Here's a **Node.js + TypeScript** project setup for a **Library Management System** with **Role-Based Authentication** using **Express, JWT, and SQL Server**.

### **Project Directory Structure**

library-management/

│── src/

│ ├── controllers/

│ │ ├── auth.controller.ts

│ │ ├── books.controller.ts

│ ├── middleware/

│ │ ├── auth.middleware.ts

│ ├── models/

│ │ ├── user.model.ts

│ │ ├── book.model.ts

│ ├── routes/

│ │ ├── auth.routes.ts

│ │ ├── book.routes.ts

│ ├── services/

│ │ ├── auth.service.ts

│ │ ├── book.service.ts

│ ├── utils/

│ │ ├── db.ts

│ │ ├── jwt.ts

│ ├── app.ts

│ ├── server.ts

│── .env

│── .gitignore

│── package.json

│── tsconfig.json

│── README.md

### **1. Install Dependencies**

Run the following commands to set up the project:

mkdir library-management && cd library-management

npm init -y

npm install express jsonwebtoken bcryptjs dotenv mssql

npm install -D typescript ts-node @types/node @types/express @types/jsonwebtoken @types/bcryptjs

### **2. Configure TypeScript (**tsconfig.json**)**

{

"compilerOptions": {

"outDir": "./dist",

"module": "CommonJS",

"target": "ES6",

"rootDir": "./src",

"strict": true

}

}

### **3. Configure Database Connection (**src/utils/db.ts**)**

import sql from 'mssql';

import dotenv from 'dotenv';

dotenv.config();

const dbConfig = {

user: process.env.DB\_USER,

password: process.env.DB\_PASS,

database: process.env.DB\_NAME,

server: process.env.DB\_SERVER,

options: {

encrypt: true,

trustServerCertificate: true,

},

};

export const connectDB = async () => {

try {

await sql.connect(dbConfig);

console.log("Connected to SQL Server");

} catch (error) {

console.error("Database connection failed:", error);

}

};

### **4. Authentication Middleware (**src/middleware/auth.middleware.ts**)**

import { Request, Response, NextFunction } from "express";

import jwt from "jsonwebtoken";

export const verifyToken = (roles: string[]) => {

return (req: Request, res: Response, next: NextFunction) => {

const token = req.header("Authorization")?.split(" ")[1];

if (!token) return res.status(401).json({ message: "Access Denied" });

try {

const decoded = jwt.verify(token, process.env.JWT\_SECRET!) as any;

if (!roles.includes(decoded.role)) {

return res.status(403).json({ message: "Forbidden" });

}

req.user = decoded;

next();

} catch (err) {

res.status(400).json({ message: "Invalid Token" });

}

};

};

### **5. Authentication Service (**src/services/auth.service.ts**)**

import bcrypt from "bcryptjs";

import jwt from "jsonwebtoken";

import sql from "mssql";

export const registerUser = async (username: string, password: string, role: string) => {

const hashedPassword = await bcrypt.hash(password, 10);

await sql.query`INSERT INTO Users (username, password, role) VALUES (${username}, ${hashedPassword}, ${role})`;

return { message: "User registered successfully" };

};

export const loginUser = async (username: string, password: string) => {

const result = await sql.query`SELECT \* FROM Users WHERE username = ${username}`;

const user = result.recordset[0];

if (!user || !(await bcrypt.compare(password, user.password))) {

throw new Error("Invalid credentials");

}

const token = jwt.sign({ id: user.id, role: user.role }, process.env.JWT\_SECRET!, { expiresIn: "1h" });

return { token };

};

### **6. Authentication Routes (**src/routes/auth.routes.ts**)**

import express from "express";

import { registerUser, loginUser } from "../services/auth.service";

const router = express.Router();

router.post("/register", async (req, res) => {

try {

const { username, password, role } = req.body;

const response = await registerUser(username, password, role);

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

} catch (error) {

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

}

});

router.post("/login", async (req, res) => {

try {

const { username, password } = req.body;

const response = await loginUser(username, password);

res.status(200).json(response);

} catch (error) {

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

}

});

export default router;

### **7. Book Routes (**src/routes/book.routes.ts**)**

import express from "express";

import { verifyToken } from "../middleware/auth.middleware";

const router = express.Router();

router.post("/add", verifyToken(["Admin"]), async (req, res) => {

res.send("Book added");

});

router.put("/update", verifyToken(["Admin"]), async (req, res) => {

res.send("Book updated");

});

router.delete("/remove", verifyToken(["Admin"]), async (req, res) => {

res.send("Book removed");

});

router.get("/search", verifyToken(["Student", "Admin"]), async (req, res) => {

res.send("Books fetched");

});

router.post("/borrow", verifyToken(["Student"]), async (req, res) => {

res.send("Book borrowed");

});

router.post("/return", verifyToken(["Student"]), async (req, res) => {

res.send("Book returned");

});

export default router;

### **8. Main App (**src/app.ts**)**

import express from "express";

import dotenv from "dotenv";

import { connectDB } from "./utils/db";

import authRoutes from "./routes/auth.routes";

import bookRoutes from "./routes/book.routes";

dotenv.config();

const app = express();

app.use(express.json());

app.use("/api/auth", authRoutes);

app.use("/api/books", bookRoutes);

connectDB();

export default app;

### **9. Server File (**src/server.ts**)**

import app from "./app";

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {

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

});

### **10. Environment Variables (**.env**)**

PORT=5000

JWT\_SECRET=your\_secret\_key

DB\_USER=your\_db\_user

DB\_PASS=your\_db\_password

DB\_NAME=your\_db\_name

DB\_SERVER=your\_db\_server

### **11. Run the Project**

1. **Compile TypeScript:**
2. npx tsc
3. **Run Server:**
4. node dist/server.js
5. **Use Postman or a frontend to test authentication and book management.**

### **Features Implemented**

✅ Role-based authentication using JWT  
✅ Secure password hashing using bcrypt  
✅ Authorization middleware for Admin & Student  
✅ API routes for user registration, login, and book management  
✅ SQL Server connection for user & book storage

This is a **scalable** and **secure** foundation for your **Library Management System** in **Node.js + TypeScript**. Let me know if you need any modifications!