# ShoppyGlobe RESTful API - Project Documentation

---

### Overview

This is a backend project built using Node.js, Express.js, and MongoDB (Mongoose) for an e-commerce application named **ShoppyGlobe**. It includes user authentication (JWT-based), product management (single and bulk), and a cart system with full CRUD operations.

---

## Github Link

https://github.com/SravanGunaganti/ShoppyGlobe_Backend.git

---

## Project Setup Instructions

- Clone the Repository

  git clone https://github.com/SravanGunaganti/ShoppyGlobe_Backend.git

  cd ShoppyGlobe_Backend

- Install Dependencies

  `npm install`

- Ensure MongoDB Is Running

  Before starting the server, make sure MongoDB is running locally or that your MongoDB Atlas cluster is accessible.

- **Create .env File:**

  PORT=5050 or yours

  MONGO_URI=mongodb://127.0.0.1:27017/shoppyglobe or yours local or atlas mongouri

  JWT_SECRET=your_jwt_secret

- Start the server

  npm start

---

## Technologies Used

- Node.js
- Express.js
- MongoDB + Mongoose
- JWT for authentication
- Thunder Client for testing

---

## Project Structure and Descriptions

```
└── 📁project
    └── 📁controllers
        ├── cart.controller.js
        ├── product.controller.js
        ├── user.controller.js
    └── 📁middlewares
        ├── errorHandlers.js
        ├── userValidation.js
        ├── validateBulkProducts.js
        ├── validateCartItem.js
        ├── validateObjectId.js
        ├── validateProduct.js
        ├── verifyToken.js
    └── 📁models
        ├── Cart.model.js
        ├── Product.model.js
        ├── User.model.js
    └── 📁routes
        ├── cart.routes.js
        ├── product.routes.js
        ├── routes.js
        ├── user.routes.js
    └── 📁utils
        ├── generateToken.js
        ├── sendErrorResponse.js
        ├── sendSuccessResponse.js
        ├── validators.js
    ├── .env
    ├── index.js
    ├── package-lock.json
    ├── package.json
    ├── products.json
    └── Screenshots_Submission.pdf
```

### 1. controllers/

Houses the logic for handling API requests and responses.

- **user.controller.js**: Handles registration and login operations.
- **product.controller.js**: Handles product creation, bulk insert, update, retrieval, and deletion.
- **cart.controller.js**: Manages cart operations like add, update, delete, and retrieve cart items.

### 2. middlewares/

Middleware functions to handle validation, authentication, and error management.

- **errorHandlers.js**: Global error handling and fallback route.
- **userValidation.js**: Validates user registration and login data.
- **validateBulkProducts.js**: Validates incoming bulk product data.

- **validateCartItem.js**: Ensures cart item request has valid productId and quantity.
- **validateObjectId.js**: Middleware to check valid MongoDB ObjectId in route params.
- **validateProduct.js**: Validates single product creation or update.
- **verifyToken.js**: Middleware to verify JWT token and decode user info.

**3. models/**

Defines Mongoose schemas for the database collections.

- **User.model.js**: Schema for storing users with validation for name, email, and hashed password.
- **Product.model.js**: Schema with validation for name, description, price (can be decimals), and stock.
- **Cart.model.js**: Stores user cart details including an array of product references and quantities.

**4. routes/**

Defines and organizes RESTful API routes.

- **user.routes.js**: Routes for user registration and login.
- **product.routes.js**: Routes to handle product operations.
- **cart.routes.js**: Routes for all cart-related operations.

**5. utils/**

Common utility functions shared across the project.

- **generateToken.js**: Creates JWT token for authenticated users.
- **sendErrorResponse.js**: Formats consistent error responses.
- **sendSuccessResponse.js**: Formats successful response messages, optionally includes a token.
- **validators.js**: Utility functions for object and type checks (e.g., `isValidObject`, `isNumber`).

---

## Server Configuration

**.env File:**

Stores sensitive and environment-specific variables like:

PORT=5050

MONGO_URI=mongodb://127.0.0.1:27017/shoppyglobe

JWT_SECRET=your_jwt_secret

**index.js:**

- Connects to MongoDB
- Applies middlewares and Mounts all API routes

- Starts the server

---

## API Endpoints & Description

- **User Routes**

  `POST /api/register`

  Registers a new user.

  `POST /api/login`

  Logs in a user and returns a JWT token.
- **Product Routes**

  `POST /api/products/bulk`

  Insert multiple products (protected).

  `POST /api/product`

  Create a single product (protected).

  `GET /api/products`

  Retrieve all products.

  `GET /api/products/:id`

  Get a product by ID.

  `PUT /api/products/:id`

  Update product by ID (protected).

  `DELETE /api/products/:id`

  Delete product by ID (protected).
- **Cart Routes**

  `POST /api/cart`

  Add product to cart (or increment if exists).

  `GET /api/cart`

  Fetch logged-in user's cart.

  `PUT /api/cart`

  Update quantity of product in cart.

  `DELETE /api/cart/:productId`

  Remove product from cart.

  `DELETE /api/cart`

  Clear the cart completely.

---

## Key Features

1. **Authentication**:
   - User registration and login

    - JWT token generation and verification
    - **Protected Routes** require `Authorization: JWT <token>` header.
    - Middleware `verifyToken` verifies and attaches user info to `req.user`.
    -
2. **Product APIs**:
    - Single and bulk product creation
    - Product validation and error handling
    - Get, update, and delete by ID
3. **Cart APIs**:
    - Add product (or increment if it exists)
    - Update product quantity
    - Remove a product
    - Clear entire cart
    - Auto-removal of product if quantity becomes 0
4. **Middleware Validations**:
    - Field-level validation for products, users, cart
    - ObjectId checks
    - Proper error messages using `sendErrorResponse`

---

## Testing

ThunderClient was used to test all API endpoints including:

- Positive test cases with valid inputs
- Negative test cases with missing/invalid fields, invalid routes, and token absenceAll results were captured in screenshots and submitted under the PDF file `Screenshots_submission.pdf`.