Creating a production-ready RESTful API structure with authentication using TypeScript and MySQL involves several steps. Below is a basic scaffolding for such a project. This structure includes:

1. \*\*Project Setup\*\*

2. \*\*Database Configuration\*\*

3. \*\*Authentication (JWT)\*\*

4. \*\*Routing\*\*

5. \*\*Controllers\*\*

6. \*\*Models\*\*

7. \*\*Middleware\*\*

8. \*\*Error Handling\*\*

9. \*\*Environment Configuration\*\*

### 1. Project Setup

First, initialize a new Node.js project and install the necessary dependencies.

```bash

mkdir my-rest-api

cd my-rest-api

npm init -y

npm install express body-parser cors dotenv jsonwebtoken bcryptjs mysql2

npm install --save-dev typescript @types/node @types/express @types/cors @types/jsonwebtoken ts-node nodemon

npx tsc --init

```

### 2. Directory Structure

```

my-rest-api/

│

├── src/

│ ├── config/

│ │ └── db.ts

│ ├── controllers/

│ │ ├── authController.ts

│ │ └── userController.ts

│ ├── middleware/

│ │ ├── authMiddleware.ts

│ │ └── errorHandler.ts

│ ├── models/

│ │ └── userModel.ts

│ ├── routes/

│ │ ├── authRoutes.ts

│ │ └── userRoutes.ts

│ ├── utils/

│ │ └── jwtUtils.ts

│ ├── app.ts

│ └── server.ts

├── .env

├── .gitignore

├── package.json

├── tsconfig.json

└── README.md

```

### 3. Database Configuration

Create a `db.ts` file in the `config` directory to handle MySQL connections.

```typescript

// src/config/db.ts

import mysql from 'mysql2/promise';

import dotenv from 'dotenv';

dotenv.config();

const pool = mysql.createPool({

host: process.env.DB\_HOST,

user: process.env.DB\_USER,

password: process.env.DB\_PASSWORD,

database: process.env.DB\_NAME,

waitForConnections: true,

connectionLimit: 10,

queueLimit: 0

});

export default pool;

```

### 4. Authentication (JWT)

Create a utility for handling JWT tokens.

```typescript

// src/utils/jwtUtils.ts

import jwt from 'jsonwebtoken';

import dotenv from 'dotenv';

dotenv.config();

const SECRET\_KEY = process.env.JWT\_SECRET || 'your-secret-key';

export const generateToken = (userId: number): string => {

return jwt.sign({ userId }, SECRET\_KEY, { expiresIn: '1h' });

};

export const verifyToken = (token: string): any => {

return jwt.verify(token, SECRET\_KEY);

};

```

### 5. Models

Create a user model to interact with the database.

```typescript

// src/models/userModel.ts

import pool from '../config/db';

interface User {

id?: number;

username: string;

password: string;

email: string;

}

export const createUser = async (user: User): Promise<User> => {

const [result] = await pool.query(

'INSERT INTO users (username, password, email) VALUES (?, ?, ?)',

[user.username, user.password, user.email]

);

return { id: (result as any).insertId, ...user };

};

export const findUserByEmail = async (email: string): Promise<User | null> => {

const [rows] = await pool.query('SELECT \* FROM users WHERE email = ?', [email]);

return (rows as any)[0] || null;

};

```

### 6. Controllers

Create controllers for handling authentication and user-related requests.

```typescript

// src/controllers/authController.ts

import { Request, Response } from 'express';

import { generateToken } from '../utils/jwtUtils';

import { findUserByEmail } from '../models/userModel';

import bcrypt from 'bcryptjs';

export const login = async (req: Request, res: Response) => {

const { email, password } = req.body;

const user = await findUserByEmail(email);

if (!user) {

return res.status(401).json({ message: 'Invalid credentials' });

}

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

if (!isPasswordValid) {

return res.status(401).json({ message: 'Invalid credentials' });

}

const token = generateToken(user.id!);

res.json({ token });

};

export const register = async (req: Request, res: Response) => {

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

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

const user = await createUser({ username, password: hashedPassword, email });

const token = generateToken(user.id!);

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

};

```

### 7. Middleware

Create middleware for authentication and error handling.

```typescript

// src/middleware/authMiddleware.ts

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

import { verifyToken } from '../utils/jwtUtils';

export const authenticate = (req: Request, res: Response, next: NextFunction) => {

const token = req.header('Authorization')?.replace('Bearer ', '');

if (!token) {

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

}

try {

const decoded = verifyToken(token);

(req as any).user = decoded;

next();

} catch (error) {

res.status(400).json({ message: 'Invalid token.' });

}

};

```

```typescript

// src/middleware/errorHandler.ts

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

export const errorHandler = (err: Error, req: Request, res: Response, next: NextFunction) => {

console.error(err.stack);

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

};

```

### 8. Routing

Create routes for authentication and user-related endpoints.

```typescript

// src/routes/authRoutes.ts

import express from 'express';

import { login, register } from '../controllers/authController';

const router = express.Router();

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

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

export default router;

```

```typescript

// src/routes/userRoutes.ts

import express from 'express';

import { authenticate } from '../middleware/authMiddleware';

import { getUserProfile } from '../controllers/userController';

const router = express.Router();

router.get('/profile', authenticate, getUserProfile);

export default router;

```

### 9. App and Server Configuration

Create the main application and server files.

```typescript

// src/app.ts

import express from 'express';

import cors from 'cors';

import bodyParser from 'body-parser';

import authRoutes from './routes/authRoutes';

import userRoutes from './routes/userRoutes';

import { errorHandler } from './middleware/errorHandler';

const app = express();

app.use(cors());

app.use(bodyParser.json());

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

app.use('/api/users', userRoutes);

app.use(errorHandler);

export default app;

```

```typescript

// src/server.ts

import app from './app';

import dotenv from 'dotenv';

dotenv.config();

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

app.listen(PORT, () => {

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

});

```

### 10. Environment Configuration

Create a `.env` file to store environment variables.

```

DB\_HOST=localhost

DB\_USER=root

DB\_PASSWORD=yourpassword

DB\_NAME=mydatabase

JWT\_SECRET=your-secret-key

PORT=3000

```

### 11. Start the Server

Add the following scripts to your `package.json`:

```json

"scripts": {

"start": "node dist/server.js",

"build": "tsc",

"dev": "nodemon --exec ts-node src/server.ts"

}

```

Now, you can start the server in development mode:

```bash

npm run dev

```

Or build and start the server in production mode:

```bash

npm run build

npm start

```

### Conclusion

This scaffolding provides a basic structure for a production-ready RESTful API with authentication using TypeScript and MySQL. You can expand upon this by adding more features, such as role-based access control, logging, and more sophisticated error handling.

Here’s a `README.md` file for the project described above. This file provides an overview of the project, setup instructions, and usage details.

```markdown

# RESTful API with Authentication (TypeScript + MySQL)

This is a production-ready RESTful API scaffolding built with \*\*TypeScript\*\*, \*\*Express\*\*, and \*\*MySQL\*\*. It includes authentication using \*\*JSON Web Tokens (JWT)\*\* and follows best practices for structuring a scalable and maintainable API.

---

## Features

- \*\*Authentication\*\*: JWT-based authentication for secure user login and registration.

- \*\*Database\*\*: MySQL database integration using `mysql2` for efficient query execution.

- \*\*TypeScript\*\*: Full TypeScript support for type safety and better developer experience.

- \*\*Environment Configuration\*\*: Uses `dotenv` for managing environment variables.

- \*\*Error Handling\*\*: Centralized error handling middleware for consistent error responses.

- \*\*Middleware\*\*: Authentication middleware to protect routes.

- \*\*Scalable Structure\*\*: Modular and organized codebase for easy scaling.

---

## Prerequisites

Before running the project, ensure you have the following installed:

- \*\*Node.js\*\* (v16 or higher)

- \*\*MySQL\*\* (or a MySQL-compatible database)

- \*\*npm\*\* (Node Package Manager)

---

## Setup Instructions

### 1. Clone the Repository

```bash

git clone https://github.com/your-username/my-rest-api.git

cd my-rest-api

```

### 2. Install Dependencies

```bash

npm install

```

### 3. Set Up Environment Variables

Create a `.env` file in the root directory and add the following variables:

```env

DB\_HOST=localhost

DB\_USER=your-database-username

DB\_PASSWORD=your-database-password

DB\_NAME=your-database-name

JWT\_SECRET=your-secret-key

PORT=3000

```

Replace the placeholders with your actual database credentials and JWT secret.

### 4. Set Up the Database

Create a `users` table in your MySQL database. Run the following SQL query:

```sql

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(255) NOT NULL,

password VARCHAR(255) NOT NULL,

email VARCHAR(255) NOT NULL UNIQUE

);

```

### 5. Run the Application

#### Development Mode

```bash

npm run dev

```

This will start the server using `nodemon` and `ts-node` for hot-reloading.

#### Production Mode

1. Build the TypeScript project:

```bash

npm run build

```

2. Start the server:

```bash

npm start

```

The server will start on the port specified in the `.env` file (default: `3000`).

---

## API Endpoints

### Authentication

- \*\*POST `/api/auth/register`\*\*: Register a new user.

- Request Body:

```json

{

"username": "john\_doe",

"password": "password123",

"email": "john@example.com"

}

```

- Response:

```json

{

"token": "jwt-token"

}

```

- \*\*POST `/api/auth/login`\*\*: Log in an existing user.

- Request Body:

```json

{

"email": "john@example.com",

"password": "password123"

}

```

- Response:

```json

{

"token": "jwt-token"

}

```

### User

- \*\*GET `/api/users/profile`\*\*: Get the authenticated user's profile.

- Headers:

```json

{

"Authorization": "Bearer jwt-token"

}

```

- Response:

```json

{

"id": 1,

"username": "john\_doe",

"email": "john@example.com"

}

```

---

## Project Structure

```

my-rest-api/

├── src/

│ ├── config/ # Database configuration

│ ├── controllers/ # Route controllers

│ ├── middleware/ # Custom middleware

│ ├── models/ # Database models

│ ├── routes/ # API routes

│ ├── utils/ # Utility functions

│ ├── app.ts # Express app configuration

│ └── server.ts # Server entry point

├── .env # Environment variables

├── .gitignore # Git ignore file

├── package.json # Node.js dependencies and scripts

├── tsconfig.json # TypeScript configuration

└── README.md # Project documentation

```

---

## Technologies Used

- \*\*Node.js\*\*: Runtime environment for the API.

- \*\*Express\*\*: Web framework for building the API.

- \*\*TypeScript\*\*: Adds static typing to JavaScript.

- \*\*MySQL\*\*: Relational database for storing data.

- \*\*JWT\*\*: JSON Web Tokens for authentication.

- \*\*Bcrypt\*\*: Password hashing for security.

- \*\*Dotenv\*\*: Manages environment variables.

- \*\*CORS\*\*: Enables cross-origin resource sharing.

---

## Contributing

Contributions are welcome! If you find any issues or have suggestions for improvement, please open an issue or submit a pull request.

---

## License

This project is licensed under the MIT License. See the [LICENSE](LICENSE) file for details.

---

## Author

[Your Name](https://github.com/your-username)

```

### How to Use the README

1. Replace placeholders like `your-username`, `your-database-username`, `your-database-password`, etc., with actual values.

2. Add a `LICENSE` file if you want to include licensing information.

3. Update the \*\*Author\*\* section with your name and GitHub profile link.

This `README.md` provides a comprehensive guide for anyone looking to use or contribute to your project.