Backend pour le suivi des élèves - Step5

Maintenant que nous avons mis en place un CRUD simple des élèves, nous devons apporter plusieurs améliorations.

Promesse, async/await

Pour les plus observateurs d'entre vous, vous avez peut-être remarqué que, parfois, dans les méthodes des contrôleurs, nous avons utilisé le mot-clé `await`.

> 🧠 **Pré-requis**
> Comprendre ce qu'est une **Promesse** en JavaScript, ainsi que le fonctionnement de `async/await`.
> Lien : https://fr.javascript.info/async

🧪 Exemple 1 – Sans `await`

```
async index({}: HttpContext) {
  return Student.query().orderBy('name').orderBy('firstname')
}
```

Ici, `Student.query()` ne retourne **pas une Promesse**, mais un objet spécial : un **Query Builder**.

Ce **Query Builder** permet de construire une requête SQL **sans l'exécuter immédiatement**.

🔍 Que se passe-t-il si on tente d'accéder directement aux résultats ?

```
async index({}: HttpContext) {
  const students = Student.query().orderBy('name').orderBy('firstname')
  console.log(students.length)
  return students
}
```

- L'API retourne bien la liste d'étudiants (Adonis attend et exécute automatiquement la requête à cause de son moteur interne).
- MAIS dans la console, `console.log(students.length)` affiche **`undefined`**.

> 💡 **Pourquoi ?**
> Parce que `students` est un **QueryBuilder**, **pas encore un tableau**.
> Un QueryBuilder **n'a pas de propriété** `.length`.

✅ Exemple 2 – Avec `await`

```
async index({}: HttpContext) {
  const students = await Student.query().orderBy('name').orderBy('firstname')
  console.log(students.length)
  return students
}
```

✅ Ici, la requête est **exécutée immédiatement** grâce à `await`.
Et `students` est maintenant un **tableau d'objets** (instances de `Student`).
Donc `students.length` fonctionne parfaitement.

🧠 Petit point technique

> Dans AdonisJS (et donc dans Lucid ORM), un QueryBuilder n'est **pas une Promesse**,
> mais il est ***thenable*** : il possède une méthode `.then()`.

Cela signifie que tu peux faire :

```
await Student.query().orderBy('name') // fonctionne
```

et même :

```
Student.query().orderBy('name').then(...)
```

🔁 Cela permet à Adonis d'exécuter la requête **même si tu oublies** `await`, car il "devine" qu'il faut attendre l'objet retourné.

⚠️ Pourquoi c'est dangereux ?

- Le QueryBuilder n'est pas encore exécuté
- Le code cache l'asynchronisme
- Si tu veux intercepter une erreur ou ajouter une étape, ça ne marche pas

> ⚠️ REGLE :
> Donc nous allons maintenant toujours préciser les `await` même si le code fonctionne sans.

> 💪 Entraînement :
> A vous d'essayer d'enlever les `await` pour vous convaincre que cela fonctionne toujours même si cela n'est pas recommandé.

Statuts HTTP

Voila une liste non exaustive des statuts HTTP :

Statut	Méthode	Explication
200	`response.ok(data)`	OK (valeur par défaut implicite)
201	`response.created()`	Création réussie
204	`response.noContent()`	Pas de contenu à retourner
400	`response.badRequest()`	Erreur de validation ou de format
401	`response.unauthorized()`	Accès non autorisé
403	`response.forbidden()`	Interdit (auth ok mais droit insuffisant)
404	`response.notFound()`	Ressource introuvable
422	`response.unprocessableEntity()`	Erreur de validation (par défaut avec validator)
500	`response.internalServerError()`	Erreur interne du serveur

Pour l'instant, nous allons utiliser les `2xx`

> EXERCICE :
> A vous de modifier le code du contrôleur `students_controller` afin de retourner le bon statut `2xx`.

Vous pourrez comparer avec ma version.