T2112
Développement informatique III (Pratique)

Persistance & Frontend basique
TP 02

Auteur(s) :
Jonathan Noël

Dernière mise à jour le
2 février 2026

Table des matières

Légendes

Conseil du professeur.
Une information ...
Pré-requis pour le TP.
Exercice à réaliser.
Exercice d'observation ou de lecture.

1 Introduction

🚨 Pour réaliser ce TP, vous devez absolument avoir terminé le TP précédent. Si ce n'est pas le cas, c'est que vous êtes en retard et que vous devriez le rattraper au plus vite. Attention que tous les TP suivants auront comme prérequis le précédent.

L'objectif de ce TP est de remplacer les données fictives par une vraie base de données relationnelle via un ORM et de créer une interface pour consommer l'API.
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 seconde séance. Aidez-vous un maximum des documentations officielles liées aux technologies que vous utiliserez dans le tp. Si dans les 30 dernières minutes du TP vous n'arrivez pas à avancer, aidez-vous de l'IAg, mais pas avant !

2 Mise en place de la DB

Nous allons utiliser **Sequelize**, un ORM populaire pour Node.js, comme vu au cours. Il nous permettra de définir nos entités et relations directement dans le code source et ensuite il s'occupera, en tant qu'intermédiaire, de travailler sur la base de données.

2.1 Installation

👨‍🏫 Rappel : `npm install ...`

1. Installez le paquet `sequelize` dans le projet
2. Installez le pilote de base de données dans le projet.

   👨‍🏫 Le cours mentionne PostgreSQL (pg et pg-hstore). Pour ce TP, vous pouvez utiliser SQLite (sqlite3) pour éviter d'avoir à configurer un serveur de base de données externe sur vos machines.

2.2 Configuration de Sequelize

1. Créez un fichier `'src/config/database.ts'`
2. Initialisez une instance de Sequelize. Nous travaillons par module, il est donc important d'exporter votre connecteur sequelize dans ce fichier.

   ```
   import { Sequelize } from "sequelize";

   const sequelize = ...
   [...]
   export default sequelize;
   ```

   Aidez-vous de la documentation suivante :
   https://sequelize.org/docs/v6/getting-started/

   👨‍🏫 En utilisant SQLite, le dialect sera `'sqlite'` et le `storage` pointera vers un fichier (ex : `'./database.sqlite'`). Ce fichier sera créé automatiquement lors de la synchronisation.

3. Mettez à jour `"src/server.ts"` pour importer `sequelize` depuis votre fichier de configuration.
4. Assurez-vous de la bonne connectivité avec un `console.log()` après la connexion.

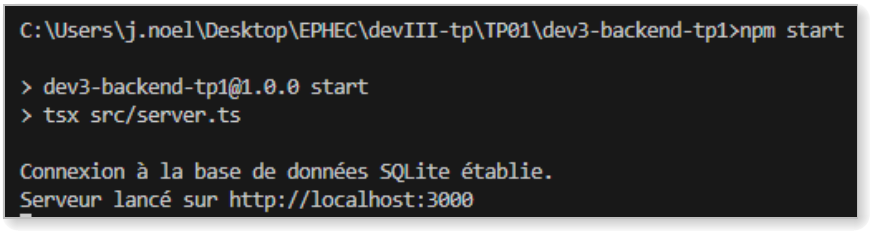

FIGURE 1 – Sequelize bien connecté

2.3 Création d'un modèle

1. Créez un dossier `'src/models'`
2. Créez le fichier `User.ts` dedans
3. A l'aide de la documentation https://sequelize.org/docs/v6/core-concepts/model-basics/, définissez le modèle `User` en étendant la classe `Model` de Sequelize. Champs à prévoir : nom (String), prenom (String).

   👨‍🏫 TypeScript demandera peut-être de déclarer les types des attributs dans la classe. Faites-le pour exploiter la force de celui-ci.

2.4 Synchronisation

Dans votre fichier principal `'server.ts'`, ajoutez une instruction pour synchroniser les modèles avec la base de données au démarrage du serveur. C'est normal que votre `app.listen()` soit englobé dans `sequelize.sync()`, ça permet de d'abord vérifier la connectivité DB avant de lancer le serveur.

👨‍🏫 Renseignez-vous sur `sequelize.sync()`

2.5 Ajout d'un utilisateur en CLI

Si vous avez bien configuré et lancé votre application, un fichier `"database.sqlite"` a été créé par l'ORM à la racine du projet. Pour démarrer, on va ajouter un utilisateur en utilisant le CMD.

1. Commencez par aller à la racine du projet en cmd.
2. Entrez dans le CLI sqlite via

   ```
   sqlite3 database.sqlite
   ```

3. Analysez le schéma du modèle User

   ```
   .schema users
   ```

4. Ensuite, ajoutez un utilisateur grâce à la commande suivante

   ```
   INSERT INTO users (nom, prenom, createdAt, updatedAt)
   VALUES ('John', 'Doe', datetime('now'), datetime('now'));
   ```

5. Pour terminer, vérifiez la présence de la ligne dans la table. Le ; est indispensable !

   ```
   SELECT * FROM users;
   ```

ℹ️ Pensez à commit votre travail !

3 Refactoring de l'API

Modifiez vos routes existantes (dans `'src/routes/userRoutes.ts'` si vous avez fait le bonus du TP 1, sinon dans `'server.ts'`) pour interagir avec la base de données au lieu du tableau statique. Utilisez les méthodes de Sequelize vues en cours théorique et ensuite testez vos routes avec cURL avant de passer à la suite.

👨‍💻 N'oubliez pas de configurer express pour parser le JSON (`app.use(express.json())`) sinon `req.body` sera vide.

```
GET / : Récupérer tous les utilisateurs (User.findAll()).
POST / : Créer un utilisateur (User.create(...)) .
DELETE /:id : Supprimer un utilisateur par son ID (User.destroy(...)).
```

ℹ️ Pensez à commit votre travail !

4 Introduction Frontend basique

Nous allons créer une interface minimale pour visualiser nos données. Comme vu en cours, le Frontend est la partie "client side" visible par l'utilisateur. Pour commencer, pas de Framwork ou de technologies complexes. Node.js avec Express.js permet de rendre des pages lorsque l'utilisateur en demande. C'est minimaliste, mais c'est un bon début.

4.1 Servir des fichiers statiques

1. Créez un dossier `"public"` à la racine.
2. Configurez Express dans `'server.ts'` pour servir ce dossier comme statique.

   👨‍💻 Cherchez de la documentation à propos d'`express.static`
3. Par la suite, on aura un fichier index.html qui sera servi par défaut lorsqu'on appelle la racine de notre application.

4.2 Structure HTML

L'objectif ici est de créer un fichier HTML basique, qui sera rendu ensuite à l'utilisateur. Commencez par créer un fichier `'public/index.html'`.

1. Ajoutez une structure HTML5 de base.
2. Importez une librairie CSS via un CDN pour rendre ça plus joli rapidement (ex : Bootstrap, MaterialUI, etc.). Explorez donc les librairies existantes importables via un CDN et choissisez celle qui vous inspire.

   👨‍💻 Pour rappel, un CDN est un ensemble de serveurs répartis géographiquement qui mettent en cache du contenu. Cela permet donc de récupérer des librairies en important simplement celles-ci dans nos fichiers html.
3. Ajoutez :
   - Un titre `<h1>`.
   - Une liste `<ul>` vide qui contiendra les utilisateurs.
   - Un formulaire simple (`Input` Nom, `Input` Prénom, `Bouton` "Ajouter").

   Voici mon résultat pour cela en utilisant Bootstrap :

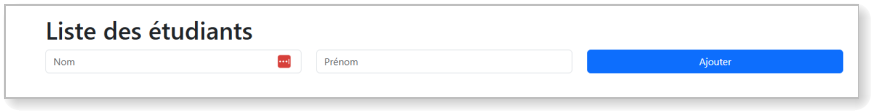

FIGURE 2 – Page HTML statique basique rendue par Express.js

4.3 Interaction JavaScript (Consommation d'API)

🚨 Pour cette partie, j'espère que vous avez encore des restes du cours de "Logique et Programmation" de BAC1 😄

Les données seront récupérées grâce à la fonction `fetch()` en utilisant l'API native du navigateur. Dans un fichier `'public/script.js'`, implémentez le scénario suivant :

👨‍💻 Pour quelle raison est-ce que le fichier .js suivant est créé dans le dossier public ?

1. Au chargement de la page, faites un appel `GET /api/users`.
2. Pour chaque utilisateur reçu, ajoutez un `<li>` dans la liste HTML.
3. Lors de la soumission du formulaire, interceptez l'événement, récupérez les valeurs, et faites un appel `POST /api/users`.
4. Si l'ajout réussit, rafraîchissez la liste sans recharger la page.

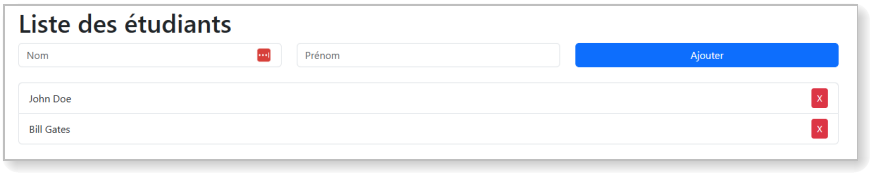

FIGURE 3 – Ajout après clic sur le bouton ajouter

ℹ️ Pensez à commit votre travail !

5 Exercices

5.1 Méthode DELETE

Ajoutez un bouton "X" à côté de chaque utilisateur (voir figure 3) dans la liste et lorsque vous cliquez dessus, cela supprime l'utilisateur en base de données.

ℹ️ Pensez à commit votre travail !

5.2 UI

Amusez-vous à rendre votre page plus jolie en exploitant la librarie que vous avez importé. N'hésitez pas à creuser plus loin que les simples styles. Amusez-vous avec les grids, les éléments préfabriqués etc.

ℹ️ Pensez à commit votre travail !

6 Travail à rendre

🚨 Pour cette partie et pour éviter d'impacter le projet qui évolue durant les TPs, déplacez-vous sur une branche `"homework_1"`. Pensez-bien à commit toute votre progression avant de changer de branche, afin de figer celle-ci sur la branche `"main"`.

Pour la **veille du prochain TP à 23:59**, rendez un **rapport** sur une extension de votre projet sur Moodle. L'objectif est de définir, individuellement, une extension à votre projet reprenant les points de matières couverts par les TP 1 et 2. Vous êtes libre des choix d'améliorations et de l'ampleur du travail, mais le rendu doit démontrer que vous maîtrisez un ensemble minimal de concepts. Le rapport doit contenir le code source concerné et des captures d'écrans en annexes. Veillez à bien référencer ces annexes dans votre texte explicatif.

6.1 Consignes générales

Vous choisissez librement les améliorations ou fonctionnalités à ajouter en fonction de la matière vue en TP. Par contre, votre travail doit couvrir explicitement la checklist ci-dessous.

6.2 Concepts à valider

🚨 TOUS doivent être traités d'une manière ou d'une autre.

1. API & routes Express (création ou amélioration d'au moins 2 endpoints).
2. Utilisation appropriée de la base de données (migrations/seeds).

3. Typage TypeScript et bonne séparation des responsabilités (models / routes / services).
4. Validation des entrées et gestion des erreurs (responses HTTP correctes).
5. Interaction client <–> API (fetch, formulaire, validation côté client ou UI minimale).

6.3 Exemple de paragraphe de rapport

*"[...] Dans cet objectif j'ai ajouté un endpoint **GET /api/users** qui renvoie la liste des utilisateurs - la logique principale est de récupérer l'ensemble des utilisateurs via la méthode findAll() de l'ORM sequelize (annexe 1 - lignes 12 à 15) et le résultat (status 200 + payload attendu) est montré sur la capture d'écran (figure 1). [...]"*

```
[...]
12. app.get('/api/users', async (req, res) => {
13.     const users = await userService.findAll();
14.     res.status(200).json(users);
15. });
[...]
```

6.4 Livrables obligatoires

1. Code source sur la branche `"homework_1"`.
2. Créez une copie de ce projet Overleaf Template de rapport
3. Le rapport .pdf contenant :
   (a) **Ce que je souhaite atteindre (objectifs précis).**
   (b) **Stratégie et raisonnement (choix techniques, compromis, plan de travail).**
   (c) **Ce que j'ai réellement atteint et comment. Incluez des captures d'écran** (UI, résultats des tests, requêtes API via curl, console serveur ou DB)

👨‍💻 Soyez concis et précis dans le rapport ; privilégiez les preuves.