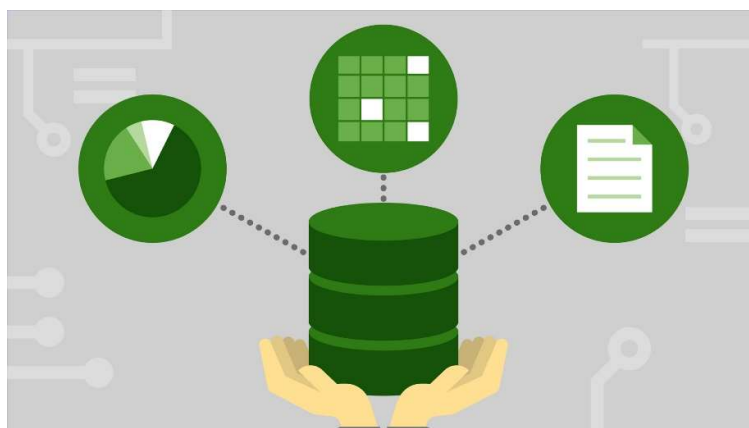


به نام خدا



دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده برق و کامپیوتر



آزمایشگاه پایگاه داده

دستورکار شماره ۶

مهلت تحویل :

۹۹/۹/۲۰

مجتبی بنائی

آنچه خواهید آموخت

در این دستور کار، به کاربرد عملی دیتابیس در سیستم های واقعی خواهیم پرداخت. برای این منظور از فریمورک nest.js استفاده شده است. این فریمورک با تکنولوژی node.js و زبان تایپ اسکریپت (زبان پیشنهادی میکروسافت برای جاوا اسکریپت که بسیار شبیه زبان های سطح بالاست) طراحی شده است و علاوه بر لحاظ الگوهای مهندسی نرم افزار مقیاس پذیر در این فریمورک، محبوبیت بسیار خوبی هم در چند سال اخیر داشته است. ORM یا کتابخانه اتصال به دیتابیس TypeORM آن هم جزء نقاط قوت این فریمورک و دلیل اصلی انتخاب آن در این دستور کار بوده است.

اگر با نود جی اس یا **نست** کار نکرده اید، نگران نباشید. هم دستور کار، ساده است و هم خود **نست** بسیار منظم و ماژولار بوده و زبان تایپ اسکریپت هم شبیه زبان های سطح بالایی مانند سی شارپ (البته تا حدود زیادی) است.

پیش نیازها

نصب نودجی اس

ابتدا کتابخانه نودجی اس را از سایت رسمی آن (<https://nodejs.org/en/>) دانلود و نصب کنید.

سپس در خط فرمان، مطمئن شوید که نود، نصب شده است:

```
node --version
```

نصب VS Code

برای کار با نود جی اس و حتی سایر زبانها، محیط محبوب vs code را توصیه می کنیم. بنابراین این محیط را هم حتما نصب کنید. (هر چند بعید میدانم روی سیستم دانشجویان رشته کامپیوتر این محیط سبک وزن و محبوب، نصب نباشد)

نصب پستمن (postman)

برای کار با توابع Rest (توابعی که از طریق پروتکل Http فراخوانی می شوند)، پستمن یکی از بهترین گزینه هاست. هر چند در ادامه، از Swagger برای کار با API هایی که خواهیم نوشت استفاده می کنیم اما برای شروع کار، بهتر است پستمن دم دستتان باشد.

نصب nestjs

به کمک npm که ابزار مدیریت کتابخانه های نودجی است، خط فرمان نست را نصب کنید:

```
npm install -g @nestjs/nest-cli
```

بعد از اجرای این دستور، می توانید از خط فرمان **نست** استفاده کنید. برای آزمایش این موضوع، این دستور را اجرا کنید:

```
nest --version
```

چند نکته:

- در دستور (npm install -g ...)، می توانید به طور خلاصه به جای install از i استفاده کنید.
- پارامتر g -، برای نصب کتابخانه ها به صورت سراسری در سیستم است و احتمالا نیاز به مجوز ادمین برای اجرا خواهد داشت. اگر از این پارامتر استفاده نکنید، کتابخانه ها در همان پوشه جاری در زیر پوشه node_modules نصب می شوند. برای کتابخانه هایی که در کل سیستم به آنها نیاز داریم و مستقل از پروژه باید نصب شوند از این پارامتر استفاده می کنیم.

اکنون همه چیز آماده است تا یک پروژه کوچک با نست ایجاد کنیم.

ساخت پروژه Hello World

برای شروع کار، یک پروژه با نام hello-project با دستور زیر در خط فرمان ایجاد کنید :

```
nest new hello-project
```

سوالاتی که پرسیده می شود را با مقادیر پیش فرض، جواب دهید. (اینتر بزنید)

حال با دستور cd hello-project وارد این پوشه شوید. پروژه را با دستور زیر اجرا کنید (از اینجا به بعد بهتر است پروژه را با vs-code باز کنید و دستورات را در خط فرمان آن اجرا کنید - منوی Terminal گزینه اول):

```
npm run start:dev
```


با این دستور، کدها کامپایل شده (از زبان ts به جاوااسکریپت تبدیل می شوند) و درون پوشه dist قرار میگیرند. اگر پیغام Nest application successfully started را مشاهده کردید، همه چیز آماده است و سرور محلی نودجی اس منتظر پاسخگویی به درخواست هاست.


آدرس <http://localhost:3000/> را در مرورگر باز کنید. باید پیغام Hello World را مشاهده کنید. درون پستمن هم یک کالکشن جدید با نام Hello ایجاد کرده، ریکوئستی با نام Hello و نوع GET و آدرس فوق ایجاد کنید. اینجا هم باید این خروجی را مشاهده کنید.

ساختار یک پروژه نست


قبل از ادامه کار، مروری سریع بر پروژه ای که ایجاد کرده ایم، خواهیم داشت.


فایلهای اصلی پروژه :


package.json  : این فایل، مشابه با تمام پروژه های نودجی اس حاوی لیست دستورات و کتابخانه های پروژه است. دستور شروع پروژه را می توانید در بین دستورات مختلف این فایل مشاهده کنید و ببینید که پشت صحنه چه چیزی اجرا میشود.

tsconfig.json  : حاوی تنظیمات کامپایل کدهای تایپ اسکریپت است .

پوشه های اصلی پروژه :


node_modules  : حاوی کتابخانه های نودجی اس پروژه است. البته اگر این پوشه پاک شود، کافی است دوباره در خط فرمان دستور npm i را اجرا کنیم تا به کمک فایل package.json، همه کتابخانه ها مجدداً دانلود و نصب شود.

src  : پوشه کدهای پروژه

test  : حاوی تست های پروژه

فایل‌های اصلی پوشه src :

پوشه src ساختار یک سرویس استاندارد نیست را دارد و بنابراین بهتر است با فایل‌های داخل آن آشنا شویم :

app.controller.ts 

فایل‌های کنترلر، حاوی کدهای توابع API ما هستند و نقطه دریافت درخواست‌ها و ارسال پاسخ به کاربر هستند. کد کنترلر پیش فرض که با زدن آن عبارت Hello World نمایش داده می‌شود از قرار زیر است :

```
@Controller()
export class AppController {
  constructor(private readonly appService: AppService) {}

  @Get()
  getHello(): string {
    return this.appService.getHello();
  }
}
```

- دکوراتور @Controller قبل از یک کلاس ، به سیستم نست اعلام می کند که این کلاس، یک کنترلر است.
- سازنده این کنترلر حاوی یک متغیر از نوع سرویس AppService است. توابع اصلی و منطق کسب و کار را درون سرویس‌ها تعریف میکنیم و درون کنترلر، فقط از آنها استفاده میکنیم . یعنی معمولا کدهای خاصی در کنترلر بجز صدازدن سرویس‌های مختلف (هر سرویس :حاوی توابع مخصوص به یک کار خاص است) نمی نویسیم. هر کنترلر می تواند از تعداد زیادی سرویس استفاده کند که همه آنها در بخش سازنده کلاس باید تعریف شوند.
- دکوراتور @Get: این دکوراتور اعلام می کند که این تابع، به درخواستهای GET پاسخ می دهد. آدرس این درخواست هم درون GET قرار میگیرد. فرض کنید ساختار فایل به صورت زیر است :

```
@Controller("/hello")
export class AppController {
  constructor(private readonly appService: AppService) {}

  @Get("/hi")
  getHello(): string {
    return this.appService.getHello();
  }

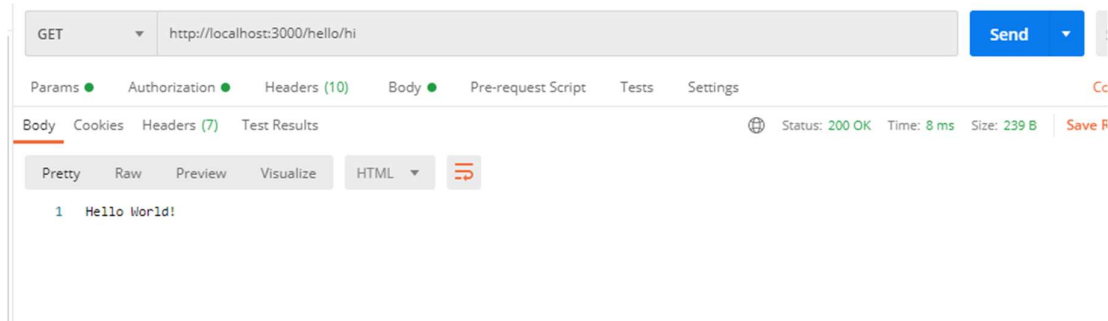
  @Get("/bye")
  getBye(): string {
    return this.appService.getHello();
  }
}
```

چون در دکواتور کنترلر اعلام کرده ایم که این کنترلر، آدرس hello را پاسخ می‌دهد، بنابراین هر یک از دو آدرس زیر که متناظر با هر یک از دو تابع GET است، عبارت Hello World را نشان می‌دهد:

<http://localhost:3000/hello/hi>

<http://localhost:3000/hello/bye>

(کافی است فایل کنترلر را ذخیره کنید. چون پروژه را با دستور `npm run start:dev` در حالت وچ اجرا کرده اید، سرور به صورت خودکار تغییرات فایلها را مانتیور کرده و ریستارت شده، تغییرات را سریعاً مشاهده میکنید.)



`app.module.ts` 🚀

این فایل حاوی تنظیمات یک ماژول است. ماژول هم واحد مدیریت و سازماندهی کلاس‌ها در نست است. از این فایل برای تعریف سرویس‌هایی که در کنترلرهای ماژول، استفاده می‌شود، سایر ماژول‌ها و کنترلرهای ماژول استفاده می‌شود. این فایل، علیرغم ساختار ساده‌ای که دارد، نقش بسیار حیاتی در نست ایفا می‌کند. فایل `app.module.ts` اصلی‌ترین ماژول کل پروژه است.

`app.service.ts` 🚀: این فایل حاوی سرویس اصلی برنامه است. همانطور که قبلاً اشاره شد، سرویس‌ها، کتابخانه‌های اصلی پروژه هستند و توابع مورد نیاز کنترلرها در آنها تعریف می‌شود. همانطور که می‌بینید این سرویس، یک تابع دارد که عبارت `Hello World` را برمیگرداند و درون کنترلر، فراخوانی شده است.

به عنوان آخرین دست‌گرمی، میخواهیم یک اندپوینت (در حقیقت یک Rest API) با متود POST ایجاد کنیم که نام و سال تولد کاربر را گرفته، سن او را محاسبه کرده و در پیام `hello` به او نشان دهد.

بهبتر است این کار را به کمک یک ماژول جدید تعریف کنیم که تمام توابع مرتبط با `hello` در آن تعریف شوند و نیاز به تغییر در ساختار ماژول اصلی پروژه نداشته باشیم.

دستورات زیر را به ترتیب برای ساخت ماژول، سرویس و کنترلر `hello` در خط فرمان وارد کنید (دقت کنید که حتماً در پوشه اصلی پروژه نست باشید - پروژه اصلی را متوقف نکنید و یک ترمینال جدید در `vs-code` باز کنید):

```
nest generate module hello
nest generate service hello
nest generate controller hello
```

اکنون باید پوشه hello با ساختاری مشابه با ماژول اصلی برنامه مشاهده کنید.

درون پوشه hello یک پوشه با نام dto ایجاد کنید. به کمک dto ها، ساختار داده‌ها را برای دریافت اطلاعات و یا ارسال خروجی‌ها تعیین می‌کنیم.

قرار است سن و نام کاربر را به کمک این اندپوینت دریافت کنیم:

```
http://localhost:3000/hello/welcome (POST)
```

بنابراین نیاز به یک dto برای دریافت اطلاعات داریم (خروجی چون یک پیام متنی است نیاز به تعریف dto ندارد). فایل person.dto.ts را درون پوشه dto با محتوای زیر ایجاد کنید:

```
import { Length, IsOptional, Min, IsNumber } from 'class-validator';

export class PersonDto {
  @Length(3, 10)
  name: string;

  @IsNumber()
  @IsOptional()
  @Min(1960)
  year: number;
}
```

با تعریف این کلاس، مشاهده می‌کنید که سرور نست خطایی را گزارش میدهد که ماژول "class-validator" را نمی‌شناسد. باید با دستور npm i آنرا نصب و به صورت لوکال در فایل‌های کتابخانه خود پروژه (پوشه node_modules) ذخیره کنید. دستور زیر اینکار را انجام میدهد:

```
npm i class-validator --save
```

دقت کنید که علاوه بر تعریف دو فیلد نام و سال تولد، ولیدیتورهایی هم برای آنها تعریف کرده‌ایم که خود **نست** به صورت خودکار، چک می‌کند که ورودی و خروجی‌ها حتما این شرایط را داشته باشند وگرنه خطا تولید خواهد شد. این کار هم به کمک دکوراتورها (@) انجام میشود که نیاز به کدنویسی خاصی نداشته و بسیار انعطاف پذیر است. نام را بین ۳ تا ۱۰ کاراکتر و مینیمم سال تولد را ۱۹۶۰ تعریف کرده‌ایم و در ادامه، تست خواهیم کرد که این موضوع، توسط نست چک خواهد شد یا نه.

حال تابع welcome را در سرویس hello یعنی فایل hello.service.ts به صورت زیر تعریف می‌کنیم:

```
import { Injectable } from '@nestjs/common';
import { PersonDto } from '../dto/person.dto';
```

¹ <https://www.npmjs.com/package/class-validator>

```

@Injectable()
export class HelloService {
  async welcome(person: PersonDto): Promise<string> {
    // check if the user exists in the db
    let msg: string;
    if (person.year) {
      let current_year = new Date().getFullYear();
      console.log(`Welcome ${person.name} - your birthday is ${person
        .year}`)
      msg = `Welcome ${person.name} - you are ${current_year - perso
          n.year} years old!`
    } else {
      console.log(`Welcome ${person.name} - your birthday is Undefined
        d`)
      msg = `Welcome ${person.name} - your birthday is Undefined!!!`
    }
    return msg;
  }
}

```

در کد فوق، ابتدا dto ساخته شده را ایمپورت میکنیم و سپس تابع welcome را با ورودی ای از نوع PersonDto و خروجی استرینگ، تعریف می‌کنیم. البته تابع را از نوع async تعریف کرده‌ایم که جزء نقاط قوت نودجی اس در مدیریت درخواست‌ها به صورت آسنکرون یا غیرهمگام است. خروجی توابع async از نوع Promise² است یعنی این تابع، قول میدهد یک خروجی از نوع رشته (در این مثال) به ما برگرداند اما ممکن است همان لحظه فراخوانی، این مقدار رشته ای، آماده نباشد (مثلا از دیتابیس بخواند و بعدا مقدار را به ما برگرداند. - سعی کردم خیلی خودمانی این مفهوم را توضیح دهم وگرنه از لحاظ فنی کمی پیچیده‌تر است -).

console.log هم برای مقاصد دیباگ برنامه در این کد، گنجانده شده است. دکوراتور Injectable هم به این موضوع اشاره دارد

حال کافی است در کنترلر، این سرویس را تعریف نموده و تابع welcome را فراخوانی کنیم:

```

import { HelloService } from './hello.service'
import { Controller, Post, Body, Get, Header } from '@nestjs/common';
import { PersonDto } from './dto/person.dto';

@Controller('hello')
export class HelloController {

  constructor(
    private readonly helloService: HelloService,
  ) {}
}

```

² <https://basarat.gitbook.io/typescript/future-javascript/promise>

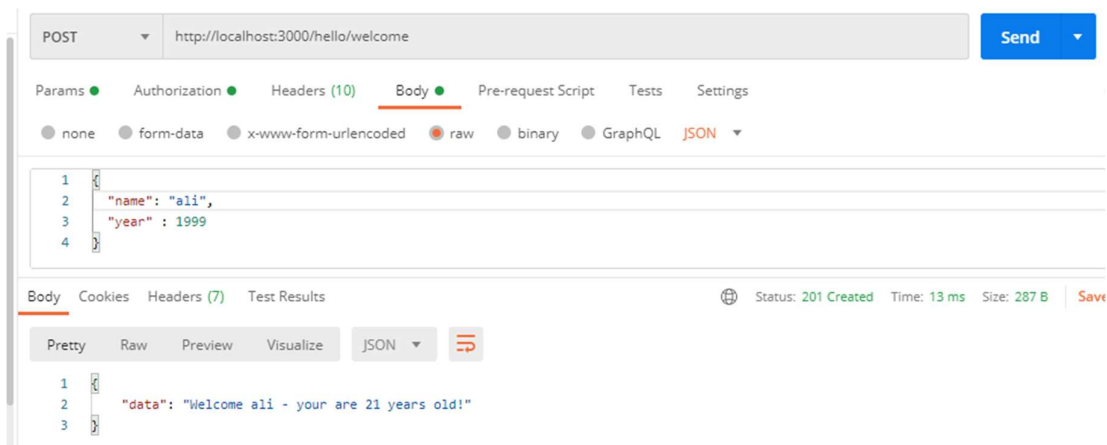

```

@Post('welcome')
@Header('Content-Type', 'application/json')
async sayWelcome(@Body() personDto: PersonDto): Promise<{data : String}> {
  > {
    let msg = await this.helloService.welcome(personDto);
    return {data : msg};
  }
}

```

خروجی تابع sayWelcome را به صورت جی‌سان تعریف کردیم که به دنیای واقعی نزدیک‌تر باشد (بهبتر است برای این موضوع هم یک Dto در همان فایل person.dto.ts تعریف کنیم و نوع خروجی را برابر <WelcomeDto> promise قرار دهیم).

حال می‌توانیم با پستمن، خروجی کار را مشاهده کنیم :



می‌توانید سن را از ورودی حذف کنید و چک کنید که خروجی مورد نظر، تولید می‌شود یا نه. طول رشته را دو حرفی و یا سال تولد را زیر ۱۹۶۰ وارد کنید و مجدداً خروجی را تست کنید.

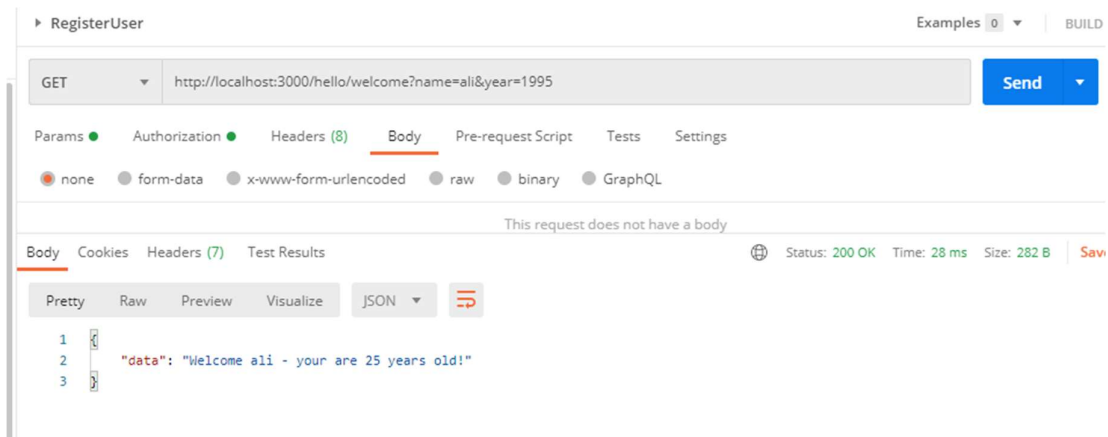
حال همین خروجی را با روش GET تولید می‌کنیم :

```

@Get('welcome')
async sayWelcome2(@Query('name') iName, @Query('year') iYear): Promise<{data : String}> {
  {
    let msg = await this.helloService.welcome({name : iName, year : iYear});
    return {data : msg};
  }
}

```

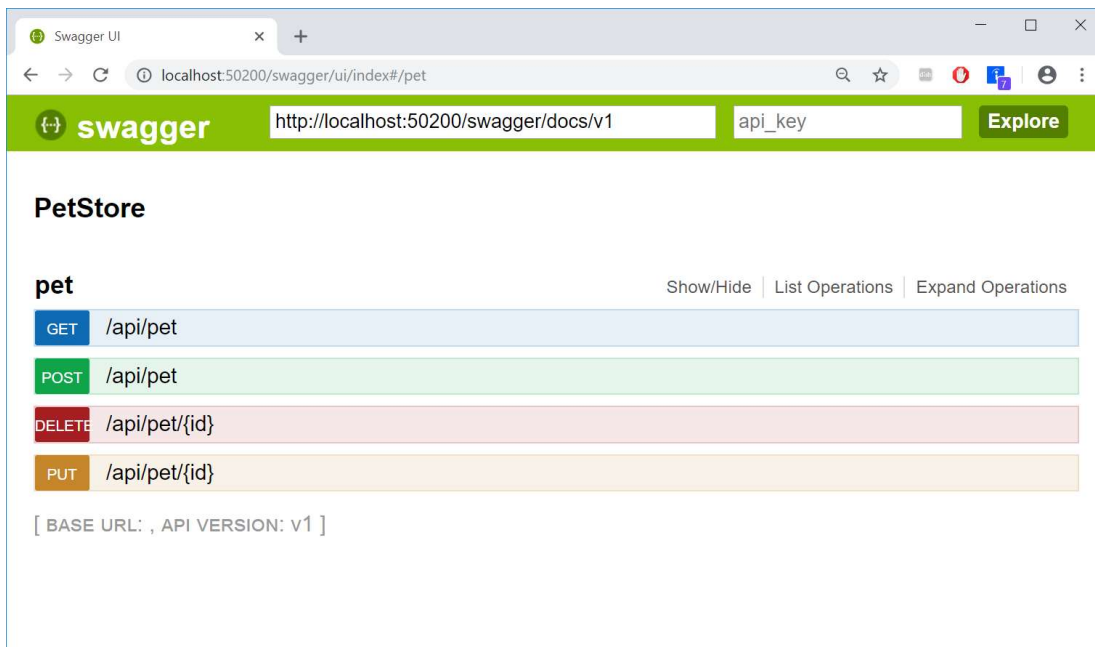
و به کمک پستمن این درخواست را ارسال می کنیم :



افزودن OpenApi به پروژه

برای مستندسازی توابع پروژه های بک اند، یکی از بهترین گزینه ها استفاده از swagger یا همان OpenApi است. به کمک سواگر^۳، به راحتی می توانیم لیست توابع را در اختیار سایر تیم ها بگذاریم و امکان تست و بررسی و مشاهده مستندات آنها را در خود پروژه فراهم کنیم.

نمونه ای از خروجی سواگر را در زیر مشاهده میکنید :



نکته : پروژه های نرم افزاری امروزی، اکثرا مبتنی بر وب و متشکل از دو بخش کاملا مجزای بک اند و فرانت اند هستند. سواگر به راحتی ارتباط بین این دو بخش را فراهم می کند و توصیه می کنم در تمامی پروژه های بک اندی خود، حتما از آن استفاده کنید .

برای افزودن سواگر به پروژه فوق، ابتدا کتابخانه زیر را نصب می کنیم :

```
npm install --save @nestjs/swagger swagger-ui-express
```

حال محتوای فایل main.ts را با کدهای زیر به روز می کنیم :

```
import { NestFactory } from '@nestjs/core';
import { SwaggerModule, DocumentBuilder } from '@nestjs/swagger';
import { AppModule } from './app.module';
```

³ <https://swagger.io/>

```

async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  const options = new DocumentBuilder()
    .setTitle('Cats example')
    .setDescription('The cats API description')
    .setVersion('1.0')
    .addTag('cats')
    .build();
  const document = SwaggerModule.createDocument(app, options);
  SwaggerModule.setup('api', app, document);

  await app.listen(3000);
}
bootstrap();

```

در دستور `SwaggerModule.setup('api', app, document);` تعیین می‌کنیم که برای دسترسی به سواگر از اندپوینت `api` استفاده می‌شود. بنابراین کافی است در مرورگر این آدرس را باز کنیم:

<http://localhost:3000/api>

اکنون می‌توانید لیست توابع پروژه را مشاهده کرده و در همین آدرس، آنها را با زدن دکمه Try It Out به ازای هر تابع، تست کنید و خروجی را همانجا مشاهده نمایید.

نکته: اگر پروژه شما، فقط بخش بک‌اند را شامل می‌شود بهتر است به جای اندپوینت `api` از بک اسلش استفاده کنید تا با رفتن به آدرس `localhost:3000` مستقیماً سواگر باز شود.

```
SwaggerModule.setup('/', app, document);
```

برای اینکه از امکانات کامل سواگر استفاده کنید، کنترلر `hello` بهتر است به صورت زیر درآید:

```

import { HelloService } from './hello.service'
import { Controller, Post, Body, Get, Header, Query } from '@nestjs/common';
import { PersonDto } from './dto/person.dto';
import { ApiResponse, ApiBearerAuth, ApiQuery } from '@nestjs/swagger';

@Controller('hello')
export class HelloController {

  constructor(
    private readonly helloService: HelloService,
  ) {}

  @Header('Content-Type', 'application/json')

```

```

    @ApiResponse({ status: 200, description: 'Say Hello!!!' })
    @Post('welcome')
    @Header('Content-Type', 'application/json')
    async sayWelcome(@Body() personDto: PersonDto): Promise<{data : String}> {
    > {
        let msg = await this.helloService.welcome(personDto);
        return {data : msg};
    }

    @ApiResponse({ status: 200})
    @ApiQuery({
        name: 'name',
        required: true,
        type: String,
        // enum : ["All", "Browser", "Device"]
    })
    @ApiQuery({
        name: 'year',
        required: false,
        type: Number,
        description : `you can ignore this`
    })

    @Get('welcome')
    async sayWelcome2(@Query('name') iName, @Query('year') iYear): Promise<
{data : String}> {
        let msg = await this.helloService.welcome({name : iName, year : iY
ear});
        return {data : msg};
    }
}

```

در اینجا هم خروجی هر تابع را مشخص کرده ایم (اگر در حالت خطا کدی غیر از ۲۰۰ برمیگردانیم بهتر است اینجا مستند شود) و هم پارامترهای ورودی تعیین شده اند. اکنون اگر سواگر را مجدداً باز کنید، متوجه تغییر و امکانات بیشتری که برای ورود داده‌ها به شما داده شده است، خواهید شد.

برای اینکه ورودی دستورات POST هم مستند شود، توضیحاتی را باید به dto اضافه کنیم. فایل person.dto.ts را به صورت زیر تغییر دهید و مجدداً سواگر را چک کنید و این دفعه، نام و سن یک نفر را وارد کرده، تست کنید :

```

import { Length , IsOptional, Min, IsNumber } from 'class-validator';
import { ApiProperty, ApiPropertyOptional } from '@nestjs/swagger';

export class PersonDto {
    @Length(3, 10)

```

```
@ApiProperty({description:'Enter Your Name > ', minLength: 3, default: 'Ali' ,maxLength:10})

name: string;

@IsNumber()
@IsOptional()
@Min(1960)
@ApiPropertyOptional({ description: 'Optional', default:1998 , minimum:1960 })
year: number;
}
```

قبل از ادامه دستورکار و شروع کار با دیتابیس، توصیه می‌کنم مثال زیر را هم که به کمک لیست‌ها و بدون استفاده از دیتابیس، انجام شده است، خودتان انجام دهید تا با جنبه‌های بیشتری از این فریم‌ورک آشنا شوید.

<https://www.digitalocean.com/community/tutorials/getting-started-with-nestjs>

نکته: در این مثال، هنگام بازیابی اطلاعات (خواندن لیست کتاب‌ها و یا یک کتاب خاص)، از متدهای ناهمگام (آسنکرون) استفاده شده است که قالبی شبیه زیر دارند:

```
getBooks(): Promise<any> {
    return new Promise(resolve => {
        resolve(this.books);
    });
}
```

که هنگام کار با دیتابیس‌های واقعی از این سبک، کمتر استفاده می‌کنیم و به جای `resolve` مستقیم (تعیین خروجی نهایی یک تابع با خروجی پرامیس)، نتایج را از دیتابیس می‌خوانیم. شما می‌توانید در این مثال، به جای استفاده از این حالت، در ابتدای تابع از کلمه `async` استفاده کنید (تا خروجی تابع به صورت خودکار از نوع پرامیس تعیین شود) و به صورت عادی، خروجی مورد نظر را برگردانید. در این حالت، عملیات `resolve`، به صورت پشت صحنه انجام شده و کدها خواناتر می‌شود (مراجعه کنید به این آدرس^۴).

^۴ <https://medium.com/jspoint/typescript-promises-and-async-await-b842b55ee3fd>

دستورکار اجرایی

برای این دستورکار، می‌خواهیم یک دیتابیس ساده برای ایجاد یک اپ لیست کار (to-do app) ایجاد کنیم و توابع API لازم برای بخش بک‌اند آنرا تولید نماییم. جزئیات آن در ادامه بیان خواهد شد.

به ترتیب کارهای زیر را انجام دهید :

1. دو تابعی که در کنترلر اصلی برنامه نوشته‌ایم را حذف کنید (تابع `getHello` و `getBye`) و فقط مازول `hello` را نگه دارید. چک کنید که سواگر به درستی بالا بیاید و خطایی در پروژه نباشد. در گیت‌هاب یک پروژه جدید با اکانت خودتان ایجاد کنید و کدهای نوشته شده تا این مرحله را به آن منتقل نمایید (بعد از ساخت پروژه در گیت‌هاب، راهنمای انتقال پروژه جاری به این پروژه را که بعد از ساخت یک پروژه خالی در گیت‌هاب نمایش داده می‌شود، در خط فرمان انجام دهید).

از این به بعد هم هر بخشی که کامل می‌کنید را کامیت کرده و در گیت‌هاب پوش کنید. خروجی اصلی شما در این دستورکار، همین پروژه گیت‌هاب شما خواهد بود و نمره شما کاملاً بسته به تعداد کامیت‌ها و تغییراتی که در هر مرحله از پروژه ایجاد کرده اید خواهد داشت.

2. در مرحله دوم، می‌خواهیم کار با دیتابیس و `typeorm` را به صورت عملی انجام دهیم .

حال مراحل انجام یک پروژه دیتابیزی ساده را مطابق دستورالعمل زیر انجام دهید :

<https://codersera.com/blog/typeorm-with-nest-js-tutorial/>

در این راهنما، نیاز دارید یک پوشه `db` (حاوی کلاس‌های موجودیت و روابط بین آنها) بسازید و سه مازول زیر را تولید کرده و سپس مرحله به مرحله، مراحل کار را انجام دهید. به ازای هر یک از سه مازول `genre.books` و `user` هر سه دستور زیر را در خط فرمان و در پوشه اصلی پروژه وارد کنید تا فایلها و پوشه‌های لازم ساخته شود :

```
nest generate module books
nest generate service books
nest generate controller books
```

این مثال با دیتابیس ساده و محبوب `SQLite` پیکر بندی شده است. بعد از اتمام کار و انجام مستندسازی‌های لازم برای نمایش توابع ساخته شده در سواگر، شروع به وارد کردن کتاب و یوزر و ژانر نمایید و به کمک `DBaver` چک کنید که داده‌ها در دیتابیس `database.sqlite` که یک فایل در پوشه اصلی پروژه است وارد شده باشند. تا مرحله جاری را کامیت کرده و پوش کنید.

3. در مرحله بعد، دیتابیس پروژه را به پستگرس تغییر دهید. ابتدا لازم است دیتابیزی در پستگرس بسازید و تنظیمات آن مانند نام دیتابیس، نام کاربری و پسورد را در فایل تنظیمات `typeorm` (`ormconfig.json` در پوشه اصلی پروژه) وارد کنید :

```
{
  "type": "postgres",
  "host": "localhost",
  "port": 5432,
  "username": "postgres",
```

```

"password": "!ChangeMePlease!",
"database": "todoapp",
"entities": ["dist/**/*.entity{.ts,.js}"],
"synchronize": true
}

```

تنظیمات فوق، برای نمونه ایجاد شده است و حتما نام دیتابیس و یوزر و پسورد خودتان را وارد کنید. دقت کنید که گزینه آخر، برابر `true` باشد تا هنگام راه اندازی پروژه و استارت سرور، به صورت خودکار، ساختار دیتابیس با فایل‌های موجودیت (فایل‌هایی که نام آنها به `entity.ts` ختم می شود) مطابقت داده شده و اگر نیاز به ساخت جداول، ارتباطات و حتی حذف یا تغییر ستونی در یک جدول باشد، به طور خودکار انجام شود. حال مجدداً پروژه را اجرا کنید. در این مرحله باید بتوانید همان کارهای قبلی را انجام دهید و این بار، داده‌ها در پستگرس ذخیره شود.

در این گام، توابع لازم برای حذف و آپدیت و اندپوینت‌های متناظر را هم به پروژه اضافه کنید. یعنی باید بتوانید در سواگر با اندپوینت `DELETE` و `PUT`، یک کتاب را حذف و یا آنرا ویرایش کنید. کارهای انجام شده را کامیت و پوش کنید.

4. در گام چهارم، می خواهیم `JWT` یا احراز هویت از طریق جی‌سان را به پروژه اضافه کنیم. یعنی تنها افرادی که احراز هویت شده‌اند و یوزر و پسورد صحیح را وارد کرده‌اند، حق صدا زدن توابع را خواهند داشت. از راهنمای زیر که مستندات رسمی خود نست است، برای افزودن بخش احراز هویت به پروژه استفاده و نتیجه را کامیت کنید.

<https://docs.nestjs.com/techniques/authentication>

تغییرات خواسته شده در موجودیت و ماژول یوزر را بر روی همین یوزری که ایجاد کرده‌اید، اعمال کنید.

در انتهای کار، پس از لاگین، یک کد دسترسی یا اکسس توکن تولید خواهد شد که برای فراخوانی توابع پروژه، این کد را باید در قسمت هدر وارد کنید. با پستمن، کافیت هنگام ارسال درخواست‌های `POST` یا `GET` یا ...، هدر را به صورت زیر تنظیم کنید:

Any DG Product over Nepal

POST `https://geobigdata.io/catalog/v1/search`

Authorization Headers (2) Body Pre-request script Tests

Header	Value
Content-Type	application/json
Authorization	Bearer [[AccessToken]] ➡ Add token here

به جای آکولاد مشخص شده در شکل، اکسس توکن را در اینجا کپی کنید و سپس تابع (مثلاً `[GET] todos`) را فراخوانی نمایید.

در سواگر هم برای فعال سازی احراز هویت، قبل از هر تابعی که باید کاربر برای فراخوانی آن، احراز هویت شود، دکوراتورهای زیر را قرار دهید:

```

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

```



```
@ApiBearerAuth()
@UseGuards(AuthGuard())
```

مثال کامل یک کنترلر که از سیستم احراز هویت **نست** استفاده می کند به شکل زیر خواهد بود :

```
@Controller("api/v1/shorten")
export class LinkShortenerController {
  constructor(private readonly shortenerService: ShortenerService) {}

  @Post('link')
  @Header('Content-Type', 'application/json')
  @ApiResponse({ status: 201, description: 'Short Code Generated Sucessfull' })
  @ApiResponse({ status: 409, description: 'Provided Code Already Exist' })
  @ApiBearerAuth()
  @UseGuards(AuthGuard())
  async create(@Req() req: Request, @Body() createShortLinkDto: CreateShortLinkDto): Promise<LinkInfoDto> {
    const user = <UserDto>req.user;
    console.log(`User : ${user}`)
    return this.shortenerService.create(user, createShortLinkDto);
  }
}
```

بعد از اینکه احراز هویت را برای تمام توابع به غیر از ثبت نام کاربر و لاگین، فعال کردید، پروژه را کامیت و تغییرات را پوش کنید.

5. حال به عنوان آخرین گام، دیتابیس برای یک اپ to-do list ایجاد کنید :

- a. هر تسک یا کار، یک گروه یا کتگوری مشخص دارد.
- b. هر تسک، یک یا چند برچسب میتواند داشته باشد. (یا اصلا برچسب نداشته باشد)
- c. هر تسک میتواند تنها یک متن خالی یا لیستی از کارها باشد (نوع آن). در حالت دوم، جدولی برای آیتمها نیاز خواهید داشت.
- d. هر تسک می تواند حذف شده یا ویرایش شود ، مثلا آیتم های آن تغییر کند. هر آیتم هم جداگانه ، می تواند حذف شود.
- e. تمام کارها، باید بعد از احراز هویت کاربر صورت گیرد.

یک مازول با نام todo ایجاد کنید، سرویس و کنترلی هم به همین نام ایجاد نمایید. موجودیت ها و روابط آنها را در پوشه db (داخل پروژه اصلی) تعریف کنید. در مازول todo یک پوشه dto ایجاد کرده و تمام dtoهای لازم را در آن قرار دهید. سپس به تکمیل کدها (ابتدا توابع سرویس ها را بنویسید و سپس به سراغ کنترلرها بروید) می توانید مازول genre و books را کلا حذف کنید.

بعد از تست تمامی قسمت ها در سواگر، پروژه را کامیت نمایید .

این مثالها می تواند به شما در انجام این مرحله ، کمک کند :

<https://hackernoon.com/how-to-create-your-first-nestjs-app-jp143t8y>

<https://codeburst.io/typeorm-by-example-part-1-6d6da04f9f23>

https://github.com/smbanaie/urlshortener_nestjs

6. گزارش کار شما باید شامل آدرس فایل گیت به همراه توضیح مختصر هر مرحله باشد . هر مرحله نهایتا نصف صفحه

نیاز به توضیح خواهد داشت. **این پروژه هم حتما باید تک نفره باشد.**

سعی کرده ام تا حد امکان، پروژه بسیار ساده ای طراحی کنم که در بیرون از دانشگاه هم کاملا برای شما مفید و کاربردی باشد.

موفق باشید.