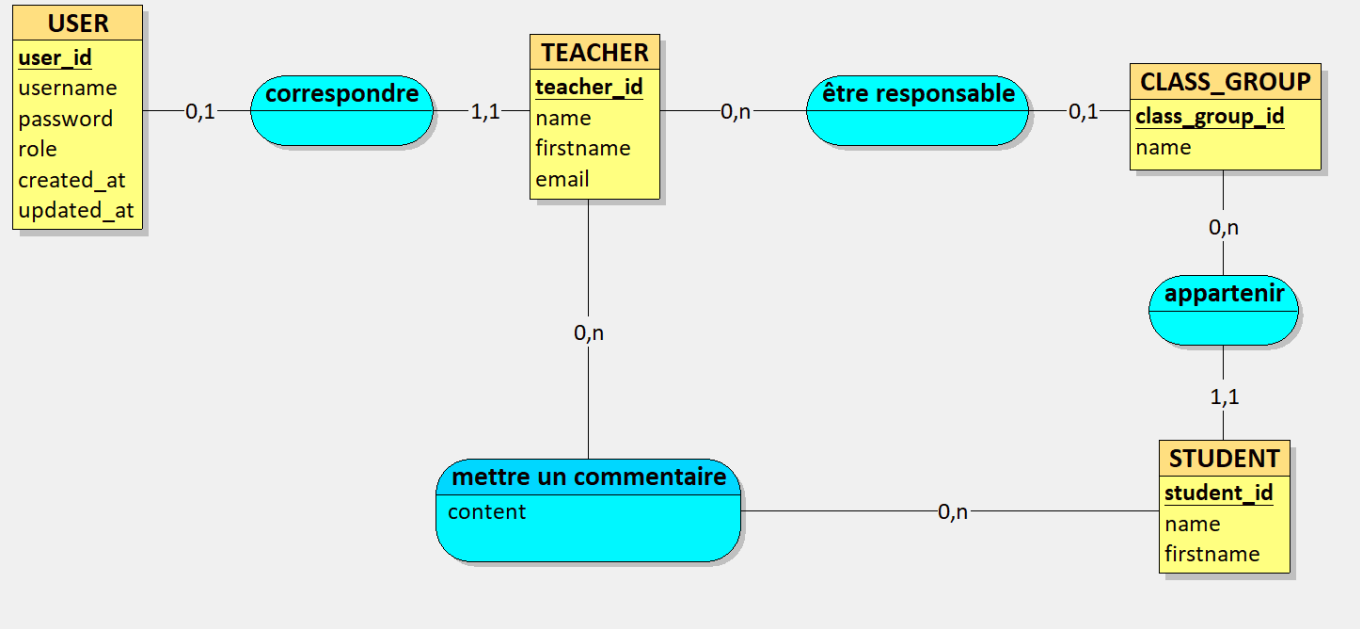
Backend pour le suivi des élèves - Step9

Nous allons maintenant nous intéresser aux relations entre les enseignants, les élèves et les classes.
Nous laissons volontairement les commentaires de côté pour l'instant ainsi que les utilisateurs.



Comme nous pouvons le voir :

- Un enseignant peut être responsable (Maître de classe) de plusieurs classes
- Une classe ne peut avoir qu'un responsable

On appelle cela une relation 1-N.

Idem pour la relation classe et élève.
C'est aussi une relation de type 1-N.

En base de données relationnelle, quand deux tables sont liées par une clé étrangère, on peut représenter cette relation dans les modèles Lucid pour simplifier les requêtes.

Lucid utilise deux décorateurs principaux pour ça :

`@hasMany` → l'entité courante possède plusieurs éléments de l'autre côté.

`@belongsTo` → l'entité courante appartient à un élément de l'autre côté.

Voyons maintenant concrètement comment intégrer cela dans les modèles.

Relation Enseignant - Classe

Les modèles

Dans le modèle Teacher :

```
import { BaseModel, column, hasMany } from '@adonisjs/lucid/orm'
import ClassGroup from './classgroup.js'
import type { HasMany } from '@adonisjs/lucid/types/relations'

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

  @column()
  declare name: string

  @column()
  declare firstname: string

  @column()
  declare email: string

  // Relation : enseignant → classes
  @hasMany(() => ClassGroup)
  declare classGroups: HasMany<typeof ClassGroup>
}
```

Dans le modèle ClassGroup :

```
import { BaseModel, belongsTo, column } from '@adonisjs/lucid/orm'
import Teacher from './teacher.js'
import type { BelongsTo } from '@adonisjs/lucid/types/relations'

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

  @column()
  declare name: string

  @column()
  declare teacherId: number | null

  // Relation : 1 classe → 1 enseignant
  @belongsTo(() => Teacher)
  declare teacher: BelongsTo<typeof Teacher>
}
```

Les migrations et les seeders

Il faut bien sûr exprimer cette relation dans la migration.
Comme on travaille au niveau de la table MySQL, c'est simplement la clé étrangère `teacher_id` qui sera précisée.

Dans la migration des classes :

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

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

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

      // Relation : 1 classe → 1 enseignant
      table
        .integer('teacher_id')
        .unsigned()
        .references('id')
        .inTable('teachers')
        .onDelete('CASCADE')

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

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

Nous devons maintenant mettre à jour le seeder des classes :

```
// database/seeders/class_group_seeder.ts
import ClassGroup from '#models/classgroup'
import Teacher from '#models/teacher'
import { BaseSeeder } from '@adonisjs/lucid/seeders'

export default class ClassGroupSeeder extends BaseSeeder {
  public async run() {
    // Récupération des enseignants
    const teachers = await Teacher.all()

    await ClassGroup.createMany([
      { name: 'CIN1A', teacherId: teachers[0].id },
      { name: 'CIN1B', teacherId: teachers[0].id },
      { name: 'CIN1C', teacherId: teachers[1].id },
      { name: 'FID1', teacherId: teachers[1].id },
      { name: 'FID2', teacherId: teachers[2].id },
      { name: 'CID2A', teacherId: teachers[2].id },
      { name: 'CID2B', teacherId: teachers[3].id },
```

```
      ])
    }
  }
```

Nous réinitialisons la db afin de prendre en compte les changements effecutés.

```
node ace migration:fresh --seed
```

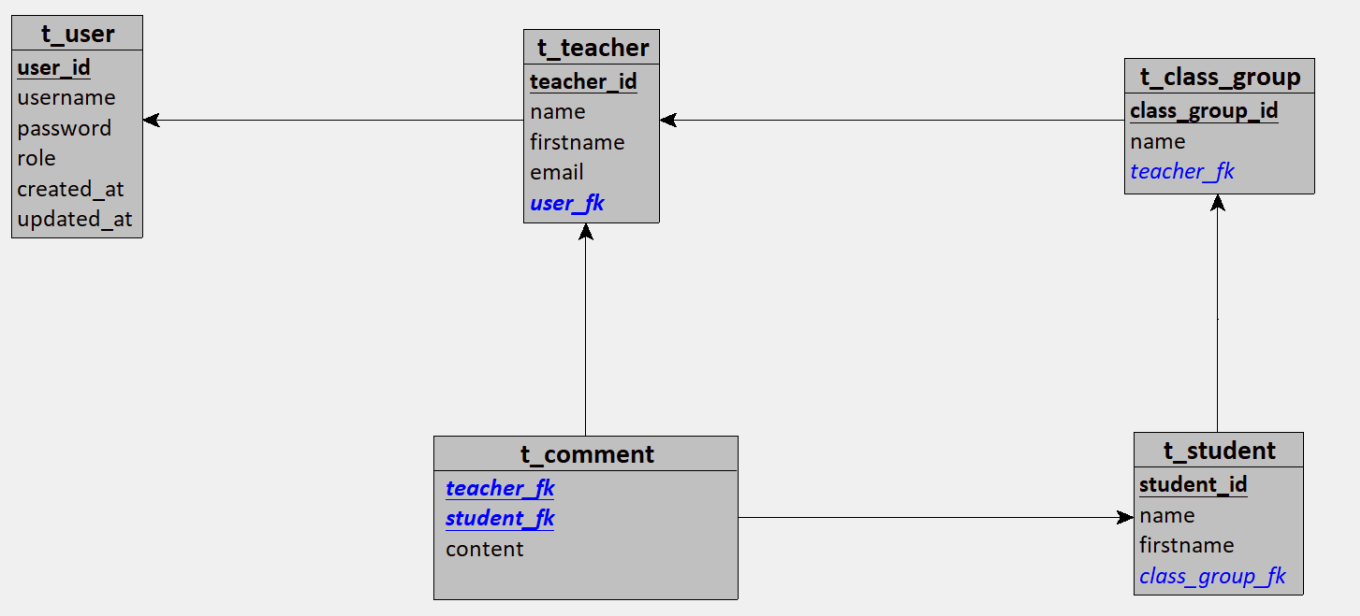
```
api-suivi-eleves-step-by-step (main)
$ node ace migration:fresh --seed
[ success ] Dropped tables successfully
[ info ] Upgrading migrations version from "1" to "2"
❯ migrated database/migrations/1746783669447_create_teachers_table
❯ migrated database/migrations/1746783673192_create_class_groups_table
❯ migrated database/migrations/1750316304397_create_users_table
❯ migrated database/migrations/1750316304401_create_access_tokens_table
❯ migrated database/migrations/1750316809384_create_students_table

Migrated in 1.36 s
❯ error     database/seeders/class_group_seeder
  Cannot read properties of undefined (reading 'id')
❯ completed database/seeders/student_seeder
❯ completed database/seeders/teacher_seeder
```

> ⚠ QUESTION
> Pourquoi cette erreur ?

Maintenant que nous mettons en place les relations entre les tables, un ordre de création des tables et des seeders doit être définit.
En effet, je ne peux, dans notre cas, pas créer de classe si je n'ai pas d'enseignants.
Je dois donc crééer les enseignants, puis les classes.

Nous allons donc définir un ordre logique dans la création des tables et des seeders.

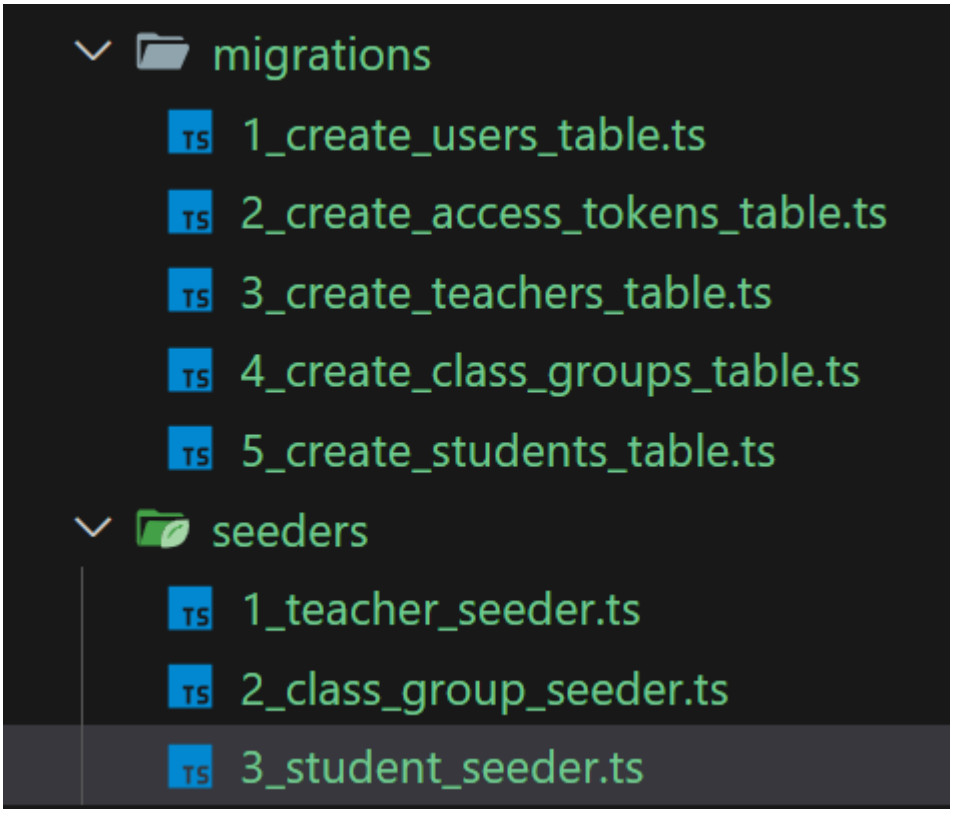
Prenons le MLD pour avoir une vision claire des clés étrangères :



Grâce au MLD, il est assez évident que nous devons respecter l'ordre suivant :

1. t_user
2. t_teacher
3. t_class_group
4. t_student
5. t_comment

Respectons cet ordre en préfixant les noms des migrations et des seeders :



La commande ci-dessous devrait maintenant passer sans problème :

```
node ace migration:fresh --seed
```

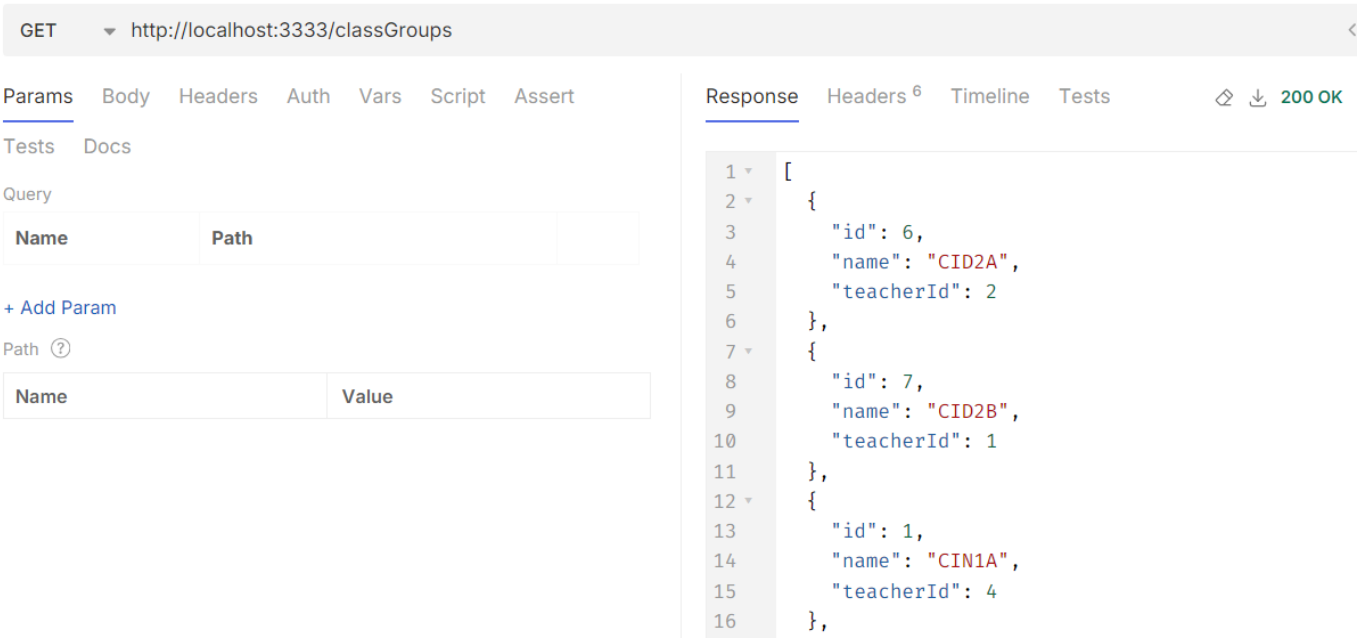
```
$ node ace migration:fresh --seed
[ success ] Dropped tables successfully
[ info ] Upgrading migrations version from "1" to "2"
> migrated database/migrations/1_create_users_table
> migrated database/migrations/2_create_access_tokens_table
> migrated database/migrations/3_create_teachers_table
> migrated database/migrations/4_create_class_groups_table
> migrated database/migrations/5_create_students_table

Migrated in 1.71 s
> completed database/seeders/1_teacher_seeder
> completed database/seeders/2_class_group_seeder
> completed database/seeders/3_student_seeder
```

Les contrôleurs

Route GET /classGroups

Maintenant, si nous testons à nouveau notre API, nous ne pouvons pas être complètement satisfait.

Par exemple :



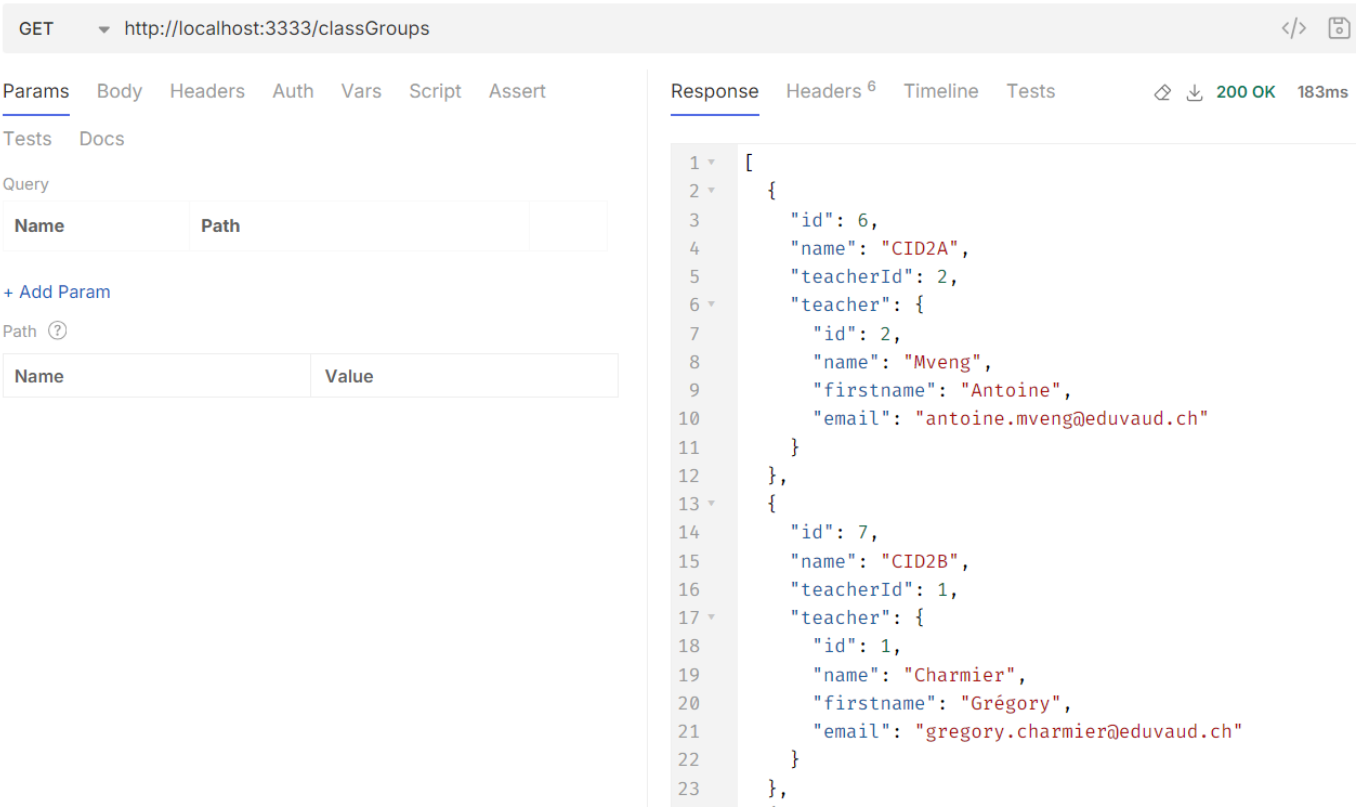
Nous aimerions bien avoir les informations du maitre de classe dans le json des classes.

> QUESTION :
> Comment faire cela ?

Avec adonis, c'est super simple !
Il suffit de précharger les données grâce au relation que l'on a défini dans le modèle.

En d'autre terme, il suffit de faire un `preload('teacher')` lors de la construction de la requête.

```
const classGroups = await ClassGroup.query().preload('teacher').orderBy('name',
'asc')
```



Route GET /classGroups/1

Pour la méthode show, nous devons un peu modifier la construction de la requête :

```
// Récupération d'une classe par son ID
const classGroup = await ClassGroup.query().preload('teacher').where('id',
params.id).firstOrFail()
```

Route POST /classGroups/

Mise à jour du validator pour ajouer le champ optionnel `teacherId`

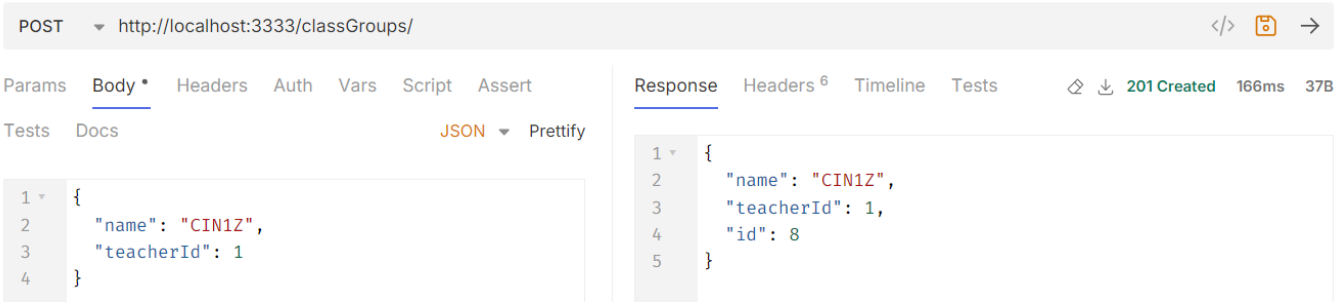
```
import vine from '@vinejs/vine'

const classGroupValidator = vine.compile(
  vine.object({
    name: vine.string().minLength(2).maxLength(255),
    teacherId: vine.number().optional(),
  })
)
```

```
export { classGroupValidator }
```

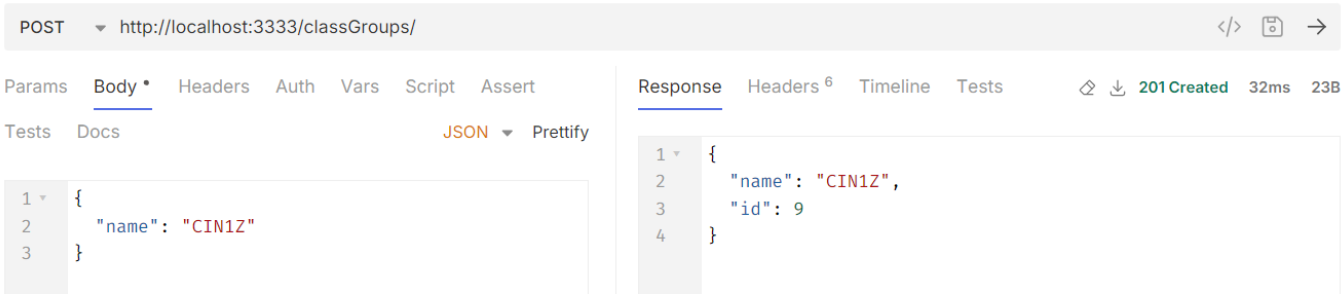
Pour le contrôleur, il y a très peu de chose à faire :

```
  /**
   * Créer une nouvelle classe
   */
  async store({ request, response }: HttpContext) {
    // Récupération des données envoyées par le client et validation des données
    const { name, teacherId } = await request.validateUsing(classGroupValidator)

    // Création d'une nouvelle classe avec les données validées
    const classGroup = await ClassGroup.create({ name, teacherId })

    // On utilise `response.created` pour retourner un code HTTP 201 avec les
données de la classe créée
    return response.created(classGroup)
  }
```

On teste l'ajout d'une classe avec un maître de classe :



On teste l'ajout d'une classe sans maître de classe :



Route PUT /classGroups/1

> EXERCICE : A vous de mettre à jour le code pour l'update.

Route DELETE /classGroups/1

Cette route n'est pas impactée par l'ajout de `teacherId`.

Conclusion

Nous mettrons en place la relation Classe - Elève dans la prochaine étape.