# Node.js Sequelize MSSQL Authentication and CRUD System

Here's a complete implementation for a Node.js application with Sequelize and MSSQL that includes:

- JWT authentication (register, login, logout)

- Department CRUD operations

- Employee CRUD operations

- Standard folder structure

## Folder Structure

```

project-root/

├── config/

│ └── config.js

├── controllers/

│ ├── auth.controller.js

│ ├── department.controller.js

│ ├── employee.controller.js

├── middlewares/

│ └── auth.middleware.js

├── migrations/

│ ├── xxxxx-create-user.js

│ ├── xxxxx-create-department.js

│ └── xxxxx-create-employee.js

├── models/

│ ├── index.js

│ ├── user.model.js

│ ├── department.model.js

│ └── employee.model.js

├── routes/

│ ├── auth.routes.js

│ ├── department.routes.js

│ ├── employee.routes.js

│ └── index.js

├── seeders/

├── services/

│ └── auth.service.js

├── utils/

│ └── jwt.util.js

├── app.js

├── package.json

└── .env

```

## Implementation Steps

### 1. Install required packages

```bash

npm install express sequelize tedious sequelize-cli jsonwebtoken bcryptjs cors dotenv

```

### 2. Configuration Files

#### config/config.js

```javascript

require('dotenv').config();

module.exports = {

development: {

username: process.env.DB\_USER,

password: process.env.DB\_PASSWORD,

database: process.env.DB\_NAME,

host: process.env.DB\_HOST,

dialect: 'mssql',

dialectOptions: {

options: {

encrypt: true,

trustServerCertificate: true

}

}

},

test: {

username: process.env.DB\_USER,

password: process.env.DB\_PASSWORD,

database: process.env.DB\_NAME,

host: process.env.DB\_HOST,

dialect: 'mssql'

},

production: {

username: process.env.DB\_USER,

password: process.env.DB\_PASSWORD,

database: process.env.DB\_NAME,

host: process.env.DB\_HOST,

dialect: 'mssql'

}

};

```

### 3. Models

#### models/user.model.js

```javascript

module.exports = (sequelize, DataTypes) => {

const User = sequelize.define('User', {

FirstName: {

type: DataTypes.STRING,

allowNull: false

},

LastName: {

type: DataTypes.STRING,

allowNull: false

},

Username: {

type: DataTypes.STRING,

allowNull: false,

unique: true

},

Email: {

type: DataTypes.STRING,

allowNull: false,

unique: true,

validate: {

isEmail: true

}

},

PasswordHash: {

type: DataTypes.STRING,

allowNull: false

}

});

return User;

};

```

#### models/department.model.js

```javascript

module.exports = (sequelize, DataTypes) => {

const Department = sequelize.define('Department', {

Name: {

type: DataTypes.STRING(100),

allowNull: false

}

});

Department.associate = (models) => {

Department.hasMany(models.Employee, {

foreignKey: 'DepartmentId',

as: 'employees'

});

};

return Department;

};

```

#### models/employee.model.js

```javascript

module.exports = (sequelize, DataTypes) => {

const Employee = sequelize.define('Employee', {

Name: {

type: DataTypes.STRING(100),

allowNull: false

},

Email: {

type: DataTypes.STRING,

allowNull: false,

validate: {

isEmail: true

}

},

Position: {

type: DataTypes.STRING(50),

allowNull: false

},

Salary: {

type: DataTypes.DECIMAL(18, 2),

allowNull: false

},

HireDate: {

type: DataTypes.DATE,

allowNull: false

},

DepartmentId: {

type: DataTypes.INTEGER,

allowNull: false,

references: {

model: 'Departments',

key: 'Id'

}

}

});

Employee.associate = (models) => {

Employee.belongsTo(models.Department, {

foreignKey: 'DepartmentId',

as: 'department'

});

};

return Employee;

};

```

#### models/index.js

```javascript

const fs = require('fs');

const path = require('path');

const Sequelize = require('sequelize');

const basename = path.basename(\_\_filename);

const env = process.env.NODE\_ENV || 'development';

const config = require(\_\_dirname + '/../config/config.js')[env];

const db = {};

let sequelize;

if (config.use\_env\_variable) {

sequelize = new Sequelize(process.env[config.use\_env\_variable], config);

} else {

sequelize = new Sequelize(config.database, config.username, config.password, config);

}

fs

.readdirSync(\_\_dirname)

.filter(file => {

return (file.indexOf('.') !== 0) && (file !== basename) && (file.slice(-3) === '.js';

})

.forEach(file => {

const model = require(path.join(\_\_dirname, file))(sequelize, Sequelize.DataTypes);

db[model.name] = model;

});

Object.keys(db).forEach(modelName => {

if (db[modelName].associate) {

db[modelName].associate(db);

}

});

db.sequelize = sequelize;

db.Sequelize = Sequelize;

module.exports = db;

```

### 4. JWT Utility

#### utils/jwt.util.js

```javascript

const jwt = require('jsonwebtoken');

const config = require('../config/config');

const generateToken = (userId) => {

return jwt.sign({ id: userId }, config.jwt.secret, {

expiresIn: config.jwt.expiresIn

});

};

const verifyToken = (token) => {

return jwt.verify(token, config.jwt.secret);

};

module.exports = {

generateToken,

verifyToken

};

```

### 5. Auth Service

#### services/auth.service.js

```javascript

const bcrypt = require('bcryptjs');

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

const jwt = require('../utils/jwt.util');

const register = async (userData) => {

const { FirstName, LastName, Username, Email, Password } = userData;

// Check if user exists

const existingUser = await db.User.findOne({ where: { Email } });

if (existingUser) {

throw new Error('User already exists');

}

// Hash password

const salt = await bcrypt.genSalt(10);

const hashedPassword = await bcrypt.hash(Password, salt);

// Create user

const user = await db.User.create({

FirstName,

LastName,

Username,

Email,

PasswordHash: hashedPassword

});

// Generate token

const token = jwt.generateToken(user.Id);

return { user, token };

};

const login = async (email, password) => {

// Find user

const user = await db.User.findOne({ where: { Email: email } });

if (!user) {

throw new Error('User not found');

}

// Check password

const isMatch = await bcrypt.compare(password, user.PasswordHash);

if (!isMatch) {

throw new Error('Invalid credentials');

}

// Generate token

const token = jwt.generateToken(user.Id);

return { user, token };

};

const logout = async (token) => {

// In a real app, you might want to implement token blacklisting here

return true;

};

module.exports = {

register,

login,

logout

};

```

### 6. Controllers

#### controllers/auth.controller.js

```javascript

const authService = require('../services/auth.service');

exports.register = async (req, res) => {

try {

const { user, token } = await authService.register(req.body);

res.status(201).json({ user, token });

} catch (error) {

res.status(400).json({ message: error.message });

}

};

exports.login = async (req, res) => {

try {

const { email, password } = req.body;

const { user, token } = await authService.login(email, password);

res.json({ user, token });

} catch (error) {

res.status(401).json({ message: error.message });

}

};

exports.logout = async (req, res) => {

try {

const token = req.headers.authorization.split(' ')[1];

await authService.logout(token);

res.json({ message: 'Logged out successfully' });

} catch (error) {

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

}

};

```

#### controllers/department.controller.js

```javascript

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

exports.createDepartment = async (req, res) => {

try {

const department = await db.Department.create(req.body);

res.status(201).json(department);

} catch (error) {

res.status(400).json({ message: error.message });

}

};

exports.getAllDepartments = async (req, res) => {

try {

const departments = await db.Department.findAll();

res.json(departments);

} catch (error) {

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

}

};

exports.getDepartmentById = async (req, res) => {

try {

const department = await db.Department.findByPk(req.params.id, {

include: [{ model: db.Employee, as: 'employees' }]

});

if (!department) {

return res.status(404).json({ message: 'Department not found' });

}

res.json(department);

} catch (error) {

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

}

};

exports.updateDepartment = async (req, res) => {

try {

const department = await db.Department.findByPk(req.params.id);

if (!department) {

return res.status(404).json({ message: 'Department not found' });

}

await department.update(req.body);

res.json(department);

} catch (error) {

res.status(400).json({ message: error.message });

}

};

exports.deleteDepartment = async (req, res) => {

try {

const department = await db.Department.findByPk(req.params.id);

if (!department) {

return res.status(404).json({ message: 'Department not found' });

}

await department.destroy();

res.json({ message: 'Department deleted successfully' });

} catch (error) {

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

}

};

```

#### controllers/employee.controller.js

```javascript

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

exports.createEmployee = async (req, res) => {

try {

const employee = await db.Employee.create(req.body);

res.status(201).json(employee);

} catch (error) {

res.status(400).json({ message: error.message });

}

};

exports.getAllEmployees = async (req, res) => {

try {

const employees = await db.Employee.findAll({

include: [{ model: db.Department, as: 'department' }]

});

res.json(employees);

} catch (error) {

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

}

};

exports.getEmployeeById = async (req, res) => {

try {

const employee = await db.Employee.findByPk(req.params.id, {

include: [{ model: db.Department, as: 'department' }]

});

if (!employee) {

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

}

res.json(employee);

} catch (error) {

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

}

};

exports.updateEmployee = async (req, res) => {

try {

const employee = await db.Employee.findByPk(req.params.id);

if (!employee) {

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

}

await employee.update(req.body);

res.json(employee);

} catch (error) {

res.status(400).json({ message: error.message });

}

};

exports.deleteEmployee = async (req, res) => {

try {

const employee = await db.Employee.findByPk(req.params.id);

if (!employee) {

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

}

await employee.destroy();

res.json({ message: 'Employee deleted successfully' });

} catch (error) {

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

}

};

```

### 7. Middleware

#### middlewares/auth.middleware.js

```javascript

const jwt = require('../utils/jwt.util');

exports.authenticate = (req, res, next) => {

try {

const token = req.headers.authorization?.split(' ')[1];

if (!token) {

return res.status(401).json({ message: 'Authentication failed: No token provided' });

}

const decoded = jwt.verifyToken(token);

req.userId = decoded.id;

next();

} catch (error) {

return res.status(401).json({ message: 'Authentication failed: Invalid token' });

}

};

```

### 8. Routes

#### routes/auth.routes.js

```javascript

const express = require('express');

const router = express.Router();

const authController = require('../controllers/auth.controller');

router.post('/register', authController.register);

router.post('/login', authController.login);

router.post('/logout', authController.logout);

module.exports = router;

```

#### routes/department.routes.js

```javascript

const express = require('express');

const router = express.Router();

const departmentController = require('../controllers/department.controller');

const authMiddleware = require('../middlewares/auth.middleware');

router.use(authMiddleware.authenticate);

router.post('/', departmentController.createDepartment);

router.get('/', departmentController.getAllDepartments);

router.get('/:id', departmentController.getDepartmentById);

router.put('/:id', departmentController.updateDepartment);

router.delete('/:id', departmentController.deleteDepartment);

module.exports = router;

```

#### routes/employee.routes.js

```javascript

const express = require('express');

const router = express.Router();

const employeeController = require('../controllers/employee.controller');

const authMiddleware = require('../middlewares/auth.middleware');

router.use(authMiddleware.authenticate);

router.post('/', employeeController.createEmployee);

router.get('/', employeeController.getAllEmployees);

router.get('/:id', employeeController.getEmployeeById);

router.put('/:id', employeeController.updateEmployee);

router.delete('/:id', employeeController.deleteEmployee);

module.exports = router;

```

#### routes/index.js

```javascript

const express = require('express');

const router = express.Router();

const authRoutes = require('./auth.routes');

const departmentRoutes = require('./department.routes');

const employeeRoutes = require('./employee.routes');

router.use('/auth', authRoutes);

router.use('/departments', departmentRoutes);

router.use('/employees', employeeRoutes);

module.exports = router;

```

### 9. Main Application File

#### app.js

```javascript

const express = require('express');

const cors = require('cors');

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

const routes = require('./routes');

const app = express();

// Middleware

app.use(cors());

app.use(express.json());

app.use(express.urlencoded({ extended: true }));

// Routes

app.use('/api', routes);

// Database connection

db.sequelize.sync()

.then(() => {

console.log('Database connected');

})

.catch(err => {

console.error('Database connection error:', err);

});

// Error handling

app.use((err, req, res, next) => {

console.error(err.stack);

res.status(500).json({ message: 'Something went wrong!' });

});

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

```

### 10. Environment Variables (.env)

```

DB\_USER=your\_db\_username

DB\_PASSWORD=your\_db\_password

DB\_NAME=your\_db\_name

DB\_HOST=your\_db\_host

JWT\_SECRET=your\_jwt\_secret

JWT\_EXPIRES\_IN=1h

```

### 11. Migrations

Run the following commands to create and run migrations:

```bash

npx sequelize-cli migration:generate --name create-user

npx sequelize-cli migration:generate --name create-department

npx sequelize-cli migration:generate --name create-employee

```

Then edit the migration files to match your schema.

### 12. Running the Application

1. Install dependencies: `npm install`

2. Set up your database and configure the connection in `.env`

3. Run migrations: `npx sequelize-cli db:migrate`

4. Start the server: `node app.js`

This implementation provides a complete RESTful API with JWT authentication and CRUD operations for departments and employees using Sequelize with MSSQL. The code follows a clean architecture with proper separation of concerns and includes error handling and validation.

Adding Swagger with Autogen in a Node.js App

Swagger (now known as OpenAPI) is a great way to document your API, and swagger-autogen is a tool that can automatically generate Swagger documentation from your route definitions. Here's how to set it up in a Node.js application:

1. Install required packages

First, install the necessary packages:

bash

Copy

Download

npm install swagger-autogen swagger-ui-express express

2. Create a basic Express app

Here's a simple Express app structure:

javascript

Copy

Download

*// app.js*

const express = require('express');

const app = express();

app.use(express.json());

*// Routes*

app.get('/users', (req, res) => {

res.json([{ id: 1, name: 'John Doe' }]);

});

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

const user = req.body;

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

});

const PORT = 3000;

app.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

3. Set up Swagger Autogen

Create a swagger.js file to configure Swagger autogen:

javascript

Copy

Download

*// swagger.js*

const swaggerAutogen = require('swagger-autogen')();

const doc = {

info: {

title: 'My API',

description: 'Description of my API',

},

host: 'localhost:3000',

schemes: ['http'],

};

const outputFile = './swagger-output.json';

const endpointsFiles = ['./app.js']; *// Point to your main app file*

*// Generate swagger.json*

swaggerAutogen(outputFile, endpointsFiles, doc);

4. Modify your app to serve Swagger UI

Update your app.js to serve the Swagger UI:

javascript

Copy

Download

*// app.js*

const express = require('express');

const swaggerUi = require('swagger-ui-express');

const swaggerFile = require('./swagger-output.json'); *// Generated swagger file*

const app = express();

app.use(express.json());

*// Routes*

app.get('/users', (req, res) => {

res.json([{ id: 1, name: 'John Doe' }]);

});

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

const user = req.body;

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

});

*// Swagger route*

app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerFile));

const PORT = 3000;

app.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

5. Add JSDoc comments for better documentation

Enhance your documentation by adding JSDoc comments to your routes:

javascript

Copy

Download

*/\*\**

*\* @swagger*

*\* /users:*

*\* get:*

*\* summary: Returns the list of all users*

*\* tags: [Users]*

*\* responses:*

*\* 200:*

*\* description: The list of users*

*\* content:*

*\* application/json:*

*\* schema:*

*\* type: array*

*\* items:*

*\* $ref: '#/components/schemas/User'*

*\*/*

app.get('/users', (req, res) => {

res.json([{ id: 1, name: 'John Doe' }]);

});

*/\*\**

*\* @swagger*

*\* /users:*

*\* post:*

*\* summary: Create a new user*

*\* tags: [Users]*

*\* requestBody:*

*\* required: true*

*\* content:*

*\* application/json:*

*\* schema:*

*\* $ref: '#/components/schemas/User'*

*\* responses:*

*\* 201:*

*\* description: The created user*

*\* content:*

*\* application/json:*

*\* schema:*

*\* $ref: '#/components/schemas/User'*

*\*/*

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

const user = req.body;

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

});

6. Define schemas in your swagger.js

Update your swagger.js to include schemas:

javascript

Copy

Download

*// swagger.js*

const swaggerAutogen = require('swagger-autogen')();

const doc = {

info: {

title: 'My API',

description: 'Description of my API',

},

host: 'localhost:3000',

schemes: ['http'],

definitions: {

User: {

type: 'object',

properties: {

id: {

type: 'integer',

format: 'int64'

},

name: {

type: 'string'

}

}

}

}

};

const outputFile = './swagger-output.json';

const endpointsFiles = ['./app.js'];

swaggerAutogen(outputFile, endpointsFiles, doc);

7. Add scripts to package.json

Add these scripts to your package.json:

json

Copy

Download

"scripts": {

"start": "node app.js",

"swagger-autogen": "node swagger.js"

}

8. Run the application

1. First generate the Swagger documentation:

bash

Copy

Download

npm run swagger-autogen

1. Then start your application:

bash

Copy

Download

npm start

1. Access the Swagger UI at http://localhost:3000/api-docs