Backend pour le suivi des élèves - Step3

Nous allons continuer à mettre en place le CRUD des élèves.

Ajouter un nouvel élève

```
  /**
   * Ajouter d'un élève
   */
  async store({ request }: HttpContext) {
    // Récupération des données envoyées par le client
    // On utilise `request.only` pour ne récupérer que les champs nécessaires
    const student = request.only(['name', 'firstname'])

    // Création d'un nouvel élève avec les données récupérées
    return await Student.create(student)
  }
```

Question : A quelle route correspond la méthode **store** ?

> ⚠ Astuce Pour voir la liste des routes, nous devons exécuter la commande offerte par le CLI à savoir `node ace liste:routes`

```
$ node ace list:routes

METHOD ROUTE ..................................................... HANDLER MIDDLEWARE
GET    /test ..................................................... closure
GET    /students (students.index) ............... StudentsController.index
POST   /students (students.store) ............... StudentsController.store
GET    /students/:id (students.show) ............ StudentsController.show
PUT    /students/:id (students.update) ........ StudentsController.update
PATCH  /students/:id (students.update) ........ StudentsController.update
DELETE /students/:id (students.destroy) ...... StudentsController.destroy
```

Mettre à jour un élève (PATCH ou PUT)

```
  /**
   * Mettre à jour un élève
   *
   * Cette méthode est utilisée pour mettre à jour les informations d'un élève que
ce soit :
   * - pour une modification complète (PUT)
   * - ou une modification partielle (PATCH)
   */
  async update({ params, request }: HttpContext) {
    // Récupération des données
    const data = request.only(['name', 'firstname'])
```

```
    // Vérification de l'existence de l'élève
    const student = await Student.findOrFail(params.id)

    // Mise à jour des données de l'élève
    student.merge(data)

    // Sauvegarde des modifications
    await student.save()

    // Retour le json de l'élève mis à jour
    return student
  }
```

Supprimer un élève

```
  /**
   * Supprimer un élève
   */
  async destroy({ params }: HttpContext) {
    // Vérification de l'existence de l'élève
    const student = await Student.findOrFail(params.id)

    // Suppression de l'élève
    return await student.delete()
  }
```

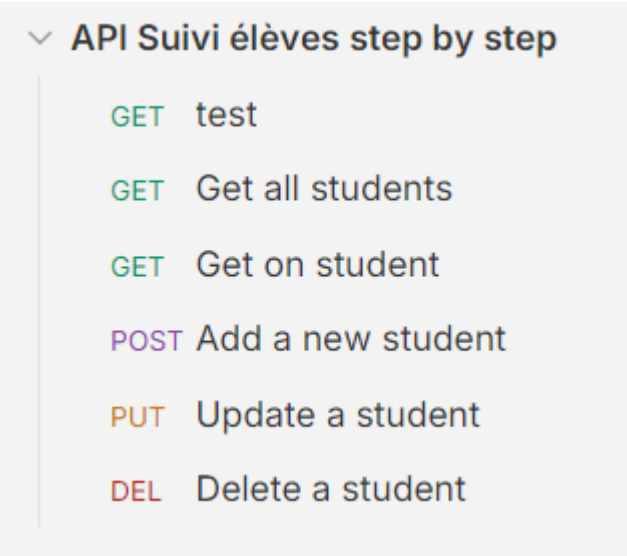
Tester nos routes

Pour tester nos routes, nous ne pouvons plus utiliser un navigateur (car il peut seulement faire des `GET` HTTP).

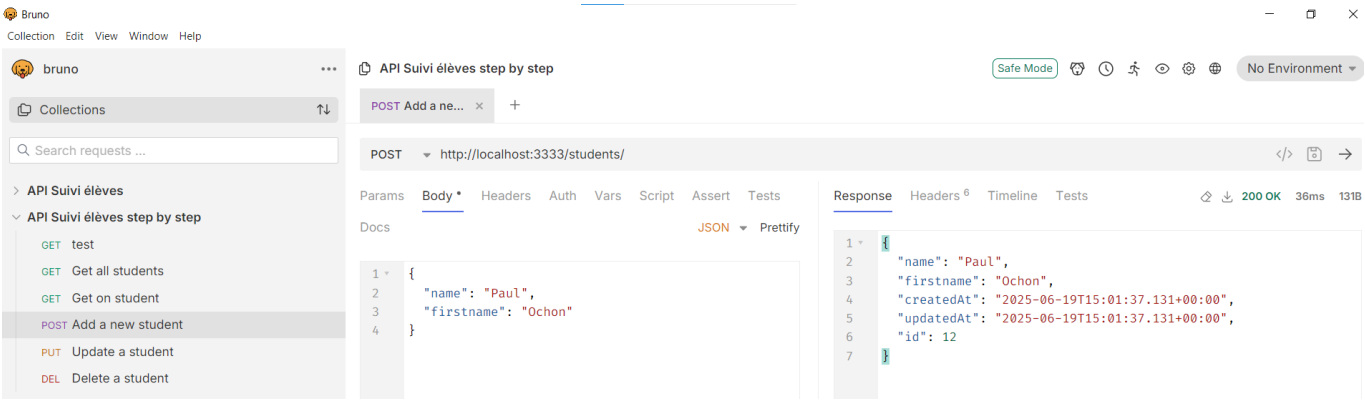
En effet, nous avons besoin de faire des `POST`, `PUT` ou `DELETE` HTTP.

Pour cela, nous pouvons utiliser des outils comme postman, insomnia ou encore Bruno.
Ce dernier devrait être déjà installé sur votre PC et c'est celui que nous allons utiliser pendant ce cours.

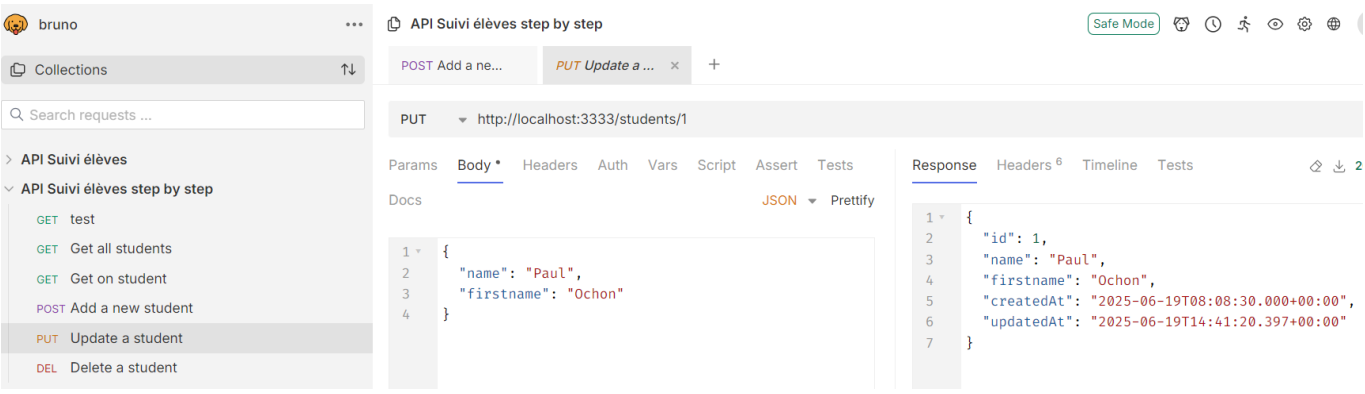
Voilà les appels HTTP dans l'interface de l'application Bruno:



Exemple : POST /students



Exemple : PUT /students/1



Exemple : DELETE /students/1