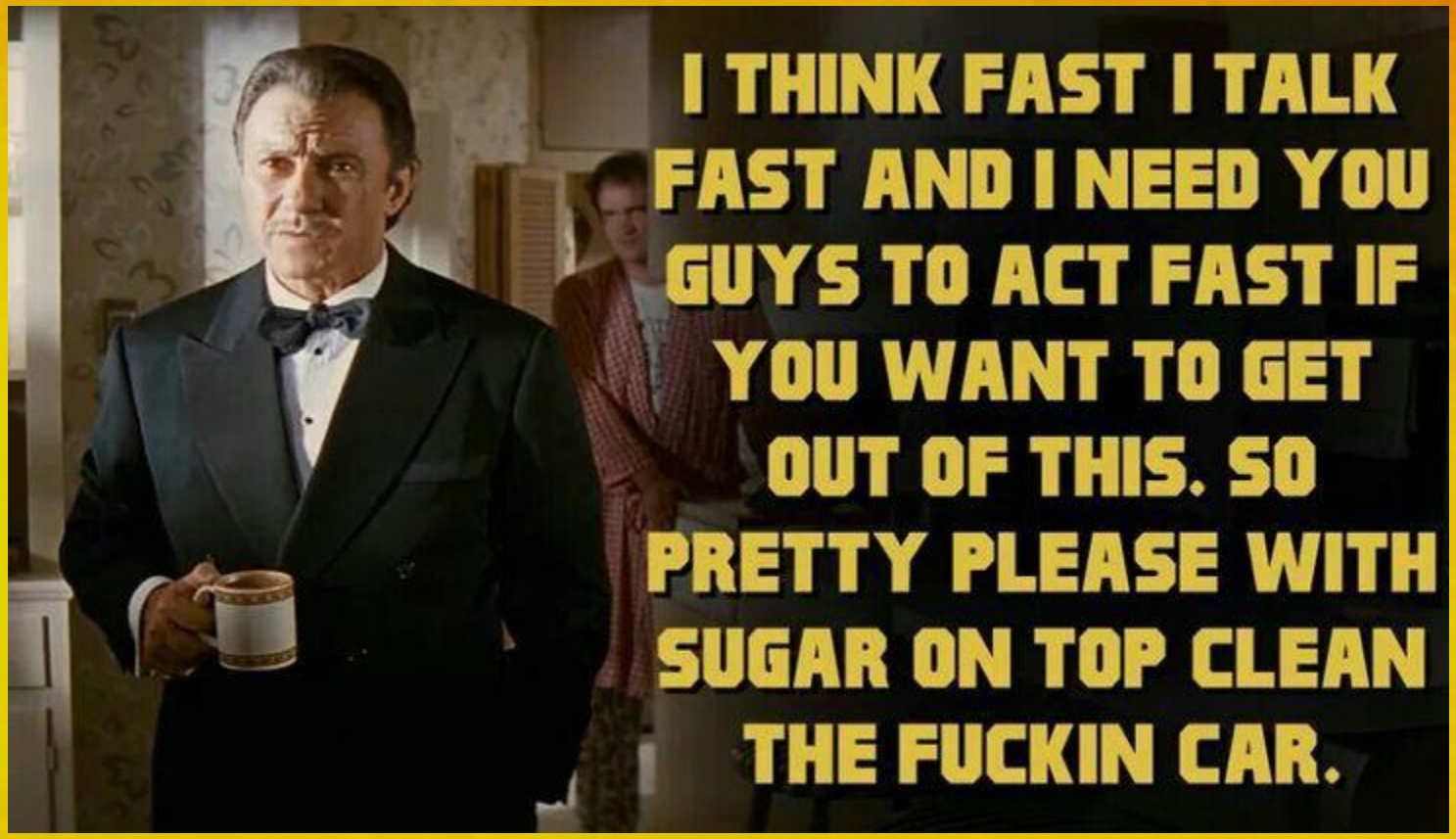
var title = “Lightning fast intro to NestJS”;

```
var info = {
    name: “Trifon Statkov”,
    occupation: “Tech Lead @ Strypes”,
    isSoftwareCraftsman: true
};
/* I might be able to show you a cool technology in less than one pomodoro time… or not SO I BETTER SKIP THIS AND HURRY THE F@#% UP!!! */
```

SOFIA JS

JS

Aug 18, 2021



SOFIA JS

JS

Aug 18, 2021

IMPORTANT:

IT'S NEST WITH AN S,

NOT AN X (AS IN NEXT)

- The other thing is also cool, but today we focus on NEST

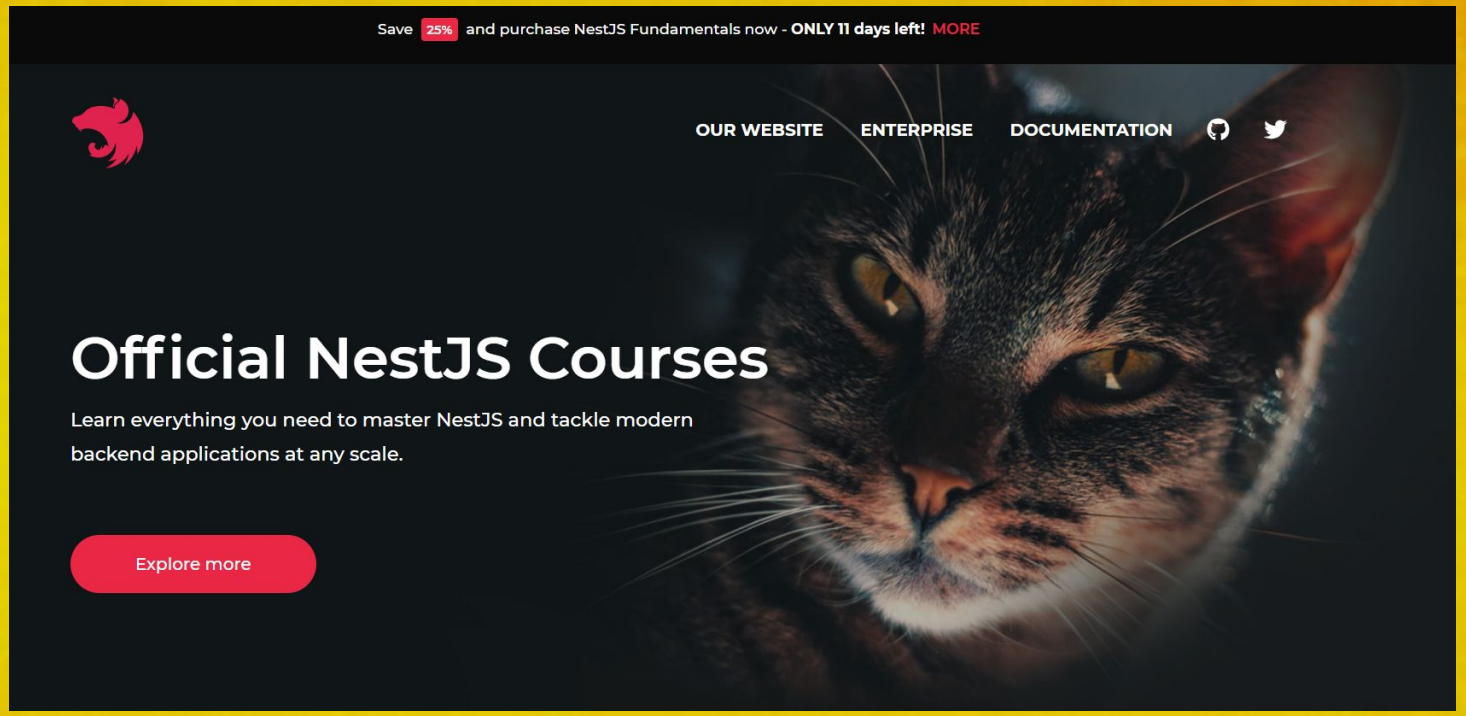
agenda();

1. What’s the problem?
2. How NestJS solves it?
3. How is a NestJS app structured?

…

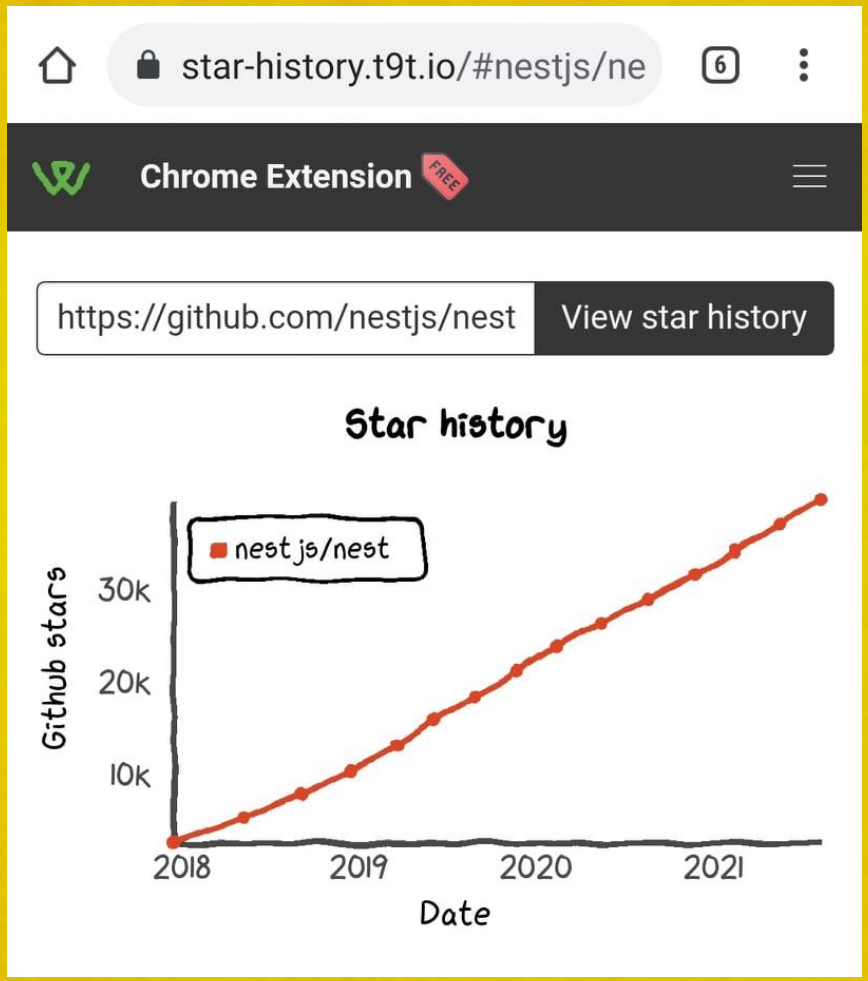
Questions

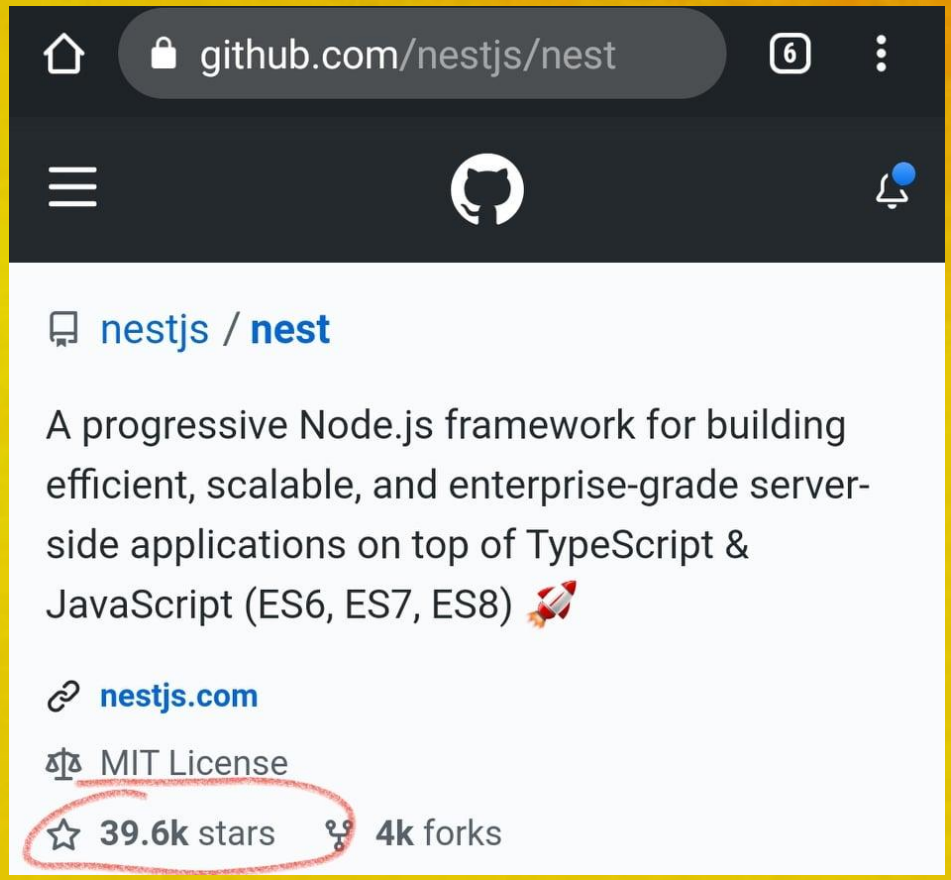
```
while (true) {
   drinkProperColdBeer();
}
```



- NestJS has nothing to do with:
  - nest-ing of Birds
  - or Cats although the official site uses cat imagery quite a lot

HELLO, NEST!


star-history.t9t.io/#nestjs/ne
6
Chrome Extension
FREE
https://github.com/nestjs/nest
View star history
Star history
nestjs/nest
Github stars
30k
20k
10k
2018
2019
2020
2021
Date


github.com/nestjs/nest
6
nestjs / nest
A progressive Node.js framework for building efficient, scalable, and enterprise-grade server-side applications on top of TypeScript & JavaScript (ES6, ES7, ES8)
nestjs.com
MIT License
39.6k stars
4k forks

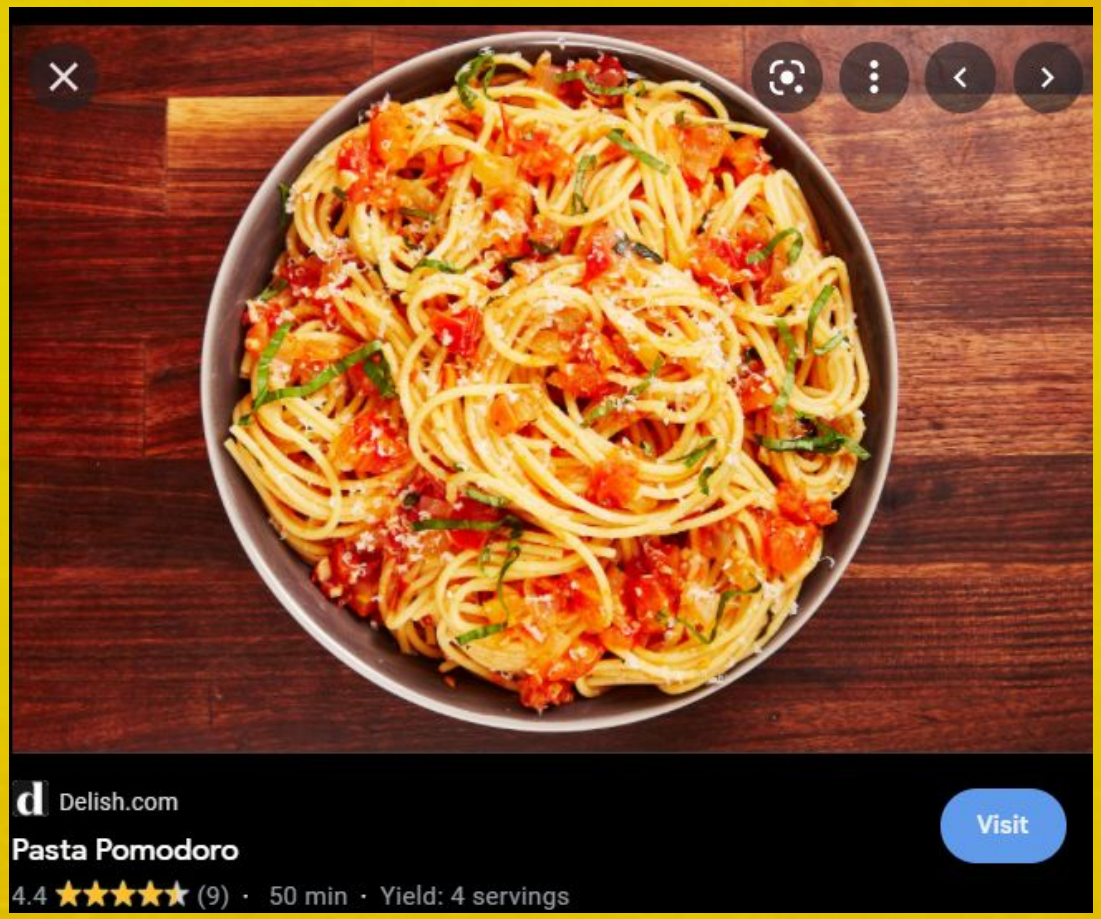

JS

WHAT AN **API** USUALLY DOES?

- Authorizes/authenticates users
- Persists some data in a database
- Guards requests so that you don’t access stuff you aren’t supposed to
- Transforms some data
- Validates some data
- Handles exceptions
- Receive requests in specific URLs and HTTP methods and return responses
- Executes some (complex) business logic

CODE DOING ALL OF THOSE **MIGHT**

LOOK LIKE THIS:



WHY NOT JUST USE EXPRESS?

- NO problem at all. You can do it, go for it!
- You **might** have to write a bit more code (which might not be a problem for you)
- You **might** have to choose **your own way to structure the code**, which might not make your application super easy to maintain
- You **might** not be very consistent
- You **might** even be constructing SQL queries yourself...

NESTJS CODE GENERATES EXPRESS APPLICATION OR FASTIFY APPLICATION

WHAT AN **API** USUALLY DOES...

IF YOU DECIDE TO USE NESTJS

- Authorizes/authenticates users
- Persists some data in a database
- Guards requests so that you don't access stuff you aren't supposed to
- Transforms some data
- Validates some data
- Handles exceptions
- Executes some (complex) business logic
- Receives requests/returns responses

ORM INTEGRATION
TypeORM / Prisma

GUARDS

PIPES
transform & validate

EXCEPTION FILTERS

CONTROLLERS
calls providers, receives requests, returns responses, sets up route URLs

PROVIDERS
useClass, useFactory, etc.

IN NESTJS THERE IS SPECIFIC COMPONENT (I.E. TYPESCRIPT FILE) FOR ALMOST EVERYTHING CALLED **BUILDING BLOCK**

GUARDS

PIPES
transform & validate

PROVIDERS
useClass, useFactory, etc.

EXCEPTION FILTERS

ORM INTEGRATION
TypeORM / Prisma

CONTROLLERS
calls providers, receives requests, returns responses, sets up route URLs

CONTROLLER

- Handles requests/responses
- nest g co <controller_name>

CONTROLLERS
calls providers, receives requests, returns responses, sets up route URLs

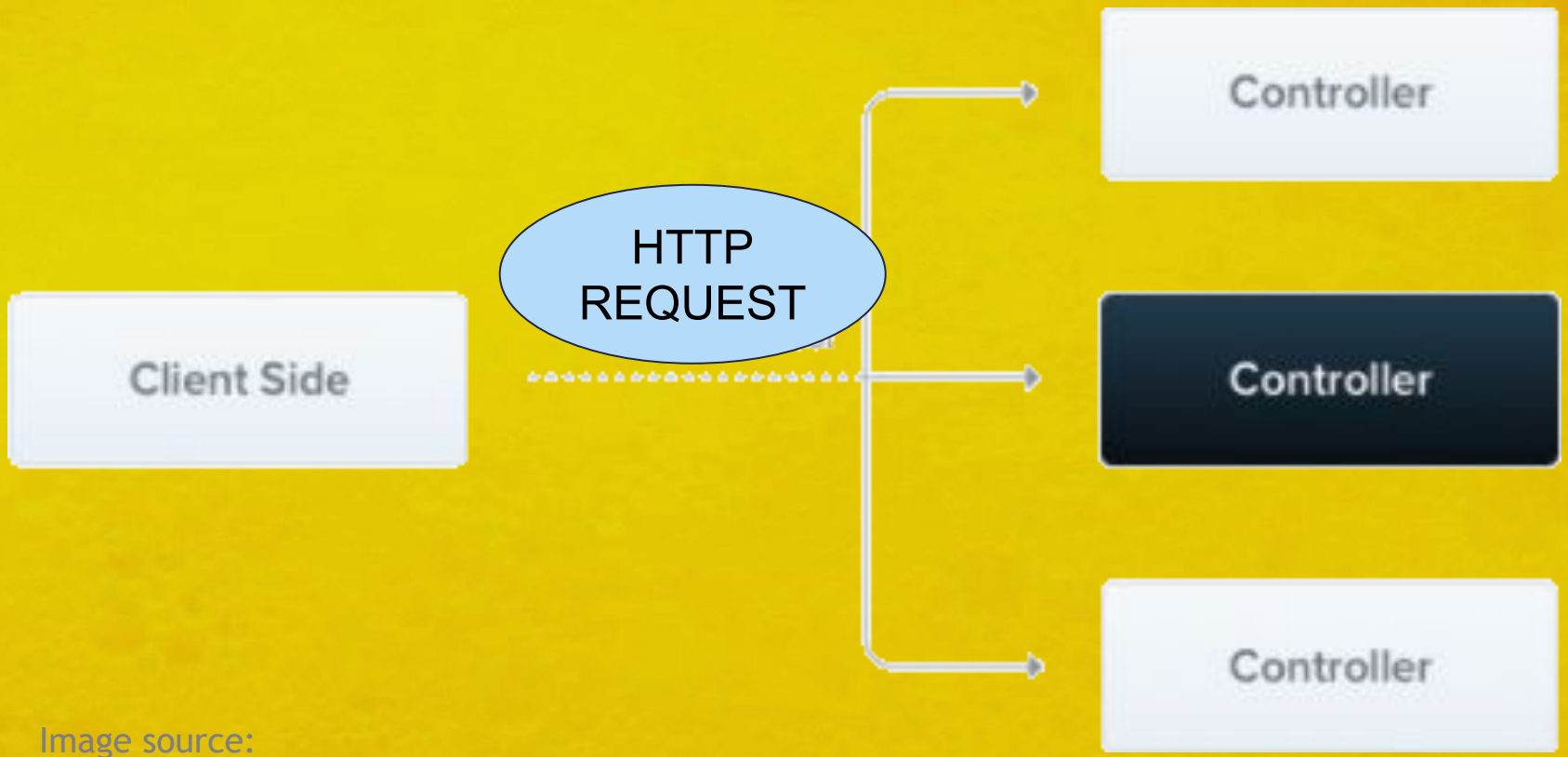


Image source:
https://docs.nestjs.com/controllers

SHOW ME THE CODEZ

WARNING: IF U DON'T LIKE .TS DON'T WATCH

```ts
import { Controller, Get, Post, Body } from
'@nestjs/common';
import { CatsService } from './cats.service';

@Controller('cats')
export class CatsController {
  constructor(private catsService: CatsService) {}

  @Post()
  async create(@Body() createCatDto: CreateCatDto) {
    this.catsService.create(createCatDto);
  }

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

// CATS.CONTROLLER.TS

SHOW ME THE CODEZ

JUST KIDDING YOU CAN ALSO USE .JS

```
import { Controller, Get, Post, Body, Bind,
Dependencies } from '@nestjs/common';
import { CatsService } from './cats.service';

@Controller('cats')
@Dependencies(CatsService)
export class CatsController {
  constructor(catsService) {
    this.catsService = catsService;
  }

  @Post()
  @Bind(Body())
  async create(createCatDto) {
    this.catsService.create(createCatDto);
  }
  /* MORE CODE */
```

// CATS.CONTROLLER.JS

HOW DO YOU WRITE CODE IN NEST?

- **GENERATE AND MODIFY**
- **Each building block** has a separate **set of responsibilities**
- There is a **CLI** that **generates building blocks** for you:
- You need a new controller? OK.
- You need a new module? Also OK.
- Example:
  - npm g co // **g** is for **g**enerate
- Easy.

PROVIDER

HINT 1: “ANGULAR”

HINT 2: “DEPENDENCY INJECTION”

PROVIDERS
useClass, useFactory, etc.

- Executes *complex* business logic
- Is injected into controllers by NestJS IoC container a.k.a. NestJS Runtime
- We are no longer concerned with initialization, yay!

SHOW ME THE CODEZ

HINT: TYPESCRIPT WITH DECORATORS

```
import { Injectable } from '@nestjs/common';
import { Cat } from './interfaces/cat.interface';

@Injectable()
export class CatsService {
  private readonly cats: Cat[] = [];

  create(cat: Cat) {
    this.cats.push(cat);
  }

  findAll(): Cat[] {
    return this.cats;
  }
}
```

// CATS.SERVICE.TS

Example source code coming almost straight from the official NestJS documentation at: https://docs.nestjs.com/providers

TIDYING THINGS UP EVEN MORE WITH MODULES

[

MODULE A / FEATURE A

- PROVIDERS
  useClass, useFactory, etc.
- GUARDS
- PIPES
  transform & validate
- EXCEPTION FILTERS
- ORM INTEGRATION
  TypeORM / Prisma
- CONTROLLERS
  calls providers, receives requests, returns responses, sets up route URLs

,

MODULE B / FEATURE B

- PROVIDERS
  useClass, useFactory, etc.
- GUARDS
- PIPES
  transform & validate
- EXCEPTION FILTERS
- ORM INTEGRATION
  TypeORM / Prisma
- CONTROLLERS
  calls providers, receives requests, returns responses, sets up route URLs

, . . .

]

INSIDE THE LIFE OF A MODULE

- Has own imports/exports for building blocks
- One module per feature
- Isolates specific building blocks into their own THING
- Separates concerns
- NestJS Inversion of Control Container handles Dependency Injection for you

SHOW ME THE CODEZ

STICKING TO .TS, THOUGH, BECAUSE OF TYPE SAFETY

```
import { Module } from '@nestjs/common';
import { CatsController } from
'./cats.controller';
import { CatsService } from
'./cats.service';

@Module({
  controllers: [CatsController],
  providers: [CatsService],
   imports: [AnotherModule],
   exports: [CatsService],
})
export class CatsModule {}
```

// CATS/CATS.MODULE.TS

THERE ARE EVEN

MORE

POSSIBILITIES

MIDDLEWARE

GIVES EVEN MORE CONTROL OVER HOW REQUESTS ARE HANDLED

MIGHT GIVING THE REQUEST TO THE NEXT MIDDLEWARE OR BUILDING BLOCK IN THE REQUEST LIFECYCLE

EACH MIDDLEWARE CAN CALL THE NEXT() METHOD OR ABORT THE REQUEST LIFECYCLE

NestJS Building Blocks: Middleware (1)

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.
- if the current middleware function does not end the request-response cycle, it must call `next()` to pass control to the next middleware function. Otherwise, the request will be left hanging.

NestJS Building Blocks: Middleware (2)

logger.middleware.ts JS

```
import { Injectable, NestMiddleware } from '@nestjs/common';
import { Request, Response, NextFunction } from 'express';

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

GRAPHQL INTEGRATION

GRAPHQL INTEGRATION

YES!

NEST SUPPORTS GRAPHQL INTEGRATION

LOWER RESPONSE SIZE FOR SUPER SMOOTH UX EVEN ON SLOWER SPEEDS

SOLVING THE SQL MESS

- Use TypeORM or Prisma or Sequalize to introduce
  - entity classes
  - migrations
- Those help you NOT write SQL
- Sometimes you might need to write a little for more complex queries

ABSTRACTING THE DB: TypeORM

- https://typeorm.io/
- Latest problem of TypeORM: dubious support as the author recently claimed he is busy with another project
- Otherwise really cool, allows smooth abstraction of all DB tables/entities + migrations

ABSTRACTING THE DB: PRISMA

- https://www.prisma.io/
- Has custom abstract language for defining the schema
- Claims to achieve higher type safety than TypeORM
- Good support

PROS OF USING NEST (RECAP)

- Helps you **manage complexity**
- You write code with **specific structure**
- It **generates under the hood** Express/Fastify/etc. applications
- You (presumably) deliver **faster**
- You (presumably) maintain **easier**
- Due to clearer code structure your team **might** produce **less bugs** while introducing **new features**, new **team members** and the size of the application **grows**

SOME USEFUL RESOURCES

- **NESTJS DOCUMENTATION (GOOD): https://docs.nestjs.com/**

- **THEY ALSO OFFER PAID CERTIFICATION COURSE (BETTER): https://courses.nestjs.com/**

CODE SESSION TO FOLLOW ON JSTALKS THIS NOVEMBER

THAT'S OF COURSE IF NESTJS AUTHORS DON'T DO SOMETHING RIDICULOUSLY STUPID IN THE MEANTIME (EMBERJS, I *MIGHT BE* LOOKING AT YOU)

Aug 18, 2021

Thanks to our Sponsors:

Aug 18, 2021