**Full Stack Development with MERN**

**Database Design and Development Report**

| Date | 17/07/2024 |
| --- | --- |
| Team ID | SWTID1720014974 |
| Project Name | BookStore |
| Maximum Marks |  |

**Project Title**: BookStore

**Date**: 17/07/2024

**Prepared by**: Manideep , Pardhiv , Shrinivasan M , Keerthi

**Objective**

The objective of this report is to outline the database design and implementation details for the BookStore 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:

**Design the Database Schema**

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

**1. Users**

**- Attributes: `\_id`, `name`, `lastname`, `email`, `userinfo`, `encry\_password`, `salt`, `role`, `purchases`, `createdAt`, `updatedAt`**

**2. Products**

**- Attributes: `\_id`, `name`, `description`, `price`, `category` (references `Category`), `stock`, `sold`, `photo`, `createdAt`, `updatedAt`**

**3. Orders**

**- Attributes: `\_id`, `products`, `transaction\_id`, `amount`, `address`, `status`, `updated`, `user` (references `User`), `createdAt`, `updatedAt`**

**4. Categories**

**- Attributes: `\_id`, `name`, `createdAt`, `updatedAt`**

Implement the Database using MongoDB

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

Database Name: Book store

**1. Collection: users**

{

\_id: ObjectId,

name: String,

lastname: String,

email: String,

userinfo: String,

encry\_password: String,

salt: String,

role: Number,

purchases: Array,

createdAt: Date,

updatedAt: Date

}

**2. Collection: products**

{

\_id: ObjectId,

name: String,

description: String,

price: Number,

category: ObjectId (references categories),

stock: Number,

sold: Number,

photo: {

data: Buffer,

contentType: String

},

createdAt: Date,

updatedAt: Date

}

**3. Collection: orders**

{

\_id: ObjectId,

products: [

{

product: ObjectId (references products),

name: String,

count: Number,

price: Number

}

],

transaction\_id: Object,

amount: Number,

address: String,

status: String,

updated: Date,

user: ObjectId (references users),

createdAt: Date,

updatedAt: Date

}

**4. Collection: categories**

{

\_id: ObjectId,

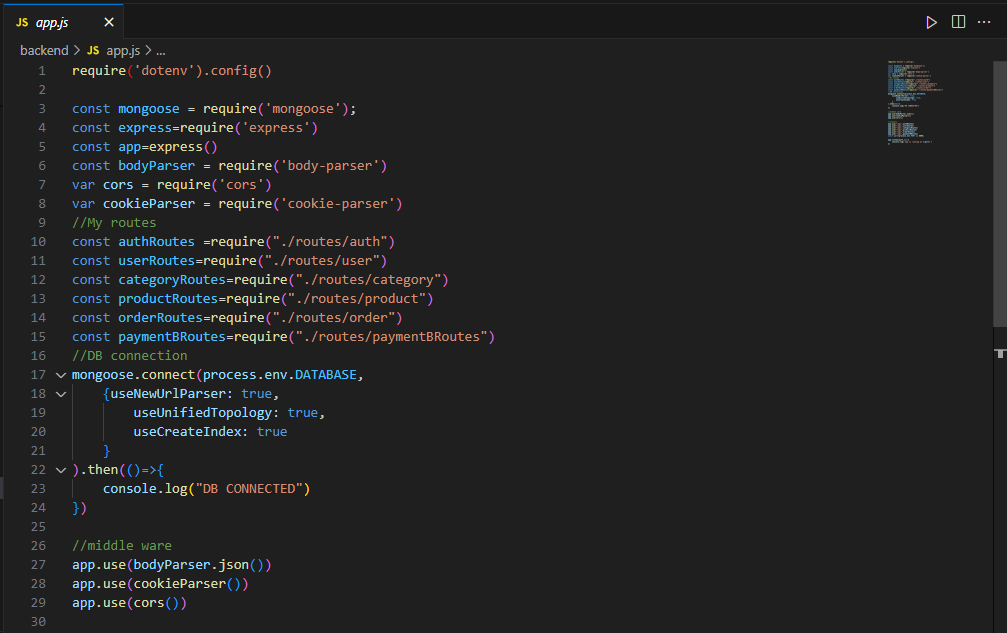
name: String,

createdAt: Date,

updatedAt: Date

}

**Integration with Backend**

* Database connection: Give Screenshot of Database connection done using Mongoose
* 

**User Management**

Create User: Register a new user with encrypted password storage.

const User = require('./models/User');

app.post('/api/users', async (req, res) => {

const user = new User(req.body);

await user.save();

res.status(201).send(user);

});

Read Users: Fetch all users or a specific user by ID.

app.get('/api/users/:id', async (req, res) => {

const user = await User.findById(req.params.id);

res.send(user);

});

Update User: Update user details.

app.put('/api/users/:id', async (req, res) => {

const user = await User.findByIdAndUpdate(req.params.id, req.body, { new: true });

res.send(user);

});

Delete User: Remove a user from the database.

app.delete('/api/users/:id', async (req, res) => {

await User.findByIdAndDelete(req.params.id);

res.status(204).send();

});

**Product Management**

Create Product: Add a new product.

const Product = require('./models/Product');

app.post('/api/products', async (req, res) => {

const product = new Product(req.body);

await product.save();

res.status(201).send(product);

});

Read Products: Fetch all products or a specific product by ID.

app.get('/api/products/:id', async (req, res) => {

const product = await Product.findById(req.params.id).populate('category');

res.send(product);

});

Update Product: Update product details.

app.put('/api/products/:id', async (req, res) => {

const product = await Product.findByIdAndUpdate(req.params.id, req.body, { new: true });

res.send(product);

});

Delete Product: Remove a product from the database.

app.delete('/api/products/:id', async (req, res) => {

await Product.findByIdAndDelete(req.params.id);

res.status(204).send();

});

**Order Management**

Create Order: Place a new order.

const Order = require('./models/Order');

app.post('/api/orders', async (req, res) => {

const order = new Order(req.body);

await order.save();

res.status(201).send(order);

});

Read Orders: Fetch all orders or a specific order by ID.

app.get('/api/orders/:id', async (req, res) => {

const order = await Order.findById(req.params.id).populate('user products.product');

res.send(order);

});

Update Order: Update order status or details.

app.put('/api/orders/:id', async (req, res) => {

const order = await Order.findByIdAndUpdate(req.params.id, req.body, { new: true });

res.send(order);

});

Delete Order: Cancel an order.

app.delete('/api/orders/:id', async (req, res) => {

await Order.findByIdAndDelete(req.params.id);

res.status(204).send();

});

**Category Management**

Create Category: Add a new category.

const Category = require('./models/Category');

app.post('/api/categories', async (req, res) => {

const category = new Category(req.body);

await category.save();

res.status(201).send(category);

});

Read Categories: Fetch all categories or a specific category by ID.

app.get('/api/categories/:id', async (req, res) => {

const category = await Category.findById(req.params.id);

res.send(category);

});

Update Category: Update category details.

app.put('/api/categories/:id', async (req, res) => {

const category = await Category.findByIdAndUpdate(req.params.id, req.body, { new: true });

res.send(category);

});

Delete Category: Remove a category from the database.

app.delete('/api/categories/:id', async (req, res) => {

await Category.findByIdAndDelete(req.params.id);

res.status(204).send();

});