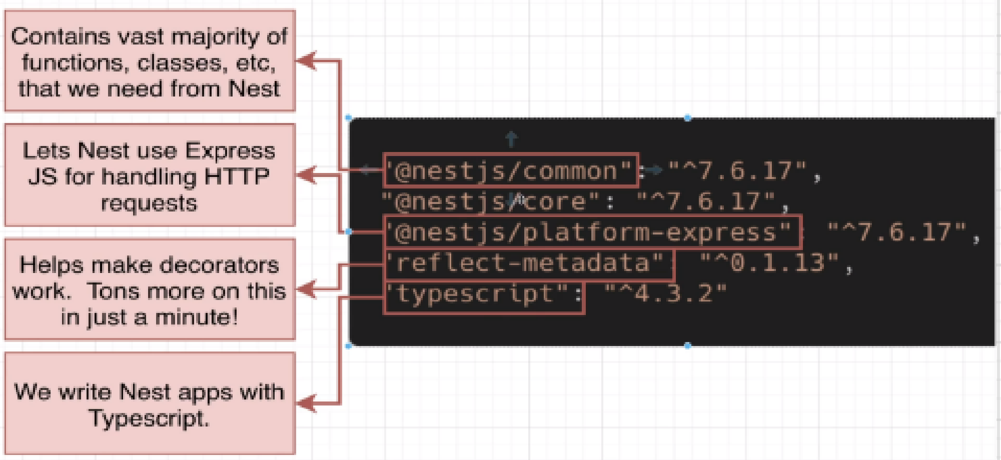
Manually start a Nestjs project

1. Create a project folder

2. Add package.json, cmd: 'npm init -y'

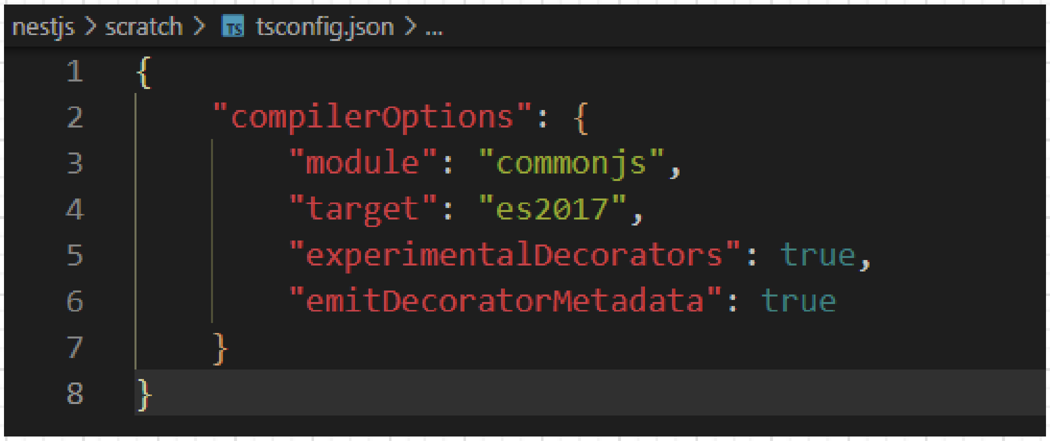
3. Install certain dependecies, cmd: 'npm i @nestjs/common@7.6.17 @nestjs/core@7.6.17 @nestjs/platform-express@7.6.17 reflect-metadata@0.1.13 typescript@4.3.2'



Contains vast majority of functions, classes, etc, that we need from Nest
Lets Nest use Express JS for handling HTTP requests
Helps make decorators work. Tons more on this in just a minute!
We write Nest apps with Typescript.

```
"@nestjs/common": "^7.6.17",
"@nestjs/core": "^7.6.17",
"@nestjs/platform-express": "^7.6.17",
"reflect-metadata": "^0.1.13",
"typescript": "^4.3.2"
```

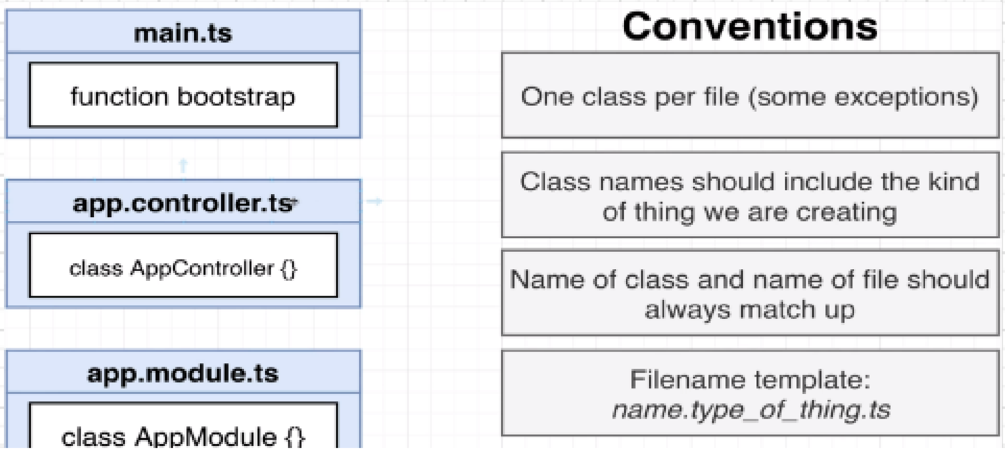
4. Create 'tsconfig.json' and ty enter following:



nestjs > scratch > tsconfig.json > ...

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

5. Create a module by:
   - Create 'src' folder in root directory
   - Create a 'main.ts' file inside 'src'
   - Create a controller function using decorator
   - Create an async function, normally using 'bootstrap'
   - Create an app.controller.ts file with a controller function
   - Create an app.module.ts file with the module class



main.ts
function bootstrap
app.controller.ts
class AppController {}
app.module.ts
class AppModule {}
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

6. Run the app, cmd: 'npx ts-node-dev src/main.ts'

7. Open localhost 3000/app/asdf and it should show 'hi there'

Start a Nestjs project with CLI

1.Create Nestjs project by using CLI

-run cmd: 'npm i -g @nestjs/cli'

-run cmd: 'nest new project', this is the 'messages' project in the folder

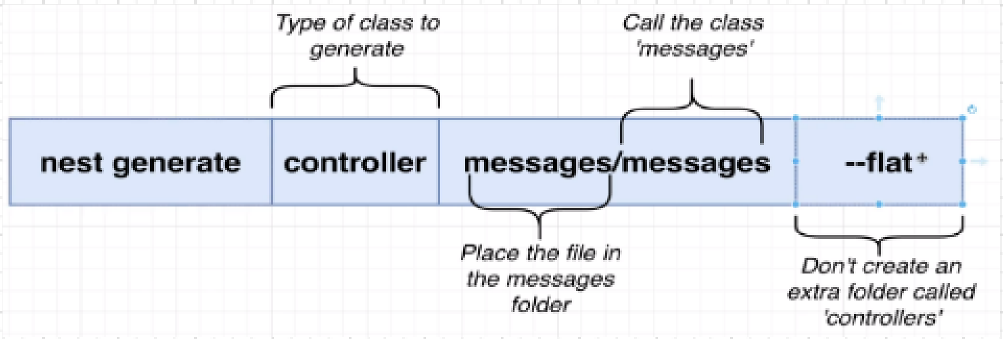
2.After creating the project, use cmd 'npm run start:dev' to run the app in watch mode

3.You can delete the auto-generated the files inside the 'src' folder except the 'main.ts' if you want to build your own files

4.Then run cmd 'nest generate module projectname' to generate your own files

5. A new fodler with projectname will be created inside 'src' with a 'projectname.module.ts' file

6. Create a contoller by using cmd: 'nest generate controller projectname/projectname --flat'


Type of class to generate
Call the class 'messages'
nest generate
controller
messages/messages
--flat
Place the file in the messages folder
Don't create an extra folder called 'controllers'

(optional) Use VSCode REST Client Extension to test APIs:

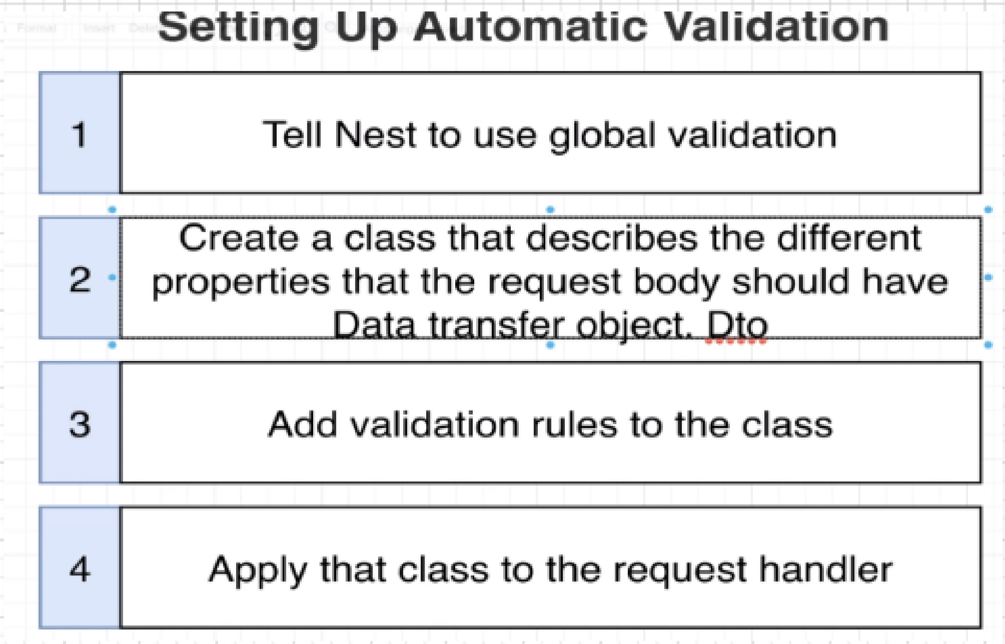
-Install the extenstion in VSCode

-Created a request.html inside root directory and check the content in example folder

Validate request data with pipes:

-Import ValidationPipe in main.ts

-Add corresponding syntax to the 'app'



Setting Up Automatic Validation

1	Tell Nest to use global validation
2	Create a class that describes the different properties that the request body should have Data transfer object. Dto
3	Add validation rules to the class
4	Apply that class to the request handler

Create dtos folders under 'src/messages'

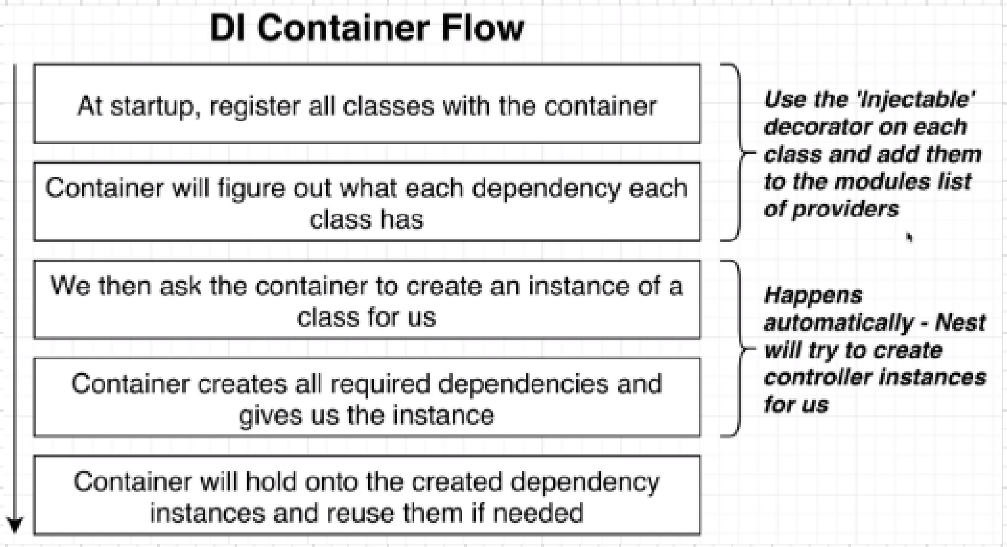
Implement a repository

-Create 'projectname.repository.ts' and 'projectname.service.ts'

-Look at the exmaple folder for content in these files

$$\begin{tikzcd}\text{Create repo} \arrow[r] & \text{Create Service} \arrow[r] & \text{Create controller}\end{tikzcd}$$

Dependency Injection: Inversion control



DI Container Flow

At startup, register all classes with the container

Container will figure out what each dependency each class has

Use the 'Injectable' decorator on each class and add them to the modules list of providers

We then ask the container to create an instance of a class for us

Container creates all required dependencies and gives us the instance

Happens automatically - Nest will try to create controller instances for us

Container will hold onto the created dependency instances and reuse them if needed