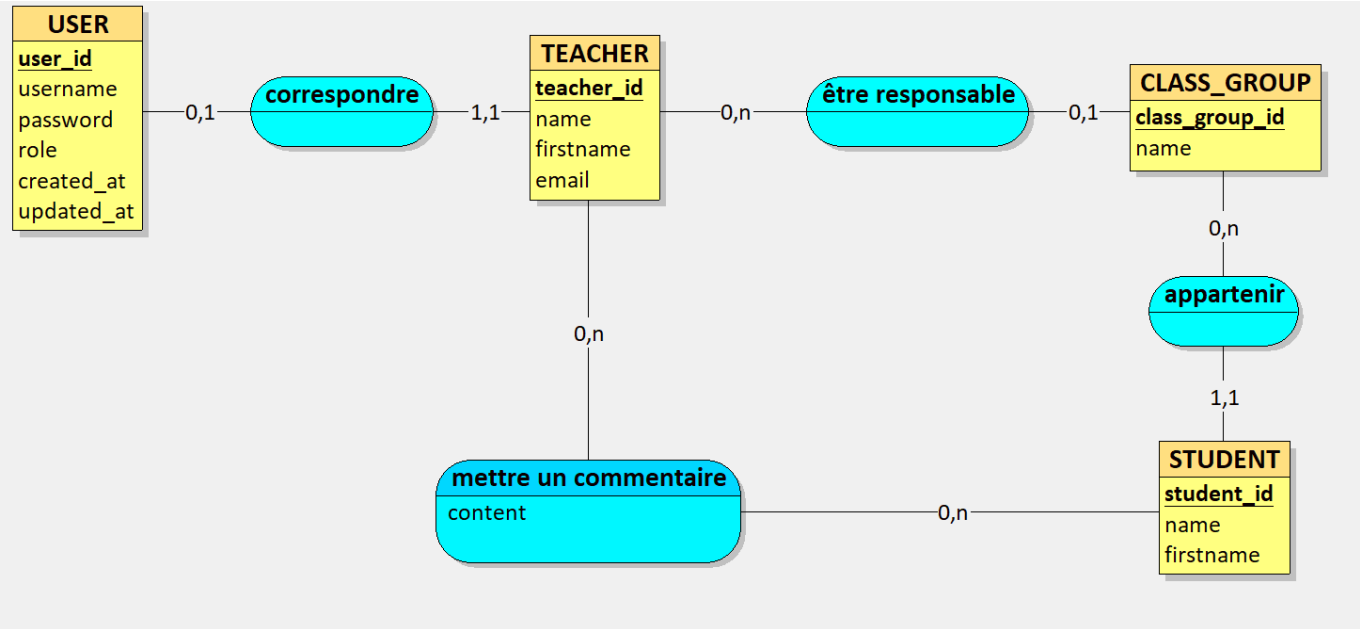
Backend pour le suivi des élèves - Step11

Dans cette étape, nous souhaitons mettre en place les commentaires.

L'idée est que pour gérer le suivi des élèves, un enseignant peut mettre un commentaire sur un élève.



Comme nous pouvons le voir :

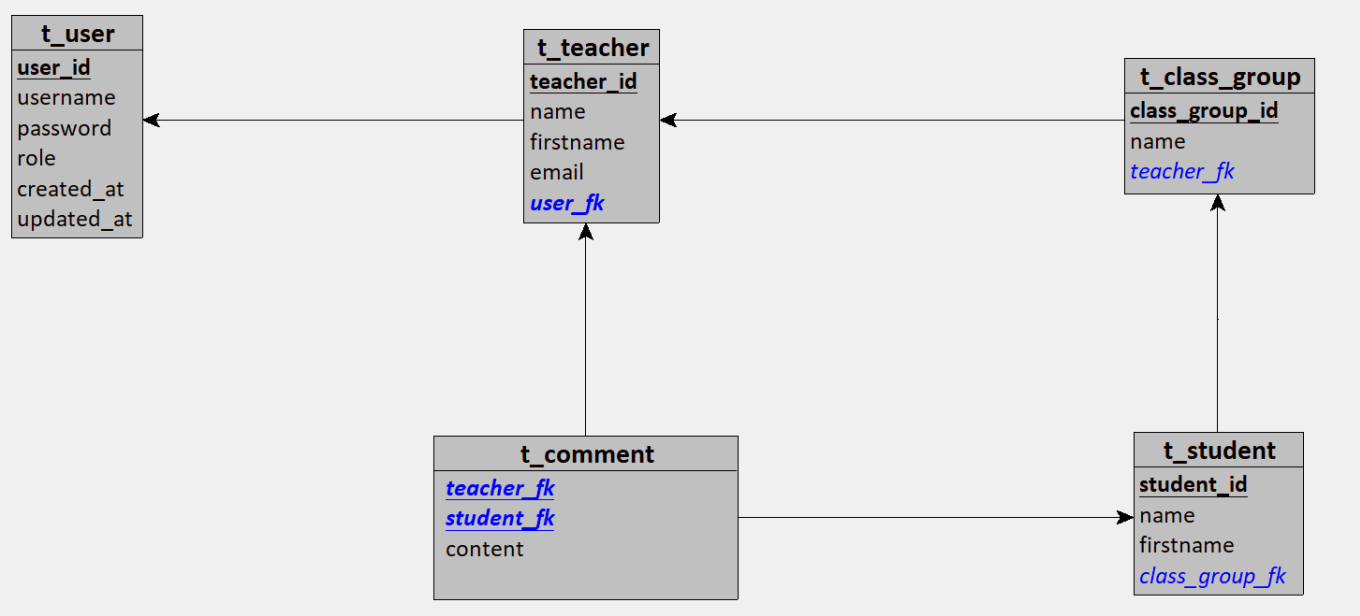
- Un enseignant peut mettre plusieurs commentaires pour un élève.
- Un élève peut recevoir plusieurs commentaires.

On appelle cela une relation N-N.

Techniquement, dans la DB, cette relation N-N est mise en place par deux relations 1-N :

- `Teacher` 1-N `Comment`
- `Student` 1-N `Comment`

En effet, nous avons les 2 clés étrangères dans la table `t_comment`



Dit autrement, la relation `Teacher` ↔ `Student` devient N-N à travers `Comment`.

Création d'un modèle et d'une migration

```
node ace make:model Comment -m
```

On en profite cette fois pour gérer les relations avec les autres tables :

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

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

  @column()
  declare content: string

  @column()
  declare studentId: number

  @column()
  declare teacherId: number

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

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

```
  // Relation : 1 commentaire → 1 élève
  @belongsTo(() => Student)
  declare student: BelongsTo<typeof Student>

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

On préfixe le nom du fichier de la migration par `6_` (`6_create_comments_table.ts`) et on renseigne les champs et relations :

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

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

  async up() {
    this.schema.createTable(this.tableName, (table) => {
      table.increments('id')
      table.text('content').notNullable() // Contenu du commentaire

      // Relation : 1 commentaire → 1 élève
      table
        .integer('student_id')
        .unsigned()
        .notNullable() // Important : un commentaire doit toujours être lié à un
élève
        .references('id')
        .inTable('students')
        .onDelete('CASCADE') // si l'élève est supprimé, le commentaire sera
supprimé également

      // Relation : 1 commentaire → 1 enseignant
      table
        .integer('teacher_id')
        .unsigned()
        .nullable() // Important : un commentaire peut ne pas être lié à un
enseignant
        .references('id')
        .inTable('teachers')
        .onDelete('SET NULL') // si l'enseignant est supprimé, le commentaire
restera en DB mais sans être lié à un enseignant

      table.timestamp('created_at', { useTz: true })
      table.timestamp('updated_at', { useTz: true })
    })
  }

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

```
  }
}
```

Pour le modèle `Student` :

```
// Relation : 1 élève -> N commentaires
@hasMany(() => Comment)
declare comments: HasMany<typeof Comment>
```

Pour le modèle `Teacher` :

```
// Relation : 1 enseignant → N commentaires
@hasMany(() => Comment)
declare comments: HasMany<typeof Comment>
```

Création d'un seeder

```
node ace make:seeder 4_comment_seeder
```

Nous allons simplement créer 2 commentaires pour un élève afin de pouvoir tester notre API REST.

```
// database/seeders/CommentSeeder.ts
import Student from '#models/student'
import Teacher from '#models/teacher'
import Comment from '#models/comment'
import { BaseSeeder } from '@adonisjs/lucid/seeders'

export default class CommentSeeder extends BaseSeeder {
  public async run() {
    // On récupère un élève et un enseignant existants
    const student = await Student.findOrFail(1)
    const teacher = await Teacher.findOrFail(1)

    await Comment.createMany([
      {
        content: 'Très bon travail sur le dernier trimestre.',
        studentId: student.id,
        teacherId: teacher.id,
      },
      {
        content: 'En constante amélioration !',
        studentId: student.id,
        teacherId: teacher.id,
      },
```

```
    ])
  }
}
```

Mise à jour de la DB

Nous pouvons réinitialiser la DB avec la commande

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

Routes imbriquées

Pour accéder à tous les commentaires d'un élève, nous allons définir la route `/students/1/comments`.
Il s'agit d'une route imbriquée qui a pour but de retourner tous les commentaires de l'élève dont l'id vaut 1.

```
/*
|--------------------------------------------------------------------------
| 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'
import TeachersController from '#controllers/teachers_controller'
import ClassGroupsController from '#controllers/class_groups_controller'
import CommentsController from '#controllers/comments_controller'

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

// Routes pour le CRUD /students
router.resource('students', StudentsController).apiOnly()

// Routes imbriquées sur les commentaires
// pour le CRUD /students/:student_id/comments
router
  .group(() => {
    router.resource('comments', CommentsController).apiOnly()
  })
  .prefix('students/:student_id')

// Routes pour le CRUD /teachers
router.resource('teachers', TeachersController).apiOnly()
```

```
// Routes pour le CRUD /classGroup
router.resource('classGroups', ClassGroupsController).apiOnly()
`
```

On peut voir l'ensemble des routes avec la commande :

```
node ace list:routes
```

Maintenant que les modifications de la DB sont faites et que les routes sont définies, nous pouvons nous intéresser au contrôleur.
Dans le contrôleur, nous aurons besoin de valider les données transmises pour l'ajout ou la modifiction d'un commentaire.
Nous allons donc commencer par créer ce validateur.

Validateur

Création du validateur :

```
node ace make:validator comment
```

Définition d'une règle pour le champ `content` :

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

const commentValidator = vine.compile(
  vine.object({
    content: vine.string().minLength(2),
  })
)

export { commentValidator }
```

Contrôleur `CommentsController`

On créé le contrôleur :

```
node ace make:controller comments -r
```

Voilà le code pour ce nouveau contrôleur :

```
import Comment from '#models/comment'
import Student from '#models/student'
```

```
import { commentValidator } from '#validators/comment'
import type { HttpContext } from '@adonisjs/core/http'

export default class CommentsController {
  /**
   * Afficher tous les commentaires de l'élève student_id
   */
  async index({ params, response }: HttpContext) {
    // Récupération de l'élève dont l'id est en paramètre
    const student = await Student.findOrFail(params.student_id)

    // Chargement des commentaires et
    // pour chaque commentaire, on précharge l'enseignant
    await student.load('comments', (query) => {
      query.preload('teacher')
    })

    return response.ok(student.comments)
  }

  /**
   * Ajouter un nouveau commentaire à l'élève student_id
   */
  async store({ params, request, response }: HttpContext) {
    // Récupération des données envoyées par le client et validation des données
    const { content } = await request.validateUsing(commentValidator)

    // Pour l'instant, on fixe "en dur" l'id de l'enseignant
    // Lorsque l'on va mettre en place l'authentification,
    // on pourra récupérer l'id de l'enseignant connecté
    const teacherId = 1

    // Création du commentaire lié à l'élève
    const comment = await Comment.create({
      content,
      studentId: params.student_id,
      teacherId,
    })

    return response.created(comment)
  }

  /**
   * Afficher un commentaire de l'élève student_id
   */
  async show({ params, response }: HttpContext) {
    // Récupère le commentaire
    // Vérifie que le commentaire appartient à l'élève
    const comment = await Comment.query()
      .where('id', params.id)
      .where('student_id', params.student_id)
      .preload('teacher')
      .firstOrFail()
```

```
    return response.ok(comment)
  }

  /**
   * Mettre à jour le commentaire de l'élève student_id
   */
  async update({ params, request, response }: HttpContext) {
    // Récupération des données envoyées par le client et validation des données
    const { content } = await request.validateUsing(commentValidator)

    // Vérifie que le commentaire appartient bien à l'élève
    const comment = await Comment.query()
      .where('id', params.id)
      .where('student_id', params.student_id)
      .firstOrFail()

    // Mise à jour
    comment.content = content
    await comment.save()

    // Réponse 200 OK avec le commentaire mis à jour
    return response.ok(comment)
  }

  /**
   * Supprimer le commentaire de l'élève student_id
   */
  async destroy({ params, response }: HttpContext) {
    const comment = await Comment.query()
      .where('id', params.id)
      .where('student_id', params.student_id)
      .firstOrFail()

    // Suppression du commentaire
    await comment.delete()

    // On utilise `response.noContent` pour retourner un code HTTP 204 sans
contenu
    return response.noContent()
  }
}
```

Le code ci-dessous ne devrait pas vous poser de problème car nous avons déjà réalisé quelque chose de similaire dans les autres contrôleurs.

Par contre, dans la méthode `index`, nous avons le code suivant qui demande quelques explications :

```
// Chargement des commentaires et
// pour chaque commentaire, on précharge l'enseignant
await student.load('comments', (query) => {
```

```
    query.preload('teacher')
  })
```

Le but de cette méthode est de retourner tous les commentaires d'un élève.
Or dans le json retourné, il est important d'avoir, pour chaque commentaire, l'enseignant qui a écrit ce commentaire.

Donc

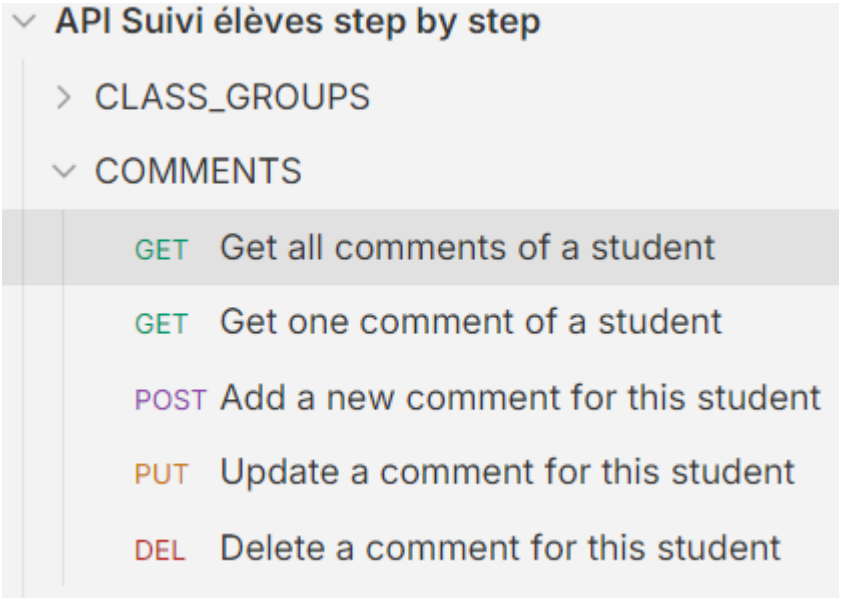
```
await student.load('comments') // charge maintenant les commentaires
```

et pour chaque commentaire, on utilise la fonction de callback ci-dessous pour charger les enseignants.

```
(query) => {
  query.preload('teacher')
}
```

Tester nos nouvelles routes

A vous de tester les nouvelles routes à l'aide de Bruno !




````
````