Backend(nodejs+sql server)

Install Dependencies

Run the following command to install required packages:

```
npm install express cors dotenv bcryptjs jsonwebtoken typeorm reflect-metadata mssql class-validator class-transformer

npm install --save-dev @types/express @types/bcryptjs @types/jsonwebtoken @types/node
```

Configure TypeORM for SQL Server

Create a **`src/config/database.ts`** file:

```
import "reflect-metadata";
import { DataSource } from "typeorm";
import dotenv from "dotenv";

dotenv.config();

export const AppDataSource = new DataSource({
  type: "mssql",
  host: process.env.DB_HOST,
  port: Number(process.env.DB_PORT) || 1433,
  username: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
  synchronize: true, // Set to false in production
  logging: true,
  entities: ["src/models/*.ts"],
  options: {
    encrypt: false,
  },
});
```

Create a `.env` file with your SQL Server credentials:

```
PORT=5000
DB_HOST=localhost
DB_PORT=1433
DB_USER=sa
DB_PASSWORD=yourStrongPassword
DB_NAME=UserDB
JWT_SECRET=your_jwt_secret_key
```

Create User Entity

Create **`src/models/User.ts`**:

```
import { Entity, PrimaryGeneratedColumn, Column } from "typeorm";
import { Exclude } from "class-transformer";

@Entity()
export class User {
  @PrimaryGeneratedColumn()
  id: number;

  @Column({ unique: true })
  email: string;

  @Column()
  @Exclude()
  password: string;

  @Column({ nullable: true })
  name: string;

  @Column({ nullable: true })
  profilePicture: string;

  @Column({ default: "user" })
  role: string;
}
```

Create User DTO (Validation)

Create **`src/dtos/UserDto.ts`**:

```
import { IsEmail, IsNotEmpty, MinLength, IsOptional } from "class-
validator";

export class RegisterDto {
  @IsNotEmpty({ message: "Email is required" })
  @IsEmail({}, { message: "Invalid email format" })
  email: string;

  @IsNotEmpty({ message: "Password is required" })
  @MinLength(6, { message: "Password must be at least 6 characters long" })
  password: string;

  @IsOptional()
  name?: string;

  @IsOptional()
  profilePicture?: string;
}

export class LoginDto {
  @IsNotEmpty()
  @IsEmail()
  email: string;

  @IsNotEmpty()
  password: string;
}
```

Create User Repository

Create **`src/repositories/UserRepository.ts`**:

```
import { AppDataSource } from "../config/database";
import { User } from "../models/User";

export const UserRepository = AppDataSource.getRepository(User);
```

Create User Service

Create **`src/services/UserService.ts`**:

```
import bcrypt from "bcryptjs";
import jwt from "jsonwebtoken";
import { User } from "../models/User";
import { UserRepository } from "../repositories/UserRepository";
import { RegisterDto, LoginDto } from "../dtos/UserDto";
import dotenv from "dotenv";

dotenv.config();

export class UserService {
  static async register(data: RegisterDto) {
    const { email, password, name, profilePicture } = data;

    const existingUser = await UserRepository.findOne({ where: { email }
});
    if (existingUser) throw new Error("User already exists");

    const hashedPassword = await bcrypt.hash(password, 10);
    const newUser = UserRepository.create({ email, password:
hashedPassword, name, profilePicture });

    return await UserRepository.save(newUser);
  }

  static async login(data: LoginDto) {
    const { email, password } = data;
    const user = await UserRepository.findOne({ where: { email } });

    if (!user || !(await bcrypt.compare(password, user.password))) {
      throw new Error("Invalid email or password");
    }

    const token = jwt.sign({ userId: user.id }, process.env.JWT_SECRET!, {
expiresIn: "1h" });
    return { token, user };
  }

  static async getProfile(userId: number) {
    return await UserRepository.findOne({ where: { id: userId } });
  }

  static async updateProfile(userId: number, data: Partial<User>) {
    await UserRepository.update(userId, data);
    return await UserRepository.findOne({ where: { id: userId } });
```

```
  }
}
```

Create User Controller

Create **`src/controllers/UserController.ts`**:

```
import { Request, Response } from "express";
import { UserService } from "../services/UserService";
import { RegisterDto, LoginDto } from "../dtos/UserDto";
import { validate } from "class-validator";

export class UserController {
  static async register(req: Request, res: Response) {
    const dto = new RegisterDto();
    Object.assign(dto, req.body);

    const errors = await validate(dto);
    if (errors.length > 0) return res.status(400).json(errors);

    try {
      const user = await UserService.register(dto);
      res.status(201).json({ message: "User registered", user });
    } catch (error) {
      res.status(400).json({ error: error.message });
    }
  }

  static async login(req: Request, res: Response) {
    const dto = new LoginDto();
    Object.assign(dto, req.body);

    const errors = await validate(dto);
    if (errors.length > 0) return res.status(400).json(errors);

    try {
      const result = await UserService.login(dto);
      res.json(result);
    } catch (error) {
      res.status(400).json({ error: error.message });
    }
  }

  static async getProfile(req: Request, res: Response) {
    const user = await UserService.getProfile(req.user.userId);
    res.json(user);
  }

  static async updateProfile(req: Request, res: Response) {
    const user = await UserService.updateProfile(req.user.userId,
req.body);
    res.json(user);
  }
}
```

Create Authentication Middleware

Create **src/middleware/authMiddleware.ts**:

```
import { Request, Response, NextFunction } from "express";
import jwt from "jsonwebtoken";

export const authMiddleware = (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 { userId:
number };
    req.user = decoded;
    next();
  } catch (error) {
    res.status(400).json({ message: "Invalid token" });
  }
};
```

Define Routes

Create **src/routes/UserRoutes.ts**:

```
import express from "express";
import { UserController } from "../controllers/UserController";
import { authMiddleware } from "../middleware/authMiddleware";

const router = express.Router();

router.post("/register", UserController.register);
router.post("/login", UserController.login);
router.get("/profile", authMiddleware, UserController.getProfile);
router.put("/profile", authMiddleware, UserController.updateProfile);

export default router;
```

FrontEnd

1 Setup Your Angular Project

Run the following command to create a new Angular project:

```
ng new user-management
cd user-management
ng add @angular/material  # Add Material UI
ng add ngx-toastr         # Notifications
npm install axios jwt-decode # HTTP & JWT decoding

// src/app/models/user.model.ts
export interface User {
```

```
  id?: number;
  email: string;
  password?: string;
  name?: string;
  profilePicture?: string;
  role?: string;
}

// src/app/services/auth.service.ts
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { User } from '../models/user.model';

@Injectable({ providedIn: 'root' })
export class AuthService {
  private apiUrl = 'http://localhost:5000';

  constructor(private http: HttpClient) {}

  register(user: User): Observable<any> {
    return this.http.post(`${this.apiUrl}/register`, user);
  }

  login(email: string, password: string): Observable<any> {
    return this.http.post(`${this.apiUrl}/login`, { email, password });
  }

  getProfile(): Observable<User> {
    return this.http.get<User>(`${this.apiUrl}/profile`, {
      headers: { Authorization: `Bearer ${localStorage.getItem('token')}`
},
    });
  }

  updateProfile(user: Partial<User>): Observable<User> {
    return this.http.put<User>(`${this.apiUrl}/profile`, user, {
      headers: { Authorization: `Bearer ${localStorage.getItem('token')}`
},
    });
  }
}

// src/app/components/register/register.component.ts

import { Component } from '@angular/core';
import { AuthService } from '../../services/auth.service';
import { Router } from '@angular/router';

@Component({ selector: 'app-register', templateUrl:
'./register.component.html' })
export class RegisterComponent {
  user = { email: '', password: '', name: '', profilePicture: '' };

  constructor(private authService: AuthService, private router: Router) {}

  register() {
    this.authService.register(this.user).subscribe(() => {
      alert('Registration successful');
      this.router.navigate(['/login']);
    }, error => alert(error.error.message));
```

```
  }
}
```

// src/app/components/login/login.component.ts

```
import { Component } from '@angular/core';
import { AuthService } from '../../services/auth.service';
import { Router } from '@angular/router';

@Component({ selector: 'app-login', templateUrl: './login.component.html'
})
export class LoginComponent {
  credentials = { email: '', password: '' };

  constructor(private authService: AuthService, private router: Router) {}

  login() {
    this.authService.login(this.credentials.email,
this.credentials.password).subscribe((res) => {
      localStorage.setItem('token', res.token);
      alert('Login successful');
      this.router.navigate(['/profile']);
    }, error => alert(error.error.message));
  }
}
```

// src/app/components/profile/profile.component.ts

```
import { Component, OnInit } from '@angular/core';
import { AuthService } from '../../services/auth.service';
import { User } from '../../models/user.model';

@Component({ selector: 'app-profile', templateUrl:
'./profile.component.html' })
export class ProfileComponent implements OnInit {
  user: User | null = null;

  constructor(private authService: AuthService) {}

  ngOnInit() {
    this.authService.getProfile().subscribe(user => {
      this.user = user;
    });
  }
}
```

// src/app/app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { RegisterComponent } from
'./components/register/register.component';
import { LoginComponent } from './components/login/login.component';
import { ProfileComponent } from './components/profile/profile.component';

const routes: Routes = [
  { path: 'register', component: RegisterComponent },
  { path: 'login', component: LoginComponent },
  { path: 'profile', component: ProfileComponent },
  { path: '', redirectTo: '/login', pathMatch: 'full' }
];
```

```
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {}
```

// src/app/app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';
import { FormsModule } from '@angular/forms';
import { AppRoutingModule } from './app-routing.module';
import { RegisterComponent } from
'./components/register/register.component';
import { LoginComponent } from './components/login/login.component';
import { ProfileComponent } from './components/profile/profile.component';

@NgModule({
  declarations: [RegisterComponent, LoginComponent, ProfileComponent],
  imports: [BrowserModule, HttpClientModule, FormsModule,
AppRoutingModule],
  providers: [],
  bootstrap: [LoginComponent]
})
export class AppModule {}
```