Backend pour le suivi des élèves - Step3

Nous allons maintenant créer nos 1ères routes et nos 1ers contrôleurs afin de mettre en place le CRUD des élèves.

Mise en place d'une route exemple

Pour commencer en douceur pour la mise en place des routes, je vous propose de créer une route qui aura seulement un but pédagogique et de test.

Dans le fichier `start/routes.ts`, nous ajouter la route suivante :

```
/*
|--------------------------------------------------------------------------
| Routes file
|--------------------------------------------------------------------------
|
| The routes file is used for defining the HTTP routes.
|
*/

import router from '@adonisjs/core/services/router'

router.get('test', async () => {
  return 'API is working!'
})
```

Pour accéder à cette route, nous devons faire un appel HTTP : `GET /test`

Nous allons tenter d'accéder à cette route depuis notre navigateur.

Pour ce faire nous devons exécuter notre serveur de développement grâce à la commande :

```
npm run dev
```

Puis à l'aide de votre navigateur préféré vous rendre à l'URL : `http://localhost:3333/test`

Le message : `API is working!` devrait s'afficher !

Maintenant que vous avez compris par la pratique ce qu'est une route, nous pouvons définir l'ensemble des routes pour le CRUD d'un élève.

Question : Que veut dire CRUD ?

Mise en place des routes

```
/*
|--------------------------------------------------------------------------
| Routes file
|--------------------------------------------------------------------------
|
| The routes file is used for defining the HTTP routes.
|
*/

import router from '@adonisjs/core/services/router'
import StudentsController from '#controllers/students_controller'

router.get('test', async () => {
  return 'API is working!'
})

router.resource('students', StudentsController).apiOnly()

/* est équivalent à :
router.group(() => {
  router.get('', [StudentsController, 'index'])
  router.get(':id', [StudentsController, 'show'])
  router.post('', [StudentsController, 'store'])
  router.put(':id', [StudentsController, 'update'])
  router.patch(':id', [StudentsController, 'update'])
  router.delete(':id', [StudentsController, 'destroy'])
}).prefix('students')
*/
```

Question : A quoi correspondent get, post, put, patch ou delete ?

Question : A quoi correspondent index, show, store, update, destroy ?

Question : A quoi sert **.apiOnly** ?

Question : A quoi sert **.prefix('students')** ?

Question : Pouvez-vous expliquer la route **router.get(':id', [StudentsController, 'show']) ?**

Mise en place du controlleur `Student`

Création du controlleur `Student`

```
node ace make:controler Student -r
```

> 📝 **À noter**
> L'option -r pour ressource

Question : A quoi sert cette option **-r** ?

Nous pouvons supprimer les 2 méthodes : `create` et `update` qui dans une application web classique ont pour but d'afficher les formulaires de création ou de modification d'un élève.

Dans le cas d'une API REST, nous n'avons pas besoin et c'est le front-end qui aura la responsabilité de gérer les formulaires.

Retourner tous les élèves

```
/**
 * Afficher la liste des élèves
 */
async index({}: HttpContext) {
  return await Student.all()
}
```

Vous pouvez voir le JSON des élèves en visitant l'URL `http://localhost:3333/students`

> 💡 **Question**
> Que fait `Student.all()` ?

`Student` correspond à notre modèle.

`Student.all()` retourne un tableau d'élèves contenant tous les élèves présents dans la table `students`.

C'est l'ORM Lucid qui va se connecter à la db est faire la requête: `SELECT * FROM students`

Grâce à l'ORM Lucid, nous n'avons pas besoin d'écrire la requête SQL mais simplement `Student.all()`

> 💡 **Question**
> Pourquoi en faisant simplement `return await Student.all()` cela retourne du JSON ?

En fait, Adonis va faire 2 choses automatiquement :

- pour le `return` Adonis va faire `ctx.response.send()`
- comme l'objet retourné par l'ORM Lucid (Model[]) est sérialisable, `ctx.response.send()` convertit automatiquement en JSON.

> 🔍 Qu'est-ce que « sérialisable » veut dire ? Un objet sérialisable peut être converti en texte (comme du JSON). En gros : un objet JavaScript devient une chaîne de texte JSON lisible par le navigateur ou Bruno.

Ordonner les élèves

> 💡 **Problème**
> Les élèves ne sont pas trier par ordre alphabétique ?

Il y a un petit problème dans le json retourné. Ce serait plus agréable d'avoir les élèves ordonnés par le nom puis en cas d'homonyme par le prénom.

```
/**
 * Afficher la liste des élèves
 */
async index({}: HttpContext) {
  return await Student.query().orderBy('name').orderBy('firstname')
}
```

Retourner un élève

```
/**
 * Afficher un élève
 */
async show({ params }: HttpContext) {
  return await Student.findOrFail(params.id)
}
```

Vous pouvez voir le JSON de l'élève dont l'id est passé en paramètre en visitant l'URL `http://localhost:3333/students/3`

> 📝 **À noter**
> `params.id` va récupérer l'id présent dans la requete HTTP

> 💡 **Question**
> Que fait `Student.findOrFail(params.id)` ?

Cela va récupérer l'élève dont l'id=3 (dans notre cas).

Dit autrement, l'ORM Lucid va (en utilisant notre modèle `Student`) exécuter la requête `SELECT * FROM students WHERE id = 3`