



Berkenalan dengan NestJS



Ahmad Arif · Follow

8 min read · Feb 19, 2019

229 3



Welcome back to Medium!

Kalo di game MOBA harusnya banyak dapat duit in game, dapet bahan untuk enhance item atau spell, dan banyak promo lainnya karena kembali pakai platformnya lagi 😊

...



Designed by Bobby Saputra

Pengenalan NestJS

Apa itu NestJS? NestJS merupakan framework yang dibuat dengan bahasa Typescript, yang memiliki arsitektur yang mirip dengan Angular. Waah, kabar baik nih bagi developer Angular. Untuk web bisa dengan Angular, untuk cross platform dan mobile bisa dengan Ionic atau Native Script, dan sekarang untuk backend juga ada yang mirip dengan arsitekturnya Angular yaitu **NestJS**.

Sebagai pengguna AdonisJS, dengan codebase Javascript. Akhirnya ada kabar gembira bagi saya yang mulai menyukai Typescript karena dikenalkan oleh Angular. Kenapa bisa suka Typescript? Wah banyak alasannya, salah satunya saya butuh **kejelasan** terhadap kode yang ditulis orang lain, seperti menggunakan sebuah fungsi yang membutuhkan parameter yang jelas tipe datanya, sehingga tidak bingung mem-passing variabel yang sesuai dengan requirementnya. Dengan arsitektur yang mirip dengan **Angular**, sangat membantu buat saya yang sudah pernah mencicipi framework frontend tersebut.

A progressive Node.js framework for building efficient, reliable and scalable server-side applications.

NestJS sendiri dibuat berdasarkan framework **Express**, sehingga tidak perlu khawatir akan kerepotan ketika menggunakan framework yang terhitung baru ini dengan berfikir minimnya library. Express kurang ngebut? Kamu bisa ganti http adapternya dengan **Fastify**. Jadi, tunggu apalagi? Mari kita mulai eksplorasi NestJS, berikut ini beberapa yang akan dibahas pada artikel kali ini:

- Instalasi Nest CLI (Nest Generator)
- Struktur NestJS
- Mengatur struktur aplikasi
- Membuat modul konfigurasi untuk pengaturan Environment Variable

. . .

Instalasi Nest CLI (Nest Generator)

Untuk melakukan instalasi Nest CLI, pastikan diperangkat kamu sudah terinstall **NodeJS** (minimal versi 8.9.0).

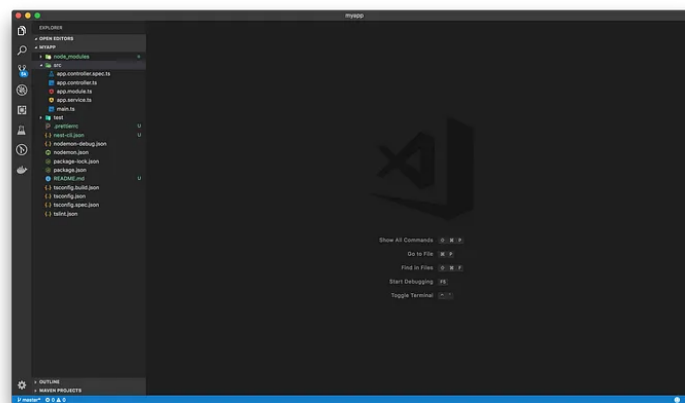
Untuk menginstall Nest CLI jalankan command berikut:

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

Setelah selesai, buat project baru dengan command seperti berikut:

```
$ nest new myapp
```

Isikan beberapa pertanyaan seperti mengisi deskripsi, versi aplikasi, dan informasi pembuat. Nest CLI akan men-generate aplikasi menggunakan skeleton dasar, dan selanjutnya memilih paket manager yang akan digunakan dalam aplikasi (npm/yarn). Setelah install dependency yang dibutuhkan, kurang lebih struktur aplikasi akan seperti ini:

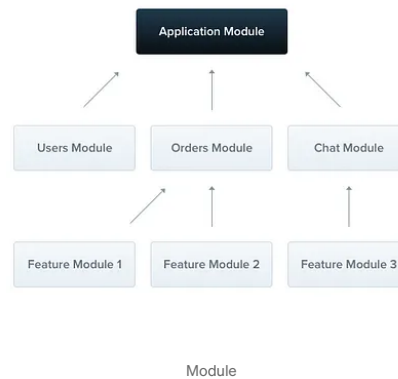


Skeleton Awal Nest App

Bootstrap file yang dijalankan oleh NestJS adalah file **main.ts**, untuk konfigurasi yang harus dijalankan saat aplikasi startup silahkan gunakan file tersebut untuk mengaturnya.

Struktur NestJS

- **Module:** Modul merupakan sebuah class yang ditandai dengan dekorator `@Module` yang menyediakan metadata yang digunakan Nest untuk mengatur struktur aplikasi. Modul merupakan root dari proses bisnis, dalam sebuah proyek Nest bisa memiliki lebih dari satu modul. Biasanya dipisahkan per-proses bisnis yang didalam terdapat proses pengolahan HTTP request, koneksi database, sampai penyajian response. Misalnya API untuk modul **users**, modul **transaksi**, dan lain sebagainya. Module memiliki beberapa bagian untuk memproses aliran data, yaitu **Controller**, **Provider/Service**, **Middleware**, dsb



- **Controller:** Digunakan untuk mengatur HTTP request dan response yang dikirim ke client. Controller mengatur endpoint yang akan digunakan, data yang harus dikirim ke server, dan data yang akan dikirimkan ke server.
- **Provider/Service:** Provider merupakan bagian yang biasa digunakan untuk melakukan proses diluar dari bagian HTTP request. Misalnya melakukan koneksi ke database, melakukan request terhadap microservice lainnya.
- **Middleware:** Middleware merupakan sebuah fungsi dijalankan sebelum **route handler** (controller), middleware memiliki akses terhadap **request** dan **response** sehingga dapat mengatur apakah request tersebut diteruskan ke route handler atau tidak dengan suatu pesan error. Misalnya middleware yang digunakan untuk mengecek otorisasi dari pengguna, atau middleware untuk menulis log setiap request, dan lain sebagainya.



Note: Dekorator menyediakan cara untuk menambahkan anotasi dan meta-programming untuk deklarasi class. Baca lebih lanjut mengenai dekorator [di sini](#).

. . .

Mengatur struktur aplikasi

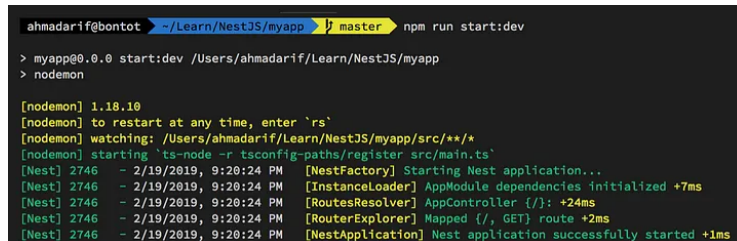
Setiap orang memiliki kebiasaan dan pattern sendiri dalam mengatur aplikasinya. Dalam penggunaan Nest, saya biasanya membuat **Application Context** di file yang terpisah dengan file main.ts, hal tersebut dilakukan untuk kemudahan akses dari luar file main.ts. Berikut kode untuk Application Context dan root file Nest:

Kemudian panggil Application Context di root file aplikasi, seperti berikut ini:

Jalankan aplikasi dengan mode development (auto reload ketika terjadi perubahan kode) dengan command:

```
$ npm run start:dev
```

Kurang lebih akan muncul tulisan seperti berikut ini:



```
ahmadarif@bontot ~/Learn/NestJS/myapp > master npm run start:dev
> myapp@0.0.0 start:dev /Users/ahmadarif/Learn/NestJS/myapp
> nodemon

[nodemon] 1.18.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: /Users/ahmadarif/Learn/NestJS/myapp/src/**/*.ts
[nodemon] starting 'ts-node -r tsconfig-paths/register src/main.ts'
[Nest] 2746 - 2/19/2019, 9:20:24 PM [NestFactory] Starting Nest application...
[Nest] 2746 - 2/19/2019, 9:20:24 PM [InstanceLoader] AppModule dependencies initialized +7ms
[Nest] 2746 - 2/19/2019, 9:20:24 PM [RoutesResolver] AppController {}: +24ms
[Nest] 2746 - 2/19/2019, 9:20:24 PM [RouterExplorer] Mapped {/, GET} route +2ms
[Nest] 2746 - 2/19/2019, 9:20:24 PM [NestApplication] Nest application successfully started +1ms
```

Silahkan akses <http://localhost:3000> menggunakan browser atau postman, dan akan muncul tulisan **Hello World!**. Mari kita bahas bagaimana tulisan tersebut bisa muncul!

Perlu diingat, bootstrap file kita adalah file main.ts. File tersebut memanggil Application Context (app.context.ts) yang menginisialisasi NestFactory dengan parameter AppModule (Nest minimal memiliki satu modul untuk melakukan bootstrapping aplikasinya). Jika dilihat file app.module.ts kita seperti ini:

AppModule Class

Terdapat 3 fungsi utama yang selalu dipanggil oleh Module, yaitu:

- **imports:** Digunakan untuk meng-import module lainnya ke dalam module tersebut
- **controllers:** Digunakan untuk meng-import Controller, dengan begitu routing yang terdapat dalam controller dapat diakses
- **providers:** Digunakan untuk meng-inject provider atau service ke dalam module, sehingga dapat langsung digunakan oleh controller maupun service lainnya yang juga didaftarkan di dalam module yang sama.

*Kembali ke cerita dari mana tulisan **Hello World!** bisa diakses*

Route handler dari Nest berada pada Controller, maka yang perlu dicek bagaimana kita bisa melakukan request GET ke basepath dari aplikasi. Jawabannya ada pada controller yang didaftarkan dalam AppModule. Berikut ini Controller dan Provider yang sudah didaftar ke dalam module:

AppService Class

Sebuah controller ditandai oleh adanya dekorator `@Controller()`, ketika dekorator tidak diisi oleh string lain maka artinya controller tersebut menggunakan path `/` atau basepath dari aplikasi. Prefix dari routing diatur pada dekorator controller, contohnya kita mau menggunakan prefix `users` maka cukup tulis `@Controller('users')`.

Dekorator controller hanya mendefinisikan prefix dari route yang kita buat, proses sebenarnya berada dalam method yang terdapat pada controller tersebut. Di dalam `AppController` terdapat sebuah fungsi `getHello` dengan dekorator `@Get()`, yang artinya kita mendefinisikan

routing method GET dengan tanpa url tambahan lainnya. Sehingga tulisan yang kita dapatkan sebagai response saat mengakses <http://localhost:3000> (akses dari browser by default menggunakan verbs/method GET) merupakan response yang dikirim oleh fungsi `getHello` dari class `AppController` setelah memanggil fungsi `getHello()` dari class `AppService` yang mengirimkan tulisan `Hello World!` yang merupakan variabel bertipe string.

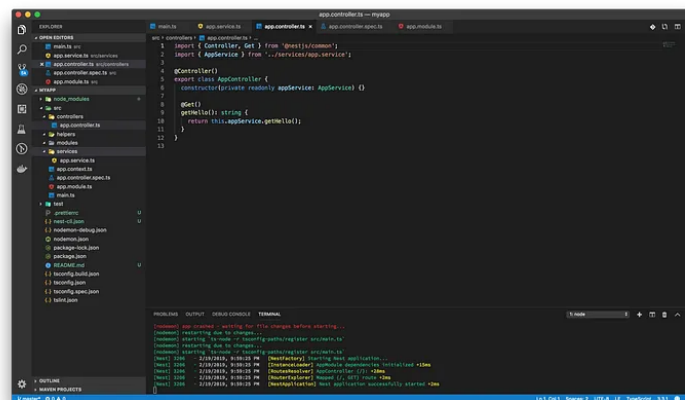
Tunggu sebentar, ada pemanggilan `this.appService.getHello()` tapi gak pernah kita inisialisasi? Konsep ini disebut dengan Dependency Injection (DI), jadi kamu dapat langsung menggunakan kelas tersebut tanpa harus menginisialisasi di setiap kelas yang menggunakan service tersebut, cukup dengan mendefinisikan kelas tersebut di constructor. Konsep DI pada Nest mirip dengan konsep DI pada Angular, silahkan baca lebih lanjut pada [link ini](#).

Say goodbye pada keyword `new` ketika inisialisasi object

Berikut ini struktur aplikasi yang biasa saya gunakan dalam membangun aplikasi menggunakan Nest:

```
src/
├── modules/
├── controllers/
├── services/
├── app.context.ts
├── app.module.ts
└── main.ts
```

Dengan menggunakan vscode, kamu cukup buat directory yang belum tersedia kemudian drag file sesuai struktur di atas, vscode akan membantu proses perbaikan path pada import file (enak kan pake TS dan vscode? 🥰), jangan lupa untuk save all dan tunggu sampai Nest me-reload semua perubahan pada kode. Kurang lebih proyek kita akan seperti ini:



Path `AppService` berubah sesuai dengan posisi file terbaru

...

Membuat modul konfigurasi untuk pengaturan Environment Variable

Aplikasi Nest yang dibuat running menggunakan port 3000, dimana port tersebut ditulis langsung di dalam file `main.ts`. Bagaimana kalo misalnya kita menggunakan banyak konfigurasi dan tersebar di beberapa file yang

berbeda? Bagaimana kalo menjalankan aplikasi menggunakan **Docker** atau **Docker Orchestration Tools** seperti **Kubernetes** dengan **Rancher** sebagai UI-nya? Akan lebih mudah jika kita membuatkan file yang dapat dibaca oleh aplikasi sebagai file konfigurasi atau menggunakan variabel `process.env` dari `nodejs`.

Sebagai bahan pembelajaran pembuatan modul di Nest, kita akan membuat sebuah module yang digunakan untuk membaca Environment Variable di aplikasi. Berikut langkah-langkahnya:

- Install **dotenv** ke dalam proyek

```
$ npm install --save dotenv
```

- Kemudian buat direktori dengan nama **modules/config**
- Lalu buat file **config.service.ts** di direktori **modules/config** yang digunakan sebagai provider untuk membaca environment variable di aplikasi, berikut ini kode config servicenya:

`ConfigService` merupakan class yang nanti digunakan untuk membaca environment variable pada aplikasi. By default tipe data yang dibaca merupakan string, maka dari itu dibuatkan beberapa fungsi `get` yang sesuai dengan tipe data yang ingin didapat.

- Selanjutnya adalah membuat class `ConfigModule` di file **modules/config/config.module.ts**. Berikut kode untuk class `ConfigModule`

Config module mendaftarkan ConfigService pada array of providers, dengan string `.env` sebagai parameternya. Di sini proses inisialisasi ConfigService, supaya dapat digunakan dengan menggunakan Dependency Injection pattern. Terdapat fungsi baru dalam module kali ini, yaitu **exports**. Fungsi tersebut digunakan untuk mengekspose class, agar dapat digunakan di hirarki module lain yang meng-import ConfigModule.

- Langkah selanjutnya adalah menambahkan ConfigModule ke dalam AppModule, AppModule merupakan root module pada aplikasi. Artinya, semua yang didaftarkan di root module, dapat digunakan di hirarki controller maupun service pada module tersebut. Kurang lebih kode terbaru dari AppModule adalah sebagai berikut:

- Update main.ts supaya menjalankan aplikasi dengan port sesuai konfigurasi, berikut kode main.ts terbaru:

- Setelah menyimpan kode tersebut, maka aplikasi akan mencari file `.env` yang disediakan oleh ConfigService di ConfigModule. Karena file tidak ditemukan, maka konfigurasi yang digunakan adalah dari `process.env`, dan berhubungan belum ada pengaturan `process.env` maka akan muncul error karena tidak mengatur port untuk aplikasi. Berikut screenshot dari penambahan ConfigModule:

```
[nodemon] restarting due to changes...
[nodemon] starting 'ts-node -r tsconfig-paths/register src/main.ts'
[Nest] 4008 - 2/19/2019, 11:00:21 PM File .env not found, app will use process.env
[Nest] 4008 - 2/19/2019, 11:00:21 PM [NestFactory] Starting Nest application... +3ms
[Nest] 4008 - 2/19/2019, 11:00:21 PM [InstanceLoader] ConfigModule dependencies initialized +9ms
[Nest] 4008 - 2/19/2019, 11:00:21 PM [InstanceLoader] AppModule dependencies initialized +0ms
[Nest] 4008 - 2/19/2019, 11:00:21 PM [RoutesResolver] AppController {}: +22ms
[Nest] 4008 - 2/19/2019, 11:00:21 PM [RouterExplorer] Mapped {/, GET} route +2ms
[Nest] 4008 - 2/19/2019, 11:00:21 PM [NestApplication] Nest application successfully started +1ms
(node:4008) UnhandledPromiseRejectionWarning: RangeError [ERR_SOCKET_BAD_PORT]: Port should be >= 0 and < 65536. Received NaN.
```

- Mencoba menjalankan aplikasi menggunakan **process.env**, pengguna mac/linux ketikkan perintah berikut (matikan proses **npm run start:dev** terlebih dulu):

```
$ APP_PORT=3333 npm run start:dev
```

Maka aplikasi akan menggunakan port 3333 sebagai listen portnya, berikut screenshot log saat aplikasi startup:

```
ahmadarif@bontot: ~/Learn/NestJS/myapp / master APP_PORT=3333 npm run start:dev
> myapp@0.0.0 start:dev /Users/ahmadarif/Learn/NestJS/myapp
> nodemon

[nodemon] 1.18.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: /Users/ahmadarif/Learn/NestJS/myapp/src/**/*.ts
[nodemon] starting 'ts-node -r tsconfig-paths/register src/main.ts'
[Nest] 4064 - 2/19/2019, 11:04:05 PM File .env not found, app will use process.env
[Nest] 4064 - 2/19/2019, 11:04:05 PM [NestFactory] Starting Nest application... +4ms
[Nest] 4064 - 2/19/2019, 11:04:05 PM [InstanceLoader] ConfigModule dependencies initialized +8ms
[Nest] 4064 - 2/19/2019, 11:04:05 PM [InstanceLoader] AppModule dependencies initialized +0ms
[Nest] 4064 - 2/19/2019, 11:04:05 PM [RoutesResolver] AppController {}: +23ms
[Nest] 4064 - 2/19/2019, 11:04:05 PM [RouterExplorer] Mapped {/, GET} route +2ms
[Nest] 4064 - 2/19/2019, 11:04:05 PM [NestApplication] Nest application successfully started +1ms
```

Mengatur variabel **process.env** bisa digunakan dengan cara di atas, maupun saat deployment workload rancher untuk mengatur **Environment Variable**.

*Note: Pengguna windows bisa menggunakan library **cross-env** untuk dapat menggunakan perintah di atas.*

- Menggunakan file **.env** menjadi alternatif lain supaya config deployment dengan development/local tidak terganggu, cukup membuat file **.env** di root direktori kemudian buat variabel yang dibutuhkan pada file tersebut. Berikut contoh file **.env**:

```
APP_PORT=3333
```

*Note: Restart development server dengan kill dan jalankan ulang, atau dengan mengetik **rs** kemudian enter pada terminal. Hal tersebut karena mengubah file **.env** tidak akan me-reload server seperti mengubah kode lainnya.*

- Untuk mengakses aplikasi, gunakan <http://localhost:3333> atau port lainnya sesuai dengan konfigurasi masing-masing.

. . .

Aplikasi simpel menggunakan Nest sudah berhasil dibuat, tinggal push kode ke repository masing-masing. Jangan lupa untuk ignore folder **node_modules** dan file **.env** supaya tidak ikut masuk ke repository. Sebagai backup file **.env** yang tidak dipush ke server, buatlah file **.env.example**


yang berisi variabel-variabel yang digunakan dalam aplikasi dengan nilai kosong atau menggunakan nilai dummy, misalnya seperti berikut:

```
APP_PORT=3000
DB_HOST=localhost
DB_PORT=3306
DB_USERNAME=YOUR_DB_USERNAME
DB_PASSWORD=YOUR_DB_PASSWORD
DB_NAME=YOUR_DB_NAME
```

. . .

Kode selengkapnya bisa cek di link berikut:

ahmadarif/medium-nest-app
NestJS Tutorial on <https://medium.com/@ahmadarif> (ID) - ahmadarif/medium-nest-app
[github.com](#)



Nestjs Typescript Tutorial

👏 229 💬 3




Written by Ahmad Arif

75 Followers

Follow

More from Ahmad Arif




 Ahmad Arif in AdonisID

Membuat Dokumentasi API di Adonis

Banyak tools yang dapat digunakan untuk membuat dokumentasi API, beberapa...

4 min read · Mar 5, 2018

👏 164 💬 1

 Ahmad Arif in AdonisID

Implementasi Authentication di Adonis

Adonis merupakan web framework Node.js berbasis MVC (Model View Controller) yang...

9 min read · Feb 21, 2018

👏 160 💬 2



Ahmad Arif in AdonisID

Eksplorasi Database di AdonisJS

Adonis mendukung penggunaan berbagai jenis database, dan juga berbagai fungsi yan...

8 min read · Feb 10, 2018

68



Ahmad Arif in AdonisID

Adonis Drive + Minio Cloud Storage

Pengelolaan file merupakan bagian yang tidak dapat dipisahkan dalam sebuah sistem. Di er...

5 min read · Apr 30, 2018

52

See all from Ahmad Arif

Recommended from Medium



Manish Kumar in Stackademic

Implementing a Custom Logger in NestJS: A Step-by-Step Guide

A custom global logging implementation in nestjs using winston module.

5 min read · Aug 18

67



Samin karki

Creating dynamic modules in nestjs

So, nestjs has two types of modules, Static and Dynamic modules. Say for example, to...

3 min read · Sep 16

8

Lists



Tech & Tools

15 stories · 60 saves



Icon Design

30 stories · 114 saves



General Coding Knowledge

20 stories · 414 saves



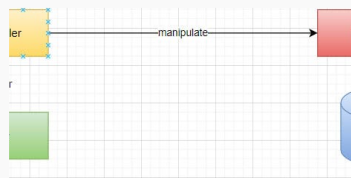
Medium Publications Accepting Story Submissions

154 stories · 797 saves



Muhammad Fahad

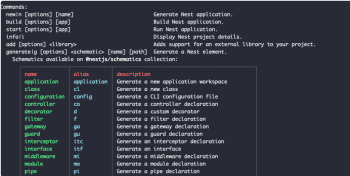
Nest.js



misostack

Create Nest.js Application with MongoDB and complete validation and authentication (JWT...

7 min read · Aug 30

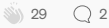


PrimaryBid Tech BL... in PrimaryBid Technology BL...

Enhancing Developer Productivity with NestJS and RxJS in a Fintech...

In the fintech industry, where agility and efficiency are paramount, developer...

5 min read · Sep 26



NestJS Course Lesson 05 — Repository Pattern

Lesson 05 Source Code

6 min read · Jul 25

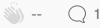


Brahim Abdelli

NestJS Boilerplate using Typescript Generics.

Just finished coding a NestJS boilerplate project (and an explicative Article about it)...

1 min read · Aug 27



See more recommendations