Src/Auth/auth

Controller

“import { Controller, Post, Body } from '@nestjs/common';

import { AuthService } from './auth.service';

import { CreateUserDto } from '../user/dto/create-user.dto';

import { LoginUserDto } from '../user/dto/login-user.dto';

import { ApiTags, ApiOperation, ApiBody } from '@nestjs/swagger';

@ApiTags('auth') // Group routes under the "auth" tag

@Controller('auth')

export class AuthController {

  constructor(private readonly authService: AuthService) {}

  @ApiOperation({ summary: 'Register a new user' }) // Description for this endpoint

  @ApiBody({

    description: 'Details required to create a new user',

    type: CreateUserDto,

  }) // Define the expected request body

  @Post('register')

  async register(@Body() createUserDto: CreateUserDto) {

    return this.authService.register(createUserDto);

  }

  @ApiOperation({ summary: 'Login a user' }) // Description for this endpoint

  @ApiBody({

    description: 'User login credentials',

    type: LoginUserDto,

  }) // Define the expected request body

  @Post('login')

  async login(@Body() loginUserDto: LoginUserDto) {

    return this.authService.login(loginUserDto.email, loginUserDto.password);

  }

}

”

Service

“import { Injectable } from '@nestjs/common';

import { JwtService } from '@nestjs/jwt';

import \* as bcrypt from 'bcrypt';

import { UserService } from '../user/user.service';

@Injectable()

export class AuthService {

  constructor(

    private readonly userService: UserService,

    private readonly jwtService: JwtService,

  ) {

    console.log('UserService:', this.userService);

  }

  async register(userData: any): Promise<any> {

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

    userData.password = hashedPassword;

    const user = await this.userService.create(userData);

    const payload = { email: user.email, sub: user.\_id };

    return {

      access\_token: this.jwtService.sign(payload),

      id: user.\_id,

    };

  }

  async login(email: string, password: string): Promise<any> {

    const user = await this.userService.findByEmail(email);

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

      throw new Error('Invalid email or password');

    }

    const payload = { email: user.email, sub: user.\_id };

    return {

      access\_token: this.jwtService.sign(payload),

      \_id: user.\_id,

    };

  }

}

”

Module

“// auth.module.ts

import { Module } from '@nestjs/common';

import { AuthService } from './auth.service';

import { AuthController } from './auth.controller';

import { JwtModule } from '@nestjs/jwt';

import { PassportModule } from '@nestjs/passport';

import { JwtStrategy } from './jwt.strategy';

import { UserModule } from '../user/user.module'; // Import UserModule

@Module({

  imports: [

    UserModule, // Add UserModule here

    PassportModule,

    JwtModule.register({

      secret: 'jwt\_secret\_key',

      signOptions: { expiresIn: '1h' },

    }),

  ],

  controllers: [AuthController],

  providers: [AuthService, JwtStrategy],

})

export class AuthModule {}

”

Jwt-auth.guard

“import { Injectable } from '@nestjs/common';

import { AuthGuard } from '@nestjs/passport';

@Injectable()

export class JwtAuthGuard extends AuthGuard('jwt') {}

”

Jwt.strategy

“import { Injectable } from '@nestjs/common';

import { PassportStrategy } from '@nestjs/passport';

import { Strategy, ExtractJwt } from 'passport-jwt';

@Injectable()

export class JwtStrategy extends PassportStrategy(Strategy) {

  constructor() {

    super({

      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),

      secretOrKey: 'jwt\_secret\_key', // Update with your secret key

    });

  }

  async validate(payload: any) {

    console.log('Decoded Payload:', payload); // Log to verify the payload

    return { sub: payload?.sub, email: payload?.email };

  }

}

”

Common/filter/

Global-exception.filter

“import {

  ExceptionFilter,

  Catch,

  ArgumentsHost,

  //   HttpException,

} from '@nestjs/common';

@Catch()

export class GlobalExceptionFilter implements ExceptionFilter {

  catch(exception: any, host: ArgumentsHost) {

    const ctx = host.switchToHttp();

    const response = ctx.getResponse();

    const status = exception.status || 500;

    response.status(status).json({

      statusCode: status,

      message: exception.message || 'Internal Server Error',

    });

  }

}

”

Schemas/address

“import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';

import { Document } from 'mongoose';

import { ApiProperty } from '@nestjs/swagger';

@Schema()

export class Address extends Document {

  @ApiProperty({

    example: '123 Main St',

    description: 'The first line of the address',

  })

  @Prop({ required: true })

  line1: string;

  @ApiProperty({

    example: 'Apt 4B',

    description: 'The second line of the address',

  })

  @Prop()

  line2: string;

  @ApiProperty({

    example: 'Near City Mall',

    description: 'A landmark for the address',

  })

  @Prop()

  landmark: string;

  @ApiProperty({ example: '700001', description: 'The postal code' })

  @Prop({ required: true })

  pin: string;

  @ApiProperty({ example: 'Kolkata', description: 'The city of the address' })

  @Prop({ required: true })

  city: string;

  @ApiProperty({

    example: 'West Bengal',

    description: 'The state of the address',

  })

  @Prop({ required: true })

  state: string;

  @ApiProperty({ example: 'India', description: 'The country of the address' })

  @Prop({ required: true })

  country: string;

}

export const AddressSchema = SchemaFactory.createForClass(Address);

”

Schemas/user

“import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';

import mongoose, { Document } from 'mongoose';

import { Address } from './address.schema';

import { ApiProperty } from '@nestjs/swagger';

@Schema()

export class User extends Document {

  @ApiProperty({ example: 'John Doe', description: 'The name of the user' })

  @Prop({ required: true })

  name: string;

  @ApiProperty({

    example: 'john.doe@example.com',

    description: 'The unique email of the user',

  })

  @Prop({ required: true, unique: true })

  email: string;

  @ApiProperty({

    example: '+1234567890',

    description: 'The mobile number of the user',

  })

  @Prop({ required: true })

  mobile: string;

  @ApiProperty({

    example: 'StrongP@ssw0rd',

    description: 'The password of the user',

  })

  @Prop({ required: true })

  password: string;

  @ApiProperty({

    type: () => Address,

    description: 'The address associated with the user',

  })

  @Prop({ type: mongoose.Schema.Types.ObjectId, ref: () => Address }) // Referencing Address

  address: mongoose.Schema.Types.ObjectId; // Storing Address as an ObjectId

}

export const UserSchema = SchemaFactory.createForClass(User);

”

User/dto

Create-user

“export class CreateUserDto {

  name: string;

  email: string;

  mobile: string;

  password: string;

  address: {

    line1: string;

    line2?: string;

    landmark?: string;

    city: string;

    pin: string;

    state: string;

    country: string;

  };

}

”

Update-user

“export class UpdateUserDto {

  mobile?: string;

  password?: string;

  address?: {

    line1?: string;

    line2?: string;

    landmark?: string;

    city?: string;

    pin?: string;

    state?: string;

    country?: string;

  };

}

”

Login-user

“export class LoginUserDto {

  email: string;

  password: string;

}

”

User/

User.controller

“import {

  Controller,

  Patch,

  Body,

  UseGuards,

  Req,

  Get,

  Delete,

  Param,

} from '@nestjs/common';

import { JwtAuthGuard } from '../auth/jwt-auth.guard'; // Make sure to import the guard

import { UserService } from './user.service';

import { UpdateUserDto } from './dto/update-user.dto';

import {

  ApiTags,

  ApiOperation,

  ApiParam,

  ApiBearerAuth,

} from '@nestjs/swagger';

@ApiTags('users') // Group routes under the "users" tag

@Controller('user')

export class UserController {

  constructor(private readonly userService: UserService) {}

  @ApiOperation({ summary: 'Get all users' }) // Describe this route

  @Get()

  async findAll() {

    return this.userService.findAll();

  }

  @ApiOperation({ summary: 'Delete a user by ID' }) // Describe the delete route

  @ApiParam({

    name: 'id',

    description: 'The ID of the user to delete',

    type: String,

  }) // Document the route parameter

  @ApiBearerAuth() // Indicate that this route requires authentication

  @UseGuards(JwtAuthGuard)

  @Delete(':id')

  async delete(@Param('id') id: string) {

    return this.userService.delete(id);

  }

  @ApiOperation({ summary: 'Update user information' }) // Describe the update route

  @ApiBearerAuth() // Indicate that this route requires authentication

  @UseGuards(JwtAuthGuard)

  @Patch('update')

  async update(@Req() req, @Body() updateUserDto: UpdateUserDto) {

    console.log(req.user);

    return this.userService.update(req.user.sub, updateUserDto);

  }

}

User.service

“import { Injectable } from '@nestjs/common';

import { InjectModel } from '@nestjs/mongoose';

import { Model } from 'mongoose';

import { User } from '../schemas/user.schema';

import { Address } from '../schemas/address.schema';

import { CreateUserDto } from './dto/create-user.dto';

@Injectable()

export class UserService {

  constructor(

    @InjectModel(User.name) private readonly userModel: Model<User>,

    @InjectModel(Address.name) private readonly addressModel: Model<Address>, // Inject Address Model

  ) {}

  async findAll(): Promise<User[]> {

    return this.userModel.find().exec();

  }

  async delete(userId: string): Promise<User | null> {

    return this.userModel.findByIdAndDelete(userId).exec();

  }

  async create(createUserDto: CreateUserDto): Promise<User> {

    // Create Address first

    const address = await new this.addressModel(createUserDto.address).save();

    // Create User and link Address ObjectId

    const user = await new this.userModel({

      ...createUserDto,

      address: address.\_id,

    }).save();

    return user;

  }

  async update(userId: string, updateData: any): Promise<User | null> {

    return this.userModel

      .findByIdAndUpdate(userId, updateData, { new: true })

      .exec();

  }

  async updateAddress(userId: string, newAddressData: any): Promise<Address> {

    const user = await this.userModel.findById(userId).exec();

    if (!user) {

      throw new Error('User not found');

    }

    // Update the linked address

    const updatedAddress = await this.addressModel

      .findByIdAndUpdate(user.address, newAddressData, { new: true })

      .exec();

    return updatedAddress;

  }

  async findByEmail(email: string): Promise<User | null> {

    return this.userModel.findOne({ email }).exec();

  }

}

”

User.module

“import { Module } from '@nestjs/common';

import { UserService } from './user.service';

import { UserController } from './user.controller';

import { MongooseModule } from '@nestjs/mongoose';

import { User, UserSchema } from '../schemas/user.schema';

import { Address, AddressSchema } from '../schemas/address.schema';

@Module({

  imports: [

    MongooseModule.forFeature([

      { name: User.name, schema: UserSchema },

      { name: Address.name, schema: AddressSchema },

    ]),

  ],

  controllers: [UserController],

  providers: [UserService],

  exports: [UserService],

})

export class UserModule {}

”

App.controll

“import { Controller, Get } from '@nestjs/common';

import { AppService } from './app.service';

@Controller()

export class AppController {

  constructor(private readonly appService: AppService) {}

  @Get()

  getHello(): string {

    return this.appService.getHello();

  }

}

”

App.service

“import { Injectable } from '@nestjs/common';

@Injectable()

export class AppService {

  getHello(): string {

    return 'Hello World!';

  }

}

”  
app.module

“import { Module } from '@nestjs/common';

import { MongooseModule } from '@nestjs/mongoose';

import { AuthModule } from './auth/auth.module';

import { UserModule } from './user/user.module';

@Module({

  imports: [

    MongooseModule.forRoot(

      'mongodb+srv://anannyamsd7:Doramike39@job-api.o82gx.mongodb.net/?retryWrites=true&w=majority&appName=JOB-API',

    ),

    AuthModule,

    UserModule,

  ],

})

export class AppModule {}

”

”

“{

"name": "Raima Maitra",

"email": "rai.maitra@example.com",

"mobile": "9876543210",

"password": "1234",

"address": {

"line1": "123 AJC Bose Road",

"landmark": "Near Mall",

"city": "Kolkata",

"pin": "700020",

"state": "West Bengal",

"country": "India"

}

}”