

## Documentation : ShoppY\_Globe

### 1. Project Overview

- Project Name: ShoppYGlobe E-commerce Backend
- Objective: Build a backend API for an e-commerce application using Node.js, Express.js, and MongoDB.
- Features:
  1. Product management (fetch products, fetch single product details).
  2. Shopping cart management (add, update, delete items).
  3. User authentication and authorization using JWT.
  4. Error handling and input validation.

### 2. Technologies Used

- Backend: Node.js, Express.js
- Database: MongoDB
- Authentication: JSON Web Tokens (JWT)
- Testing: ThunderClient (or Postman)
- Other Libraries:
  1. mongoose for MongoDB object modeling.
  2. bcryptjs for password hashing.
  3. dotenv for environment variable management.
  4. express-validator for input validation.

### 3. Setup Instructions

- **Prerequisites:**
  1. Node.js and npm installed.
  2. MongoDB installed or a MongoDB Atlas connection string.
  3. ThunderClient (or Postman) for API testing.
- **Steps to Run the Project**
  1. Clone the Repository:
    - `git clone <repository-link>`
    - `cd shoppyglobe-backend`
  2. Install Dependencies:
    - `npm install`
  3. Set Up Environment Variables:
    - `PORT=3000`
    - `MONGODB_URI=<your-mongodb-connection-string>`
    - `JWT_SECRET_KEY=<your-jwt-secret-key>`
  4. Run the Server:
    - `npm start`

## 5. Access the API:

- The API will be running at `http://localhost:${PORT.number}`.

## 4. API Endpoints

- Authentication

- POST /register:

- Register a new user.
- Request Body:

```
{
  "email": "k9am@gmail.com",
  "password": "password123"
}
```

- Response:

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

- POST /login:

- Authenticate a **user** and return a JWT token.
- Request Body:

```
{
  "email": "user@example.com",
  "password": "password123"
}
```

- Response:

```
{
  "token": "<jwt-token>"
}
```

- Products

- GET /products:

- Fetch a list of all products.
- Response:

```
{
  "_id": "64f1b2c7e4b0f5a3d4f5e6a7",
  "name": "Laptop",
  "price": 1200,
  "description": "A high-performance Laptop",
  "stockQuantity": 10
}
```

- GET /products/:id :
  - Fetch details of a single product by its ID.
  - Response:

```
{
  "_id": "64f1b2c7e4b0f5a3d4f5e6a7",
  "name": "Laptop",
  "price": 1200,
  "description": "A High-performance Laptop",
  "stockQuantity": 10
}
```

- Cart

- POST /cart:
  - Add a product to the shopping cart (protected route).
  - Request Body:

```
{
  "productId": "64f1b2c7e4b0f5a3d4f5e6a7",
  "quantity": 18
}
```

- Response

```
{
  "message": "Product added to cart"
}
```

- PUT /cart/:id :
  - Update the quantity of a product in the cart (protected route).
  - Request Body:

```
{
  "quantity": 3
}
```

- Response

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

- DELETE /cart/:id :
  - Remove a product from the cart (protected route).
  - Response:

```
{
  "message": "Product removed from cart"
}
```

## 5. Database Schema

- User Collection:

```
{
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true }
}
```

- Products Collection:

```
{
  name: { type: String, required: true },
  price: { type: Number, required: true },
  description: { type: String, required: true },
  stockQuantity: { type: Number, required: true }
}
```

- Cart Collection:

```
{
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  productId: { type: mongoose.Schema.Types.ObjectId, ref: 'Product', required: true },
  quantity: { type: Number, required: true }
}
```

## 6. Error Handling

- 400 Bad Request: Invalid input data.
- 401 Unauthorized: Missing or invalid JWT token.
- 404 Not Found: Resource not found (e.g., product or cart item not found).
- 500 Internal Server Error: Server-side errors.

## 7. Screenshots

- Include screenshots of: **ref** to ScreenShots Folder.
  1. MongoDB collections (Products, Cart, Users).
  2. Thunder Client/Postman API testing results for all endpoints.

## 8. Testing with Thunder Client

- Provide screenshots of all API requests and responses tested using Thunder Client.
- Example:
  1. Register a user.
  2. Login and get a JWT token.
  3. Fetch products.
  4. Add, update, and delete items from the cart.

## 9. GitHub Repository

- link : [https://github.com/sourabh-969/Shoppy\\_Globe-Backend.git](https://github.com/sourabh-969/Shoppy_Globe-Backend.git)

- the repository contains:

```
shoppyglobe_backend-main/
├── .env
├── .gitignore
├── config/
│   └── db.js
├── controllers/
│   ├── authController.js
│   ├── cartController.js
│   └── productController.js
├── middlewares/
│   └── authMiddleware.js
├── models/
│   ├── cartModel.js
│   ├── productModel.js
│   └── userModel.js
├── package-lock.json
├── package.json
├── Readme.md
├── routes/
│   ├── authRoutes.js
│   ├── cartRoutes.js
│   └── productRoutes.js
├── ScreenShots/
│   ├── 1.Db Connect.png
│   ├── 2.1.User Register .png
│   ├── 2.2.MongoDb Auth.png
│   ├── 3.User Login.png
│   ├── 4.ADD_Product.png
│   ├── 5.1.GET _All-Products.png
│   ├── 5.2.MongoDb products.png
│   ├── 6.GET_Product-by_Id.png
│   ├── 7.1.POST_Cart-item.png
│   ├── 7.2.POST_Cart-item_Auth-Token.png
│   ├── 7.3.MongoDb POST_Cart-item.png
│   ├── 8.1.PUT_Cart-item.png
│   ├── 8.2.PUT_Cart-item_Auth-Token.png
│   ├── 8.3.MongoDb PUT_Cart-item.png
│   └── 9.DELETE_Cart-item.png
└── server.js
```

#### 10. Comments and Code Quality

- the code is well-commented and follows best practices.
- Used meaningful variable and function names.