

Application des surnoms des enseignants - Step4

Création du contrôleur pour les enseignants

Dans cette étape, nous allons apprendre à créer un contrôleur.

Qu'est ce qu'un contrôleur ?

Un contrôleur est un composant clé dans le cadre d'une architecture MVC (Model-View-Controller).

Il agit comme un intermédiaire entre la logique métier (modèles) et l'interface utilisateur (vues). Son rôle principal est de traiter les requêtes entrantes, d'interagir avec les modèles pour récupérer ou manipuler des données, puis de retourner une réponse appropriée (souvent sous forme de vue ou de données JSON).

Créer le contrôleur teacher à l'aide du CLI

```
$ node ace make:controller teacher -r
DONE: create app/controllers/teachers_controller.ts
```

A noter l'option `-r` pour ressource qui permet d'avoir l'ensemble des signatures des méthodes présentes dans le contrôleur.

```
import type { HttpContext } from '@adonisjs/core/http'

export default class TeachersController {
  /**
   * Display a list of resource
   */
  async index({}: HttpContext) {}

  /**
   * Display form to create a new record
   */
  async create({}: HttpContext) {}

  /**
   * Handle form submission for the create action
   */
  async store({ request }: HttpContext) {}

  /**
   * Show individual record
   */
  async show({ params }: HttpContext) {}

  /**
   * Edit individual record
   */
  async edit({ params }: HttpContext) {}
```

```
/**
 * Handle form submission for the edit action
 */
async update({ params, request }: HttpContext) {}

/**
 * Delete record
 */
async destroy({ params }: HttpContext) {}
}
```

Afficher la liste des enseignants

Nous allons commencer par afficher la liste des enseignants.

Pour cela, nous allons implémenter la méthode `index`.

```
import Teacher from '#models/teacher'
import type { HttpContext } from '@adonisjs/core/http'

export default class TeachersController {
  /**
   * Afficher la liste des enseignants
   */
  async index({ view }: HttpContext) {
    //
    // Récupérer la liste des enseignants triés par ordre alphabétique sur le nom
    // et le prénom
    const teachers = await Teacher.query().orderBy('lastname',
'asc').orderBy('firstname', 'asc')

    // Appel de la vue
    return view.render('pages/home', { teachers })
  }

  /**
   * Display form to create a new record
   */
  async create({}: HttpContext) {}

  /**
   * Handle form submission for the create action
   */
  async store({ request }: HttpContext) {}

  /**
   * Show individual record
   */
  async show({ params }: HttpContext) {}
}
```

```
/**
 * Edit individual record
 */
async edit({ params }: HttpContext) {}

/**
 * Handle form submission for the edit action
 */
async update({ params, request }: HttpContext) {}

/**
 * Delete record
 */
async destroy({ params }: HttpContext) {}
}
```

Que fait ce code ?

```
// Récupérer la liste des enseignants triés par ordre alphabétique sur le nom et le prénom
const teachers = await Teacher.query().orderBy('lastname', 'asc').orderBy('firstname', 'asc')
```

Cette ligne permet de récupérer tous les enseignants sous la forme d'un tableau de **Teachers**.

C'est l'équivalent de la requête SQL **SELECT * FROM teachers ORDER BY lastname, firstname**.

Teacher correspond au modèle créée dans l'étape précédente.

Nous allons donc accéder à la base de données MySQL par l'intermédiaire du modèle et de l'ORM Lucid.

Comme l'accès à la base de données est une opération asynchrone, le mot-clé **await** suspend l'exécution de la fonction jusqu'à ce que la base de données renvoie les résultats, garantissant que la variable **teachers** contiendra bien les données finales.

```
// Appel de la vue
return view.render('pages/home', { teachers })
```

Cette ligne permet simplement de transmettre la variable **teachers** à la vue **pages/home** et d'afficher cette vue.

A noter que **view** est récupéré grâce à une déconstruction de l'objet **HttpContext**.

Maintenant que vous avons codé la 1ère méthode de notre contrôleur, nous avons besoin de connecter cette méthode à une route.

Qu'est ce qu'une route ?

Une route, dans un framework comme AdonisJS, est une définition qui associe une URL à une logique métier (une fonction ou un contrôleur).

Les routes permettent de gérer les requêtes des utilisateurs et de retourner les réponses correspondantes.

Amélioration de la route homepage

Nous allons modifier le fichier des routes `start/routes.ts`.

```
/*
|-----
| Le fichier des routes
|-----
|
| Le fichier des routes a pour but de définir toutes les routes HTTP.
|
*/

import TeachersController from '#controllers/teachers_controller'
import router from '@adonisjs/core/services/router'

router.get('/', [TeachersController, 'index']).as('home')
```

Grâce à cette route, nous pouvons maintenant faire un GET HTTP sur <http://localhost:3333/> et le code de la méthode `index` du contrôleur `TeachersController` sera exécuté.

A noter que nous avons donné le nom `home` à la route.

Ceci nous permet d'utiliser ce nom partout dans l'application.

Ainsi, nous pouvons décider de modifier l'URL d'une route sans que cela nécessite des changements dans notre code (à part le fichier `routes.ts`).

Mais avant de tester cela, nous allons apporter quelques améliorations pour exploiter les données transmises du contrôleur à la vue.

Amélioration de la vue home

Dans notre partial `header.edge` nous avons la balise HTML `<nav>` pour les liens de navigation.

Comme nous venons de définir la route pour la homepage de notre application, nous pouvons utiliser cette route de la manière suivante :

Fichier `views/partials/header.edge`

```
<nav>
  <a href="{ route('home') }">Accueil</a>&nbsp;
  <a href="addTeacher.html">Ajouter un enseignant</a>
</nav>
```

{{ route('home') }} grâce au nom de la route `home`, va donner l'URL `/` comme défini dans le fichier des routes.

Pour finir, nous allons utiliser la variable `teachers` transmise par le contrôleur pour parcourir le tableau des enseignants et afficher les différentes informations (nom, prénom et surnom)

Fichier `views/pages/home.edge`

```
@component('components/layout', { title: 'Accueil' })

<h3>
  Liste des enseignants
</h3>
<form action="#" method="post">
  <table>
    <thead>
      <tr>
        <th>
          Nom
        </th>
        <th>
          Surnom
        </th>
        <th>
          Options
        </th>
      </tr>
    </thead>
    <tbody>
      @each ((teacher, index) in teachers)
        <tr>
          <td>
            {{ teacher.lastname }} {{ teacher.firstname }}
          </td>
          <td>
            {{ teacher.nickname }}
          </td>
          <td class="containerOptions">
            <a href="#">
              
            </a>
            <a href="javascript:confirmDelete()">
              
            </a>
            <a href="detailTeacher.html">
              
            </a>
          </td>
        </tr>
      @end
    </tbody>
  </table>
```

```
</form>
@endcomponent
```

Debugger facilement grâce dd()

La méthode `dd()` (abréviation de `dump and die`) est une fonction de débogage intégrée dans AdonisJS.

Elle permet d'afficher directement dans le navigateur des informations utiles sur des variables, des objets ou d'autres données, puis d'arrêter immédiatement l'exécution du script.

Par exemple, modifions notre méthode `index` du contrôleur `TeachersController` :

```
async index({ view }: HttpContext) {
  //
  // Récupérer la liste des enseignants triés par ordre alphabétique sur le nom
  // et le prénom
  const teachers = await Teacher.query().orderBy('lastname',
    'asc').orderBy('firstname', 'asc')

  // Pour obtenir des infos sur la variable teachers
  dd(teachers)

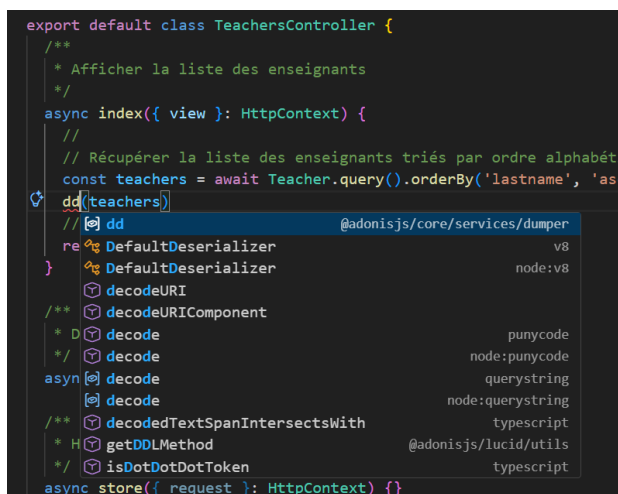
  // Appel de la vue
  return view.render('pages/home', { teachers })
}
```

Pour pouvoir utiliser `dd()` nous devons l'importer.

Heureusement, il existe un raccourci clavier pour importer automatiquement.

Astuce : comment importer automatiquement ?

placer le curseur après le `dd` mais avant les `()` et faire `ctrl + espace`



et cela va écrire la ligne ci-dessous dans votre fichier :

```
import { dd } from '@adonisjs/core/services/dumper'
```

Si maintenant vous tentez d'accéder à la homepage de l'application :

```
DUMP DIE
...\\OneDrive%20-%20Education%20Vaud\\P_Bulle\\app-teachers-adonisjs\\app\\controllers\\teachers_controller.ts:14
Array:12 [
  Teacher {
    modelOptions: Object { },
    modelTrx: undefined,
    transactionListener: [function bound listener],
    fillInvoked: false,
    cachedGetters: Object { },
    forceUpdate: false,
    $columns: Object { },
    $attributes: Object { },
    $original: Object { },
    $preloaded: Object { },
    $extras: Object { },
    $sideLoaded: Object { },
    $isPersisted: true,
    $isDeleted: false,
    $isLocal: false,
    section: undefined,
    [[Prototype]] { }
  },
  Teacher { },
  Teacher { },
  Teacher { },
  Teacher { },
  Teacher { },
  Teacher { },
  Teacher { },
  Teacher { },
  Teacher { },
  Teacher { },
  Teacher { }
]
```

Etat de l'application à la fin de cette étape

Notre application a toujours qu'une page à savoir la homepage.

Surnom des enseignants

Login Mot de passe Se connecter

[Accueil](#) [Ajouter un enseignant](#)

Liste des enseignants

Nom	Surnom	Options

Copyright GCR - P_Bulle Dev - 2025

Nous n'avons pour l'instant aucune donnée dans notre base de données.

C'est donc normal qu'aucun enseignant ne soit affiché !

Pour s'assurer que tout est normal, on peut vérifier dans la console qu'aucune erreur est indiquée.

```
MINGW64/c:/Users/px86dym/OneDrive - Education Vaud/P_Bulle/app-teachers
full-reload app\models\teacher.ts

Server address: http://localhost:3333
Watch Mode: HMR
Ready in: 1.92 s

[13:08:23.381] INFO (20140): started HTTP server on localhost:3333
```

Dans la prochaine étape [step5](#), nous allons ajouter des données à notre application.