miam | Web

Janvier–avril 2023

JUNIA
Grande école d'ingénieurs
HEI·ISEN·ISA

Typescript Frameworks

Plan:

Typescript vs javascript

Nestjs
Nestjs project installation

Rest standard

nestjs project info

Typescript vs Javascript

Pourquoi avoir un second language et pas que du js ?

TypeScript est un langage de programmation plus adapté aux grandes applications.
Open source, il a été développé par Microsoft en 2012, principalement parce que le code JavaScript devenait trop complexe à gérer lorsqu'il s'agissait d'applications à grande échelle.

TypeScript est un sur-ensemble de JavaScript : de manière générale, tout ce qui est du code en JavaScript est donc également valable en TypeScript

TypeScript transcompile vers JavaScript.
Il existe de nombreuses alternatives pour la transcompilation pour transformer TypeScript en JavaScript. On peut notamment utiliser le vérificateur TypeScript par défaut, ou bien le compilateur Babel.

Quels différences ?

1. typage

javascript : changement de type permit

```
let var1 = "Hello";
var1 = 10;
console.log(var1);
```

Typescript

```
let var1: string = "Hello";
var1 = 10;
console.log(var1);

TSError: × Unable to compile TypeScript:
src/snippet1.ts:2:1 - error TS2322: Type 'number' is not assignable to type
'string'.
```

2. les interfaces

```
function printLabel(labeledObj: { label: string }) {
 console.log(labeledObj.label);
}
 let myObj = { size: 10, label: "Size 10 Object" };
printLabel(myObj);
```

```
interface LabeledValue {
 label: string;
}
function printLabel(labeledObj: LabeledValue) {
 console.log(labeledObj.label);
}
let myObj = { size: 10, label: "Size 10 Object" };
printLabel(myObj);
```

3. levée d'erreur à la précompilation

prenons ce code javascript simple :

```
const mysql = require('mysql');


mysql.toto();
```

Aucune erreure apparente, par contre à l'exécution on a une erreure :

```
mysql.toto();
      ^

TypeError: mysql.toto is not a function
    at Object.<anonymous> (/home/alexis/junia/cours5/step1/database_test.js:3:7)
    at Module._compile (internal/modules/cjs/loader.js:999:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1027:10)
    at Module.load (internal/modules/cjs/loader.js:863:32)
```

En typescript votre IDE doit souligner cette erreure

4. Refactor

Vous avez fait une typo par exemple fotre fonction s’appelle :

‘getNumbr()’

Vous voulez refactorer votre code pour corriger cette typo.
-> vous devez trouver toutes les occurrences pour les remplacer

En javascript il vous faut une recherche textuelle

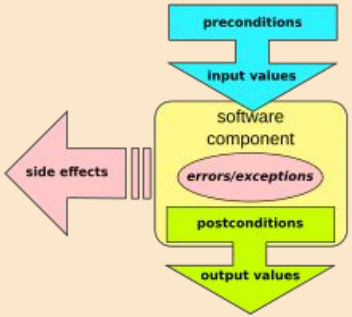
En typescript le code qui dépend de cette fonction ne fonctionnera plus et vous serez informé si vous n’avez pas renommé correctement.

5. La doc

Les classes abstraites, les interfaces, le typage etc facilite la documentation

Facilite le design by contract.

Wikipedia :
Design by contract (DbC), also known as contract programming, programming by contract and design-by-contract programming, is an approach for designing software.
It prescribes that software designers should define formal, precise and verifiable interface specifications for software components, which extend the ordinary definition of abstract data types with preconditions, postconditions and invariants. These specifications are referred to as "contracts", in accordance with a conceptual metaphor with the conditions and obligations of business contracts.



5. Et encore

Auto complete

async/await

private class field

Framework

C’est quoi ?

Un framework propose une bibliothèque de fonctionnalités dans laquelle vos développeurs vont pouvoir piocher en fonction de vos besoins.
Une boîte a outils.

Elle permet donc de gagner du temps. On ne réinvente pas la roue.
Elle gère les cas auxquels on ne pense pas forcement.

Un framework va aussi impacter la façon dont on code, nos choix d’architecture.
Par exemple nous allons voir un framework fait pour suivre le pattern MVC qui va nous faire créer des models, des controllers etc.

Nestjs vs Expressjs

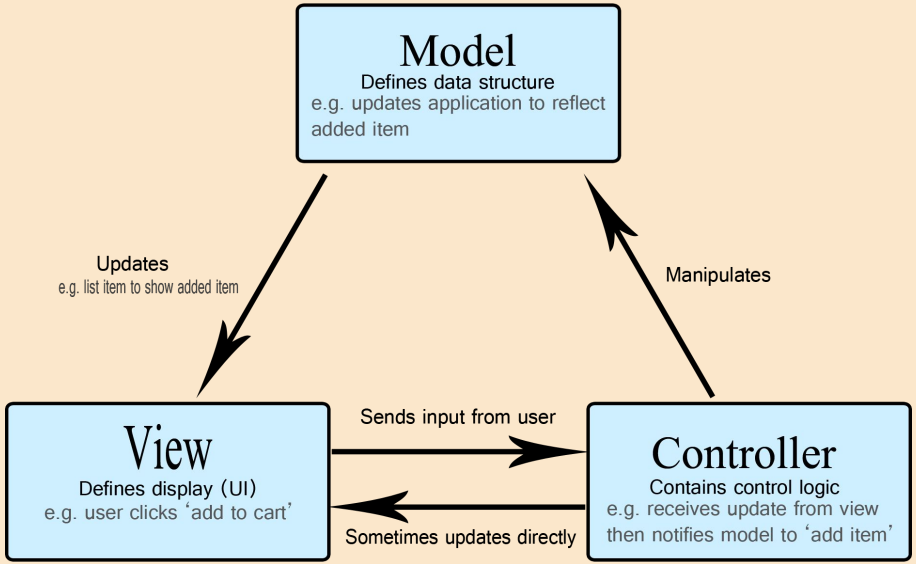
“
NestJS, on the other hand, uses the “convention over configuration” paradigm that attempts to decrease the number of decisions developers need to make, and it is expected of them to write the repos, service, and controllers in a specific manner.
“

Wikipedia :
Convention over configuration (also known as coding by convention) is a software design paradigm used by software frameworks that attempts to decrease the number of decisions that a developer using the framework is required to make without necessarily losing flexibility and don't repeat yourself (DRY) principles.

The phrase essentially means a developer only needs to specify unconventional aspects of the application. For example, if there is a class Sales in the model, the corresponding table in the database is called "sales" by default. It is only if one deviates from this convention, such as the table "product sales", that one needs to write code regarding these names.

NestJS follows the MVC architecture with components like modules, controllers, and providers, used to control logic, data, and implement the UI. ExpressJS doesn't follow MVC, which in some projects leads to ineffectiveness and inefficiency due to the lack of structure.

MVC :
Model lié à la BDD, par exemple un model 'Annonce' lié a la table 'annonces' qui a un titre, un prix etc.
Controller lié à la réception des requêtes et les réponses



Controllers: They handle the incoming requests and return responses to the client-side.

Providers: These are the fundamental concepts of NestJS that you can treat as services, repositories, factories, helpers, etc. You can create and inject them into controllers or other providers as they are designed to abstract any complexity and logic.

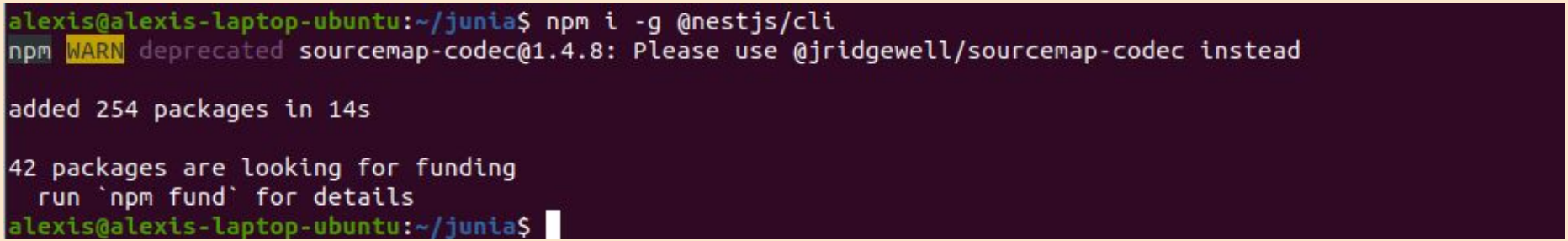
Modules: These are the classes. A module is a reusable chunk of code that has a separate functionality. The entire source code is organized and structured into modules. Each application will have at least one root module which is the starting point.

https://docs.nestjs.com/

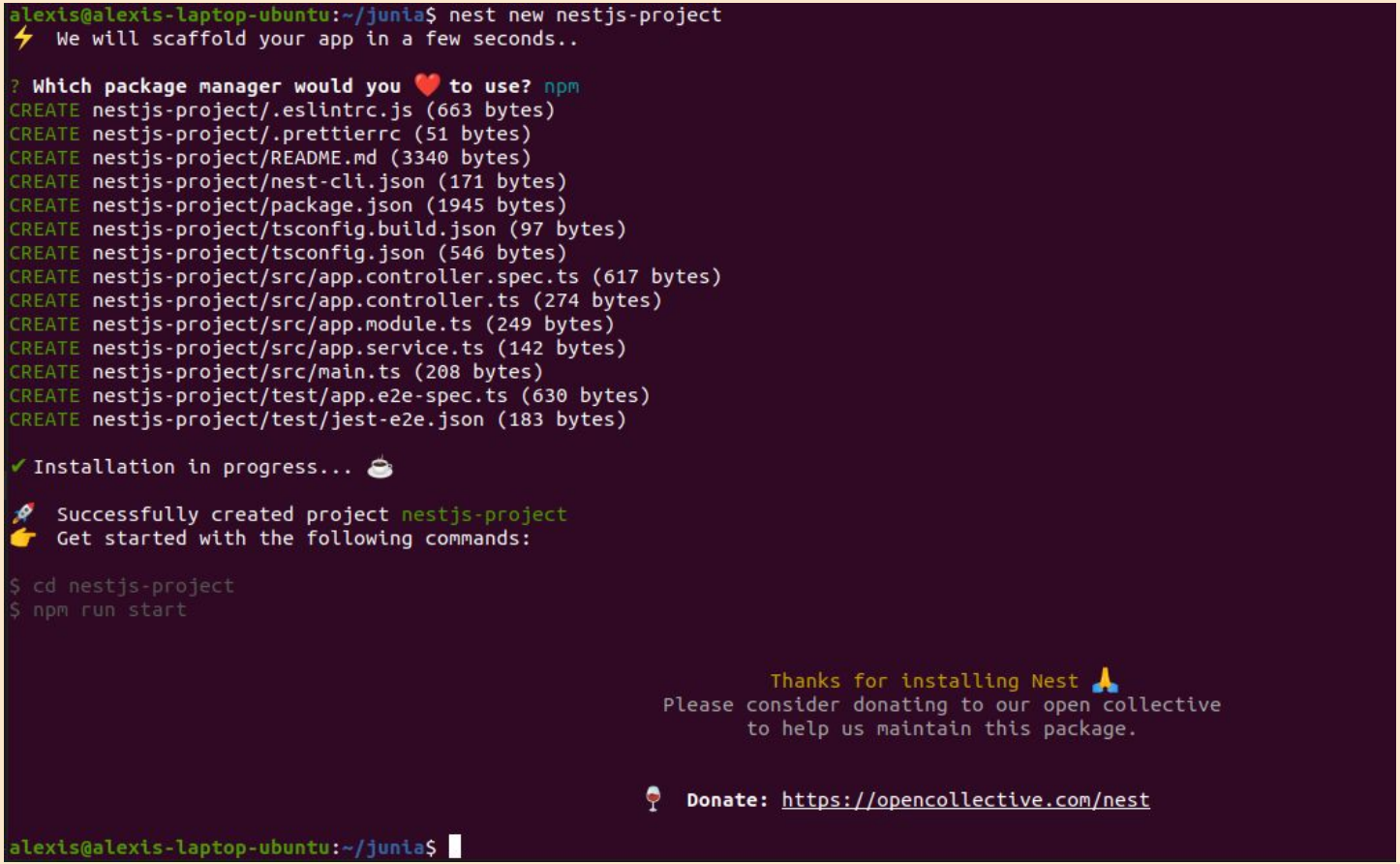
Utilisez les outils a votre disposition pour installer nestjs, notament la CLI (command line interface)

Permet de créer un projet avec une structure de base

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



```
nest new nestjs-project
```



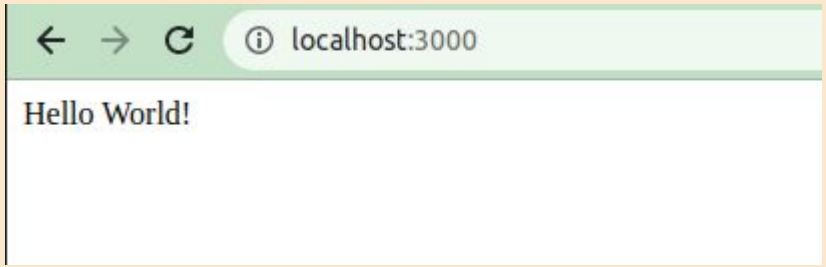
Vous avez un projet de base, voici l’architecture

```
alexis@alexis-laptop-ubuntu:~/junia$ cd nestjs-project/
alexis@alexis-laptop-ubuntu:~/junia/nestjs-project$ ll
total 380
drwxrwxr-x   6 alexis alexis   4096 mars   5 22:31 ./
drwxrwxr-x  11 alexis alexis   4096 mars   5 22:31 ../
-rw-rw-r--   1 alexis alexis    663 mars   5 22:31 .eslintrc.js
drwxrwxr-x   7 alexis alexis   4096 mars   5 22:31 .git/
-rw-rw-r--   1 alexis alexis    391 mars   5 22:31 .gitignore
-rw-rw-r--   1 alexis alexis    171 mars   5 22:31 nest-cli.json
drwxrwxr-x 450 alexis alexis  20480 mars   5 22:31 node_modules/
-rw-rw-r--   1 alexis alexis   1945 mars   5 22:31 package.json
-rw-rw-r--   1 alexis alexis 310964 mars   5 22:31 package-lock.json
-rw-rw-r--   1 alexis alexis     51 mars   5 22:31 .prettierrc
-rw-rw-r--   1 alexis alexis   3340 mars   5 22:31 README.md
drwxrwxr-x   2 alexis alexis   4096 mars   5 22:31 src/
drwxrwxr-x   2 alexis alexis   4096 mars   5 22:31 test/
-rw-rw-r--   1 alexis alexis     97 mars   5 22:31 tsconfig.build.json
-rw-rw-r--   1 alexis alexis    546 mars   5 22:31 tsconfig.json
alexis@alexis-laptop-ubuntu:~/junia/nestjs-project$
```

On peut installer et lancer le projet :

```
alexis@alexis-laptop-ubuntu:~/junia/nestjs-project$ npm install

up to date, audited 692 packages in 1s

92 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
alexis@alexis-laptop-ubuntu:~/junia/nestjs-project$ npm start

> nestjs-project@0.0.1 start
> nest start

[Nest] 39714  - 05/03/2023 22:33:58     LOG [NestFactory] Starting Nest application...
[Nest] 39714  - 05/03/2023 22:33:58     LOG [InstanceLoader] AppModule dependencies initialized +8ms
[Nest] 39714  - 05/03/2023 22:33:58     LOG [RoutesResolver] AppController {/}: +4ms
[Nest] 39714  - 05/03/2023 22:33:58     LOG [RouterExplorer] Mapped {/, GET} route +2ms
[Nest] 39714  - 05/03/2023 22:33:58     LOG [NestApplication] Nest application successfully started +4ms
```

Le serveur est lancé et accessible



Rest :

Un standard qui donne des règles pour créer votre api.

C'est-à-dire que votre serveur va offrir des routes qui vont respecter certaines contraintes qui vont garantir une facilité d'utilisation, une scalabilité, une extensibilité.

1. La **séparation entre client et serveur** : les responsabilités du côté serveur et du côté client sont séparées, si bien que **chaque côté peut être implémenté indépendamment de l'autre**. Le code côté serveur (l'API) et celui côté client peuvent chacun être modifiés sans affecter l'autre, tant que **tous deux continuent de communiquer dans le même format**. Dans une architecture REST, différents clients envoient des requêtes sur les mêmes *endpoints*, effectuent les mêmes actions et obtiennent les mêmes réponses.

ex: GET monapi.com/annonces
pour récupérer des annonces. Deux clients qui vont sur mon site qui va effectuer cette même requête auront les mêmes annonces. Je peux modifier le code côté serveur, tant que ce endpoint envoie la même data, le client continu de fonctionner correctement

2. L’**absence d’état de sessions** (*stateless*) **:** la communication entre client et serveur ne conserve pas l’état des sessions d’une requête à l’autre. Autrement dit, **l’état d’une session est inclus dans chaque requête**, ce qui signifie que ni le client ni le serveur n’a besoin de connaître l’état de l’autre pour communiquer. Chaque requête est complète et se suffit à elle-même : pas besoin de maintenir une connexion continue entre client et serveur, ce qui implique une **plus grande tolérance à l’échec**. De plus, cela permet aux APIs REST de répondre aux requêtes de plusieurs clients différents sans saturer les ports du serveur. L’exception à cette règle est **l’authentification**, pour que le client n’ait pas à préciser ses informations d’authentification à chaque requête.

ex:

1. GET monapi.com/annonces

Je récupère les annonces

2. POST monapi.com/annonces

je créé une nouvelle annonce, j’aurais pu faire le 2. de façon indépendante du 1.

3. GET monapi.com/annonces

Je recupère de nouveau les annonces, je peux peut être voir celle créé dedans

3. L’**uniformité de l’interface** : les différentes actions et/ou ressources disponibles avec leurs *endpoints* et leurs paramètres spécifiques doivent être décidés et **respectés religieusement,** de façon **uniforme** par le client et le serveur. Chaque réponse doit contenir suffisamment d’informations pour être interprétée sans que le client n’ait besoin d’autres informations au préalable. Les réponses ne doivent pas être trop longues et doivent contenir, si nécessaire, des **liens** vers d’autres *endpoints*.

ex:
GET monapi.com/annonces/42
Va me renvoyer l’annonce numéro 42, avec toutes les infos nécessaire, par exemple titre, prix, id de la catégorie (par exemple 9) et une info sur le endpoint pour avoir cette catégorie : monapi.com/categories
GET monapi.com/categories/9
Me renvoie cette catégorie, je vois que c’est la vente immobilière
Je peux aussi généralement y acceder a travers l’annonce:
GET monapi.com/annonces/42/category

4. La **mise en cache** : les réponses peuvent être mises en cache pour éviter de surcharger inutilement le serveur. **La mise en cache doit être bien gérée** : l'API REST doit préciser si telle ou telle réponse peut être mise en cache et pour combien de temps pour éviter que le client ne reçoive des **informations obsolètes**.

Ex: header

cache-control: max-age=3600, private

5. L'architecture **en couches** : un client connecté à une API REST ne peut en général pas distinguer s'il est en communication avec le serveur final ou un serveur intermédiaire. Une architecture REST permet par exemple de recevoir les requêtes sur un serveur A, de stocker ses données sur un serveur B et de gérer les authentifications sur un serveur C.

6. Le **code à la demande**. Cette contrainte est **optionnelle**. Elle signifie qu'une API peut retourner du **code exécutable** au lieu d'une réponse en JSON ou en XML par exemple. Cela signifie qu'une API RESTful peut **étendre** le code du client tout en lui simplifiant la vie en lui fournissant du code exécutable tel qu'un **script JavaScript** ou un **applet Java**.

Projets : fichiers:

tsconfig.json -> infos compil

tsconfig.build -> specifique pour le build production

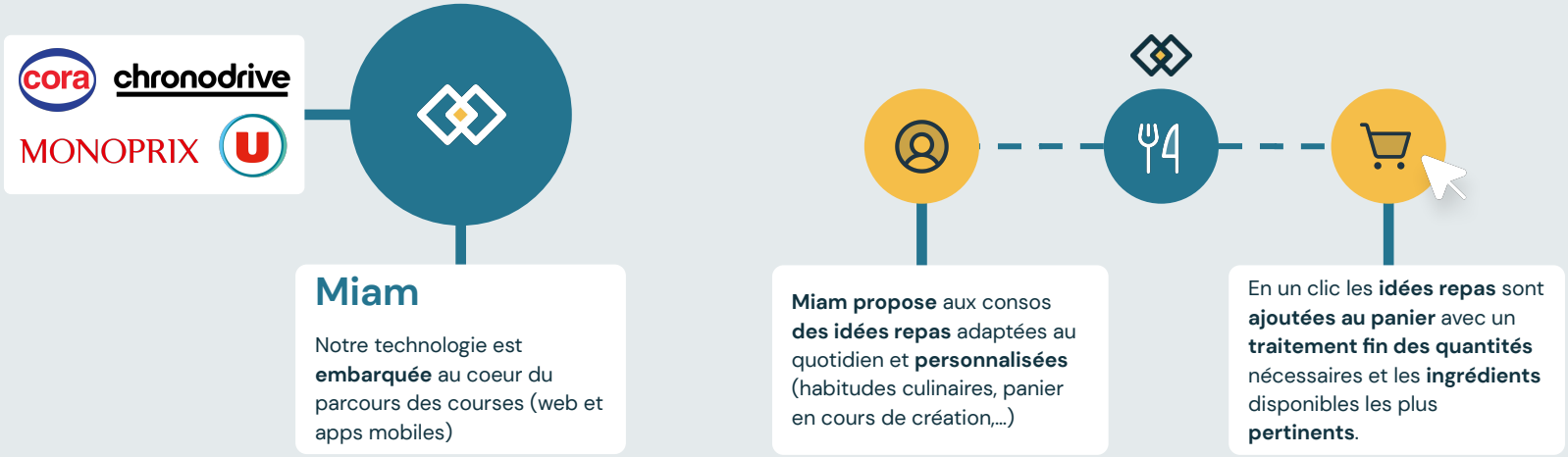
lint -> info linter, cad ce que l'ide indique comme erreur

package.json
package-lock
-> dejà vu. pour les librairies importées
+ des scripts pour réaliser des actions : build, start, format

```
"scripts": {
  "build": "nest build",
  "format": "prettier --write \"src/**/*.ts\" \"test/**/*.ts\"",
  "start": "nest start",
  "start:dev": "nest start --watch",
  "start:debug": "nest start --debug --watch",
  "start:prod": "node dist/main",
  "lint": "eslint \"{src,apps,libs,test}/**/*.ts\" --fix",
  "test": "jest",
  "test:watch": "jest --watch",
  "test:cov": "jest --coverage",
  "test:debug": "node --inspect-brk -r tsconfig-paths/register -r ts-node/register node_modules/.bin/jest --runInBand",
  "test:e2e": "jest --config ./test/jest-e2e.json"
},
```

A propos de Miam

Miam réinvente et simplifie les courses alimentaires.



Miam

Notre technologie est **embarquée** au coeur du parcours des courses (web et apps mobiles)

Miam propose aux consos **des idées repas** adaptées au quotidien et **personnalisées** (habitudes culinaires, panier en cours de création,...)

En un clic les **idées repas** sont **ajoutées au panier** avec un **traitement fin des quantités** nécessaires et les **ingrédients** disponibles les plus **pertinents**.

Qui sommes-nous ? Une startup tech lilloise, qui développe depuis 2019 des solutions d'intelligence artificielle à destination de la grande distribution.

Plus de détails sur https://miam.tech