Application des surnoms des enseignants avec Adonis - Step3

Création des tables MySQL

Dans cette étape, nous allons apprendre à créer des tables MySQL à l'aide des migrations.

Puis nous allons apprendre à réaliser des modèles (le M du pattern MVC).

Qu'est ce qu'une migration ?

Une migration est un mécanisme utilisé pour gérer les schémas de base de données dans le cadre d'un projet de développement. Elle permet de créer, modifier ou supprimer des tables et des colonnes dans une base de données de manière versionnée et contrôlée.

Les migrations sont particulièrement utiles dans les frameworks modernes comme AdonisJS, Laravel, Rails, ou Django, où elles offrent une manière structurée et collaborative de gérer l'évolution de la base de données tout au long du cycle de vie d'une application.

Migrations pour les enseignants et les sections

Nous avons besoin de créer 2 migrations, une pour les enseignants et une autre pour les sections.
Par défaut, AdonisJS nous propose une migration pour les utilisateurs de notre application.

Nous nous préoccupons pas volontairement de cela pour l'instant. Nous y reviendrons lorsque nous mettrons en place le système d'authentification.

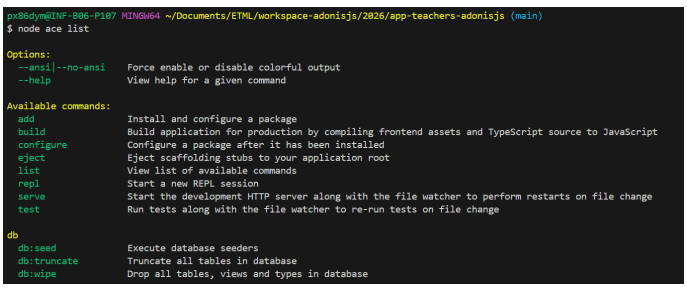
Créer les migrations à l'aide du CLI

Lorsque nous avons besoin de créer un élément (Migration, Modèle, Contrôleur, etc) avec AdonisJS, nous allons utiliser le CLI (Command Line Interface).

Pour voir la liste des actions que nous pouvons faire avec le CLI, il suffit de taper la commande :

```
node ace list
```

Voilà ce que donne la commande (Résultat partiel) :



Grâce à l'extension vs-code AdonisJS, vous pouvez aussi utiliser le CLI sans utiliser la ligne de commande.

Je vous laisse tester par vous même ou appeler votre enseignant si vous n'y parvenez pas !

Il est temps de créer nos 2 migrations :

```
px86dym@INF-N508-P104 MINGW64 ~/OneDrive - Education Vaud/P_Bulle/app-teachers
$ node ace make:migration sections
DONE:    create database/migrations/1734783986582_create_sections_table.ts

px86dym@INF-N508-P104 MINGW64 ~/OneDrive - Education Vaud/P_Bulle/app-teachers
$ node ace make:migration teachers
DONE:    create database/migrations/1734783999672_create_teachers_table.ts
```

Compléter les migrations

Grâce au CLI :

- les fichiers ont été créés dans le bon répertoire à savoir `database\migrations`
- les fichiers ont chacun le bon nom `*_create_teachers_table.ts` par exemple
- le code par défaut généré

Note : L'ordre de nos migration est important en raison de la clé étrangère `section_id`
En effet, il est impossible de créer la table `teachers` en définissant une clé étrangère `section_id` si le champ `id` de la table `section` n'existe pas.

Pour assurer que l'ordre soit correct, il est possible de renommer les noms des fichiers des migrations.
Par exemple :

- `1_create_sections_table.ts`
- `2_create_teachers_table.ts`

Ainsi, lorsque l'on va exécuter les migrations, elles seront exécutées dans le bon ordre.

Il nous reste à compléter les 2 migrations.

La migration pour les sections :

```ts
import { BaseSchema } from '@adonisjs/lucid/schema'

export default class extends BaseSchema {
  protected tableName = 'sections'

  async up() {
    this.schema.createTable(this.tableName, (table) => {
      table.increments('id')

      // La seule ligne que nous devons ajouter à cette migration
      // Nom de la section
      table.string('name').notNullable()

      table.timestamp('created_at')
      table.timestamp('updated_at')
    })
  }

  async down() {
```

```
    this.schema.dropTable(this.tableName)
  }
}
```

Une section est simplement définie par son nom.

Donc une chaine de caractère non nulle : `table.string('name').notNullable()`

La migration pour les enseignants :

Un enseignant est défini par :

- son genre
- son prénom
- son nom
- son surnom
- l'origine de son surnom
- la clé étrangère de l'id de la section auquel l'enseignant appartient

```
import { BaseSchema } from '@adonisjs/lucid/schema'

export default class extends BaseSchema {
  protected tableName = 'teachers'

  async up() {
    this.schema.createTable(this.tableName, (table) => {
      table.increments('id')

      // Champ genre pour lequel on définit un enum avec pour valeur par défaut
'woman'
      table.enum('gender', ['woman', 'man',
'other']).notNullable().defaultTo('woman')

      // Champ prénom
      table.string('firstname').notNullable()

      // Champ nom
      table.string('lastname').notNullable()

      // Champ surnom
      table.string('nickname').notNullable()

      // Champ origine du prénom
      table.string('origine').notNullable()

      // Clé étrangère de la section
      table
        .integer('section_id') // Clé étrangère
        .unsigned() // La clé ne doit pas être négative
        .references('id') // Référence la colonne `id` de la table `section`
        .inTable('sections') // Nom de la table de référence
```

```
          .onDelete('CASCADE') // Supprime les teachers si la section est supprimée
          .onUpdate('CASCADE') // Met à jour la clé étrangère si l'id change
        table.timestamp('created_at')
        table.timestamp('updated_at')
      })
    }

    async down() {
      this.schema.dropTable(this.tableName)
    }
  }
```

Compléter le fichier .env

Avant de pouvoir exécuter les migrations, nous avons besoin de donner quelques informations sur notre base de données à AdonisJS.

```
TZ=UTC
PORT=3333
HOST=localhost
LOG_LEVEL=info
APP_KEY=vafuca9u0xK5ViwXk-3OCaD4C--nh9O5
NODE_ENV=development
SESSION_DRIVER=cookie
DB_HOST=127.0.0.1
DB_PORT=6033
DB_USER=root
DB_PASSWORD=root
DB_DATABASE=db_teachers
```

Les 4 derniers éléments de configurations doivent être renseignés.

A noter le port de la DB `6033` qui correspond au port utilisé "normalement" pour nos conteneurs Docker.

De plus, la base de données `db_teachers` doit exister. A vous de la créer !

Lancer les migrations à l'aide du CLI

Ensuite nous pouvons exécuter les migrations :

```
$ node ace migration:run
[ info ] Upgrading migrations version from "1" to "2"
> migrated database/migrations/1734616322170_create_users_table
> migrated database/migrations/1734783986582_create_sections_table
> migrated database/migrations/1734783999672_create_teachers_table
```

Vérifier que les 2 tables MySQL correspondantes ont bien été créées.

En réalité, une 3ème table `user` a été créée car une migration par défaut existait.
Nous reviendrons sur cette table, cette migration et ce modèle lorsque nous mettrons en place l'authentification.

Modèles pour les enseignants et les sections

Qu'est ce qu'un modèle ?

Un modèle (ou Model) est une représentation logique et structurée des données dans une application. Il sert d'intermédiaire entre la base de données et le reste de l'application, permettant de gérer les interactions avec les données (comme la récupération, la création, la mise à jour et la suppression) tout en offrant une abstraction qui facilite le travail des développeurs.

Les modèles sont au cœur de l'architecture MVC (Model-View-Controller), où ils définissent les règles et la logique métier liées aux données.

Créer les modèles à l'aide du CLI

```
$ node ace make:model section
DONE:    create app/models/section.ts

px86dym@INF-N508-P104 MINGW64 ~/OneDriv
$ node ace make:model teacher
DONE:    create app/models/teacher.ts
```

Comme nous l'indique le CLI, les modèles sont générés dans le dossier `/app/models`.

Compléter les modèles

Le code pour le modèle Section :

```
import { DateTime } from 'luxon'
import { BaseModel, column } from '@adonisjs/lucid/orm'

export default class Section extends BaseModel {
  @column({ isPrimary: true })
  declare id: number

  @column()
  declare name: String

  @column.dateTime({ autoCreate: true })
  declare createdAt: DateTime

  @column.dateTime({ autoCreate: true, autoUpdate: true })
  declare updatedAt: DateTime
}
```

Le code pour le modèle Teacher :

```
import { DateTime } from 'luxon'
import { BaseModel, column, belongsTo } from '@adonisjs/lucid/orm'
import Section from '#models/section'
import type { BelongsTo } from '@adonisjs/lucid/types/relations'
```

```
export default class Teacher extends BaseModel {
  @column({ isPrimary: true })
  declare id: number

  @column()
  declare gender: String

  @column()
  declare firstname: String

  @column()
  declare lastname: String

  @column()
  declare nickname: String

  @column()
  declare origine: String

  @column()
  declare sectionId: number // Colonne correspondant à la clé étrangère

  @belongsTo(() => Section)
  declare section: BelongsTo<typeof Section> // Relation vers le modèle Section

  @column.dateTime({ autoCreate: true })
  declare createdAt: DateTime

  @column.dateTime({ autoCreate: true, autoUpdate: true })
  declare updatedAt: DateTime
}
```

Nous n'utilisons pas pour l'instant les modèles.

Prochaine étape

Dans la prochaine étape step4, nous allons créer notre 1er contrôleur, utiliser le modèle et envoyer des données à la vue.