NestJS | 1

Bootcamp

Discord | Official Documentation

Kushagra Acharya

Disclaimer

- This is an optional course and will not effect your academic credit
- If you're not interested and cannot fullfill any requirement or class rules you will be resulted for class dropout.

General Rules

- Having a laptop and a separate notebook is compulsory
- Faliure to answer at least 3 viva question will result in dissmissal.
- Faliure to complete homework/classwork without any valid result will be unacceptable.

Prerequisite

- Separate notebook/copy for notes
- NVM with Node Installed
- PC with VS Code Installed
- Stable Internet Connection

Introduction & Getting Started

CLI and Project

Firstly, We install nestjs in our machine

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

Navigate to work directory and run the following

> nest new <project_name>

```
nest new myapp
```

Actions

- We installed nestjs cli globally
- We used that CLI to make a new project

Result

- In terminal, there is success message of new project creation

Now?

- Go inside the new project `cd myapp`
- Open VS Code `code .`

Base Structure

- node_modules
  - delete and install `npm i`
- package.json
  - scripts
  - dependencies
- gitignore
  - what and why?

Discussion

- NPM vs Yarn
- NodeJs vs ReactJs vs NestJs
- Framework

Project Structure

- src folder
- test folder

Running the project

- refer to package.json
    - scripts
    - `npm run start:dev`
    - localhost:3000
    - should see `Hello World`
    - edit `app.service.ts` for new Message!

Task!

- Create a project called myapp
- Run the project in browser `localhost`
- Output must be:
  - Hello NestJs
- Submit your task

Congratulations

You Completed :

- NodeJS Basics
- Backend Concepts
- CLI Concepts
- Capable of making a new NestJS Project
- Running a NestJS Project

Homework :

- NodeJS and NestJS working principle diagram (classwork)
- What is the terminal code to `install` nestjs cli?
- What is the terminal code to `create new nestjs` project?

Viva questions:

- Authorization vs Authentication

NestJS | 2

Bootcamp

Discord | Official Documentation

Kushagra Acharya

Disclaimer

- This is an optional course and will not effect your academic credit
- If you're not interested and cannot fullfill any requirement or class rules you will be resulted for class dropout.

General Rules

- Having a laptop and a separate notebook is compulsory
- Faliure to answer at least 3 viva question will result in dissmissal.
- Faliure to complete homework/classwork without any valid result will be unacceptable.

Prerequisite

- Separate notebook/copy for notes
- NVM with Node Installed
- PC with VS Code Installed
- Stable Internet Connection

Project Setup from Scratch

Something from Nothing?

- Will be hard ... very hard .. because we are staring from scratch
- What is scratch?
- We will learn behind-the-scenes stuff
- Know how NestJS works so will make all of Nest easy

Steps

- Open bash
- Go to `D:\backend\nestjs-bootcamp`
- Create a folder called `scratch`
- `cd` into `scratch`
- do `npm init -y` to make `package.json`

Steps

Install the following dependencies from terminal

- `npm install @nestjs/common@9.0.0`
- `npm install @nestjs/core@9.0.0`
- `npm install @nestjs/platform-express@9.0.0`
- `npm install reflect-metadata@0.1.13`
- `npm install typescript@4.7.4`

All are the basic necessary dependencies for NestJS

Open your `scratch` folder in VSCode after installation

package.json

@nestjs/common

> has functions, classes (libs) that we need from Nest

@nestjs/platform-express

> lets Nest use ExpressJs to handle http requests/response

reflect-metadata

> helps make decorators work (more later!)

typescript

> Nest app with typescript

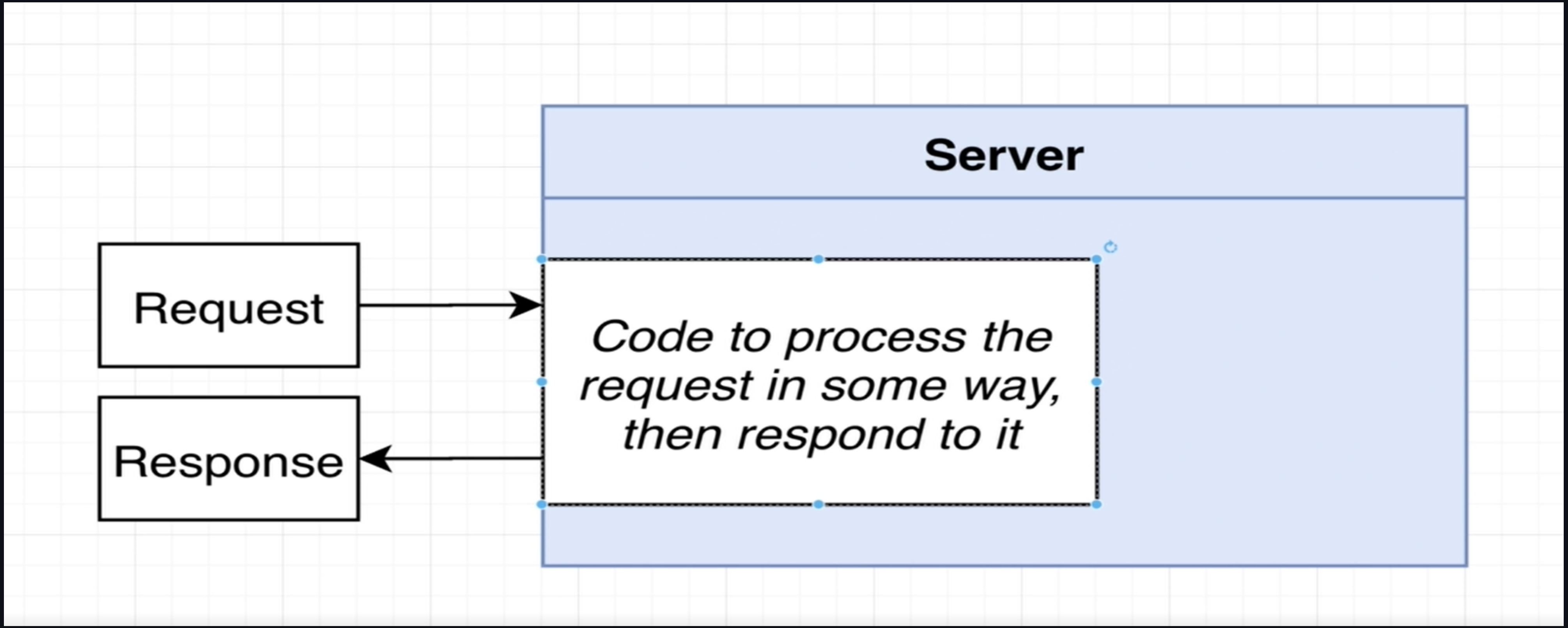
Configure TypeScript

- create file `tsconfig.json` in root
- write the following

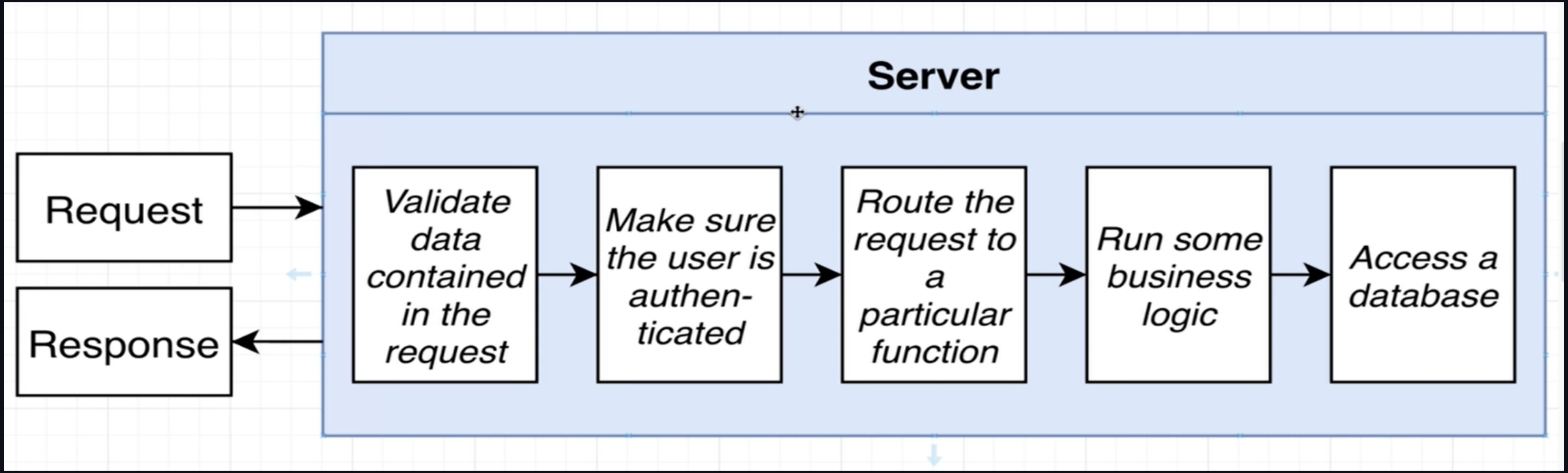
```json
{
  "compilerOptions": {
    "module": "commonjs",
    "target": "es2017",
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true
  }
}
```

Nest Module and Controller

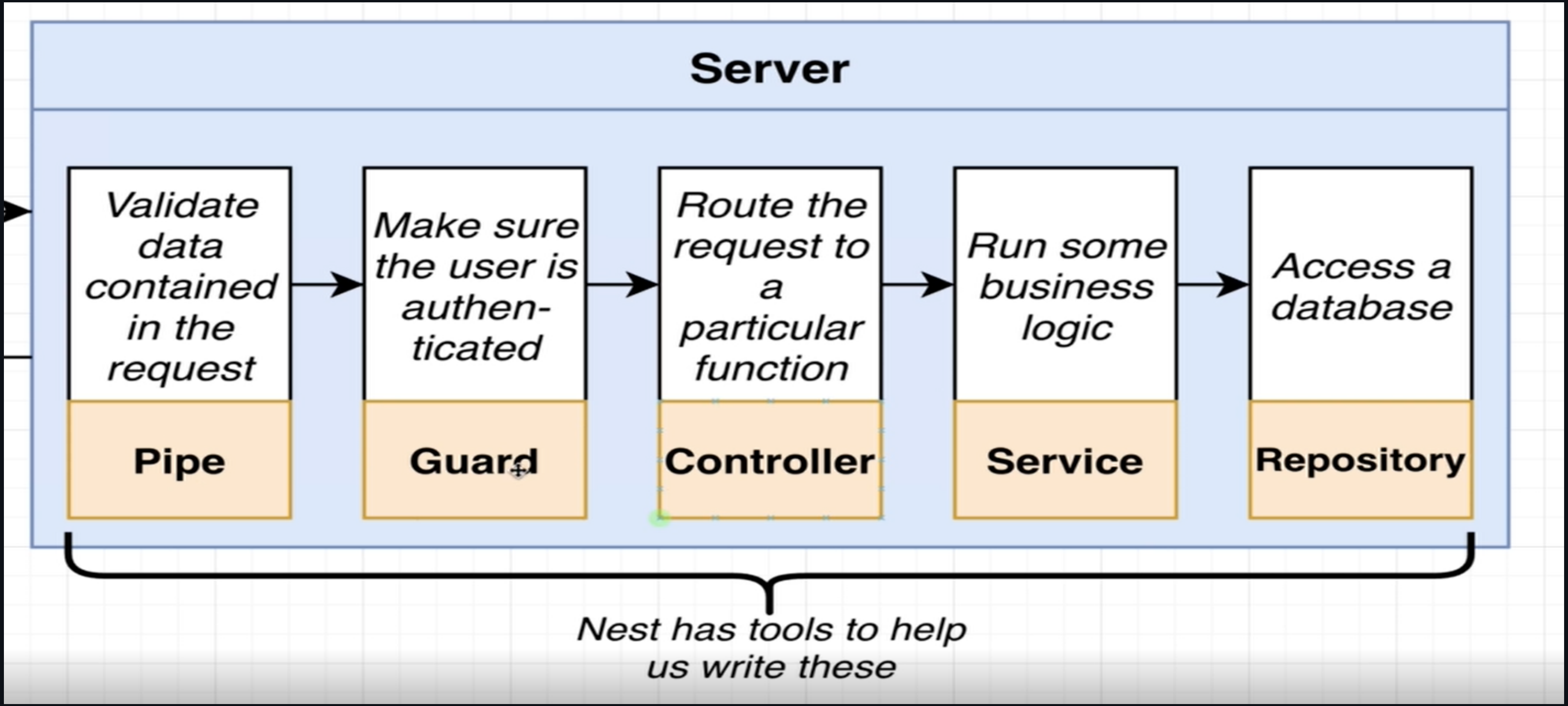
Basic Concept of Server



Detail Concept of Server

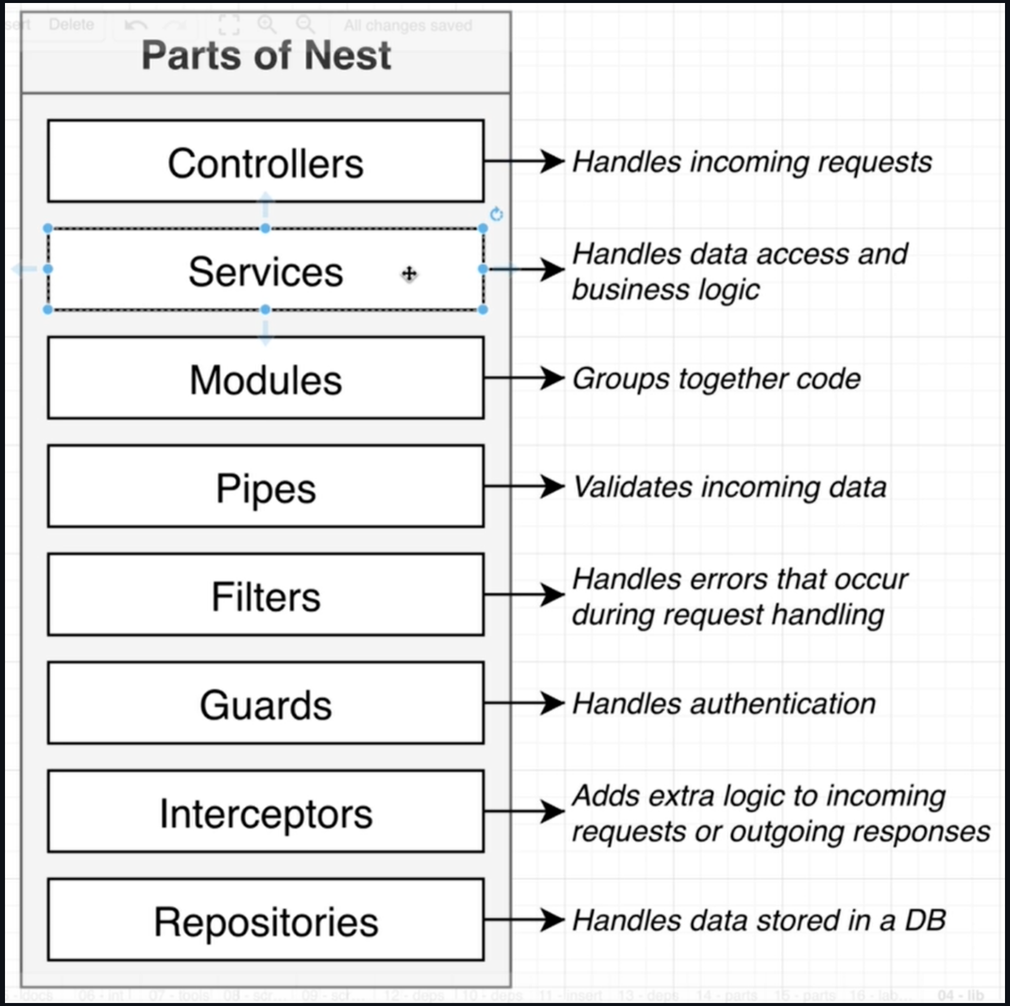

Server
Request
Response
Validate data contained in the request
Make sure the user is authen-ticated
Route the request to a particular function
Run some business logic
Access a database

Nest has special tools for each steps


Server
Validate data contained in the request
Pipe
Make sure the user is authen-ticated
Guard
Route the request to a particular function
Controller
Run some business logic
Service
Access a database
Repository
Nest has tools to help us write these

We will learn each tools in this series

Parts of Nest


Parts of Nest
Controllers
Handles incoming requests
Services
Handles data access and business logic
Modules
Groups together code
Pipes
Validates incoming data
Filters
Handles errors that occur during request handling
Guards
Handles authentication
Interceptors
Adds extra logic to incoming requests or outgoing responses
Repositories
Handles data stored in a DB

Basic Nest App

- to exist as an app
- a simple nest app will contain
    - a controller
    - a module
- which is the minimum criteria for it's existance

Creating the basics of Nest

- in your scratch folder, make `main.ts` file
- main.ts is the 1st file to get executed in any Nest project
- Complete the following code in `main.ts`

```
import {Controller, Module} from '@nestjs/common`
```

- these tools provided by nest for us to create our own controllers and modules

Creating a controller

- below import, write

```
class AppController {}
```

- decorate the `AppController` with `@Controller()`

What did we do?

```
@Controller()
class AppController {}
```

- We made own own controller using a decorator which tells nest that AppController is a controller
- Controller is made to handle and route incoming request
- Inside controller we will creating functions that will be able to handle specific requests

Task!

- Delete this `main.ts` file
- Make a new `main.ts` and do creation of a controller

Method Route in Controller

- import `Get` decorator and use for `getRootRoute`
- Add a method with following content

```
import { Controller, Module, Get } from "@nestjs/common";

@Controller()
class AppController {
  @Get()
  getRootRoute() {
    return "Hello Root Route!";
  }
}
```

Now Module

- Module is going to `wrap-up` a controller
- Every app we create must have at least one module
- Create a module below the previous `controller`

Create Module

- create a class `AppModule`
- decorate it with `@Module()`
- Try it yourself!

Passing configuration object to Module

- There is error in Module decorator
- It needs configuration as

```
@Module({
  controllers: [AppController],
})
class AppModule {}
```

- controllers property will list all controllers in the application

What will happen?

Whenever our nest app starts

- It will look into this `AppModule`
- It will find all the `Controllers` listed in `Module`
- Automatically creates instances of all controller classes
- So `AppController` instance will be created
- Will check all the decorators, eg: `@Get()`, and define `route handlers`

Review

What we did :

- main.ts as entry point
- Nest needs Module and Controller so we made both in main.ts
- `Module` wraps `Controller`
- `Controller` wraps `routes` with `methods` like `@Get()`
- Module needs `configuration`

Bootstrap

- main entrypoint needs a function
- async function called bootstrap()

```
import {NestFactory} from '@nestjs/core

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

bootstrap();
```

Run Application

Terminal

- make sure you're inside the project in cmd
- `npx ts-node-dev src/main.ts`
- will run the app
- "Nest Application Successfully Started"
- other program using port 3000?
- now see `localhost:3000`

Congratulations

You've completed:

- Core understanding of Nest and its components
- Module and Controller concept and creation
- main.ts file structure and use

Homework

- create a Nest project from scratch where:
  - in main.ts there is a module
  - module will have `Hello EEC Student`
  - submit the code via github from your account in discord

Viva questions:

- Module and Controller

NestJS | 3

Bootcamp

Discord | Official Documentation

Kushagra Acharya

Disclaimer

- This is an optional course and will not effect your academic credit
- If you're not interested and cannot fullfill any requirement or class rules you will be resulted for class dropout.

General Rules

- Having a laptop and a separate notebook is compulsory
- Faliure to answer at least 3 viva question will result in dissmissal.
- Faliure to complete homework/classwork without any valid result will be unacceptable.

Prerequisite

- Separate notebook/copy for notes
- NVM with Node Installed
- PC with VS Code Installed
- Stable Internet Connection

Setup and Modification

Extensions

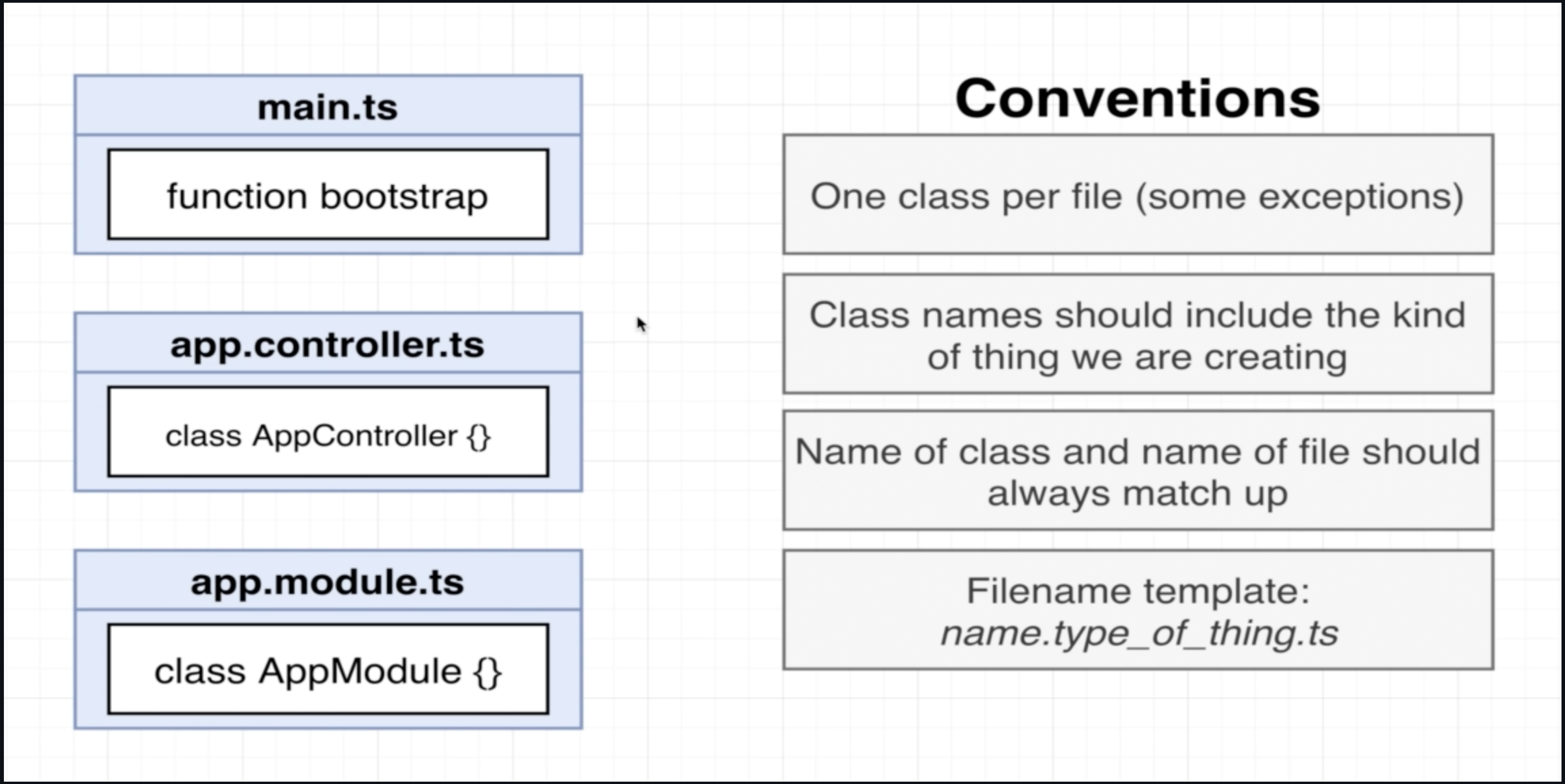
Install the following in VSCode

- NestJS
- Prettier

Structuring

- Currently we have all our classes in a single file
- This is not a very good approach for development
- We will now extract the app controller and the app module to a separate file

Before that: Conventions



Conventions

- One class per file (some exceptions)
- Class names should include the kind of thing we are creating
- Name of class and name of file should always match up
- Filename template: *name.type_of_thing.ts*

Creating files

- Create `app.controller.ts` file
- Move your `AppController class` to this file
- Fix any imports if necessary
- Do the same for `AppModule class` in `app.module.ts` file

main.ts

- The `main.ts` file should now only contain the `bootstrap()`
- But you will need to fix the import for `AppModule` in *main.ts* as it is being used as `NestFactory.create(AppModule)`

Run the application

- `npx ts-node-dev src/main.ts`

Discussion

- Importance of naming conventions
- What is separation of concern
- Help in project structuring
- Project scalability

Congratulations

Level Completed!

- Structuring for files and naming

NestJS | 4

Bootcamp

Discord | Official Documentation

Kushagra Acharya

Disclaimer

- This is an optional course and will not effect your academic credit
- If you're not interested and cannot fullfill any requirement or class rules you will be resulted for class dropout.

General Rules

- Having a laptop and a separate notebook is compulsory
- Faliure to answer at least 3 viva question will result in dissmissal.
- Faliure to complete homework/classwork without any valid result will be unacceptable.

Prerequisite

- Separate notebook/copy for notes
- NVM with Node Installed
- PC with VS Code Installed
- Stable Internet Connection

Routing Decorators

Decorators

- In your `app.controller.ts` file you can see some decorators
- `@Get()`
- `@Controller()`
- Decorators start with `@` and may or may not take arguments

Routing Rules

- `@Conroller()` decorator can take argument to change routing routes
- `@Get()` decorator as well can take arguments to modify routing
- Run the project to test any routing changes in upcoming slides

Get Route

```
@Controller()
export class AppController {
  @Get("/one")
  getRootRoute() {
    return "hey there!";
  }
}
```

- Test this change in your `localhost:3000/one`

Controller Routing

```
@Controller("/app")
export class AppController {
  @Get("/one")
  getRootRoute() {
    return "hey there!";
  }
}
```

- Figure out the endpoint for result 'hey there!'
- We can see that `controller routing` changes for all inner routes functions for higher level routing

Task

- Create another method inside AppController as `getByeThere` which returns the string `'bye there'` with a get-decorater having argument as `'bye'`
- Test your output in localhost

Run the application

- `npx ts-node-dev src/main.ts`

Discussion

- Decorators
- Routing
- Arguments

Congratulations

Level Completed!

- Routing with Decorators