Introduction to NestJS

Introduction

`Nest` (**`NestJS`**) is a framework for building efficient, scalable Node.js server-side applications

It uses progressive JavaScript, is built with and fully supports TypeScript and combines elements of OOP (Object Oriented Programming), FP (Functional Programming), and FRP (Functional Reactive Programming)

https://docs.nestjs.com/

Installation

Prerequisite

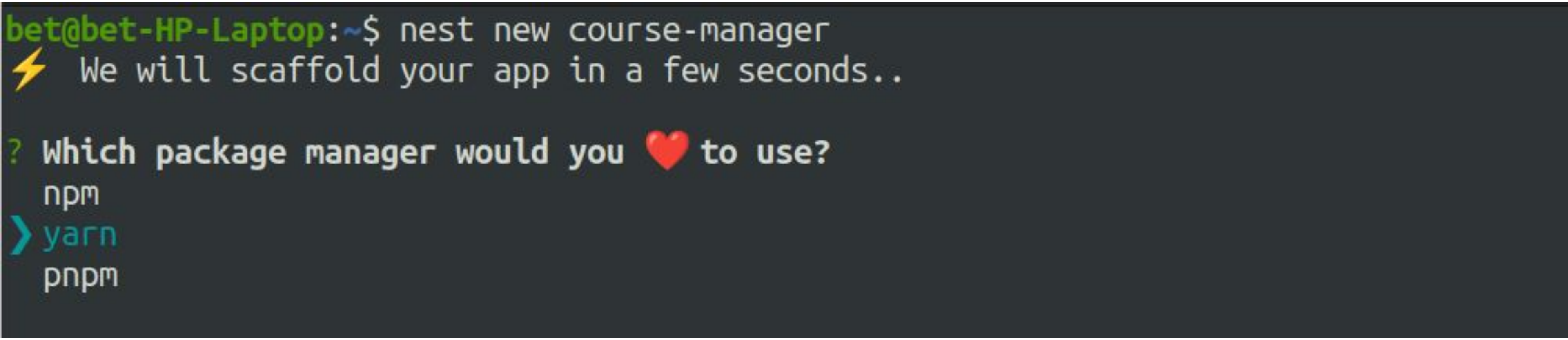
Make sure that `Node.js` (`version >= 12`, except for v13) is installed on your operating system

Installation Command

```
npm i -g @nestjs/cli
```

Creating New Project

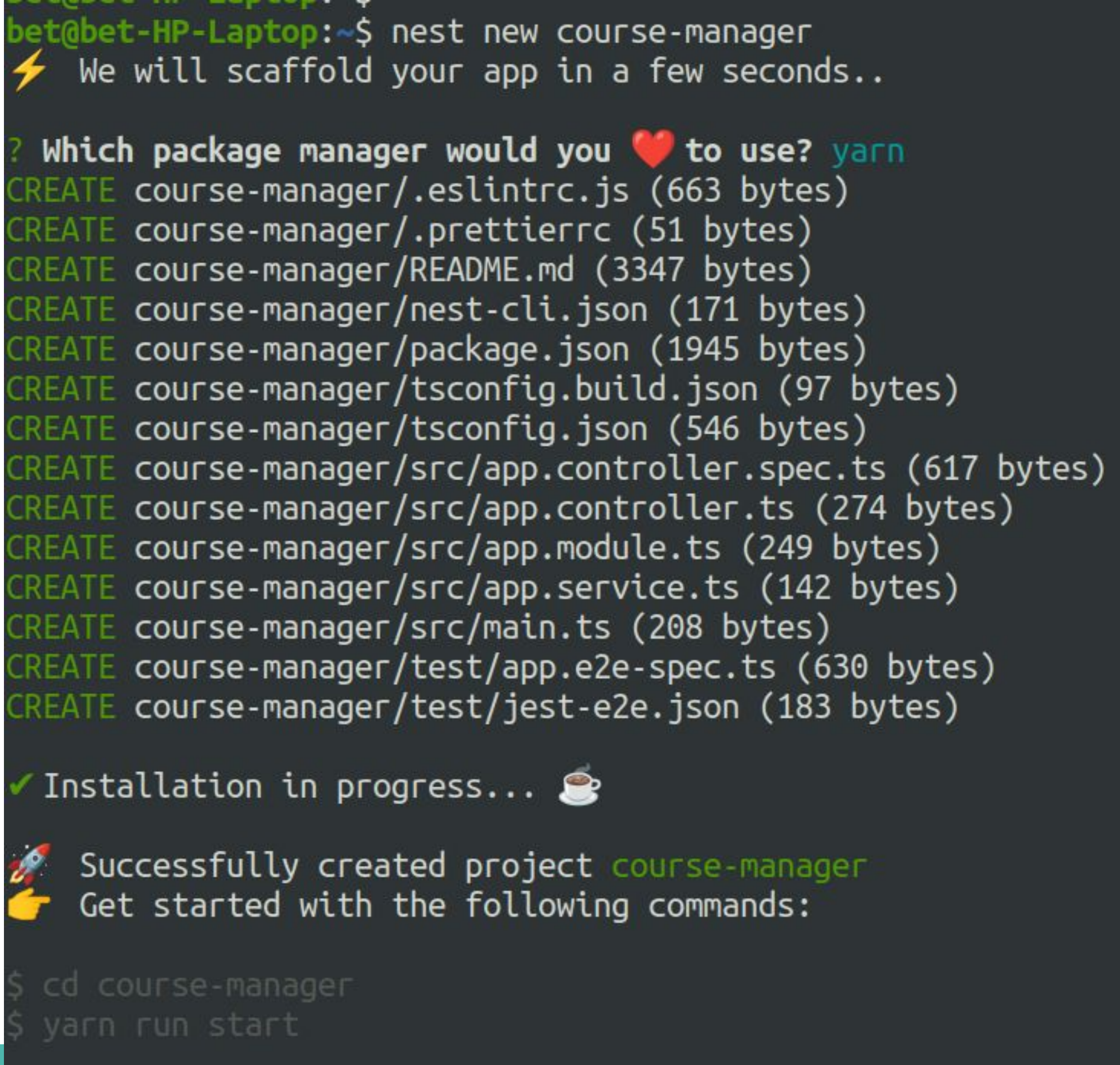
```
nest new project-name
```



```
bet@bet-HP-Laptop:~$ nest new course-manager
⚡  We will scaffold your app in a few seconds..

? Which package manager would you ❤️ to use?
  npm
❯ yarn
  pnpm
```

Creating New Project

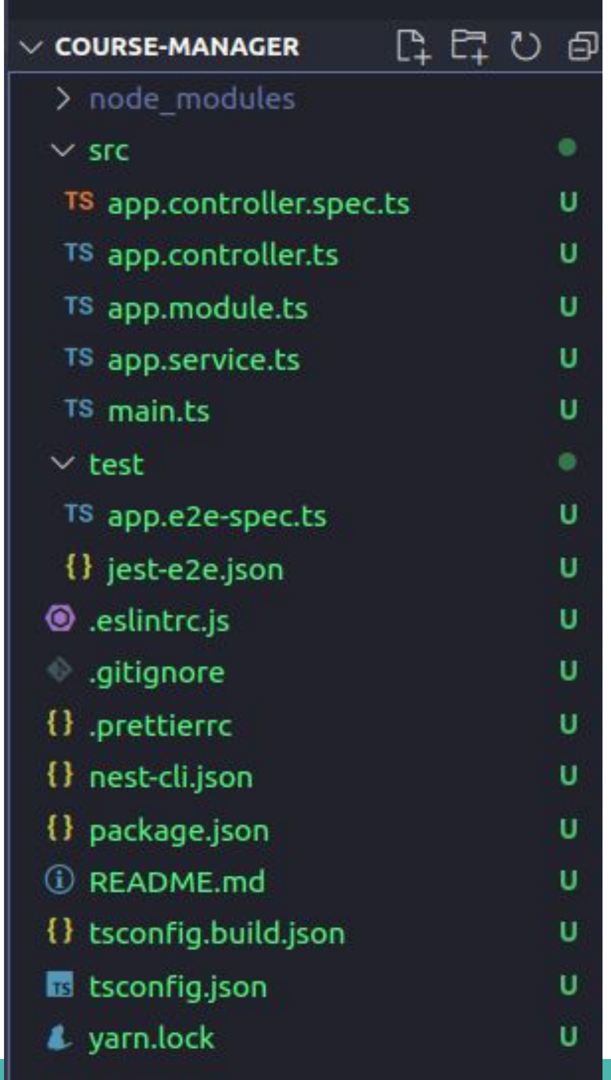
Command output



```
bet@bet-HP-Laptop:~$
bet@bet-HP-Laptop:~$ nest new course-manager
⚡  We will scaffold your app in a few seconds..

? Which package manager would you ❤️ to use? yarn
CREATE course-manager/.eslintrc.js (663 bytes)
CREATE course-manager/.prettierrc (51 bytes)
CREATE course-manager/README.md (3347 bytes)
CREATE course-manager/nest-cli.json (171 bytes)
CREATE course-manager/package.json (1945 bytes)
CREATE course-manager/tsconfig.build.json (97 bytes)
CREATE course-manager/tsconfig.json (546 bytes)
CREATE course-manager/src/app.controller.spec.ts (617 bytes)
CREATE course-manager/src/app.controller.ts (274 bytes)
CREATE course-manager/src/app.module.ts (249 bytes)
CREATE course-manager/src/app.service.ts (142 bytes)
CREATE course-manager/src/main.ts (208 bytes)
CREATE course-manager/test/app.e2e-spec.ts (630 bytes)
CREATE course-manager/test/jest-e2e.json (183 bytes)

✔ Installation in progress... ☕

🚀  Successfully created project course-manager
👉  Get started with the following commands:

$ cd course-manager
$ yarn run start
```

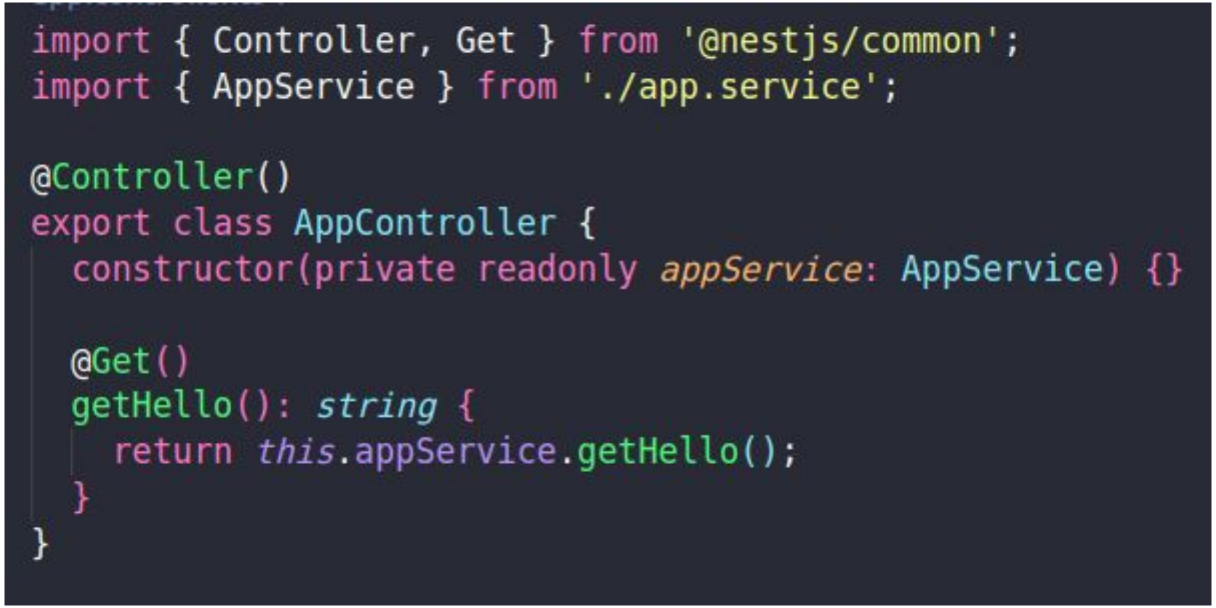
Creating New Project

Open the directory created by the project name - `course-manager` to access the generated files

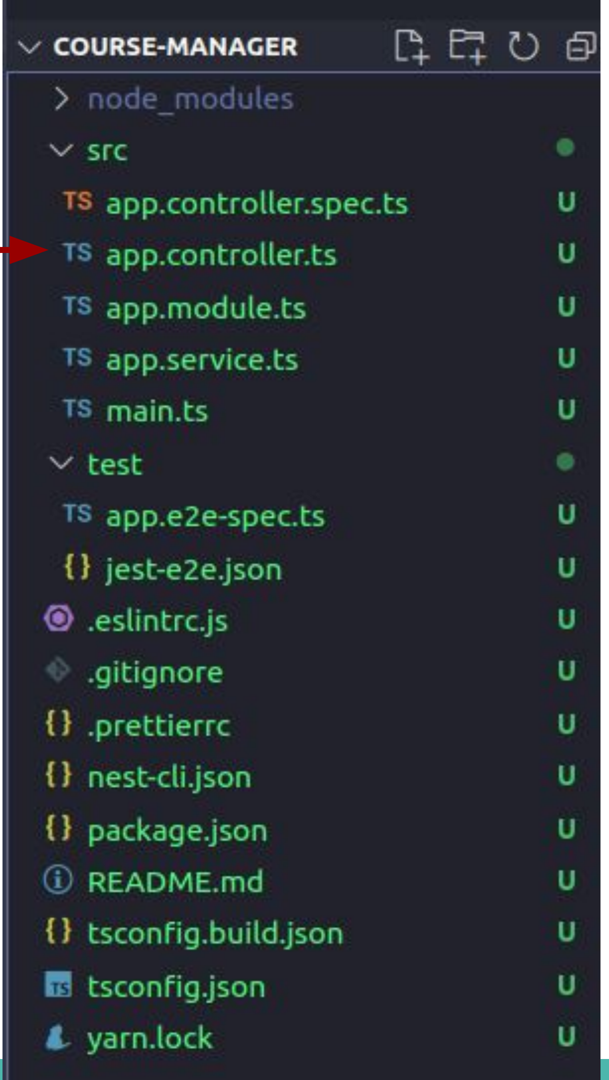


```
COURSE-MANAGER
> node_modules
v src
    app.controller.spec.ts      U
    app.controller.ts           U
    app.module.ts               U
    app.service.ts              U
    main.ts                     U
v test
    app.e2e-spec.ts             U
    jest-e2e.json               U
  .eslintrc.js                  U
  .gitignore                    U
  .prettierrc                   U
  nest-cli.json                 U
  package.json                  U
  README.md                     U
  tsconfig.build.json           U
  tsconfig.json                 U
  yarn.lock                     U
```

A basic controller with a single route

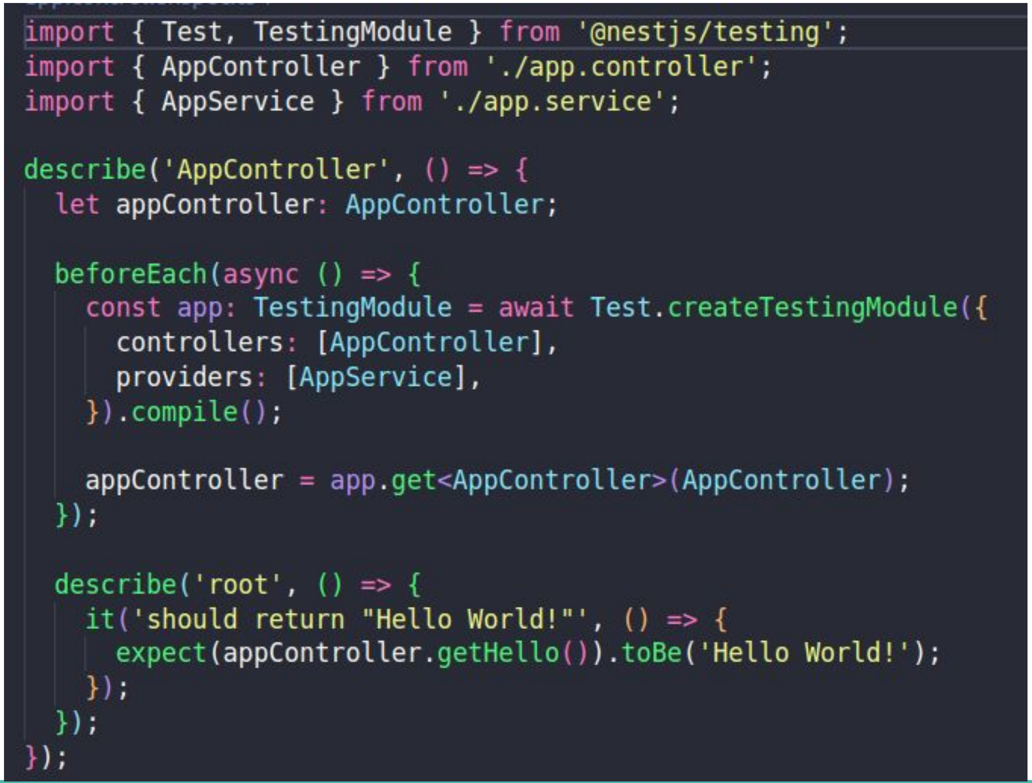


```
import { Controller, Get } from '@nestjs/common';
import { AppService } from './app.service';

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

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

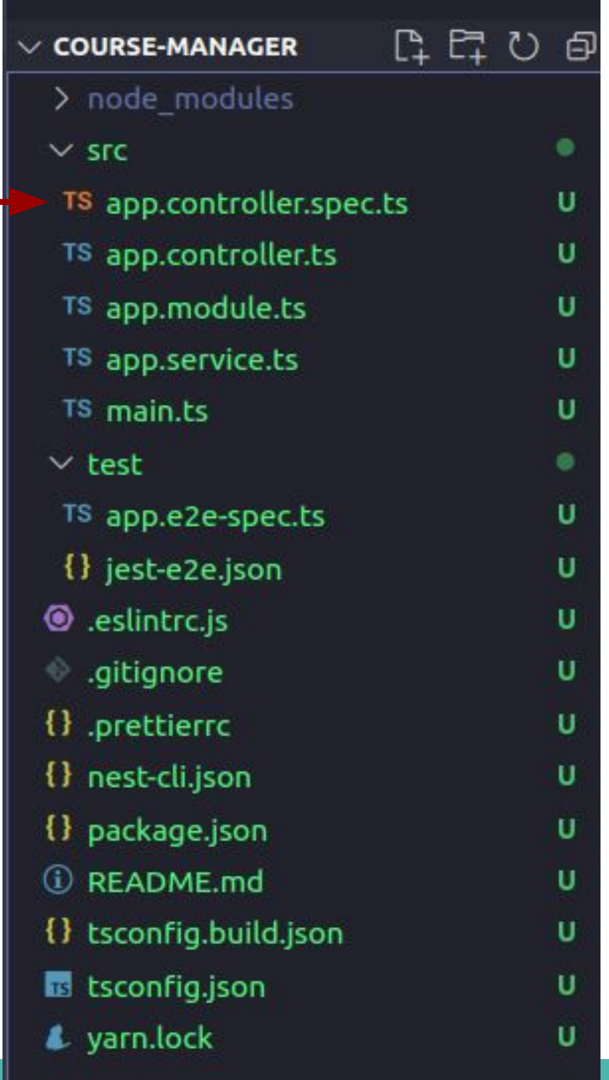


```
COURSE-MANAGER
  > node_modules
  v src
      app.controller.spec.ts      U
      app.controller.ts           U
      app.module.ts               U
      app.service.ts              U
      main.ts                     U
  v test
      app.e2e-spec.ts             U
      jest-e2e.json               U
    .eslintrc.js                  U
    .gitignore                    U
    .prettierrc                   U
    nest-cli.json                 U
    package.json                  U
    README.md                     U
    tsconfig.build.json           U
    tsconfig.json                 U
    yarn.lock                     U
```

The unit tests for the controller

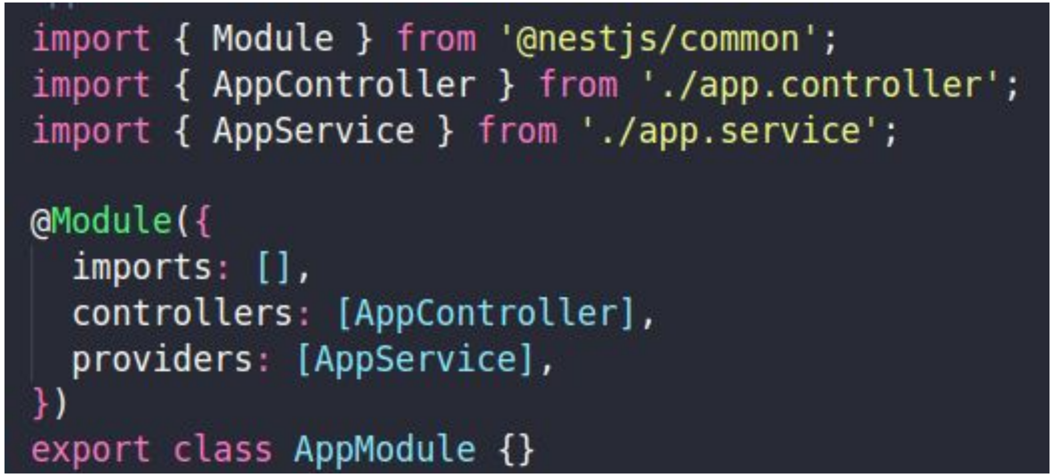


```
import { Test, TestingModule } from '@nestjs/testing';
import { AppController } from './app.controller';
import { AppService } from './app.service';

describe('AppController', () => {
  let appController: AppController;

  beforeEach(async () => {
    const app: TestingModule = await Test.createTestingModule({
      controllers: [AppController],
      providers: [AppService],
    }).compile();

    appController = app.get<AppController>(AppController);
  });

  describe('root', () => {
    it('should return "Hello World!"', () => {
      expect(appController.getHello()).toBe('Hello World!');
    });
  });
});
```

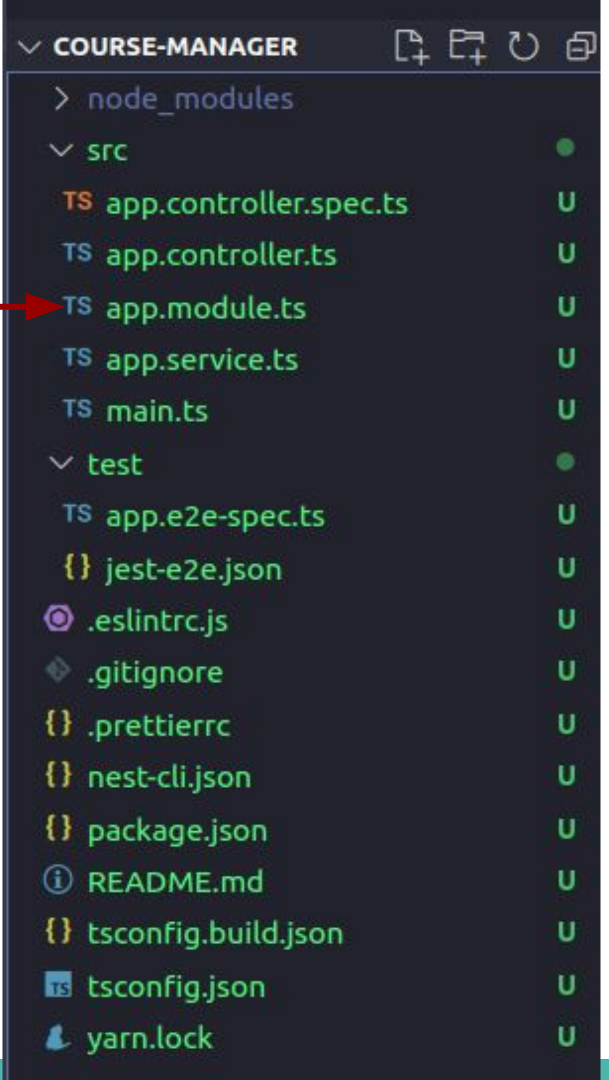


```
COURSE-MANAGER
  > node_modules
  v src
      app.controller.spec.ts   U
      app.controller.ts        U
      app.module.ts            U
      app.service.ts           U
      main.ts                  U
  v test
      app.e2e-spec.ts          U
      jest-e2e.json            U
    .eslintrc.js               U
    .gitignore                 U
    .prettierrc                U
    nest-cli.json              U
    package.json               U
    README.md                  U
    tsconfig.build.json        U
    tsconfig.json              U
    yarn.lock                  U
```

The root module of the application

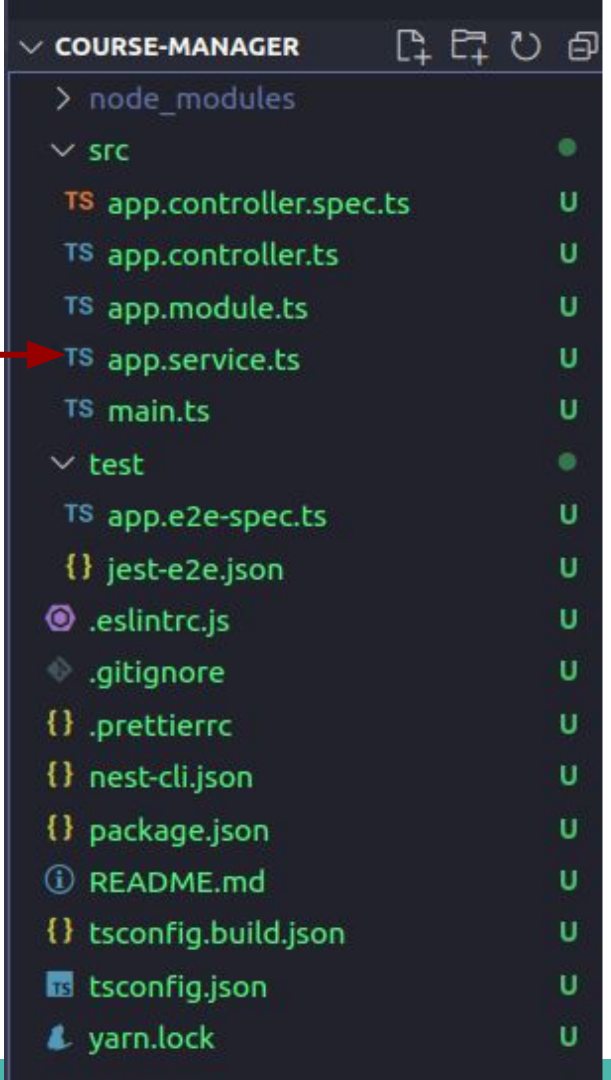


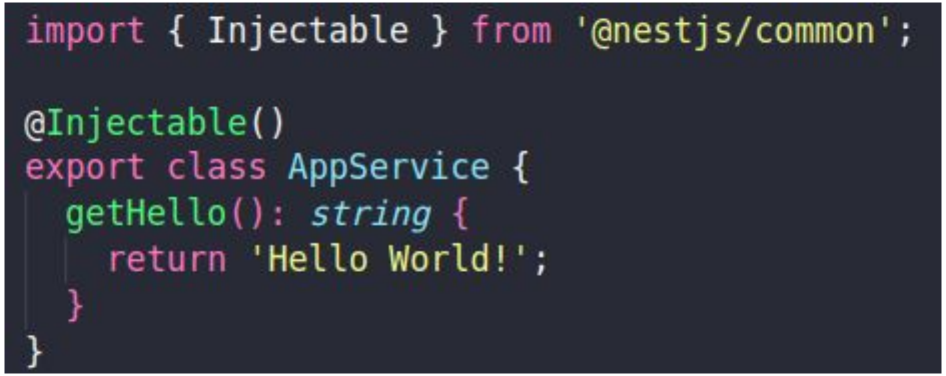
```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';

@Module({
  imports: [],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```



```
COURSE-MANAGER
  > node_modules
  v src
      app.controller.spec.ts   U
      app.controller.ts        U
      app.module.ts            U
      app.service.ts           U
      main.ts                  U
  v test
      app.e2e-spec.ts          U
      jest-e2e.json            U
    .eslintrc.js               U
    .gitignore                 U
    .prettierrc                U
    nest-cli.json              U
    package.json               U
    README.md                  U
    tsconfig.build.json        U
    tsconfig.json              U
    yarn.lock                  U
```

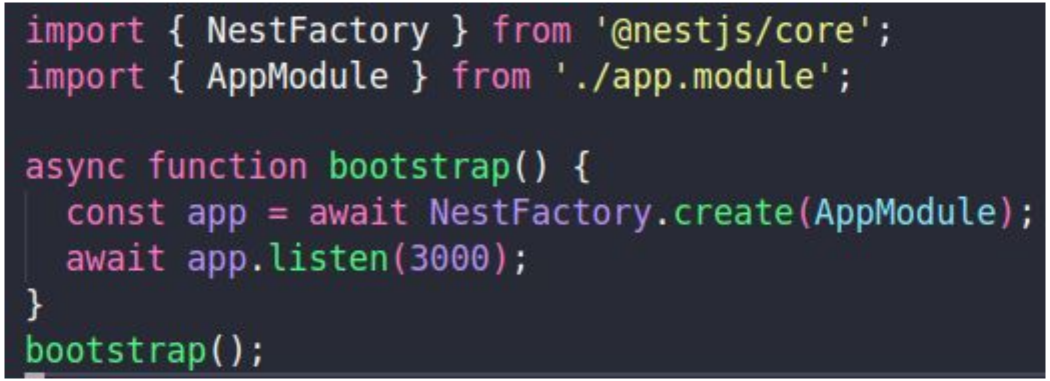
A basic service with a single method



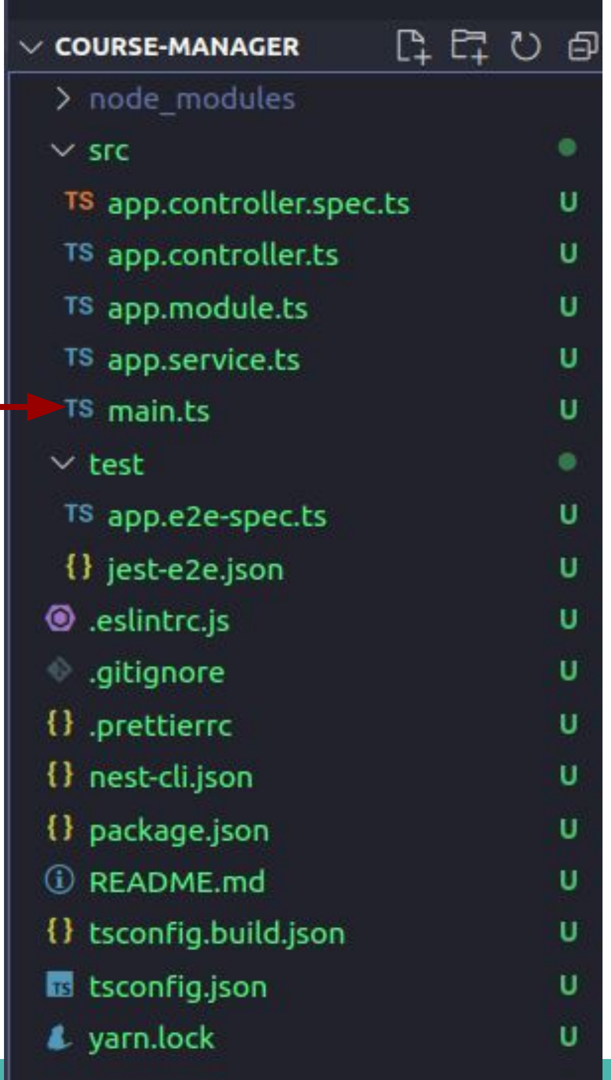


```
import { Injectable } from '@nestjs/common';

@Injectable()
export class AppService {
  getHello(): string {
    return 'Hello World!';
  }
}
```

The entry file of the application which uses the core function `NestFactory` to create a Nest application instance



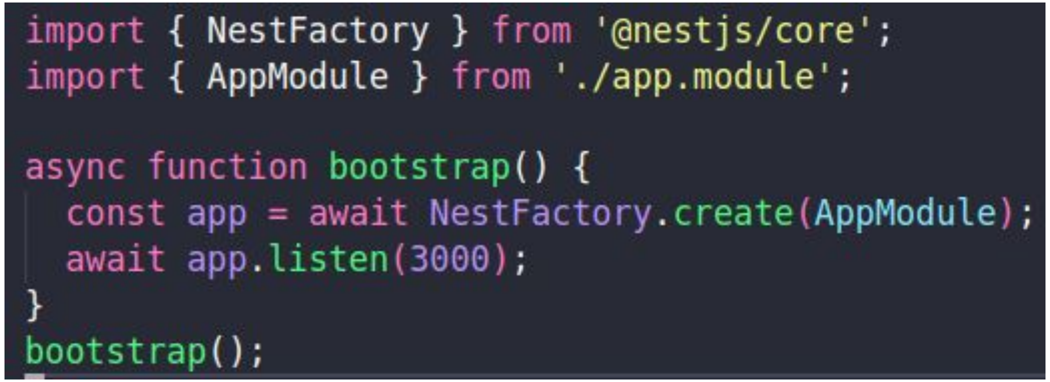
```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  await app.listen(3000);
}
bootstrap();
```



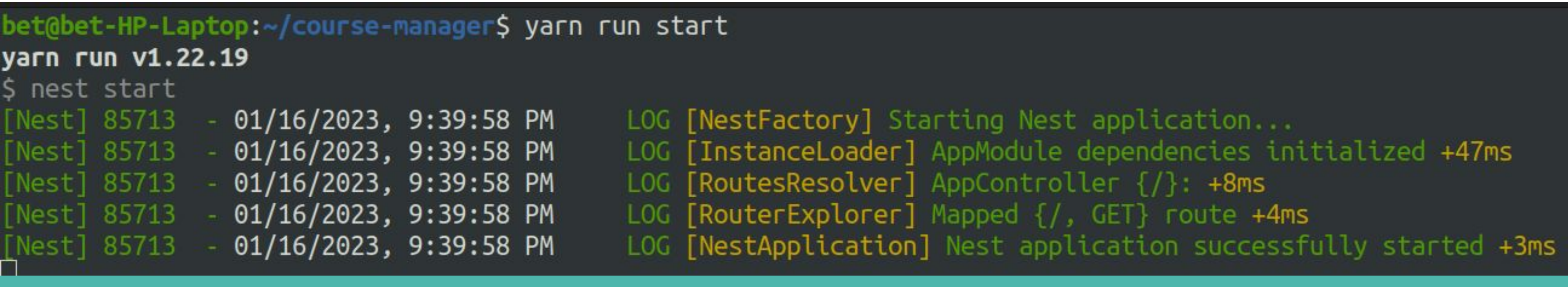
```
COURSE-MANAGER
  > node_modules
  v src
      app.controller.spec.ts   U
      app.controller.ts        U
      app.module.ts            U
      app.service.ts           U
      main.ts                  U
  v test
      app.e2e-spec.ts          U
      jest-e2e.json            U
    .eslintrc.js               U
    .gitignore                 U
    .prettierrc                U
    nest-cli.json              U
    package.json               U
    README.md                  U
    tsconfig.build.json        U
    tsconfig.json              U
    yarn.lock                  U
```

Running the application

```
yarn run start
```



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

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  await app.listen(3000);
}
bootstrap();
```

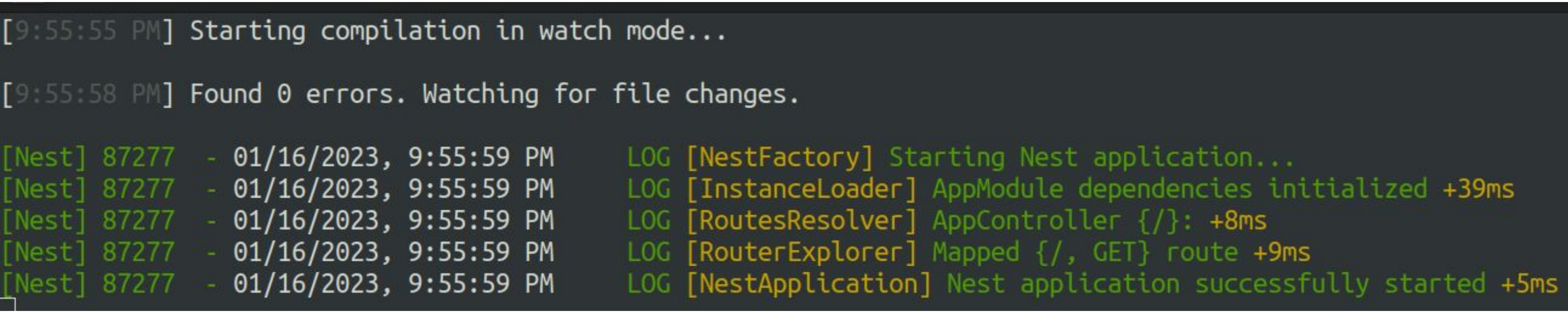


```
bet@bet-HP-Laptop:~/course-manager$ yarn run start
yarn run v1.22.19
$ nest start
[Nest] 85713  - 01/16/2023, 9:39:58 PM     LOG [NestFactory] Starting Nest application...
[Nest] 85713  - 01/16/2023, 9:39:58 PM     LOG [InstanceLoader] AppModule dependencies initialized +47ms
[Nest] 85713  - 01/16/2023, 9:39:58 PM     LOG [RoutesResolver] AppController {/}: +8ms
[Nest] 85713  - 01/16/2023, 9:39:58 PM     LOG [RouterExplorer] Mapped {/, GET} route +4ms
[Nest] 85713  - 01/16/2023, 9:39:58 PM     LOG [NestApplication] Nest application successfully started +3ms
```

Running the application in Watch mode

To watch for changes in your files, you can run the following command

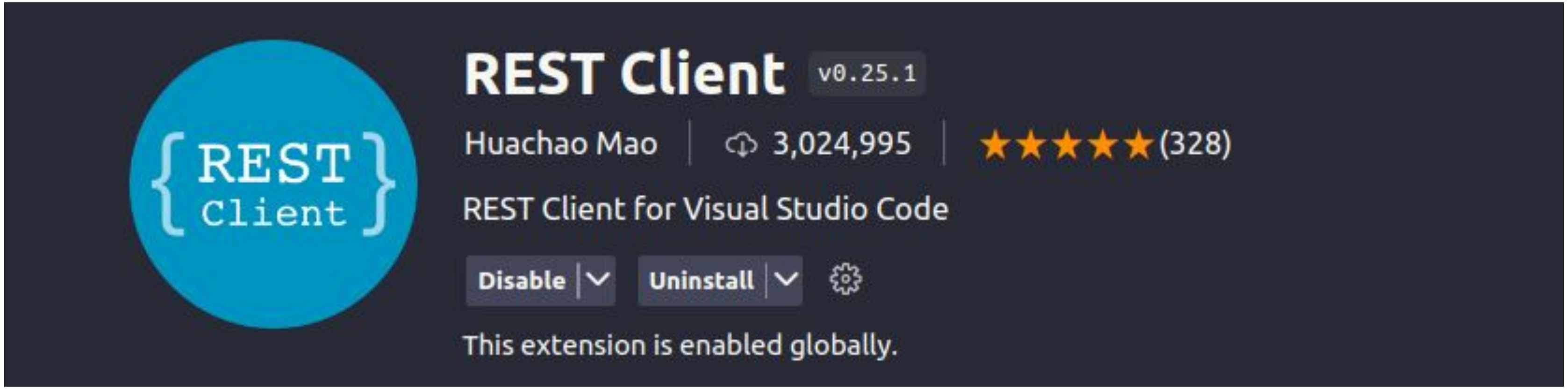
```
yarn run start:dev
```



```
[9:55:55 PM] Starting compilation in watch mode...

[9:55:58 PM] Found 0 errors. Watching for file changes.

[Nest] 87277  - 01/16/2023, 9:55:59 PM     LOG [NestFactory] Starting Nest application...
[Nest] 87277  - 01/16/2023, 9:55:59 PM     LOG [InstanceLoader] AppModule dependencies initialized +39ms
[Nest] 87277  - 01/16/2023, 9:55:59 PM     LOG [RoutesResolver] AppController {/}: +8ms
[Nest] 87277  - 01/16/2023, 9:55:59 PM     LOG [RouterExplorer] Mapped {/, GET} route +9ms
[Nest] 87277  - 01/16/2023, 9:55:59 PM     LOG [NestApplication] Nest application successfully started +5ms
```

Testing the API with REST Client

You can use REST Client VS Code extension to test APIs


REST Client v0.25.1

Huachao Mao | 3,024,995 | ★★★★★ (328)

REST Client for Visual Studio Code

This extension is enabled globally.

Testing the API with REST Client

Create a file named **`api-requests.rest`** under **`src`** directory

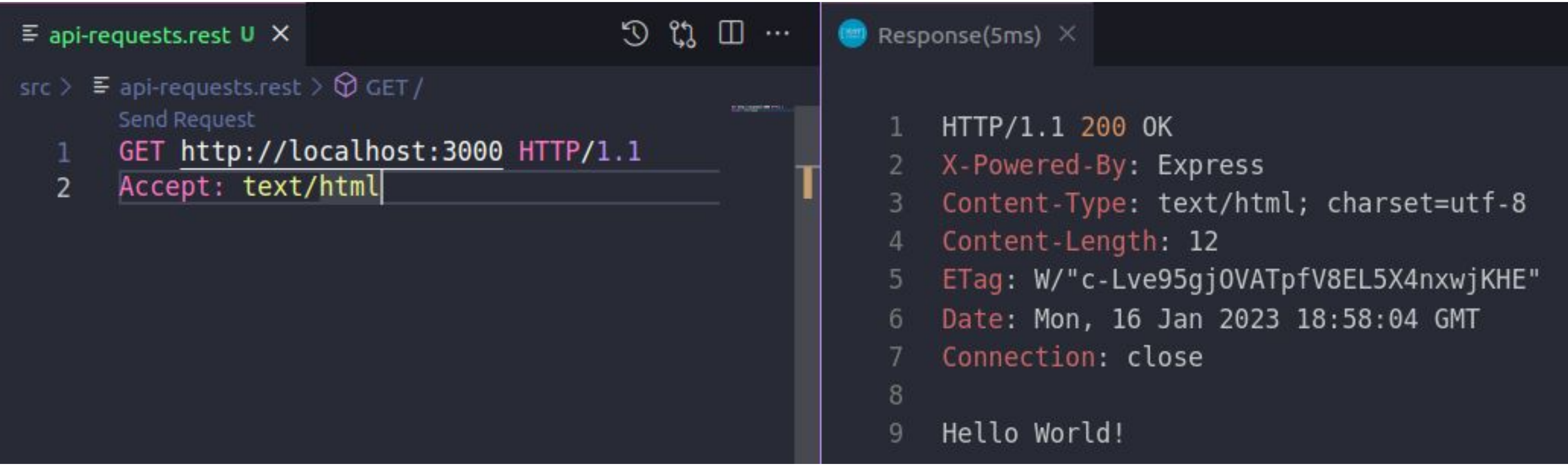
Add the following line in the **`api-requests.rest`** file

```
GET http://localhost:3000 HTTP/1.1
```

Click the **`Send Request`** link shown above the above line

You should see the API response in a different tab as shown in the next slide

Testing the API with REST Client



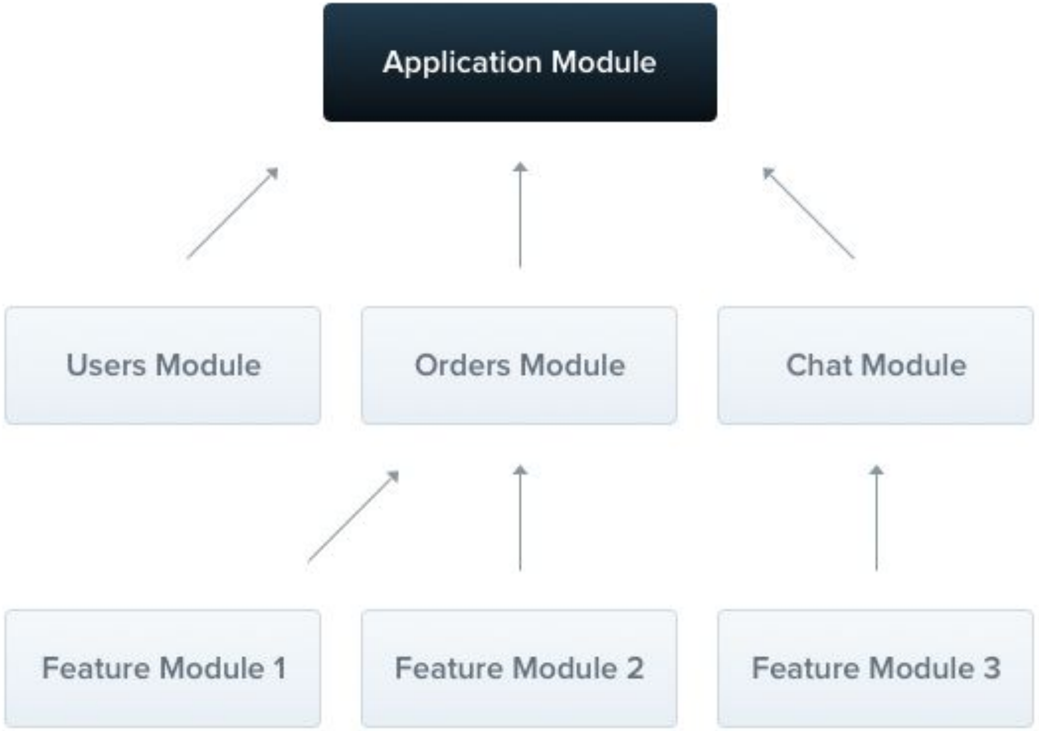
```
GET http://localhost:3000 HTTP/1.1
Accept: text/html
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 12
ETag: W/"c-Lve95gjOVATpfV8EL5X4nxwjKHE"
Date: Mon, 16 Jan 2023 18:58:04 GMT
Connection: close

Hello World!
```

Modules

A `module` is a class annotated with a `@Module()` decorator

The `@Module()` decorator provides **metadata** that `Nest` makes use of to **organize the application structure**


Application Module
Users Module
Orders Module
Chat Module
Feature Module 1
Feature Module 2
Feature Module 3

Modules

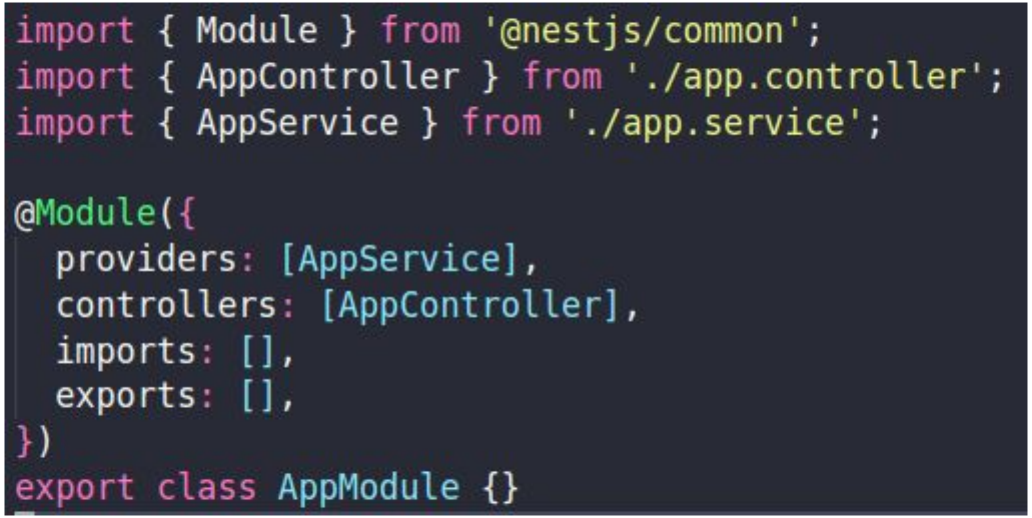
Each application has at least one module, a `root` module

The `root` module is the starting point Nest uses to build the **application graph**

The **application graph** is the internal data structure Nest uses to resolve module and provider relationships and dependencies

Modules

The `@Module()` decorator takes a single object whose properties describe the module



```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';

@Module({
  providers: [AppService],
  controllers: [AppController],
  imports: [],
  exports: [],
})
export class AppModule {}
```

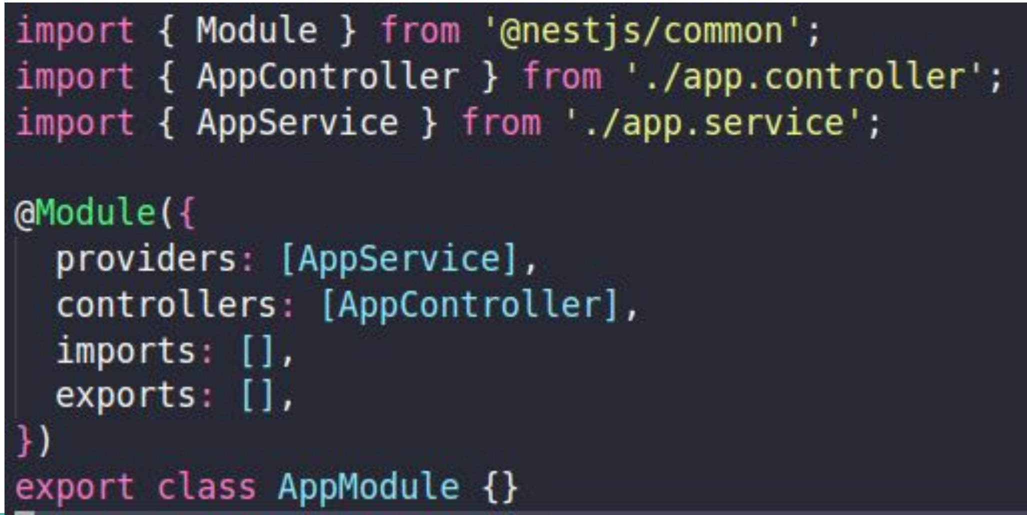
Modules

The `@Module()` decorator takes a single object whose properties describe the module

`providers`

will be instantiated by the Nest injector

they may be shared at least across this module



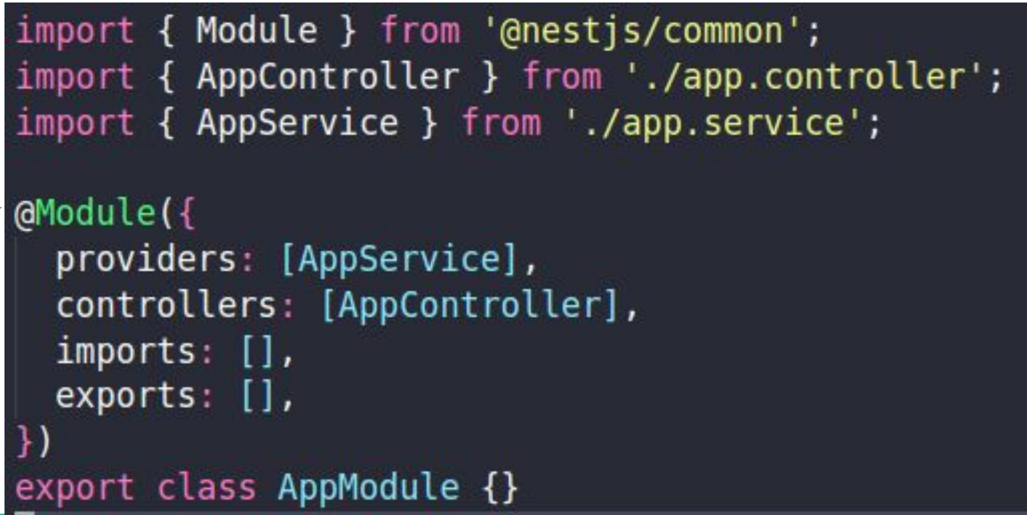
```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';

@Module({
  providers: [AppService],
  controllers: [AppController],
  imports: [],
  exports: [],
})
export class AppModule {}
```

Modules

The `@Module()` decorator takes a single object whose properties describe the module

`controllers`

the set of controllers defined in this module which have to be instantiated



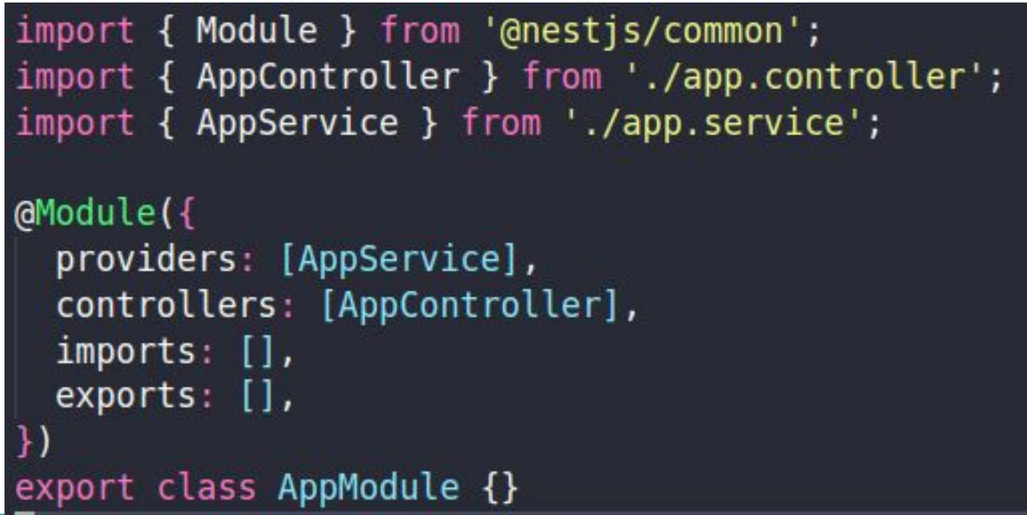
```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';

@Module({
  providers: [AppService],
  controllers: [AppController],
  imports: [],
  exports: [],
})
export class AppModule {}
```

Modules

The `@Module()` decorator takes a single object whose properties describe the module

imports

the list of imported modules that export the providers which are required in this module



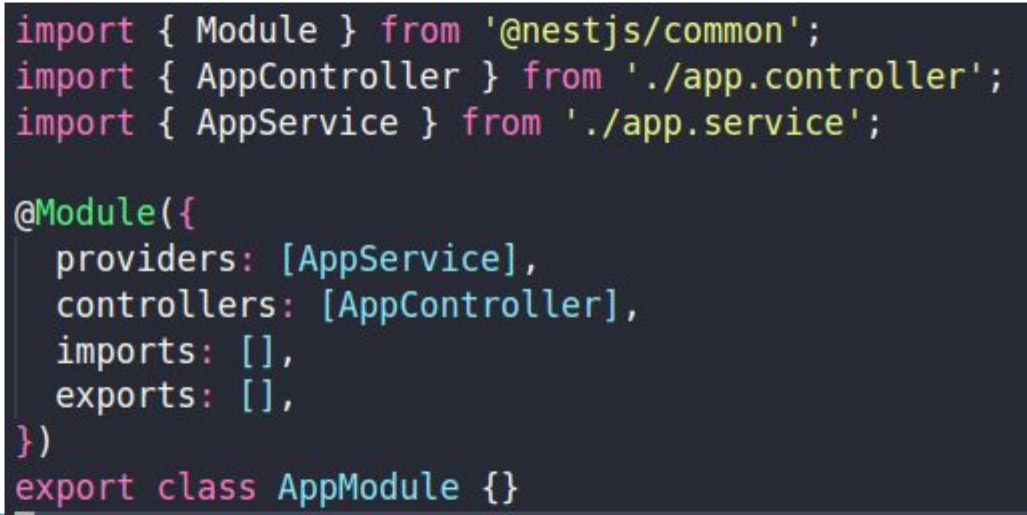
```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';

@Module({
  providers: [AppService],
  controllers: [AppController],
  imports: [],
  exports: [],
})
export class AppModule {}
```

Modules

The `@Module()` decorator takes a single object whose properties describe the module

`exports`

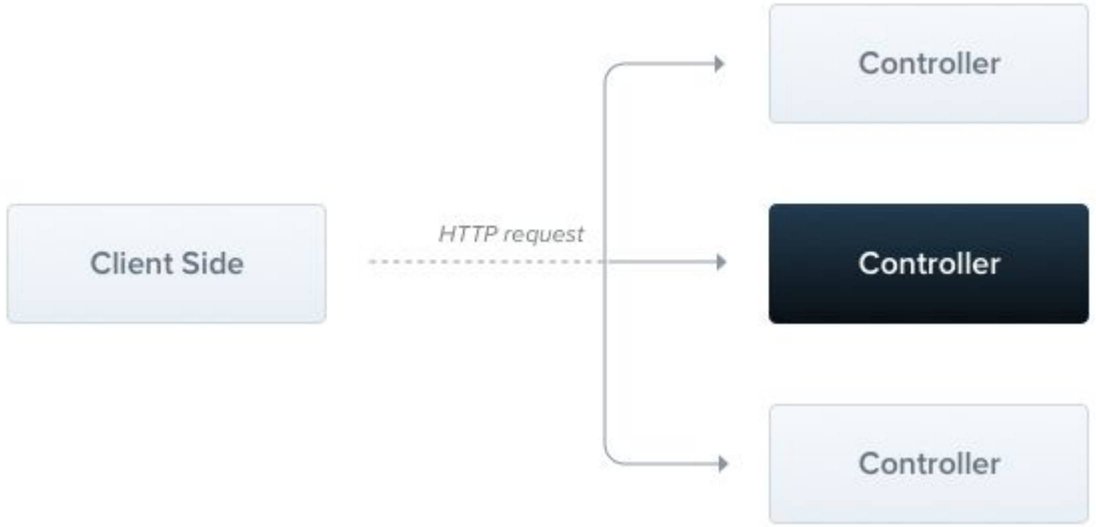
the subset of providers that are provided by this module and should be available in other modules which import this module



```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';

@Module({
  providers: [AppService],
  controllers: [AppController],
  imports: [],
  exports: [],
})
export class AppModule {}
```

Controllers

`Controllers` are responsible for **handling incoming requests** and **returning responses** to the client



Controllers

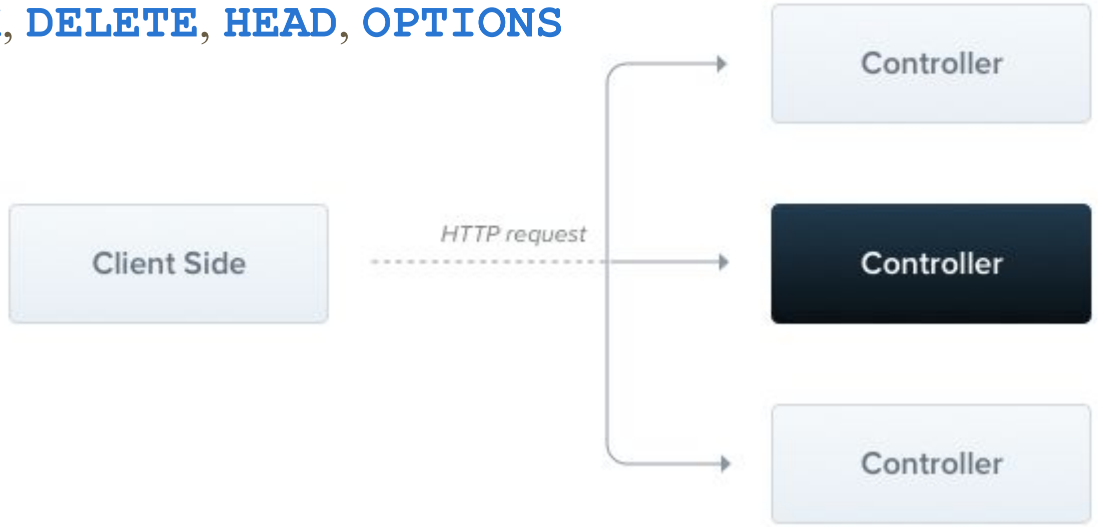
Routing

The **routing** mechanism controls which controller receives which requests

HTTP Requests

Http Requests contains the following information about the request

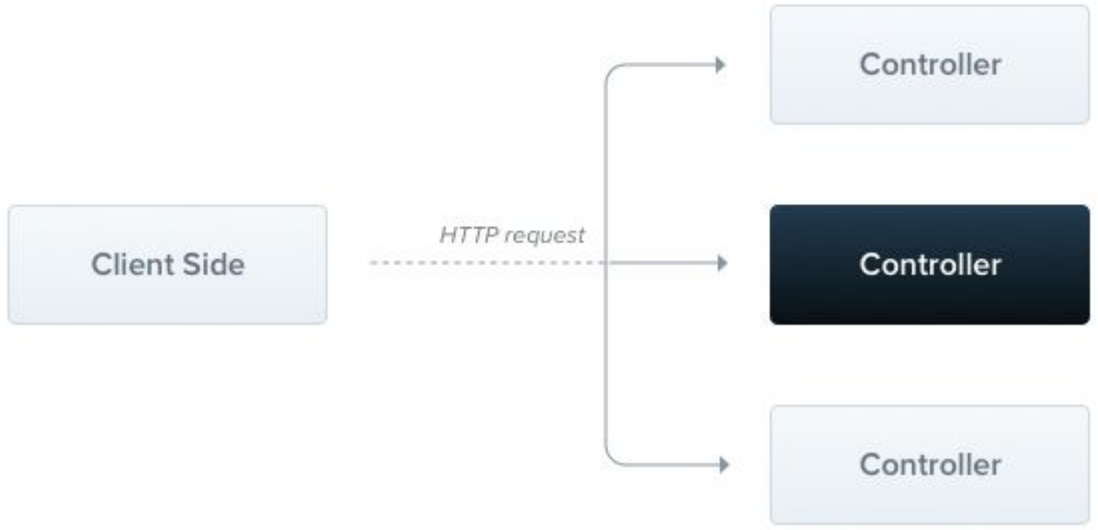
- Request Method
    - **`GET`**, **`POST`**, **`PUT`**, **`PATCH`**, **`DELETE`**, **`HEAD`**, **`OPTIONS`**
- Request Path
- Path or Query Parameter
- Request Body
- Request Header



Controllers

`Controllers` provides mechanisms to access the information contained in the HTTP request

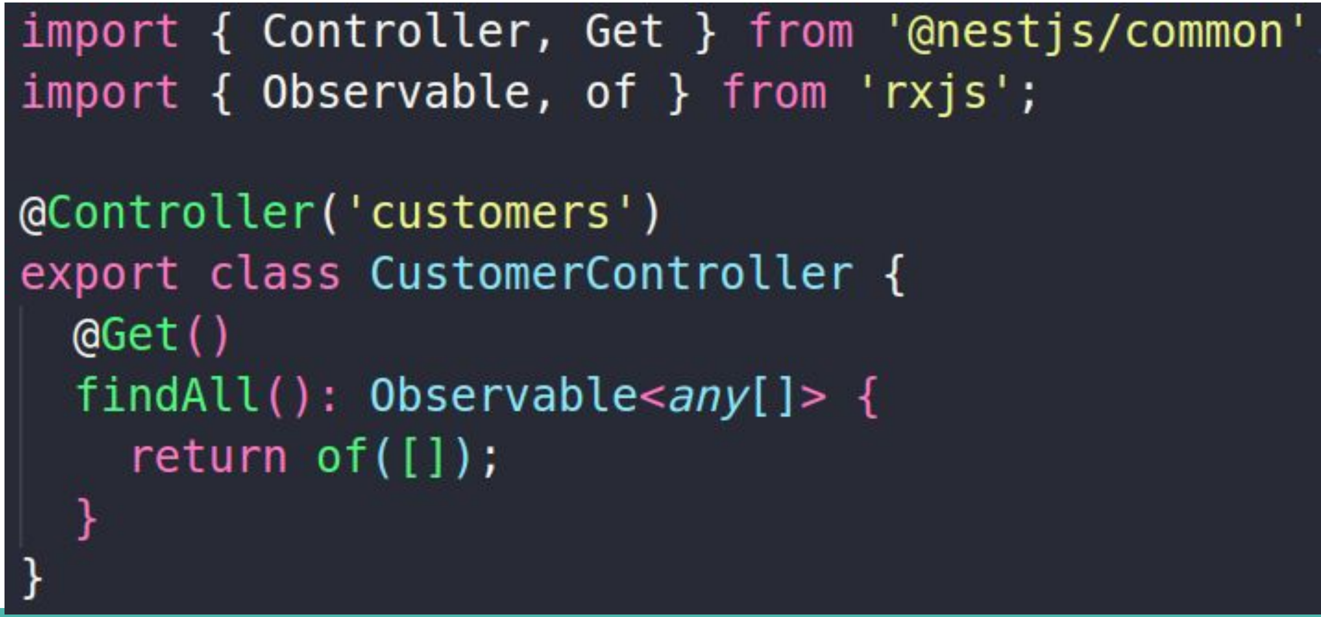
They are responsible for **handling incoming requests** and **returning responses** to the client



Controllers

The controller shown below handles

`GET /customers` request

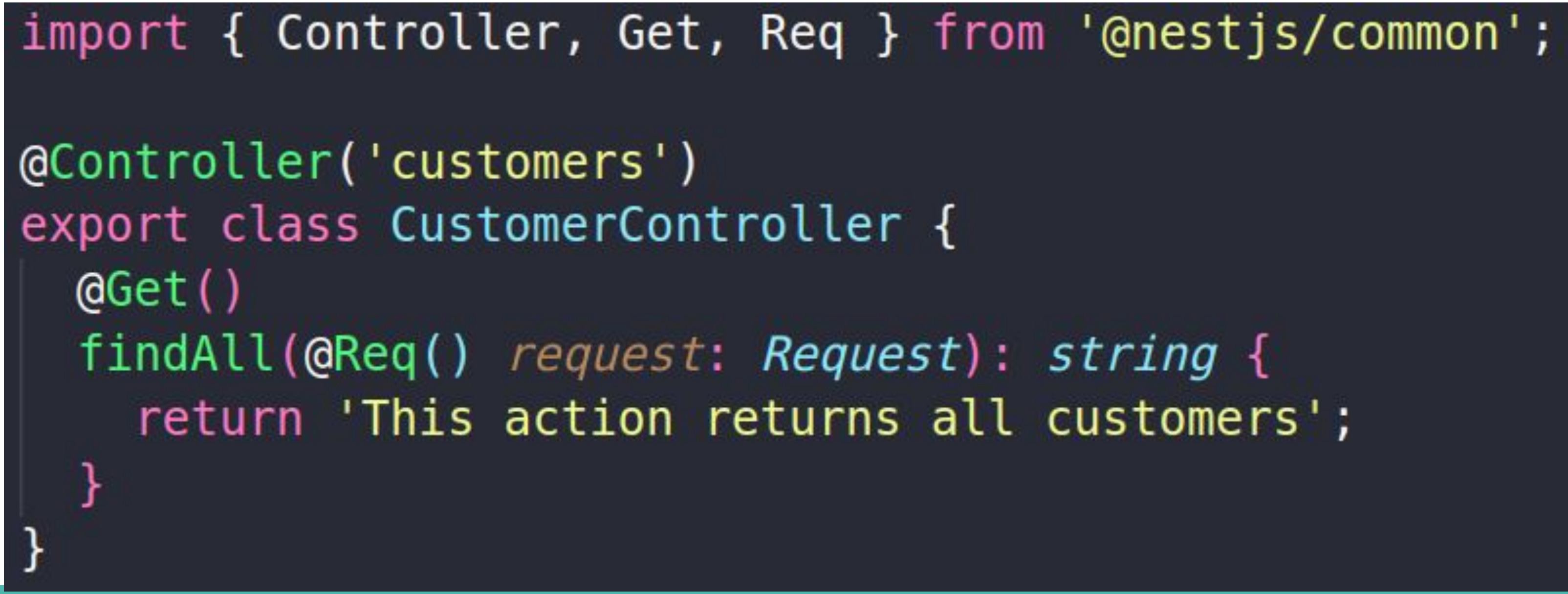


```
import { Controller, Get } from '@nestjs/common'
import { Observable, of } from 'rxjs';

@Controller('customers')
export class CustomerController {
  @Get()
  findAll(): Observable<any[]> {
    return of([]);
  }
}
```

Controllers

Request object

You can get request details using `@Req()` decorator



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

@Controller('customers')
export class CustomerController {
  @Get()
  findAll(@Req() request: Request): string {
    return 'This action returns all customers';
  }
}
```

Controllers

Request object

The request object represents the HTTP request and has properties for the request **query string**, **parameters**, HTTP **headers**, and **body**

In most cases, it's not necessary to grab these properties manually

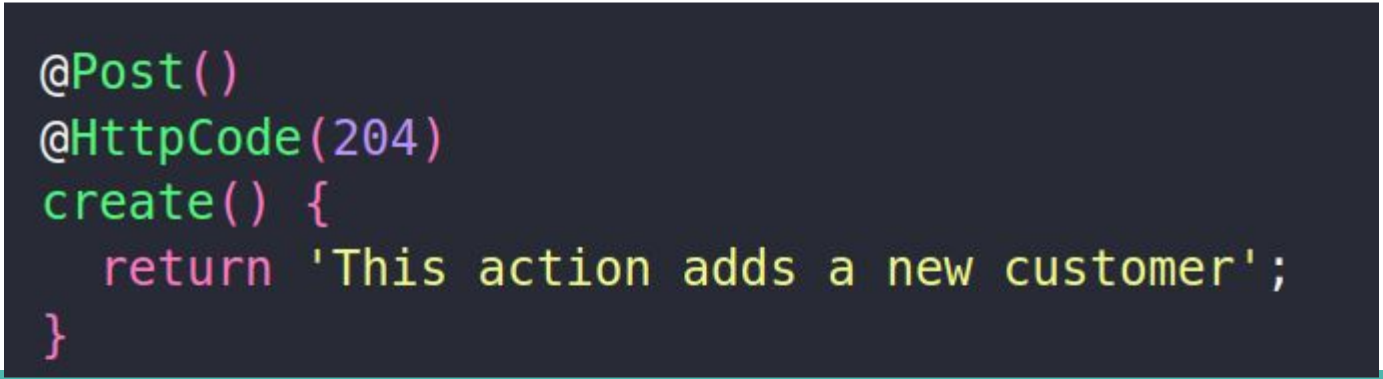
We can use dedicated decorators instead, such as `@Body()` or `@Query()`

Controllers

Status code

The response status code is always `200` by default, except for `POST` requests which are `201`

We can easily change this behavior by adding the `@HttpCode(...)` decorator at a handler level

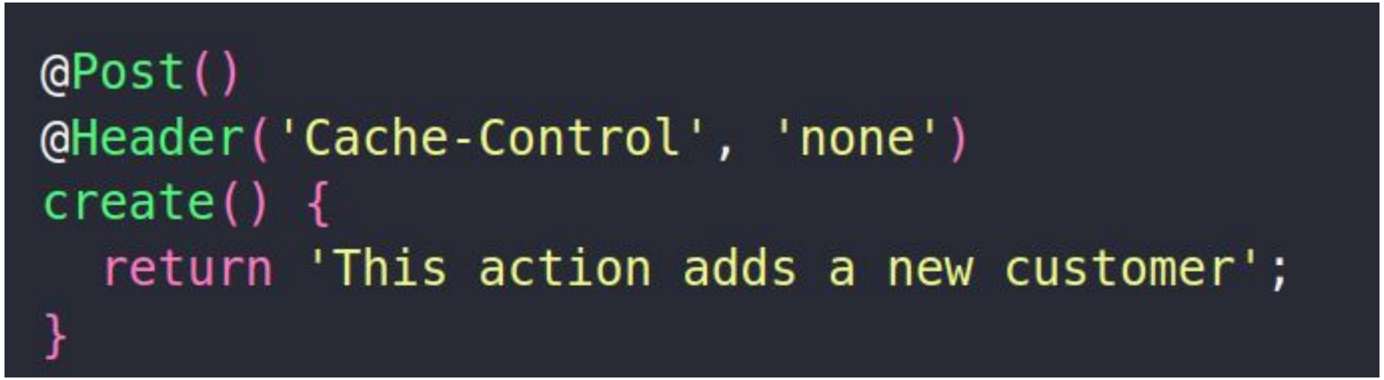


```
@Post()
@HttpCode(204)
create() {
  return 'This action adds a new customer';
}
```

Controllers

Headers

To specify a custom response header, you can either use a **`@Header()`** decorator or a library-specific response object (and call **`res.header()`** directly)

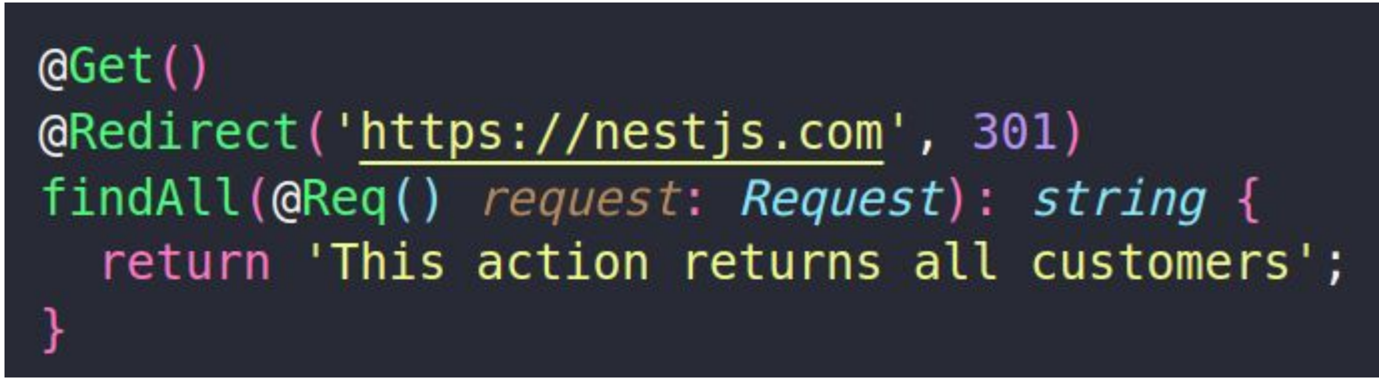


```
@Post()
@Header('Cache-Control', 'none')
create() {
  return 'This action adds a new customer';
}
```

Controllers

Redirection

To redirect a response to a specific URL, you can either use a **`@Redirect()`** decorator or a library-specific response object (and call **`res.redirect()`** directly)

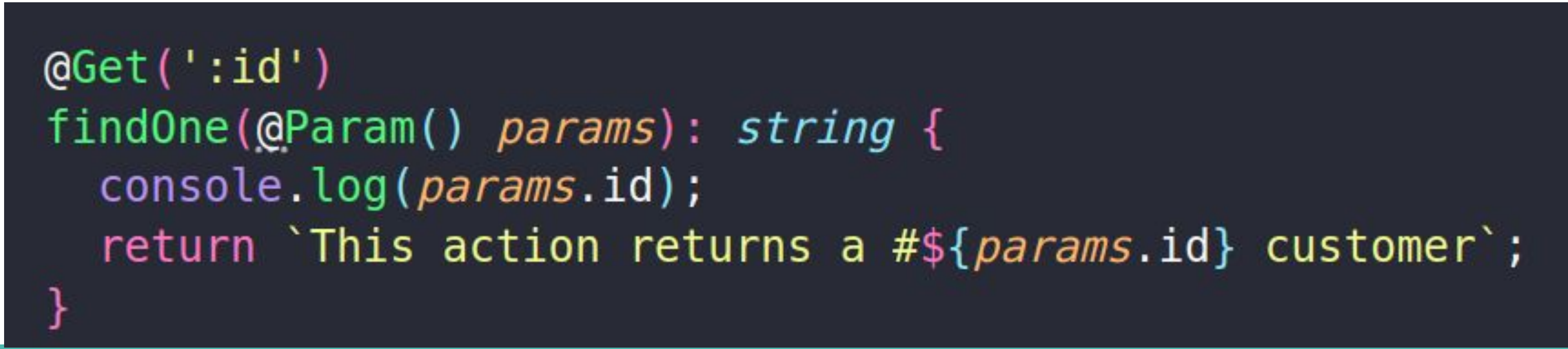


```
@Get()
@Redirect('https://nestjs.com', 301)
findAll(@Req() request: Request): string {
  return 'This action returns all customers';
}
```

Controllers

Route parameters

In order to define routes with parameters, we can add route parameter tokens in the path of the route to capture the dynamic value at that position in the request URL

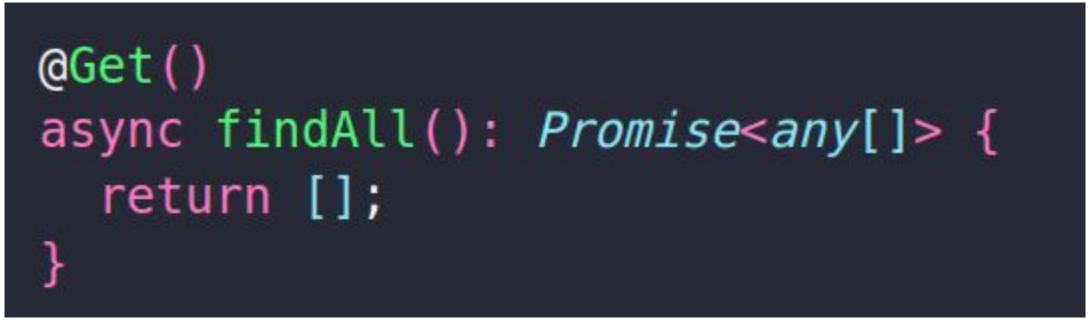


```
@Get(':id')
findOne(@Param() params): string {
  console.log(params.id);
  return `This action returns a #${params.id} customer`;
}
```

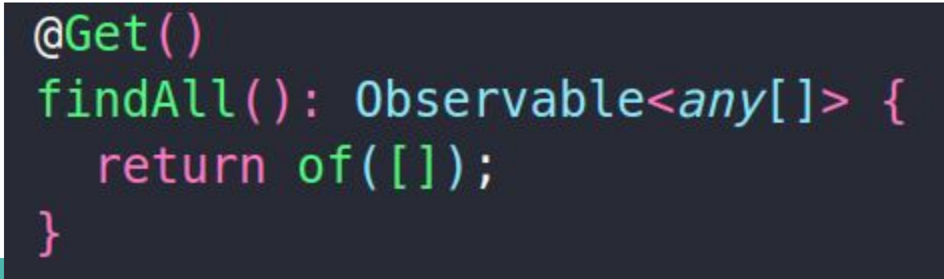
Controllers

Asynchronicity

Nest supports and works with `async` functions



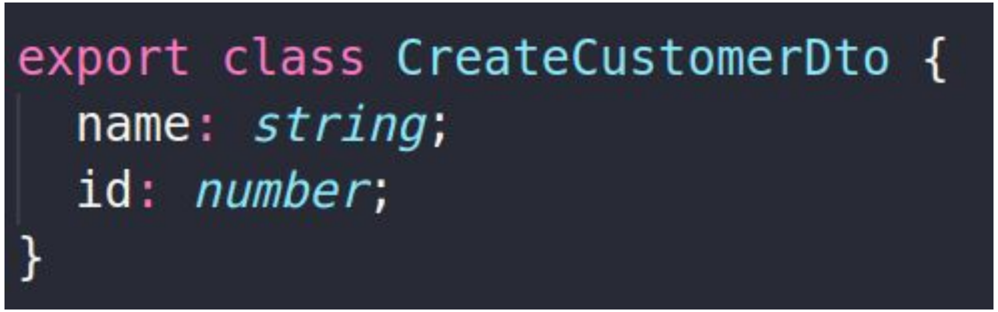
```
@Get()
async findAll(): Promise<any[]> {
  return [];
}
```



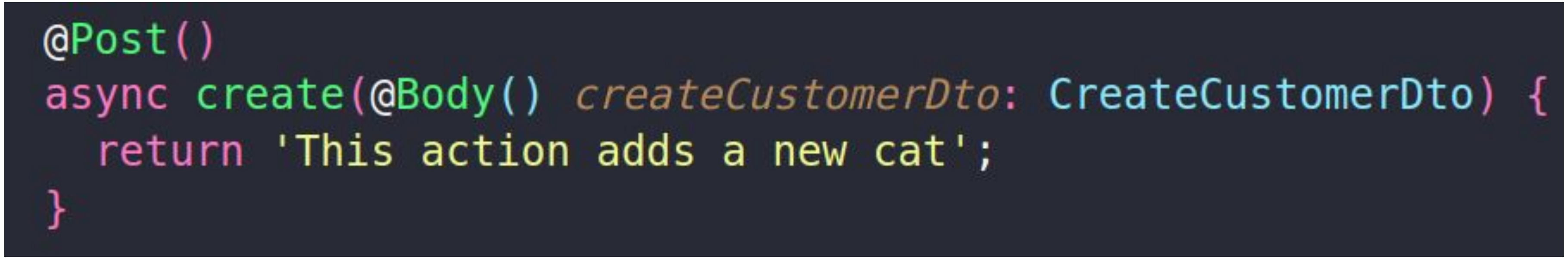
```
@Get()
findAll(): Observable<any[]> {
  return of([]);
}
```

Controllers

Request payloads



```
export class CreateCustomerDto {
  name: string;
  id: number;
}
```



```
@Post()
async create(@Body() createCustomerDto: CreateCustomerDto) {
  return 'This action adds a new cat';
}
```

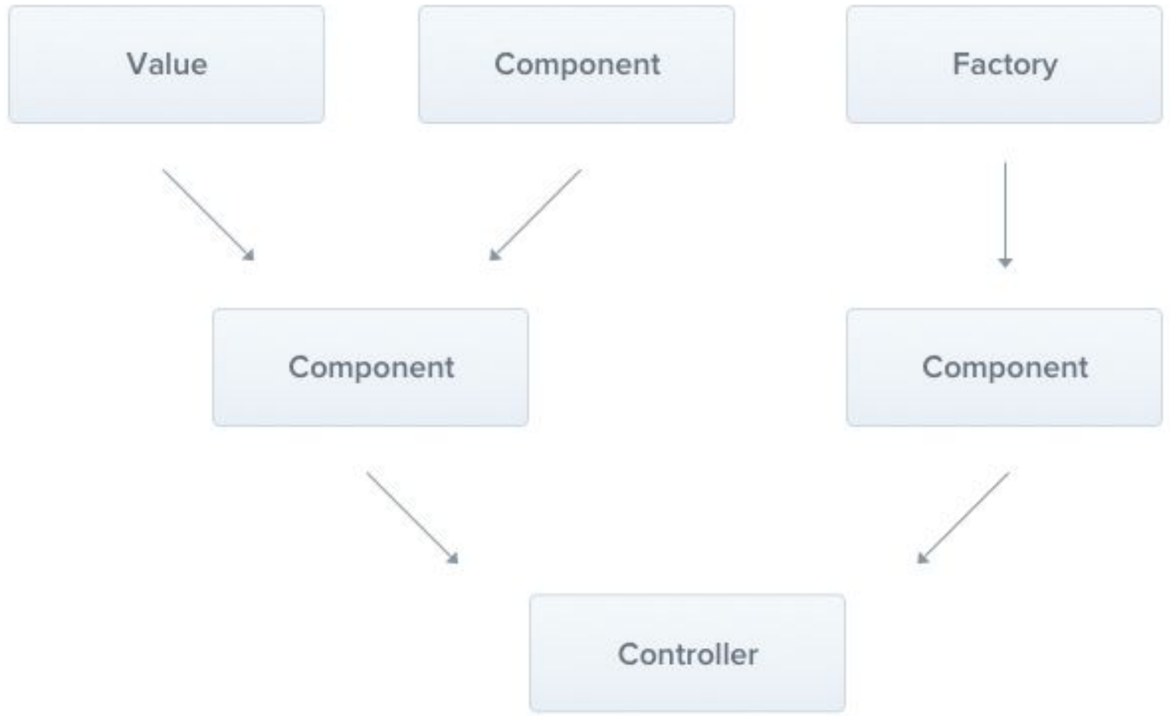
Providers

Providers are a fundamental concept in Nest

Many of the basic Nest classes may be treated as a `provider` – `services`, `repositories`, `factories`, `helpers`, and so on

The main idea of a provider is that it can be injected as a dependency

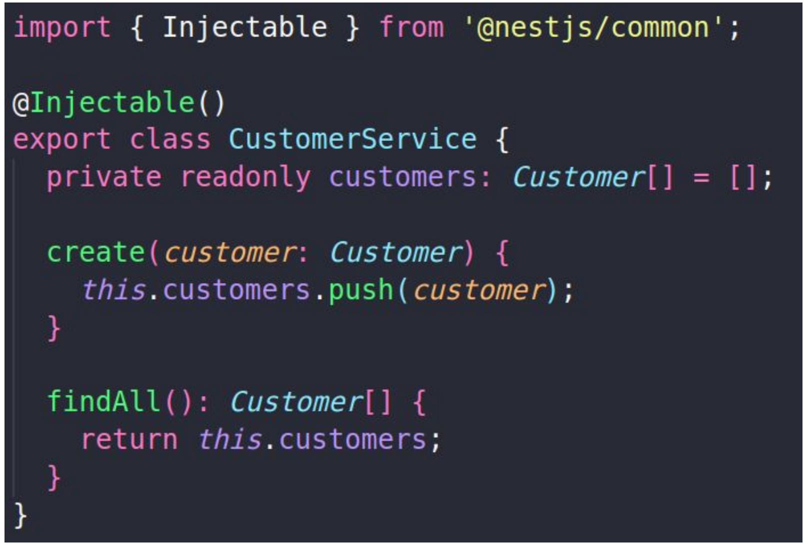
> this means objects can create various relationships with each other
>
> the function of "**wiring up**" instances of objects can largely be delegated to the Nest runtime system.


Value
Component
Factory
Component
Component
Controller

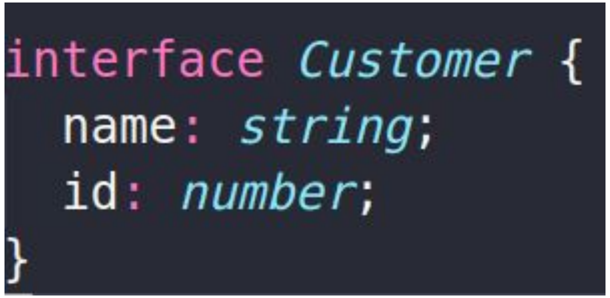
Providers

Service

The `@Injectable()` decorator attaches metadata, which declares that `CustomerService` is a class that can be managed by the **Nest IoC container**



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

@Injectable()
export class CustomerService {
  private readonly customers: Customer[] = [];

  create(customer: Customer) {
    this.customers.push(customer);
  }

  findAll(): Customer[] {
    return this.customers;
  }
}
```



```
interface Customer {
  name: string;
  id: number;
}
```

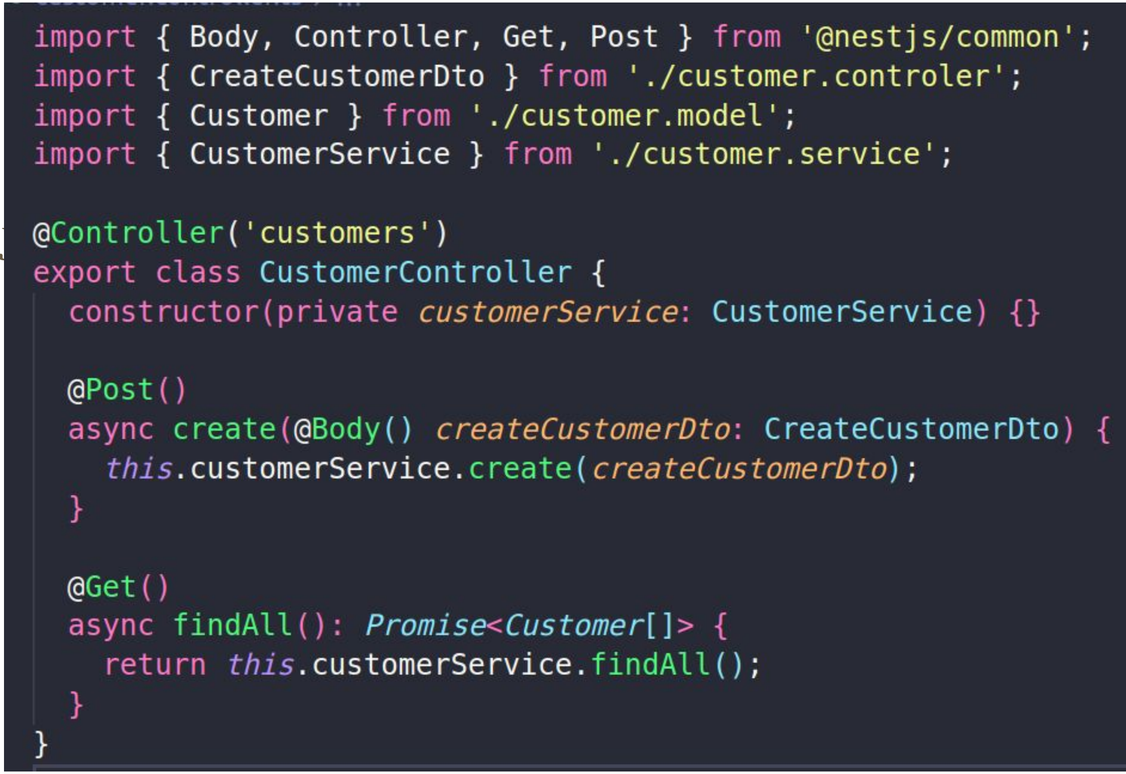
Providers

Using the service

Dependency injection

The `CustomerService` is in

through the class **constructor**



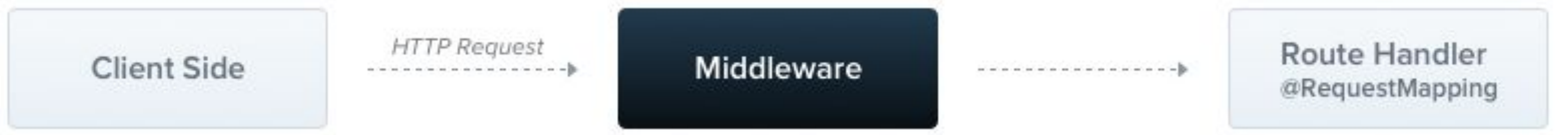
```
import { Body, Controller, Get, Post } from '@nestjs/common';
import { CreateCustomerDto } from './customer.controler';
import { Customer } from './customer.model';
import { CustomerService } from './customer.service';

@Controller('customers')
export class CustomerController {
  constructor(private customerService: CustomerService) {}

  @Post()
  async create(@Body() createCustomerDto: CreateCustomerDto) {
    this.customerService.create(createCustomerDto);
  }

  @Get()
  async findAll(): Promise<Customer[]> {
    return this.customerService.findAll();
  }
}
```

Middleware

`Middleware` is a function which is **called before the route handler**

`Middleware` functions **have access to the request and response objects**, and the `next()` middleware function in the application's request-response cycle

The next middleware function is commonly denoted by a variable named `next`



Middleware

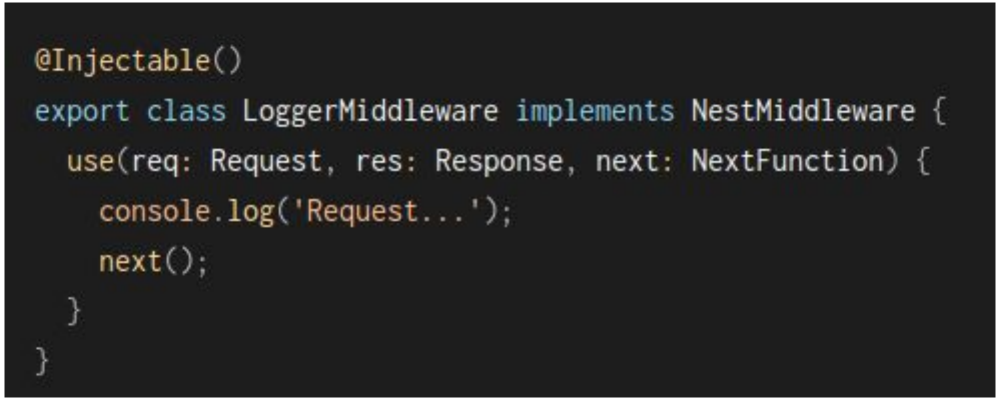
Middleware functions can perform the following tasks:

- execute any code
- make changes to the request and the response objects
- end the request-response cycle
- call the next middleware function in the stack

Middleware

You implement custom Nest middleware in either a **`function`**, or in a **`class`** with an **`@Injectable()`** decorator

The class should implement the **`NestMiddleware`** interface, while the function does not have any special requirements



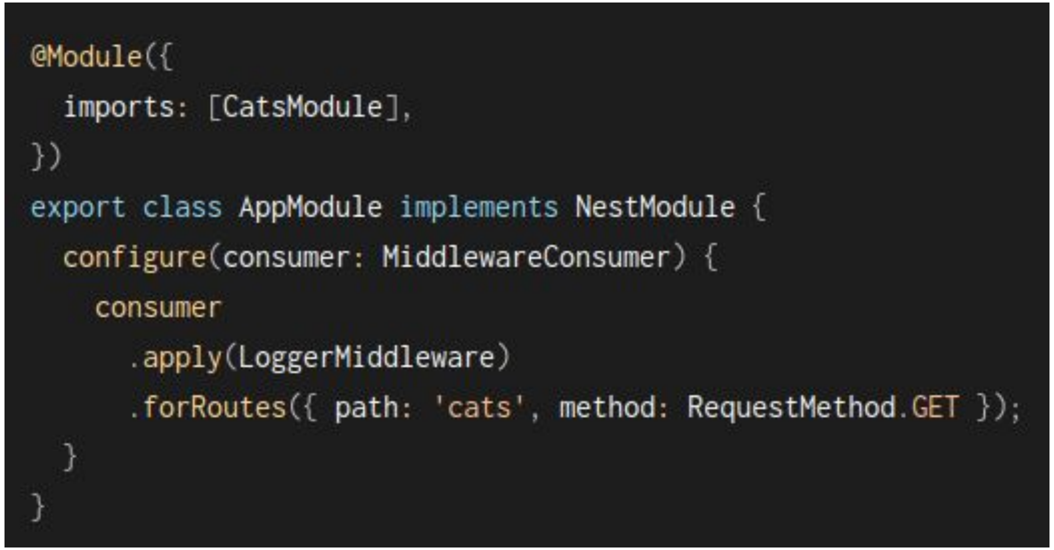
```
@Injectable()
export class LoggerMiddleware implements NestMiddleware {
  use(req: Request, res: Response, next: NextFunction) {
    console.log('Request...');
    next();
  }
}
```

Middleware

Applying middleware

Modules that include middleware have to implement the **`NestModule`** interface

We then use the **`configure()`** method



```
@Module({
  imports: [CatsModule],
})
export class AppModule implements NestModule {
  configure(consumer: MiddlewareConsumer) {
    consumer
      .apply(LoggerMiddleware)
      .forRoutes({ path: 'cats', method: RequestMethod.GET });
  }
}
```

Middleware

Functional middleware

```
import { Request, Response, NextFunction } from 'express';

export function logger(req: Request, res: Response, next: NextFunction) {
  console.log(`Request...`);
  next();
};
```

use it within the AppModule

```
consumer
  .apply(logger)
  .forRoutes(CatsController);
```

Middleware

Multiple Middleware

```
consumer.apply(cors(), helmet(), logger).forRoutes(CatsController);
```

Global middleware

```
const app = await NestFactory.create(AppModule);
app.use(logger);
await app.listen(3000);
```

Other Concepts

Exception filters

Pipes

Guards

Interceptors

ORM

Authentication/Authorization

References

https://docs.nestjs.com/

https://www.youtube.com/watch?v=GHTA143_b-s

https://www.youtube.com/watch?v=uAKzFhE3rxU

https://github.com/nestjs/nest/tree/master/sample