Node.js + Express + MySQL + Sequelize Demo Project (TypeScript)

---

1. Project Setup

```
mkdir node-sequelize-ts-demo
cd node-sequelize-ts-demo
npm init -y
npm install express sequelize mysql2 dotenv
npm install -D typescript ts-node @types/node @types/express nodemon
npx tsc --init
```

package.json scripts:

```
"scripts": {
  "dev": "nodemon src/index.ts"
}
```

---

2. Folder Structure

```
src/
├─ config/
│  └─ db.config.ts
├─ controllers/
│  └─ user.controller.ts
├─ models/
│  └─ user.model.ts
├─ routes/
│  └─ user.routes.ts
├─ usecases/
│  └─ createUser.usecase.ts
└─ index.ts
.env
```

---

3. Database Configuration

.env:

```
DB_HOST=localhost
DB_USER=root
DB_PASSWORD=yourpassword
DB_NAME=demo_db
DB_PORT=3306
PORT=3000
```

src/config/db.config.ts:

```ts
import { Sequelize } from "sequelize";
import dotenv from "dotenv";
dotenv.config();

export const sequelize = new Sequelize(
  process.env.DB_NAME!,
  process.env.DB_USER!,
  process.env.DB_PASSWORD!,
  {
    host: process.env.DB_HOST!,
    dialect: "mysql",
    logging: false,
  }
);

export const testConnection = async () => {
  try {
    await sequelize.authenticate();
    console.log("Database connected successfully.");
  } catch (error) {
    console.error("Unable to connect to the database:", error);
  }
};
```

---

4. Model

src/models/user.model.ts:

```ts
import { DataTypes, Model } from "sequelize";
import { sequelize } from "../config/db.config";

export class User extends Model {
  public id!: number;
```

```
  public firstName!: string;
  public lastName!: string;
  public email!: string;

  public readonly createdAt!: Date;
  public readonly updatedAt!: Date;
}

User.init(
  {
    id: {
      type: DataTypes.INTEGER.UNSIGNED,
      autoIncrement: true,
      primaryKey: true,
    },
    firstName: {
      type: DataTypes.STRING,
      allowNull: false,
    },
    lastName: {
      type: DataTypes.STRING,
      allowNull: false,
    },
    email: {
      type: DataTypes.STRING,
      allowNull: false,
      unique: true,
    },
  },
  {
    tableName: "users",
    sequelize,
  }
);
```

---

5. Use Case

src/usecases/createUser.usecase.ts:

```
import { User } from "../models/user.model";

interface CreateUserDTO {
  firstName: string;
  lastName: string;
```

```
  email: string;
}

export class CreateUserUseCase {
  async execute(data: CreateUserDTO): Promise<User> {
    const existing = await User.findOne({ where: { email: data.email } });
    if (existing) throw new Error("User already exists");

    const user = await User.create(data);
    return user;
  }
}
```

---

6. Controller

src/controllers/user.controller.ts:

```
import { Request, Response } from "express";
import { CreateUserUseCase } from "../usecases/createUser.usecase";
import { User } from "../models/user.model";

const createUserUseCase = new CreateUserUseCase();

export class UserController {
  async createUser(req: Request, res: Response) {
    try {
      const { firstName, lastName, email } = req.body;
      const user = await createUserUseCase.execute({ firstName, lastName,
email });
      res.status(201).json(user);
    } catch (error: any) {
      res.status(400).json({ message: error.message });
    }
  }

  async getAllUsers(req: Request, res: Response) {
    try {
      const users: User[] = await User.findAll();
      res.status(200).json(users);
    } catch (error: any) {
      res.status(500).json({ message: error.message });
    }
```

```
  }
}
```

---

7. Routes

src/routes/user.routes.ts:

```
import { Router } from "express";
import { UserController } from "../controllers/user.controller";

const router = Router();
const controller = new UserController();

router.post("/users", controller.createUser.bind(controller));
router.get("/users", controller.getAllUsers.bind(controller));

export default router;
```

---

8. Main Entry

src/index.ts:

```
import express from "express";
import dotenv from "dotenv";
import userRoutes from "./routes/user.routes";
import { sequelize, testConnection } from "./config/db.config";

dotenv.config();

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

app.use("/api", userRoutes);

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

app.listen(PORT, async () => {
  console.log(`Server running on http://localhost:${PORT}`);
  await testConnection();
  await sequelize.sync({ alter: true });
```

```
  console.log("Database synced.");
});
```

---

9. Testing

POST `http://localhost:3000/api/users`
Body:

```
{
  "firstName": "Alice",
  "lastName": "Smith",
  "email": "alice@example.com"
}
```

GET `http://localhost:3000/api/users`

Key Concepts

- **Model** → represents MySQL table.
- **Use Case** → business logic.
- **Controller** → HTTP request/response.
- **Sequelize** → ORM mapping.

Extensions / More Examples

- Add `UpdateUserUseCase`, `DeleteUserUseCase`.
- Implement transactions with Sequelize.
- Input validation with `Joi` or `express-validator`.
- Error handling middleware.
- Add additional models (e.g., Posts, Comments) with relationships.