**Full Stack Development with MERN**

**Database Design and Development Report**

|  |  |
| --- | --- |
| Date | 18-07-2024 |
| Team ID | PNT2022TMID SWTID1720175019 |
| Project Name | Nexus Learn - Online learning platform |
| Maximum Marks |  |

**Project Title**: Nexus Learn - Online learning platform

**Date**: 18-07-2024

**Prepared by**: Sathvik(Team Lead), Shanmukh, Sashank, Sudheer

**Objective**

The objective of this report is to outline the database design and implementation details for the Nexus Learn - Online learning platform project, including schema design and database management system (DBMS) integration.

**Technologies Used**

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

**Database Schema for NEXUS LEARN**

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

1. **Users**
   * Attributes: \_id, name, lastname, email, userinfo, encry\_password, salt, role, createdAt, updatedAt
2. **Courses**
   * Attributes: \_id, title, description, category, price, teacher (references User), modules, students, createdAt, updatedAt
3. **Modules**
   * Attributes: \_id, title, content, course (references Course), createdAt, updatedAt
4. **Enrollments**
   * Attributes: \_id, course (references Course), student (references User), progress, completed, certificate, createdAt, updatedAt

**MongoDB Database Implementation for NEXUS LEARN**

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

**Database Name:** NEXUS\_LEARN

1. **Collection: users**

{

\_id: ObjectId,

name: String,

lastname: String,

email: String,

userinfo: String,

encry\_password: String,

salt: String,

role: String, // e.g., "student", "teacher", "admin"

createdAt: Date,

updatedAt: Date

}

1. **Collection: courses**

{

\_id: ObjectId,

title: String,

description: String,

category: String, // Category name or ID

price: Number,

teacher: ObjectId (references users),

modules: [

{

moduleId: ObjectId (references modules),

title: String,

content: String

}

],

students: [

ObjectId (references users)

],

createdAt: Date,

updatedAt: Date

}

1. **Collection: modules**

{

\_id: ObjectId,

title: String,

content: String,

course: ObjectId (references courses),

createdAt: Date,

updatedAt: Date

}

1. **Collection: enrollments**

{

\_id: ObjectId,

course: ObjectId (references courses),

student: ObjectId (references users),

progress: Number, // Percentage or progress indicator

completed: Boolean,

certificate: String, // Link or reference to the certificate

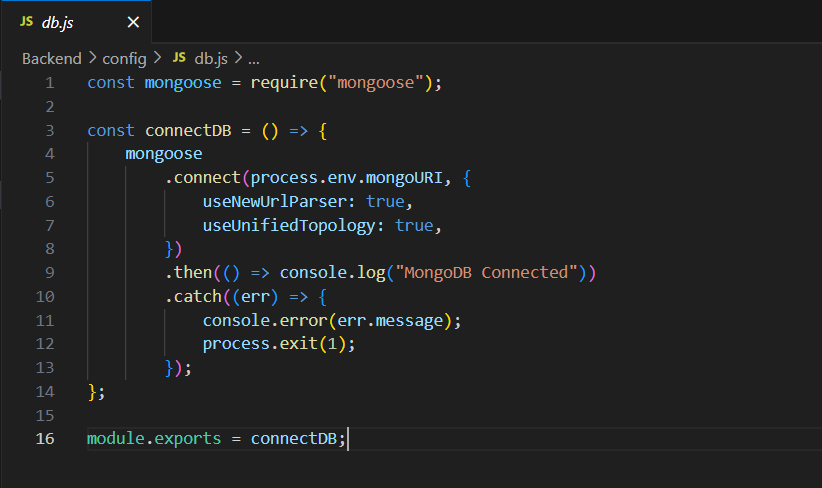
createdAt: Date,

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();

});

**Course Management**

* **Create Course: Add a new course.**

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

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

const course = new Course(req.body);

await course.save();

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

});

* **Read Courses: Fetch all courses or a specific course by ID.**

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

const course = await Course.findById(req.params.id).populate('teacher modules');

res.send(course);

});

* **Update Course: Update course details.**

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

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

res.send(course);

});

* **Delete Course: Remove a course from the database if no students are enrolled.**

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

const course = await Course.findById(req.params.id);

if (course.students.length > 0) {

return res.status(400).send({ error: 'Course has enrolled students and cannot be deleted' });

}

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

res.status(204).send();

});

**Module Management**

* **Create Module: Add a new module to a course.**

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

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

const module = new Module(req.body);

await module.save();

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

});

* **Read Modules: Fetch a specific module by ID.**

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

const module = await Module.findById(req.params.id);

res.send(module);

});

* **Update Module: Update module details.**

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

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

res.send(module);

});

* **Delete Module: Remove a module from the database.**

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

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

res.status(204).send();

});

**Enrollment Management**

* **Create Enrollment: Enroll a student in a course.**

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

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

const enrollment = new Enrollment(req.body);

await enrollment.save();

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

});

* **Read Enrollments: Fetch all enrollments for a student or a specific enrollment by ID.**

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

const enrollments = await Enrollment.find({ student: req.params.studentId });

res.send(enrollments);

});

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

const enrollment = await Enrollment.findById(req.params.id);

res.send(enrollment);

});

* **Update Enrollment: Update enrollment details.**

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

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

res.send(enrollment);

});

* **Delete Enrollment: Unenroll a student from a course.**

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

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

res.status(204).send();

});

**Certificate Management**

* **Generate Certificate: Retrieve a course completion certificate for a student.**

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

const enrollment = await Enrollment.findById(req.params.enrollmentId);

if (enrollment.completed) {

// Assume certificate generation logic here

res.send({ certificate: `Certificate for ${enrollment.student}` });

} else {

res.status(400).send({ error: 'Course not completed yet' });

}

});