Scenario: Employee Leave Management API

A company wants to build a simple backend for employees to apply for leave and for managers to view, approve, or reject leaves.
You must build REST APIs using:

- Express.js
- Router-level & global middleware
- Custom logger
- Token validation middleware
- Sequelize ORM
- MySQL database
- Routes with params & query strings
- Transactions for approval/rejection
- Error handling middleware

1. Project Description

You are building an **Employee Leave Management API** that supports:

Features

1. Add Employee
2. Apply for Leave
3. Manager approves/rejects leave
4. Search employee leaves by query params
5. Custom logger for every API call
6. Authentication using `x-auth-token`
7. Sequelize ORM for DB operations
8. Transaction handling during leave approval

2. Project Structure

```
employee-leave-api/
├── config/
│   └── db.js
├── middleware/
│   ├── logger.js
│   ├── auth.js
│   └── errorHandler.js
├── models/
│   ├── index.js
│   ├── employee.js
│   └── leave.js
├── routes/
│   ├── employee.routes.js
│   └── leave.routes.js
├── controllers/
│   ├── employee.controller.js
│   └── leave.controller.js
├── app.js
└── server.js
```

3. Sequelize Models

employee.js

```
module.exports = (sequelize, DataTypes) => {
  const Employee = sequelize.define("Employee", {
    name: { type: DataTypes.STRING, allowNull: false },
    department: DataTypes.STRING
  });

  Employee.associate = (models) => {
    Employee.hasMany(models.Leave, { foreignKey: "employeeId"
});
  };

  return Employee;
};
```

leave.js

```
module.exports = (sequelize, DataTypes) => {
  const Leave = sequelize.define("Leave", {
```

```
    fromDate: DataTypes.DATE,
    toDate: DataTypes.DATE,
    status: { type: DataTypes.STRING, defaultValue: "PENDING"
}
  });

  Leave.associate = (models) => {
    Leave.belongsTo(models.Employee, { foreignKey:
"employeeId" });
  };

  return Leave;
};
```

4. Middleware

logger.js

```
module.exports = (req, res, next) => {
  const time = new Date().toISOString();
  const route = req.originalUrl;
  const method = req.method;
  const token = req.headers["x-auth-token"] ? "***" : "NONE";

  console.log(`[${time}] ${method} ${route} - token: ${token}
`);
  next();
};
```

auth.js

```
module.exports = (req, res, next) => {
  const token = req.headers["x-auth-token"];
  if (!token || token !== "manager123")
    return res.status(401).json({ error: "Invalid or missing
token" });

  next();
};
```

errorHandler.js

```
module.exports = (err, req, res, next) => {
  console.error("Error:", err.message);
  res.status(500).json({ error: err.message });
};
```

5. Controller Logic

employee.controller.js

```
const { Employee } = require("../models");

exports.createEmployee = async (req, res, next) => {
  try {
    const emp = await Employee.create(req.body);
    res.status(201).json(emp);
  } catch (err) {
    next(err);
  }
};
```

leave.controller.js

```
const { Leave, Employee, sequelize } = require("../models");

exports.applyLeave = async (req, res, next) => {
  try {
    const leave = await Leave.create({
      employeeId: req.body.employeeId,
      fromDate: req.body.fromDate,
      toDate: req.body.toDate
    });
    res.status(201).json(leave);
  } catch (err) {
    next(err);
  }
};

exports.approveLeave = async (req, res, next) => {
  const t = await sequelize.transaction();
  try {
    const leave = await Leave.findByPk(req.params.id);
    if (!leave) throw new Error("Leave not found");
```

```
    leave.status = "APPROVED";
    await leave.save({ transaction: t });

    await t.commit();
    res.json({ message: "Leave approved" });
  } catch (err) {
    await t.rollback();
    next(err);
  }
};

exports.searchLeaves = async (req, res, next) => {
  try {
    const { status } = req.query;
    const leaves = await Leave.findAll({
      where: { ...(status && { status }) },
      include: Employee
    });

    res.json(leaves);
  } catch (err) {
    next(err);
  }
};
```

6. Routes

employee.routes.js

```
const router = require("express").Router();
const controller = require("../controllers/
employee.controller");

router.post("/", controller.createEmployee);

module.exports = router;
```

leave.routes.js

```
const router = require("express").Router();
const controller = require("../controllers/
leave.controller");
const auth = require("../middleware/auth");
```

```
router.post("/", controller.applyLeave);
router.put("/:id/approve", auth, controller.approveLeave);
router.get("/", controller.searchLeaves);

module.exports = router;
```

7. app.js

```
const express = require("express");
const logger = require("./middleware/logger");
const errorHandler = require("./middleware/errorHandler");

const employeeRoutes = require("./routes/employee.routes");
const leaveRoutes = require("./routes/leave.routes");

const app = express();
app.use(express.json());
app.use(logger);

app.use("/employees", employeeRoutes);
app.use("/leaves", leaveRoutes);

app.use(errorHandler);

module.exports = app;
```

8. server.js

```
const app = require("./app");
const { sequelize } = require("./models");

sequelize.sync().then(() => {
  console.log("DB Connected");
  app.listen(5000, () => console.log("Server running on
5000"));
});
```

9. Sample Inputs & Outputs

Create a new employee

POST `/employees`

Payload:

```
{
  "name": "Syed",
  "department": "Engineering"
}
```

Response:

```
{
  "id": 1,
  "name": "Syed",
  "department": "Engineering"
}
```

Apply for Leave

POST `/leaves`

Payload:

```
{
  "employeeId": 1,
  "fromDate": "2025-01-01",
  "toDate": "2025-01-03"
}
```

Response:

```
{
  "id": 1,
  "employeeId": 1,
  "fromDate": "2025-01-01T00:00:00.000Z",
  "toDate": "2025-01-03T00:00:00.000Z",
  "status": "PENDING"
}
```

Approve Leave

PUT `/leaves/1/approve`

Headers:

```
x-auth-token: manager123
```

Response:

```
{ "message": "Leave approved" }
```

Search Leaves

GET `/leaves?status=APPROVED`

Response:

```
[
  {
    "id": 1,
    "status": "APPROVED",
    "Employee": { "id": 1, "name": "Syed" }
  }
]
```

Online Course Enrollment API

This example includes:

✔ Project description
✔ Requirements
✔ Project structure
✔ Sequelize models
✔ Advanced routing
✔ Custom middleware (logger, token auth, validation)
✔ Controllers
✔ Transactions
✔ Error handling
✔ Sample inputs & outputs
✔ Clear explanation of each part

1. Scenario Description

A training platform wants a backend system that allows:

- Students to register

- Instructors to create courses
- Students to enroll in courses
- Students to view their enrolled courses
- Managers to approve or reject enrollments
- Filtering enrollments using query parameters

Your job is to design backend APIs using:

- Express.js
- Sequelize ORM
- MySQL
- Middleware (logging → validation → token auth → route)
- Advanced routing (route params, query)
- Transactions for enrollment approval workflow

Role-based auth requirement:
Manager must use token `"manager-token-123"` for approving enrollments.

2. Project Structure

```
online-course-api/
├── config/
│   └── db.js
├── middleware/
│   ├── logger.js
│   ├── validate.js
│   ├── auth.js
│   └── errorHandler.js
├── models/
│   ├── index.js
│   ├── student.js
│   ├── course.js
│   └── enrollment.js
├── routes/
│   ├── student.routes.js
│   ├── course.routes.js
│   └── enrollment.routes.js
├── controllers/
```

```
│       ├── student.controller.js
│       ├── course.controller.js
│       └── enrollment.controller.js
├── app.js
└── server.js
```

3. Sequelize Models

student.js

```
module.exports = (sequelize, DataTypes) => {
  const Student = sequelize.define("Student", {
    name: { type: DataTypes.STRING, allowNull: false },
    email: { type: DataTypes.STRING, unique: true }
  });

  Student.associate = (models) => {
    Student.hasMany(models.Enrollment, { foreignKey:
"studentId" });
  };

  return Student;
};
```

course.js

```
module.exports = (sequelize, DataTypes) => {
  const Course = sequelize.define("Course", {
    title: DataTypes.STRING,
    instructor: DataTypes.STRING
  });

  Course.associate = (models) => {
    Course.hasMany(models.Enrollment, { foreignKey:
"courseId" });
  };

  return Course;
};
```

enrollment.js

```
module.exports = (sequelize, DataTypes) => {
  const Enrollment = sequelize.define("Enrollment", {
    status: {
      type: DataTypes.STRING,
      defaultValue: "PENDING"
    }
  });

  Enrollment.associate = (models) => {
    Enrollment.belongsTo(models.Student, { foreignKey:
"studentId" });
    Enrollment.belongsTo(models.Course, { foreignKey:
"courseId" });
  };

  return Enrollment;
};
```

4. Middleware

logger.js

```
module.exports = (req, res, next) => {
  const time = new Date().toISOString();
  const route = req.originalUrl;
  const method = req.method;
  const token = req.headers["x-auth-token"] ? "***" : "NONE";

  console.log(`[${time}] ${method} ${route} - token: ${token}
`);
  next();
};
```

auth.js

Used only for actions requiring a manager.

```
module.exports = (req, res, next) => {
  const token = req.headers["x-auth-token"];
  if (token !== "manager-token-123")
```

```
    return res.status(401).json({ error: "Unauthorized
manager access" });

  next();
};
```

validate.js

Checks required fields.

```
module.exports = (fields) => {
  return (req, res, next) => {
    for (let f of fields) {
      if (!req.body[f]) {
        return res.status(400).json({ error: `${f} is
required` });
      }
    }
    next();
  };
};
```

errorHandler.js

```
module.exports = (err, req, res, next) => {
  console.error("Error:", err.message);
  res.status(500).json({ error: err.message });
};
```

5. Controllers

student.controller.js

```
const { Student } = require("../models");

exports.createStudent = async (req, res, next) => {
  try {
    const student = await Student.create(req.body);
    res.status(201).json(student);
  } catch (err) {
```

```
    next(err);
  }
};
```

course.controller.js

```
const { Course } = require("../models");

exports.createCourse = async (req, res, next) => {
  try {
    const course = await Course.create(req.body);
    res.status(201).json(course);
  } catch (err) {
    next(err);
  }
};
```

enrollment.controller.js

```
const { Enrollment, Student, Course, sequelize } =
require("../models");

// Student enrolls for a course
exports.enroll = async (req, res, next) => {
  try {
    const enrollment = await Enrollment.create({
      studentId: req.body.studentId,
      courseId: req.body.courseId
    });

    res.status(201).json(enrollment);
  } catch (err) {
    next(err);
  }
};

// Manager approves enrollment
exports.approveEnrollment = async (req, res, next) => {
  const t = await sequelize.transaction();

  try {
```

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

    if (!enrollment) throw new Error("Enrollment not found");

    enrollment.status = "APPROVED";
    await enrollment.save({ transaction: t });

    await t.commit();
    res.json({ message: "Enrollment approved" });
  } catch (err) {
    await t.rollback();
    next(err);
  }
};

// Query filter based on status
exports.getEnrollments = async (req, res, next) => {
  try {
    const { status } = req.query;

    const enrollments = await Enrollment.findAll({
      where: { ...(status && { status }) },
      include: [Student, Course],
    });

    res.json(enrollments);
  } catch (err) {
    next(err);
  }
};
```

Routes

student.routes.js

```
const router = require("express").Router();
const validate = require("../middleware/validate");
const controller = require("../controllers/
student.controller");
```

```
router.post("/", validate(["name", "email"]),
controller.createStudent);

module.exports = router;
```

course.routes.js

```
const router = require("express").Router();
const validate = require("../middleware/validate");
const controller = require("../controllers/
course.controller");

router.post("/", validate(["title", "instructor"]),
controller.createCourse);

module.exports = router;
```

enrollment.routes.js

```
const router = require("express").Router();
const validate = require("../middleware/validate");
const auth = require("../middleware/auth");
const controller = require("../controllers/
enrollment.controller");

router.post("/", validate(["studentId", "courseId"]),
controller.enroll);
router.put("/:id/approve", auth,
controller.approveEnrollment);
router.get("/", controller.getEnrollments);

module.exports = router;
```

7. app.js

```
const express = require("express");
const logger = require("./middleware/logger");
const errorHandler = require("./middleware/errorHandler");

const studentRoutes = require("./routes/student.routes");
const courseRoutes = require("./routes/course.routes");
```

```
const enrollmentRoutes = require("./routes/
enrollment.routes");

const app = express();
app.use(express.json());
app.use(logger);

app.use("/students", studentRoutes);
app.use("/courses", courseRoutes);
app.use("/enrollments", enrollmentRoutes);

app.use(errorHandler);

module.exports = app;
```

8. server.js

```
const app = require("./app");
const { sequelize } = require("./models");

sequelize.sync().then(() => {
  console.log("DB Connected");
  app.listen(5000, () => console.log("Server running on
5000"));
});
```

9. Sample Inputs & Outputs

Create a Student

POST `/students`

Request

```
{
  "name": "David",
  "email": "david@gmail.com"
}
```

Response

```
{
  "id": 1,
  "name": "David",
  "email": "david@gmail.com"
}
```

Create a Course

POST `/courses`

Request

```
{
  "title": "Node.js Masterclass",
  "instructor": "John Watson"
}
```

Response

```
{
  "id": 1,
  "title": "Node.js Masterclass",
  "instructor": "John Watson"
}
```

Student Enrolls in Course

POST `/enrollments`

Request

```
{
  "studentId": 1,
  "courseId": 1
}
```

Response

```
{
  "id": 1,
  "studentId": 1,
  "courseId": 1,
  "status": "PENDING"
}
```

Manager Approves Enrollment

PUT `/enrollments/1/approve`

Header:

```
x-auth-token: manager-token-123
```

Response

```
{ "message": "Enrollment approved" }
```

Filter Enrollments

GET `/enrollments?status=APPROVED`

Response

```
[
  {
    "id": 1,
    "status": "APPROVED",
    "Student": {
      "id": 1,
      "name": "David"
    },
    "Course": {
      "id": 1,
      "title": "Node.js Masterclass"
    }
  }
]
```

1. Install Dependencies

Run inside your project folder:

```
npm install express sequelize mysql2
```

For development:

```
npm install --save-dev nodemon
```

2. Start MySQL Server

Make sure MySQL is running locally.

Create database:

```
CREATE DATABASE online_course_db;
```

Update your `config/db.js` connection details if needed.

3. Run the Server

Option A — Using Node

```
node server.js
```

Option B — Using Nodemon (Auto-restart on changes)

Add this script in `package.json`:

```
"scripts": {
  "dev": "nodemon server.js"
}
```

Then run:

```
npm run dev
```

Expected Output in Terminal

```
DB Connected
Server running on 5000
```

Test APIs Using Postman / Thunder Client / Swagger

Example:

```
POST http://localhost:5000/students
```

Install ONLY the migration-related packages

1. Install Sequelize CLI (local)

```
npm install --save-dev sequelize-cli
```

2. Install Sequelize + MySQL adapter (required for migrations to run)

```
npm install sequelize mysql2
```

These are the **minimum required packages** for migrations to work.

After installation — create migration folders

```
npx sequelize-cli init
```

This creates:

```
config/
models/
migrations/
seeders/
```

Create a migration

Example:

```
npx sequelize-cli migration:generate --name create-students
```

Run migrations

```
npx sequelize-cli db:migrate
```

What is the Use of Migration Files?

Migration files are **versioned instructions** that tell Sequelize:

✔ What to create
✔ What to update
✔ What to remove
✔ How to revert changes

They manage **database schema changes** over time.

Database Version Control

A migration file works like **Git for your database**.

Example:

- Day 1 → Create `students` table
- Day 5 → Add `phone` column
- Day 10 → Add `status` column

Every change is saved in a migration file.

This lets your entire team have the **same database structure** without manually writing SQL.