NEST JS OVERVIEW

Nest Introduction:

- Nest is a superset of NodeJs.
- Angular based architecture pattern.
- Typescript Support and TypeORM feature.

TypeORM (Object Relational Mapping**) Introduction:**

- Easy to link TypeScript application with relational database.
- ORMs are used to make database access easier.

Setup Nestjs Application:

- **Install Nest Cli**

  npm install -g @nestjs/cli

- **Create New Nest Project**

  nest new project-name

- **Install Dependencies for an application**

  1. Mysql

     npm install mysql2

  2. TypeORM

     npm install --save @nestjs/typeorm typeorm

  3. Config for .env

     npm install @nestjs/config

- **Add Configuration for Database with env**

```
TypeOrmModule.forRoot({
    type: 'mysql',
    host: process.env.DATABASE_HOST,
    port: 3306,
    username: process.env.DATABASE_USER,
    password: process.env.DATABASE_PASSWORD,
    database: process.env.DATABASE_NAME,
    synchronize: false,
    autoLoadEntities: false,
```

}),

Run Nestjs Application:

npm run start:dev

CRUD using Nestsjs:

- **Create Resource**

  nest g resource <name>

  Ex : nest g resource user

- **Create Entities**
- **Create Dto for Validation**
- **Create API Endpoints**
- **Create Service File Methods for Endpoints**
- **Check Endpoints with Swagger**

Swagger Setup:

- **Install Swagger**

  npm install --save @nestjs/swagger

- **Config Swagger in main.ts**

```
const config = new DocumentBuilder()
    .setTitle('Simple CRUD API')
    .setDescription('CRUD Using NestJS and MySQL') .
    setVersion('1.0')
    .addTag('CRUD') .build();
const document = SwaggerModule.createDocument(app, config); SwaggerModule.setup('api',
app, document);
```

Add DTO Validation:

- **Install Class Validator & Class Transformer.**

  npm i --save class-validator class-transformer

- **Add Validation Pipe Configuration in main.ts**

  app.useGlobalPipes(new ValidationPipe({ transform : true }));

- **Add Validtions in DTO File**

  There are many decorators used for validations.

  For Example Decorators:

@ApiProperty() - Must use for Swagger.

@IsNotEmpty() - Checks if given value is not empty.

@IsString() - Checks if given value is a string.

@IsOptional() - Checks if value is missing and if so, ignores all validators.

Use Like this in DTO File:

```
@ApiProperty()
        @IsNotEmpty()
        @IsString()
        @Matches('^[a-zA-Z0-9.-@_ ]+$')     (You can use pattern like this)
        @Length(2, 100)                     (You can limit the value with length)
        userName: string;
```

Session Implementation:

- **Install Session**

```
npm i express-session
```

- **Add Configuration in main.ts**

```
app.use(
        session({
        secret: process.env.sessionSecret,
        resave: false,
        saveUninitialized: true,
        cookie: {
                maxAge: 24 * 60 * 60 * 1000,
                },
        }),
);
```

- **Add Session id in response after login:**

```
sessionId : req.session.id
```

- **Add Options like below in frontend service:**

{withCredentials : true}

- **Set SessionId in Localstorage:**
- **Add sessionid in headers (get from localstorage):**
- **Compare req headers and req session id:**

add passReqToCallback : true in JWT Strategy

Logout:

req.session.destroy(cb)

Mail Service:

npm install --save @nestjs-modules/mailer nodemailer
npm install --save-dev @types/nodemailer

npm install --save handlebars

Logger:

npm install --save nest-winston winston

npm install winston-daily-rotate-file

- **Add Config in app.module.ts**

```
WinstonModule.forRoot({
        level: "info",
        format: winston.format.combine(
        winston.format.timestamp(),
        winston.format.json(),
        winston.format.colorize(),
        winston.format.errors()
        ),
        transports: [
            new winston.transports.DailyRotateFile({
            filename : 'name-%DATE%.log',
            level: "info",
            dirname : 'logs/',
            handleExceptions : true,
            json : false,
            zippedArchive : true,
```

maxSize : '50m'

})

]

}),

- **Add in Logger using controller or Service:**

@Inject(WINSTON_MODULE_PROVIDER) private readonly logger: Logger,

- **Format for Logger:**

${Controller Name} | Function Name() | ${User Id} | RequestId | Proper Message | ${Error}

<u>JWT Strategy:</u>

- **Install dependencies:**

npm install --save @nestjs/jwt

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

- **Add config in app.module.ts :**

JwtModule.register({

secret: “”,

signOptions: {

expiresIn: '60s'

},

})

- **Create Jwt-Strategy Service as jwt-strategy.service.ts :**

Add below code in that file :

constructor() {

super({

jwtFromRequest : ExtractJwt.fromAuthHeaderAsBearerToken(),

ignoreExpiration : false,

secretOrKey : “”

passReqToCallback : true

})

}

async validate(req : Request, payload : any) {

```
                try {
                        return payload;
                } catch (error) {
                        console.log(error);
                        return false;
                        }
                }
```

- **Add the above service file in providers that module :**

providers : [JwtStartegyService]

- **Use this guard in controller like below :**

@UseGuards(AuthGuard('jwt'))