

Full Stack Development with MERN

Database Design and Development Report

Date	12 th July 2024
Team ID	SWTID1719933594
Project Name	SHOPEZ -E-COMMERCE APP
Maximum Marks	

Project Title: SHOPEZ -E-COMMERCE APP

Date: 12th July 2024

Prepared by: Nandini J Nair

Objective

The objective of this report is to outline the database design and implementation details for the SHOPEZ -E-COMMERCE APP project, including schema design and database management system (DBMS) integration.

Technologies Used

- **Database Management System (DBMS):** MongoDB
- **Object-Document Mapper (ODM):** Mongoose

Design the Database Schema

The database schema is designed to accommodate the following entities and relationships:

1. Users

- Attributes: [

name,

email,

password,

phone,

address,

answer,

role]

2. Product

- Attributes: [name,
 Slug,
 Description,
 Price,
 Category,
 Quantity,
 Photo,
 shipping]

3. orders

-Attributes: [products,
 Payment{buyer},
 Status]

4. category

Attributes: [name,
 Slug]

Implement the Database using MongoDB

The MongoDB database is implemented with the following collections and structures:

Database Name: shopez

1. Collection: users

```
JS userModle.js X
models > JS userModle.js > userSchema > address
1 import mongoose from "mongoose"; 849k (gzipped: 228.3k)
2
3 const userSchema = new mongoose.Schema({
4   name:{
5     type: String,
6     required:true,
7     trim:true,
8   },
9   email:{
10    type: String,
11    required:true,
12    unique:true,
13  },
14  password:{
15    type: String,
16    required:true,
17  },
18  phone:{
19    type: String,
20    required:true,
21  },
22  address:{
23    type: String,
24    required:true,
25  },
26
27  answer:{
28    type:String,
29    required:true,
30  },
31  role:{
32    type: Number,
33    default:0,
34  },
35
36 },{timeStamps:true}) // timeStamps is used as when an user is created the time on which user is created will be recorded
37
38 export default mongoose.model('users',userSchema);
```

2. Collection: Product

JS productModel.js X

models > JS productModel.js > ...

```
1  import mongoose from "mongoose"; 849k (gzipped: 228.3k)
2  |
3  const productSchema = new mongoose.Schema(
4    {
5      name: {
6        type: String,
7        required: true,
8      },
9      slug: {
10       type: String,
11       required: true,
12     },
13     description: {
14       type: String,
15       required: true,
16     },
17     price: {
18       type: Number,
19       required: true,
20     },
21     category: {
22       type: mongoose.ObjectId,
23       ref: "Category",
24       required: true,
25     },
26     quantity: {
27       type: Number,
28       required: true,
29     },
30     photo: {
31       data: Buffer,
32       contentType: String,
33     },
34     shipping: {
35       type: Boolean,
36     },
37   },
```

```

37     },
38     { timestamps: true }
39   );
40
41
42   export default mongoose.model('Product', productSchema)

```

3. Collection: Order

```

JS orderModel.js X
models > JS orderModel.js > [o] orderSchema
1  import mongoose from "mongoose"; 849k (gzipped: 228.3k)
2
3  const orderSchema = new mongoose.Schema(
4    {
5      products: [
6        {
7          type: mongoose.ObjectId,
8          ref: "Product",
9        },
10     ],
11     payment: {},
12     buyer: {
13       type: mongoose.ObjectId,
14       ref: "users",
15     },
16     status: {
17       type: String,
18       default: "Not Process",
19       enum: ["Not Process", "Processing", "Shipped", "delivered", "cancel"],
20     },
21   },
22   { timestamps: true }
23 );
24
25 export default mongoose.model("Order", orderSchema);

```

4 collection:Category

```
JS categoryModel.js X
models > JS categoryModel.js > ...
1 import mongoose from "mongoose"; 849k (gzipped: 228.3k)
2
3 const categorySchema = new mongoose.Schema({
4   name: {
5     type: String,
6     required: true,
7     unique: true,
8   },
9   slug: {
10    type: String,
11    lowercase: true,
12  },
13 });
14
15 export default mongoose.model("Category", categorySchema);
```

Integration with Backend

- Database connection: Database connection done using Mongoose

```
JS db.js X
config > JS db.js > [⌘] default
1 import mongoose from "mongoose"; 849k (gzipped: 228.3k)
2 const connectDB = async()=>{
3   try{
4     const conn = await mongoose.connect(process.env.MONGODB_URL)
5     console.log(`Connected to mongoDB Database ${conn.connection.host}`)
6   } catch(error){
7     console.log(`error in mongoDB ${error}`)
8   }
9 }
10
11
12 export default connectDB;
```

- The backend APIs interact with MongoDB using Mongoose ODM Key interactions include:
 - User Management: CRUD operations for users.

JS authRoute.js X

routes > JS authRoute.js > ...

```
1 import express from 'express';
2 import {registerController,loginController,testController, forgetPasswordC
3 import { isAdmin, requireSignIn } from '../middlewares/authMiddlewares.js'
4 //router object
5 const router = express.Router()
6
7 //routing
8 //Register || Method POST
9 router.post('/register',registerController);
10
11 //Login || Method POST
12 router.post('/login',loginController)
13
14 //forget password
15 router.post('/forgot-password',forgetPasswordController)
16 //testroutes
17 router.get('/test',requireSignIn,isAdmin,testController)
18
19 // protected user route-auth
20 router.get('/user-auth',requireSignIn, (req,res)=>{
21 |   res.status(200).send({ok:true})
22 | })
23 // protected admin route auth
24 router.get('/admin-auth',requireSignIn,isAdmin, (req,res)=>{
25 |   res.status(200).send({ok:true})
26 | })
27
28 //update profile
29 router.put('/profile',requireSignIn,updateProfileController)
30
31 //orders
32 //orders
33 router.get("/orders", requireSignIn, getOrdersController);
34
35 // all orders
36
37 router.get('/all-orders',requireSignIn, isAdmin, getAllOrdersController)
```

```

37 router.get('/:id/orders', requireSignIn, isAdmin, getOrderController);
38
39
40 //order status update
41 router.put(
42   "/order-status/:orderId",
43   requireSignIn,
44   isAdmin,
45   orderStatusController
46 );
47
48
49 export default router;

```

- Product Management: CRUD operations for products

```

# productRoutes.js X
routes > JS productRoutes.js > ...
1 import express from 'express';
2 import { isAdmin, requireSignIn } from '../middlewares/authMiddlewares.js';
3 import { braintreePaymentController, braintreeTokenController, createProductController, deleteProductController, getProductController, getSingleProductController, productPhotoController } from '../controllers/product.js';
4 import formidable from 'express-formidable';
5
6 const router = express.Router()
7
8 //routes
9 router.post('/create-product', requireSignIn, isAdmin, formidable(), createProductController);
10
11 //routes
12 router.put(
13   "/update-product/:pid",
14   requireSignIn,
15   isAdmin,
16   formidable(),
17   updateProductController
18 );
19
20 //get products
21 router.get("/get-product", getProductController);
22
23 //single product
24 router.get("/get-product/:slug", getSingleProductController);
25
26 //get photo
27 router.get("/product-photo/:pid", productPhotoController);
28
29 //delete rproduct
30 router.delete("/delete-product/:pid", deleteProductController);
31
32 //filter product
33 router.post("/product-filters", productFiltersController);
34
35 //product count
36 router.get("/product-count", productCountController);
37
38

```

```

# productRoutes.js X
routes > JS productRoutes.js > ...
34 //product count
35 router.get("/product-count", productCountController);
36
37 //product per page
38 router.get("/product-list/:page", productListController);
39
40 //search product
41 router.get("/search/:keyword", searchProductController);
42
43 //similar product//pid product id //cid category id
44 router.get("/related-product/:pid/:cid", relatedProductController);
45
46 //category wise product
47 router.get("/product-category/:slug", productCategoryController);
48
49 //payments routes
50 //token
51 router.get("/braintree/token", braintreeTokenController);
52
53 //payments
54 router.post("/braintree/payment", requireSignIn, braintreePaymentController);
55
56
57 export default router

```

- Categories Management: CRUD operations for categories

JS categoryRoutes.js X

routes > JS categoryRoutes.js > ...

```
1 import express from "express";
2 import { isAdmin, requireSignIn } from "../middlewares/authMiddlewares.js";
3 import {
4   categoryControlller,
5   createCategoryController,
6   deleteCategoryController,
7   singleCategoryController,
8   updateCategoryController,
9 } from "../controllers/categoryController.js";
10
11 const router = express.Router();
12
13 //routes
14 // create category
15 router.post(
16   "/create-category",
17   requireSignIn,
18   isAdmin,
19   createCategoryController
20 );
21
22 //update category
23 router.put(
24   "/update-category/:id",
25   requireSignIn,
26   isAdmin,
27   updateCategoryController
28 );
29
30 //getAll category
31 router.get("/get-category", categoryControlller);
32
33 //single category
34 router.get("/single-category/:slug", singleCategoryController);
35
36 //delete category
37 router.delete(
```

```
35
36 //delete category
37 router.delete(
38   "/delete-category/:id",
39   requireSignIn,
40   isAdmin,
41   deleteCategoryController
42 );
43
44 export default router;
```