by Category

Request: `GET /api/products?category=electronics&limit=1`

```
{
  "products": [
    {
      "_id": "61fa9bcf5c130b2e6d675432",
      "name": "Wireless Headphones",
      "price": 89.99,
      "category": "electronics",
```

```
      "stockQuantity": 45
    }
  ],
  "pagination": { "total": 12, "page": 1, "limit": 1, "pages": 12 }
}
```

Example 2: Price Range and Sorting
Request: `GET /api/products?minPrice=20&maxPrice=100&sort=price&order=asc`

```
{
  "products": [
    {
      "_id": "61fa9bcf5c130b2e6d675439",
      "name": "T-Shirt",
      "price": 24.99,
      "category": "clothing"
    }
  ],
  "pagination": { "total": 18, "page": 1, "limit": 20, "pages": 1 }
}
```

2.4 Special Considerations

- **Data Consistency:** The API uses MongoDB. IDs must be 24-character hex strings.
- **Pagination Logic:** Total pages are calculated as $\lceil TotalProducts/Limit \rceil$.
- **Rate Limiting:** Currently, there is no software-level rate limiting implemented in the provided code; however, standard production practices recommend a limit of 100 requests per minute per IP.
- **Stock Filter:** The `inStock` filter only triggers when the string 'true' is passed, filtering for `stockQuantity > 0`.

3 Swagger Documentation

The following is the `ProductAPI Endpoint` documentation this was written using prompt 2.

```
Swagger Documentation
openapi: 3.0.0
info:
  title: Product API
  description: |
    API documentation for managing and retrieving product information.

    **Special Considerations:**
    * **Data Consistency:** The API uses MongoDB. IDs must be 24-character hex
        strings.
    * **Pagination Logic:** Total pages are calculated as `ceil(Total Products
        / Limit)`.
    * **Rate Limiting:** Recommended limit of 100 requests per minute per IP (
        not currently enforced in software).
  version: 1.0.0

servers:
  - url: http://localhost:3000
    description: Local Development Server

paths:
  /api/products:
```

```yaml
    get:
      summary: List Products
      description: Get a list of products with flexible filtering, sorting, and
          pagination.
      operationId: getProducts
      parameters:
        - in: query
          name: category
          schema:
            type: string
          description: Filter products by category
        - in: query
          name: minPrice
          schema:
            type: number
          description: Filter products with price     value
        - in: query
          name: maxPrice
          schema:
            type: number
          description: Filter products with price     value
        - in: query
          name: sort
          schema:
            type: string
            default: createdAt
          description: Field to sort by
        - in: query
          name: order
          schema:
            type: string
            enum: [asc, desc]
            default: desc
          description: Sort order
        - in: query
          name: page
          schema:
            type: integer
            default: 1
          description: Page number for pagination
        - in: query
          name: limit
          schema:
            type: integer
            default: 20
          description: Number of items per page
        - in: query
          name: inStock
          schema:
            type: boolean
          description: When 'true', only show products with stock > 0
      responses:
        '200':
          description: Successful response with list of products and pagination
              info
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ProductListResponse'
              example:
                products:
                  - _id: "61fa9bcf5c130b2e6d675432"
                    name: "Wireless Headphones"
```

```
                    description: "High-quality wireless headphones with noise
                        cancellation"
                    price: 89.99
                    category: "electronics"
                    stockQuantity: 45
                    createdAt: "2023-02-01T15:32:47Z"
                    updatedAt: "2023-03-15T09:21:08Z"
                pagination:
                  total: 42
                  page: 1
                  limit: 20
                  pages: 3
      '500':
        description: Internal Server Error
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ErrorResponse'
            example:
              error: "Server error"
              message: "Failed to fetch products"

/api/products/{productId}:
  get:
    summary: Get Product by ID
    description: Retrieve detailed information for a specific product using
        its unique identifier.
    operationId: getProductById
    parameters:
      - in: path
        name: productId
        schema:
          type: string
        required: true
        description: The unique MongoDB ObjectId of the product
    responses:
      '200':
        description: Product details found
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Product'
            example:
              _id: "61fa9bcf5c130b2e6d675432"
              name: "Wireless Headphones"
              description: "High-quality wireless headphones with noise
                  cancellation"
              price: 89.99
              category: "electronics"
              stockQuantity: 45
              createdAt: "2023-02-01T15:32:47Z"
              updatedAt: "2023-03-15T09:21:08Z"
      '400':
        description: Bad Request - Invalid ID format
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ErrorResponse'
            example:
              error: "Invalid ID"
              message: "Invalid product ID format"
      '404':
        description: Product Not Found
```

```yaml
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'
              example:
                error: "Not found"
                message: "Product not found"

components:
  schemas:
    Product:
      type: object
      properties:
        _id:
          type: string
          description: MongoDB ObjectId
          example: "61fa9bcf5c130b2e6d675432"
        name:
          type: string
          example: "Wireless Headphones"
        description:
          type: string
          example: "High-quality wireless headphones with noise cancellation"
        price:
          type: number
          format: float
          example: 89.99
        category:
          type: string
          example: "electronics"
        stockQuantity:
          type: integer
          example: 45
        createdAt:
          type: string
          format: date-time
          example: "2023-02-01T15:32:47Z"
        updatedAt:
          type: string
          format: date-time
          example: "2023-03-15T09:21:08Z"

    Pagination:
      type: object
      properties:
        total:
          type: integer
          description: Total number of products found
        page:
          type: integer
          description: Current page number
        limit:
          type: integer
          description: Items per page
        pages:
          type: integer
          description: Total number of pages

    ProductListResponse:
      type: object
      properties:
        products:
          type: array
```

```
          items:
            $ref: '#/components/schemas/Product'
        pagination:
          $ref: '#/components/schemas/Pagination'

    ErrorResponse:
      type: object
      properties:
        error:
          type: string
        message:
          type: string
```

4 Product API Developer Guide

Introduction

Welcome to the Product API integration guide. This document is designed to help you integrate product management features into your application. Our API is built on REST principles, uses JSON for data exchange, and is backed by a MongoDB database.

4.1 1. Getting Started

Base URL

All API requests should be made to the following base URL:

`https://api.yourdomain.com`

(Replace with your actual environment URL)

Authentication

Currently, the Product API is public. No API keys or OAuth tokens are required to access the endpoints documented below.

Request Headers

While authentication is not required, we recommend including the standard JSON content type header for clarity:

```
Content-Type: application/json
```

Listing 1: Header Example

Data Formats

- **Requests:** URL Query parameters for GET requests.
- **Responses:** All responses are returned in JSON format.
- **Timestamps:** All dates (`createdAt`, `updatedAt`) are returned in ISO 8601 format (UTC).

4.2 2. Common Operations

4.2.1 A. List and Filter Products

Endpoint: `GET /api/products`

This is your primary tool for browsing the catalog. It supports comprehensive filtering, sorting, and pagination.

Basic Request:

```
curl "https://api.yourdomain.com/api/products"
```

Advanced Filtering: To build a catalog view, you will often need to combine multiple filters.

Scenario: A user wants to see "Electronics" between $50 and $200, sorted by price (low to high), showing only items currently in stock.

Request:

```
GET /api/products?category=electronics&minPrice=50&maxPrice=200&inStock=true&
    sort=price&order=asc
```

Key Parameter Logic:

- **Pagination:** Use `page` and `limit` to handle large datasets.
  - Formula: The API skips $(page - 1) \times limit$ items.
  - Note: The default limit is 20 items.
- **Stock Filter:** The `inStock` parameter is a string check. You must send `inStock=true`. Sending `inStock=1` or boolean `true` (depending on your HTTP client) might not trigger the filter correctly.

4.2.2 B. Get Single Product Details

Endpoint: `GET /api/products/:productId`

Use this to load the "Product Detail Page" for a specific item.

Request:

```
GET /api/products/61fa9bcf5c130b2e6d675432
```

Response Snapshot:

```
{
  "_id": "61fa9bcf5c130b2e6d675432",
  "name": "Wireless Headphones",
  "price": 89.99,
  "stockQuantity": 45,
  ...
}
```

4.3 3. Error Handling & Troubleshooting

The API uses standard HTTP status codes to indicate the success or failure of a request.

Common Troubleshooting Scenarios

- **Issue:** "I'm sending `inStock=true` but seeing out-of-stock items."
  - *Check:* Ensure you are passing the literal string 'true'.
  - *Verify:* Check the `stockQuantity` field in the response. If it is 0, the filter might be ignored due to a typo in the parameter name (e.g., `instock` vs `inStock`).

Code	Meaning	Common Cause	Solution
200	OK	Request was successful.	N/A
400	Bad Request	Invalid ID Format.	Ensure `productId` is a 24-character hex string.
404	Not Found	ID is valid format, but does not exist.	Check if the product was deleted or the ID is correct.
500	Internal Error	Server-side issue or DB connection failure.	Retry the request later. If persistent, contact support.

Table 3: HTTP Status Codes

- **Issue:** "I get a 400 error when requesting a product."
  - *Check:* The ID provided must be a valid MongoDB ObjectId.
  - *Example:* `123` is invalid. `61fa9bcf5c130b2e6d675432` is valid.

4.4 4. Integration Best Practices

Validate IDs Client-Side

Before sending a request to `GET /api/products/:id`, validate that the ID is a 24-character hexadecimal string. This prevents unnecessary network calls that are guaranteed to result in a 400 Bad Request.

```
// Example JavaScript Regex for MongoDB ID
const isValidObjectId = (id) => /^[0-9a-fA-F]{24}$/.test(id);
```

Listing 2: ID Validation Regex

Respect Rate Limits

While there is no strict software-level blocking currently enabled, we monitor for abusive traffic.

- **Guideline:** Limit your requests to 100 requests per minute per IP.
- **Implementation:** If you are writing a script to scrape or sync products, add a small delay (sleep) between requests.

Handle Pagination Gracefully

The `pagination` object in the response is your source of truth.

- Always check `pagination.pages` (total pages) before requesting the next page.
- If `pagination.page >= pagination.pages`, stop requesting.

Type Safety

Since query parameters are parsed as strings:

- Prices (`minPrice`, `maxPrice`) are parsed as floats.
- Pagination (`page`, `limit`) are parsed as integers.
- Ensure your client application sanitizes these inputs to avoid injection of non-numeric characters.