Folders

* modules:

File dto: Nơi data đc truyền vào qua controller và validate data

File schema/entity: Model hoặc Table

File controller: Route của API (end point), gọi service

@Controller('users') // localhost:8080/api/v1/users

export class UsersController {

  constructor(private readonly usersService: UsersService) {}

  @Post()

  create(@Body() createUserDto: CreateUserDto) {

    return this.usersService.create(createUserDto);

  }

  @Get()

  findAll() {

    return this.usersService.findAll();

  }

File service: Giao tiếp với database

@Injectable()

export class UsersService {

  create(createUserDto: CreateUserDto) {

    return 'This action adds a new user';

  }

@Injectable(): Export và cho phép file khác dùng trực tiếp class

@Controller('todos')

export class TodosController {

  constructor(private readonly todosService: TodosService) {} // Inject

  @Post()

  create(@Body() createTodoDto: CreateTodoDto) {

    return this.todosService.create(createTodoDto);

  }

File module: Nơi tổng hợp service, controller, model, config,...

* imports: import các modules từ folder khác: Thư viện, model,… để gọi trong module này
* controllers: Controller của bản thân module
* providers: Các service của bản thân module (Có thể là service của modules khác đc dùng trong service này)

@Module({

  imports: [

    MongooseModule.forFeature([{ name: User.name, schema: UserSchema }]),

  ],

  controllers: [UsersController],

  providers: [UsersService],

})

export class UsersModule { }

| **Decorator** | **Mô tả** |
| --- | --- |
| @Body() | Lấy dữ liệu từ body của request |
| @Param(paramName?) | Lấy route parameter (ví dụ: /users/:id) |
| @Query(paramName?) | Lấy dữ liệu từ query string (ví dụ: /users?page=2) |
| @Headers(headerName?) | Lấy dữ liệu từ HTTP headers |
| @Req() | Truy cập trực tiếp Request object (Express) |
| @Res() | Truy cập trực tiếp Response object |
| @Next() | Truy cập next() middleware function |

# Default value

    @Prop({ default: false })

    isChecked: boolean;

# Time stamp

@Schema({ timestamps: true })

**CRUD CLI**

# Tạo REST API:

*npx @nestjs/cli g resource users*

Hoặc

*nesjt g resource todos*

*Nên thêm timestamps vào*

@Schema({ timestamps: true })

**Configuration (.env)**

# .env global

*app.modules.ts*

    ConfigModule.forRoot({

      isGlobal: true,

    }),

* *Ở module khác ko cần import ConfigModule nữa*

*JwtModule*.registerAsync({

      imports: [ConfigModule], // No need ConfigModule

      useFactory: async (configService: ConfigService) => ({

        secretOrPrivateKey: configService.get<string>('SECRET\_KEY'),

        signOptions: {

          expiresIn: '1d',

        },

      }),

      inject: [ConfigService], // Vẫn dùng đc service mà ko cần module

    }),],

# Get .env value:

  const configService = app.get(ConfigService);

  const PORT = configService.get<number>('PORT');

# 2 loại get

configService.get<string>('MONGODB\_URI'), // Ép kiểu, ko lỗi

configService.getString('SECRET\_KEY'), // Lỗi

**MongoDB ORM**

# Host db on docker

*docker-composet.yml*

services:

  mongo:

    image: mongo

    environment:

      MONGO\_INITDB\_ROOT\_USERNAME: root

      MONGO\_INITDB\_ROOT\_PASSWORD: a

    ports:

      - '27017:27017'

    volumes:

      - mongo\_nest:/data/db

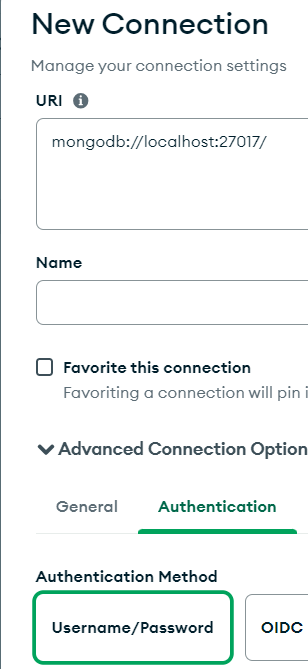
volumes:

  mongo\_nest:

**cmd:**

***docker compose -p nestjsdb up -d***

# Connect mongoDB username/password



**cmd:**

***npm i @nestjs/mongoose mongoose***

# Rename folder ***entities => schemas***

# Connect nestjs with mongoDB

    MongooseModule.forRootAsync({

      imports: [ConfigModule],

      useFactory: async (configService: ConfigService) => ({

        uri: configService.get<string>('MONGODB\_URI'),

      }),

      inject: [ConfigService],

    }),

# Create table in DB

Create schema => import to modules

*user.schema.ts*

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

import { HydratedDocument } from 'mongoose';

export type UserDocument = HydratedDocument<User>;

@Schema({ timestamps: true })

export class User {

  @Prop()

  name: string;

  @Prop()

  age: number;

}

export const UserSchema = SchemaFactory.createForClass(User);

*users.modules.ts*

imports: [MongooseModule.forFeature([{ name: User.name, schema: UserSchema }])],

# Relation

@Schema()

export class Todo {

    @Prop({ type: mongoose.Schema.Types.ObjectId, ref: 'User' })

    user: User;

# Route prefix + Exclude route

  app.setGlobalPrefix('api/v1', { exclude: ['/'] });

# Query param

  @Get()

  findAll(@Query('page') page: number, @Query('limit') limit: number) {

    return this.usersService.findAll(page | 1, limit | 5);

  }

# Create service

  async create(createUserDto: CreateUserDto) {

    const createdUser = await this.UserModel.create(

      { ...createUserDto }

    );

    return {

      \_id: createdUser.\_id

    };

  }

**Class validator**

Dùng với file dto

# White list + Forbid

Whitelist bỏ properties thừa của req, forbidNonWhitelisted trả về lỗi nếu thừa properties

  app.useGlobalPipes(

    new ValidationPipe({

      whitelist: true,

      forbidNonWhitelisted: true

    }),

  );

# Error message

    @IsEmail({},{ message: smail' })

# Debug

JWT

# Thêm secret key vào

*auth.module.ts*

    JwtModule.registerAsync({

      useFactory: async (configService: ConfigService) => ({

        secretOrPrivateKey: configService.get<string>('SECRET\_KEY'),

        signOptions: {

          expiresIn: configService.get<string>('JWT\_EXPIRES'),

        },

      }),

      inject: [ConfigService],

    }),],

# Sử dụng jwt service

*auth.module.ts*

export class AuthService {

  constructor(

    private usersService: UsersService,

    private jwtService: JwtService,

const access\_token = await this.jwtService.signAsync(payload)

**Passport guard**

# Mục đích auth guard:

* Phải có user mới login đc => Guard lấy data truyền lên tìm user
* Guard tìm ko thấy thì trả về error
* Tìm thấy thì thêm user vào req => req.user

# Controller sẽ gọi đến guard, xử lý, validate tại guard

* Chỉ có Guard gọi đến service
* Guard sẽ tự động bắt @Body trong controller

# Passport local (username/password): Dùng với login

*local.strategy.ts (Guard)*

@Injectable()

export class LocalStrategy extends PassportStrategy(Strategy) {

    constructor(private authService: AuthService) {

        super();

    }

// Controller gọi hàm này khi có decorator @UseGuards(AuthGuard('local'))

// @Body được parse thành tham số email,password

    async validate(email: string, password: string): Promise<any> {  
 // Chỉ có Guard gọi đến service (vd: authService)

        const user = await this.authService.validateUser(email, password);

        if (!user) {

            throw new UnauthorizedException();

        }

        return user;

    }

}

*auth.service.ts*

  async validateUser(email: string, pass: string): Promise<any> {

    // Tìm user trong db

    const user = await this.usersService.findUserByEmail(email);

    if (user && user.password === pass) {

      const { password, ...result } = user;

      return result;

    }

    return null; // Ko thấy thì trả về null

  }

# Khai báo kiểu class guard để ko cần code cứng loại guard ví dụ:

Code cứng:

*AuthGuard(‘jwt’)*

Thay bằng:

*@Injectable() // Dùng đc ở class khác*

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

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

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

@Injectable() // Dùng đc ở class khác

export class LocalAuthGuard extends AuthGuard('local') {}

  imports: [

    ...,

    PassportModule

    ],

  providers: [..., LocalStrategy],

**cmd:**

*npm install --save @nestjs/jwt passport-jwt*

*npm install --save-dev @types/passport-jwt*

Mục đích auth guard:

* Phải có user mới login đc => Guard lấy data truyền lên tìm user
* Guard tìm ko thấy thì trả về error
* Tìm thấy thì thêm user vào req => req.user

  @UseGuards(LocalAuthGuard)

  @Post('auth/login')

  async login(@Request() req: any) {

    return this.authService.loginJWT(req.user);

  }

*auth.service.ts*

 async loginJWT(user: any) {

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

    return {

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

       ...payload

    };

  }

# JWT auth guard

@Injectable()

export class JwtStrategy extends PassportStrategy(Strategy) {

    constructor(private configService: ConfigService) {

        super({

            // Tách bearer token

            jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),

            ignoreExpiration: false,

            secretOrKey: configService.get<string>('JWT\_SECRET\_KEY') || ''

        })

    }

    // Giải mã JWT Token

    async validate(payload: any) {

        return { userId: payload.sub, username: payload.username };

    }

}

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

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

@Injectable()

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

  @UseGuards(JwtAuthGuard)

  @Get('profile')

  getProfile(@Request() req) {

    return req.user;

  }