Full Stack Development with MERN

Database Design and Development Report

Date	21/Jul/2024
Team ID	SWTID1720103759
Project Name	Resolve Radar
Maximum Marks	

Project Title: Complaint Management System

Date: 21st July 2024

Prepared by: Resolve Radar

Objective

The objective of this report is to outline the database design and implementation details for the Complaint Management System 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. StudentSchema

- Attributes: studentId, email, password

2. employeeSchema

- Attributes: employeeId, email, password

3. complaintSchema

- Attributes: complaintText, category, date, studentId, status

Implement the Database using MongoDB

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

Database Name: Schema

1. Collection: studentSchema

   - Schema:

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

2. Collection: employeeSchema

   - Schema:

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

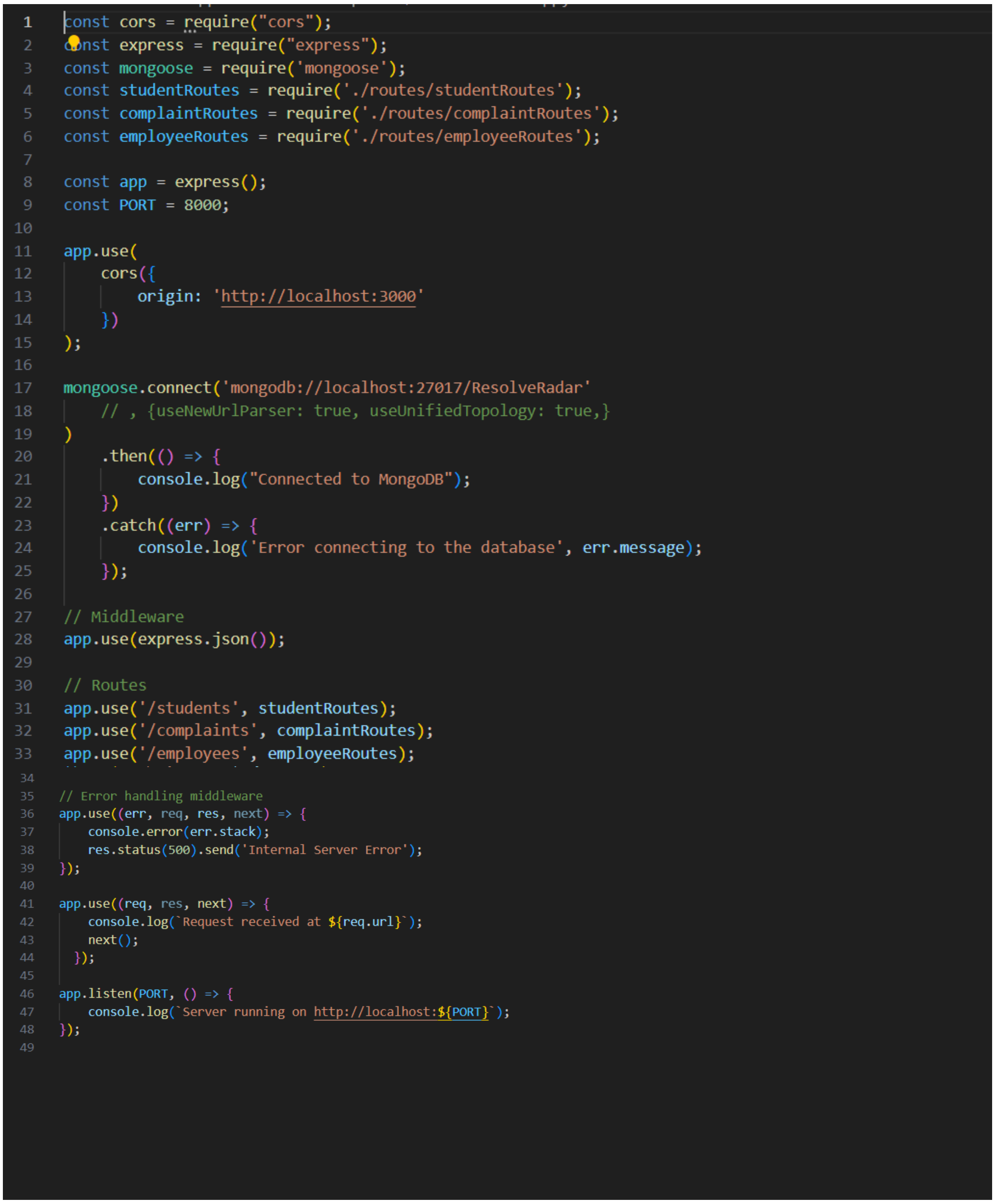
3. Collection: complaintSchema

   - Schema:

```
{
  complaintText: { type: String, required: true },
  category: { type: String, required: true },
  date: { type: Date, default: () => new Date().toISOString().split('T')[0] },
  studentId: { type: String, required: true },
  status: {type:String, default:'Pending'}
}
```

Integration with Backend

- Database connection:



```javascript
const cors = require("cors");
const express = require("express");
const mongoose = require('mongoose');
const studentRoutes = require('./routes/studentRoutes');
const complaintRoutes = require('./routes/complaintRoutes');
const employeeRoutes = require('./routes/employeeRoutes');

const app = express();
const PORT = 8000;

app.use(
    cors({
        origin: 'http://localhost:3000'
    })
);

mongoose.connect('mongodb://localhost:27017/ResolveRadar'
    // , {useNewUrlParser: true, useUnifiedTopology: true,}
)
    .then(() => {
        console.log("Connected to MongoDB");
    })
    .catch((err) => {
        console.log('Error connecting to the database', err.message);
    });

// Middleware
app.use(express.json());

// Routes
app.use('/students', studentRoutes);
app.use('/complaints', complaintRoutes);
app.use('/employees', employeeRoutes);

// Error handling middleware
app.use((err, req, res, next) => {
    console.error(err.stack);
    res.status(500).send('Internal Server Error');
});

app.use((req, res, next) => {
    console.log(`Request received at ${req.url}`);
    next();
  });

app.listen(PORT, () => {
    console.log(`Server running on http://localhost:${PORT}`);
});
```

- The backend APIs interact with MongoDB using Mongoose ODM Key interactions include:
    - **Students Management:**

1. **Create a new student**:

```
exports.createStudent = async (req, res) => {
  try {
    const { studentId, email, password } = req.body;
    console.log("Request body:", req.body);

    const existingStudent = await Student.findOne({ email });

    if (existingStudent) {
      return res.status(400).json({ message: 'Student already exists' });
    }

    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(password, salt);
    const newStudent = new Student({ studentId, email, password:
hashedPassword });
    await newStudent.save();

    res.status(201).json(newStudent);
  } catch (error) {
    console.error('Error creating student:', error); // Log the full error stack
    res.status(500).json({ message: 'Error creating student', error:
error.message });
  }
};
```

2. **Get student by studentId**:

```
exports.getStudentByStudentId = async (req, res) => {
  try {
    const student = await Student.findOne({ studentId: req.params.studentId
}).populate('complaints');
    if (!student) {
      return res.status(404).json({ message: 'Student not found' });
    }

    res.status(200).json(student);
  } catch (error) {
    res.status(500).json({ message: 'Error fetching student', error });
  }
};
```

3. **Update student profile:**

```
exports.updateStudentProfile = async (req, res) => {
  try {
    const { studentId, email, password } = req.body;
    const student = await Student.findOne({ studentId: req.params.studentId
});
```

```
  if (!student) {
    return res.status(404).json({ message: 'Student not found' });
  }

  if (email) {
    student.email = email;
  }
  if (password) {
    const salt = await bcrypt.genSalt(10);
    student.password = await bcrypt.hash(password, salt);
  }
  await student.save();
  res.status(200).json({ message: 'Profile updated successfully', student });
 } catch (error) {
  res.status(500).json({ message: 'Error updating profile', error });
 }
};
```

- **Employee Management**:
  1. **Create a new employee**:

```
exports.createEmployee = async (req, res) => {
 try {
  const { employeeId, email, password } = req.body;
  console.log("Request body:", req.body);

  const existingEmployee = await Employee.findOne({ email });
  if (existingEmployee) {
    return res.status(400).json({ message: 'Employee already exists' });
  }
  const salt = await bcrypt.genSalt(10);
  const hashedPassword = await bcrypt.hash(password, salt);
  const newEmployee = new Employee({ employeeId, email, password:
hashedPassword });
  await newEmployee.save();
  res.status(201).json(newEmployee);
 } catch (error) {
  res.status(500).json({ message: 'Error creating employee', error });
 }
};
```

  2. **Get employee by ID**

```
exports.getEmployeeById = async (req, res) => {
 try {
  const employee = await
Employee.findById(req.params.id).populate('complaints');
  if (!employee) {
    return res.status(404).json({ message: 'Employee not found' });
  }
  res.status(200).json(employee);
 } catch (error) {
   res.status(500).json({ message: 'Error fetching employee', error });
 }
};
```

3. **Update employee profile:**

```
exports.updateEmployeeProfile = async (req, res) => {
 try {
   const { employeeId, email, password } = req.body;
   const employee = await Employee.findOne({ employeeId:
req.params.employeeId });

   if (!employee) {
     return res.status(404).json({ message: 'Employee not found' });
   }

   if (email) {
     employee.email = email;
   }
   if (password) {
     const salt = await bcrypt.genSalt(10);
     employee.password = await bcrypt.hash(password, salt);
   }
   await employee.save();

   res.status(200).json({ message: 'Profile updated successfully', employee });
 } catch (error) {
   res.status(500).json({ message: 'Error updating profile', error });
 }
};
```

- **Complaints management:**
  1. **Create a new complaint**:

```
exports.createComplaint = async (req, res) => {
 try {
   const { complaintText, category, studentId } = req.body;

   const newComplaint = new Complaint({
     complaintText,
     category,
     studentId
   });
   const savedComplaint = await newComplaint.save();
   res.status(201).json(savedComplaint);
 } catch (error) {
   res.status(500).json({ message: 'Error creating complaint', error });
 }
};
```

2. **Get all complaints**:

```
exports.getAllComplaints = async (req, res) => {
  try {
    const complaints = await Complaint.find();
    res.status(200).json(complaints);
  } catch (error) {
    res.status(500).json({ message: 'Error fetching complaints', error });
  }
};
```

3. **Get complaint by ID**:

```
exports.getComplaintById = async (req, res) => {
  try {
    const complaint = await Complaint.find({ studentId: req.params.studentId
});
    if (!complaint) {
      return res.status(404).json({ message: 'Complaint not found' });
    }
    res.status(200).json(complaint);
  } catch (error) {
    res.status(500).json({ message: 'Error fetching complaint', error });
  }
};
```

4. **Get complaints by Status:**

```
exports.getComplaintByStatus = async (req, res) => {
  try {
    const complaint = await Complaint.find({ status: req.params.status });
    if (!complaint) {
      return res.status(404).json({ message: 'Complaint not found' });
    }
    res.status(200).json(complaint);
  } catch (error) {
    res.status(500).json({ message: 'Error fetching complaint', error });
  }
};
```

5. **Mark a complaint as Done**:

```
exports.markAsDone = async (req, res) => {
  try {
    const complaint = await Complaint.findById(req.params.id);
    if (!complaint) {
      return res.status(404).json({ message: 'Complaint not found' });
    }
    complaint.status = 'Done';
    await complaint.save();
    res.json(complaint);
```

```
 } catch (error) {
   res.status(500).json({ message: error.message });
 }
};
```

6. **Mark a complaint as In Progress**:

```
exports.markAsInProgress = async (req, res) => {
 try {
   const complaint = await Complaint.findById(req.params.id);
   if (!complaint) {
     return res.status(404).json({ message: 'Complaint not found' });
   }
   complaint.status = 'In Progress';
   await complaint.save();
   res.json(complaint);
 } catch (error) {
   res.status(500).json({ message: error.message });
 }
};
```

7. **Delete a complaint**:

```
exports.deleteComplaint = async (req, res) => {
 try {
   const complaint = await Complaint.findById(req.params.id);
   if (!complaint) {
     return res.status(404).json({ message: 'Complaint not found' });
   }
   const student = await Student.findById(complaint.studentId);
   if (student) {
     student.complaints.pull(complaint._id);
     await student.save();
   }

   await complaint.remove();

   res.status(200).json({ message: 'Complaint deleted successfully' });
 } catch (error) {
   console.error('Error deleting complaint:', error);
   res.status(500).json({ message: 'Error deleting complaint', error });
 }
};
```