T2112
Développement informatique III (Pratique)

Initialisation Backend & TypeScript
TP 01

Auteur(s) :
Jonathan Noël

Dernière mise à jour le
2 février 2026

Table des matières

Légendes

👨‍🏫 Conseil du professeur.
ℹ️ Une information ...
🚨 Pré-requis pour le TP.
🖥️ Exercice à réaliser.
👓 Exercice d'observation ou de lecture.

1 Introduction

L'objectif de ce premier TP est de mettre en place l'environnement de développement qui sera utilisé pour le projet de fin de semestre. Nous allons passer de la théorie (Architecture Client/Serveur , Node.js, ...) à la pratique en initialisant un serveur web basique mais strictement typé.
Pour certaines séances de travail pratique, il est interdit d'utiliser les outils d'intelligence artificielle génératives afin que vous appreniez les fondamentaux correctement, ce qui est le cas de cette première séance. Aidez-vous un maximum des documentations officielles liées aux technologies que vous utiliserez dans le tp.

2 Environnement de travail

Nous allons partir de zéro. Pas de générateur de code magique pour l'instant, afin de comprendre ce que nous installons. L'objectif est de préparer un environnement de travail propre pour ce TP et les TPs à venir.

1. Créez un dossier `dev3-backend-tp1` et ouvrez-le dans un IDE (VS Code recommandé).
2. Initialisez un projet Node.js pour créer le fichier `package.json`.

   ℹ️ Quelle commande permet de le faire rapidement sans répondre aux questions une par une ?

3. Installez les dépendances de développement nécessaires pour `TypeScript`.

   ℹ️ Nous avons besoin de `typescript`, `ts-node` (pour exécuter du TS directement) et `@types/node` (pour que TS reconnaisse les objets globaux de Node).
   Renseignez-vous sur la commande : `npm install ... --save-dev`

4. Générez le fichier de configuration `tsconfig.json`. Quelle est son utilité ?

   ℹ️ Cherchez la commande via `npx tsc ...`
   Dans ce fichier, assurez-vous de décommenter/activer l'option `"outDir": "./dist"` et `"rootDir": "./src"`.

Le résultat de votre architecture doit ressembler à la figure 1.

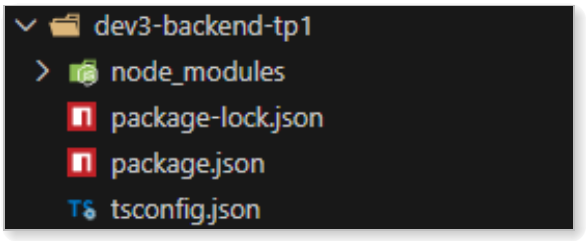

FIGURE 1 – Résultat attendu

3 Lier le projet à un répertoire Github Classroom

Pour commencer, rendez-vous sur le lien suivant (https://classroom.github.com/a/m4jEJxZW). Connectez-vous avec votre compte Github (EPHEC de préférence) et sélectionnez votre nom dans la liste. Ensuite, liez votre projet local avec votre répertoire distant. Pour ce faire, rendez-vous en CMD dans le dossier de projet et suivez les étapes suivantes, qui sont aussi présentent lors de votre premier accès à votre nouveau répertoire.

```
git commit -m "Init"
git branch -M main
git remote add origin git@github.com:JN-EPHEC/individual-practice-sessions-<USERNAME_GITHUB>.git
git push -u origin main
```

4 Hello World en TypeScript

Sachant qu'on travaille avec TypeScript et avant de faire du web, il faut s'assurer que celui-ci fonctionne correctement grâce à un simple code de test. Pour ce faire,

1. Commencez par créer un dossier `src` dans votre dossier de tp.
2. Ensuite, dans ce dossier `src`, créez un fichier `server.ts`
3. Écrivez une fonction simple `greet(name: string): string` (comme vu dans le cours théorique) qui retourne une phrase, peu importe laquelle.

   🤦 Il est indispensable de bien travailler avec la force du typage de TypeScript. Évidemment, la fonction `greet(name)` fonctionnera aussi bien sans, mais ce n'est pas le but ici.

4. Appelez cette fonction dans un `console.log`.
5. Essayez d'exécuter directement le fichier via `node src/server.ts`. Analysez l'erreur qui vous sera renvoyée. Pourquoi recevez-vous cette erreur ?

   🤦 Tips : C'est en lien avec TypeScript !

6. Ajoutez un **script** `"start"` dans le ficher `package.json` pour lancer votre fichier via `ts-node src/server.ts`
7. Lancez ensuite ce script. Si la phrase s'affiche en console, vous êtes prêts pour la suite.

```
C:\Users\j.noel\Desktop\EPHEC\devIII-tp\TP01\dev3-backend-tp1>npm run start

> dev3-backend-tp1@1.0.0 start
> ts-node src/server.ts

Hello, Étudiants de BAC2!
```

FIGURE 2 – Résultat de l'exécution

ℹ️ Pensez à commit votre travail !

5 Création d'un serveur express.js

Maintenant, nous allons transformer ce script en serveur web capable d'écouter des requêtes HTTP.

5.1 Installation

Installez le framework Express dans votre projet.

ℹ️ Attention : Express est écrit en JS. Comme nous sommes en TypeScript, si vous installez juste express, TS va crier.

🤦 Quelle dépendance supplémentaire (commençant par @types/) devez-vous installer en devDependency pour qu'Express soit reconnu par TypeScript ?

5.2 Le code minimal

Dans `src/server.ts`, implémentez le code vu dans les slides du cours théorique `"P1.2-Backend;s.15"` en l'adaptant :

1. Commencez par importer express en haut du fichier.
2. Définissez un port (ex : 3000).
3. Créez une route `GET` sur la racine `/` qui renvoie `"Bienvenue sur mon serveur API"`.

4. Lancez le serveur et testez-le via votre navigateur (http://localhost:<PORT>).

   🚨 Pour que le serveur puisse être lancé correctement sans erreurs liées à TypeScript et les fichiers .ts, il faut changer le script `'start'` pour qu'il utilise la commande `tsx` et non `ts-node`. Pour ce faire, modifier le script start par `"tsx src/server.ts"`. Si la commande n'existe pas, il vous manque un package npm, installez-le.

5. **Défi typage** : Dans la callback de `app.get('/', (req, res) => { ... })`, TypeScript peut déduire les types implicitement. Cependant, pour être rigoureux, essayez de trouver quels sont les types exacts de `req` et `res` fournis par Express et typez-les explicitement.

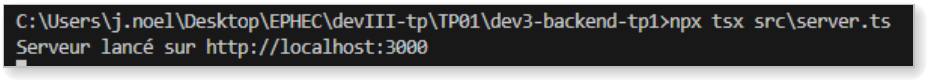

FIGURE 3 – Serveur démarré

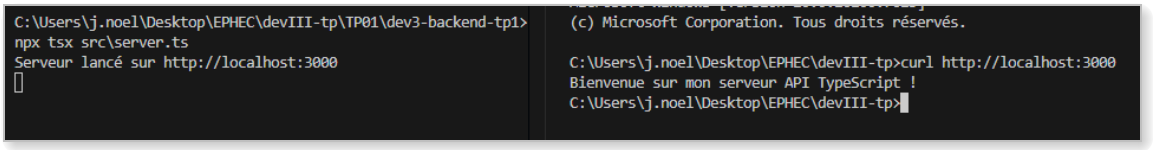

FIGURE 4 – Serveur testé via cUrl

ℹ️ Pensez à commit votre travail !

6 Exercices

Maintenant que le serveur tourne, implémentez les fonctionnalités suivantes. Vous devez chercher dans la documentation d'Express comment réaliser ces tâches. En plus de cela, forcez vous à travailler avec le typage statique de TypeScript !

🚨 Attention que pour le moment nous ne travaillons pas avec un service qui rafraichit l'exécution de notre serveur lors des modifications. Cela signifie que si vous modifiez des lignes d'instructions, il est nécessaire de redémarrer le serveur.

6.1 Exercice 1 : API JSON et `Array`

Le cours théorique rappelle le fonctionnement des `Array` et du format JSON. Créez une route GET `/api/data`. Cette route doit retourner un tableau d'objets JSON représentant les étudiants (id, nom, prenom).

```
const etudiants = [
  { id: 1, nom: "Dupont", prenom: "Jean" },
  { id: 2, nom: "Martin", prenom: "Sophie" },
  { id: 3, nom: "Doe", prenom: "John" },
];
```

👨‍🏫 Utilisez `res.json()` au lieu de `res.send()`

FIGURE 5 – Résultat attendu

ℹ️ Pensez à commit votre travail !

6.2 Exercice 2 : Paramètres dynamiques

Créez une route GET `/api/hello/:name`. Si j'appelle `/api/hello/Yves`, le serveur doit répondre en JSON et le timestamp est la datetime exacte au moment du traitement de la requête.
`{ "message": "Bonjour Yves", "timestamp":"2026-01-29T12:00:19.821Z" }`

👨‍🏫 Regardez du côté de `req.params`

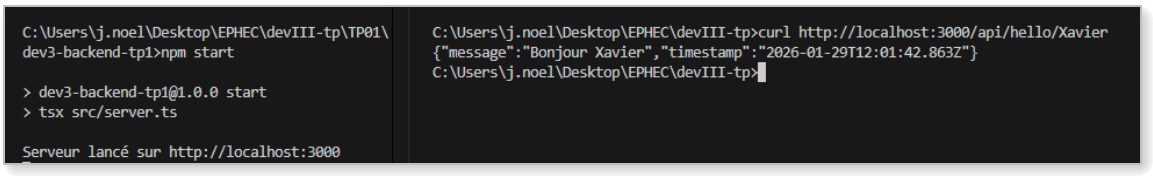

FIGURE 6 – Résultat attendu

ℹ️ Pensez à commit votre travail !

6.3 Exercice 3 : Structure et modularité

Pour l'instant, tout est dans `server.ts` et c'est une mauvaise pratique. En vous inspirant de la structure vue dans les slides de théorie :

1. Créez un fichier `src/routes/userRoutes.ts`.
2. Écrivez vos routes liées aux utilisateurs dedans, pour commencer un simple `GET` de `/api/users` qui renvoie, en JSON, la liste des utilisateurs suivante :

```
const users = [
  { id: 1, name: "Alice" },
  { id: 2, name: "Bob" },
];
```

👨‍🏫 Informez-vous et utilisez `express.Router`

3. Importez ce routeur dans le fichier `server.ts`

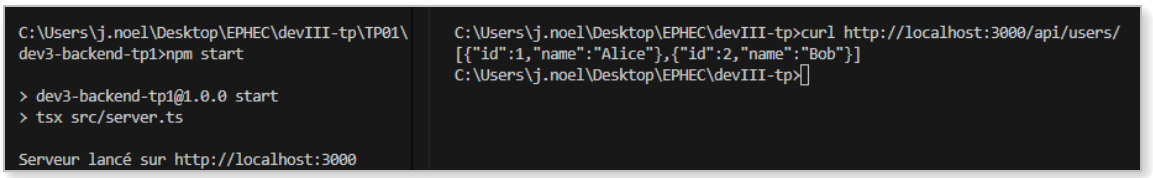

FIGURE 7 – Résultat attendu

ℹ️ Pensez à commit votre travail !

7 Pour la semaine prochaine

Assurez-vous que votre environnement est stable. La semaine prochaine, nous connecterons ce serveur à une base de données. **Cela signifie que vous devez avoir terminer la totalité de ce tp avant de commencer le tp suivant.**