Backend pour le suivi des élèves - Step14

Objectif

Dans cette étape, nous souhaitons mettre en place **la pagination** sur la route `GET /students` afin de retourner les élèves de manière paginée, avec possibilité de tri, recherche et filtrage.

Pourquoi ?

Sans pagination, une route peut retourner des centaines d'élèves, ce qui :

- alourdit la réponse HTTP,
- complique l'affichage côté client,
- empêche un bon référencement des ressources.

Avec la pagination, on respecte le **niveau 2 du modèle de maturité de Richardson**, en exposant proprement les ressources.

Création du validateur de requête

On commence par créer un validateur `getStudentsQueryValidator.ts` :

```
const getStudentsQueryValidator = vine.compile(
  vine.object({
    page: vine.number().min(1).optional(),
    limit: vine.number().min(1).max(100).optional(),
    sort: vine.string().in(['name', 'firstname', 'created_at']).optional(), //
trier par nom, prénom ou date de création
    order: vine.string().in(['asc', 'desc']).optional(), // ordre de tri
    classGroupId: vine.number().optional(),
    search: vine.string().trim().minLength(1).optional(),
  })
)
```

Contrôleur `StudentsController`

Voici la méthode `index()` mise à jour dans `StudentsController.ts` :

```
  /**
   * Afficher la liste des élèves
   */
  async index({ response, request }: HttpContext) {
    // Récupère les paramètres de pagination de la requête
    const {
      page = 1,
      limit = 10,
      sort = 'name',
```

```
      order = 'asc',
      classGroupId, // ID de la classe pour filtrer les étudiants
      search,
    } = await request.validateUsing(getStudentsQueryValidator)

    // Récupère tous les étudiants avec leurs classes et commentaires
    // Les étudiants sont triés par nom et prénom, puis paginés
    // A noter que await n'est pas présent
    const query = Student.query().preload('classGroup').preload('comments')

    if (classGroupId) {
      query.where('class_group_id', classGroupId) // Filtre les étudiants par ID
de classe
    }

    // Recherche sur le nom et le prénom des étudiants
    if (search) {
      query.where((subQuery) => {
        subQuery.whereILike('name', `%${search}%`).orWhereILike('firstname',
`%${search}%`)
      })
    }

    // Tri des étudiants par le champ spécifié (sort) et l'ordre (asc ou desc)
    query.orderBy(sort, order as 'asc' | 'desc')

    // A noter que le await est nécessaire pour exécuter la requête
    // et volontairement omis précédemment pour éviter l'exécution prématurée
    const students = await query.paginate(page, limit) // Pagination des résultats

    // affiche correctement le chemin (/students),
    students.baseUrl('/students')

    // conserve les paramètres (recherche, tri, etc.).
    students.queryString({ page, limit, sort, order, classGroupId, search })

    // On utilise `response.ok` pour retourner un code HTTP 200 avec les données
des élèves
    return response.ok(students)
  }
```

Exemple de réponse JSON


```
{
  "meta": {
    "total": 59,
    "perPage": 10,
    "currentPage": 1,
    "lastPage": 6,
    "firstPageUrl": "/students?page=1&limit=10&sort=name&order=asc",
    "lastPageUrl": "/students?page=6&limit=10&sort=name&order=asc",
```

```
    "nextPageUrl": "/students?page=2&limit=10&sort=name&order=asc",
    "previousPageUrl": null
  },
  "data": [
    { "id": 1, "name": "Durand", "firstname": "Marie", ... },
    { "id": 2, "name": "Dupont", "firstname": "Luc", ... }
  ]
}
```

Conclusion

Cette pagination respecte les bonnes pratiques REST, garde une URL claire (`/students`), tout en permettant le tri, la recherche et le filtrage sur les classes. On reste ainsi fidèle à l'esprit du niveau 2 du modèle de Richardson.