Backend pour le suivi des élèves - Step12

Dans cette étape, nous souhaitons mettre en place l'authentification.

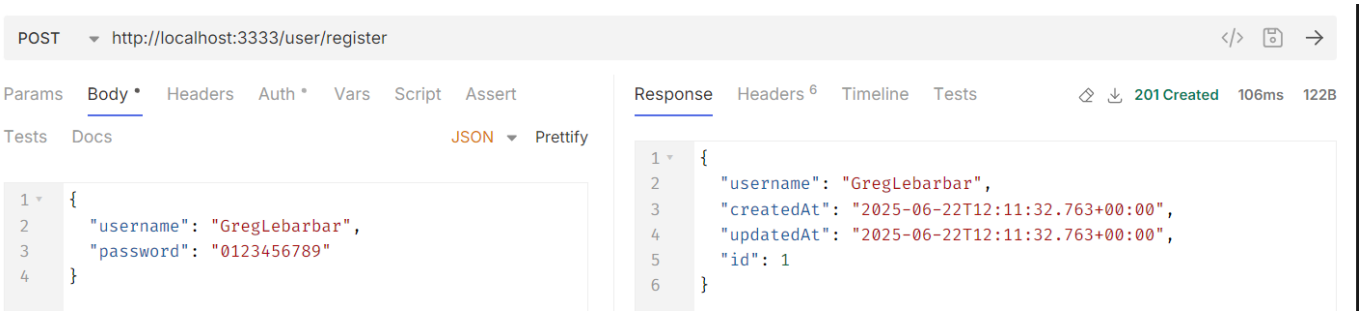
> **Remarque :**
> N'oublions pas que lorsque nous avons créé le projet à l'étape 1, nous avons précisé l'option `--auth-guard=access_tokens`.
> Cette option a permis de configurer l'authentification basée sur tokens persistés en base de données.

Introduction

Quelles sont les différentes étapes pour qu'un utilisateur puisse s'authentifier.

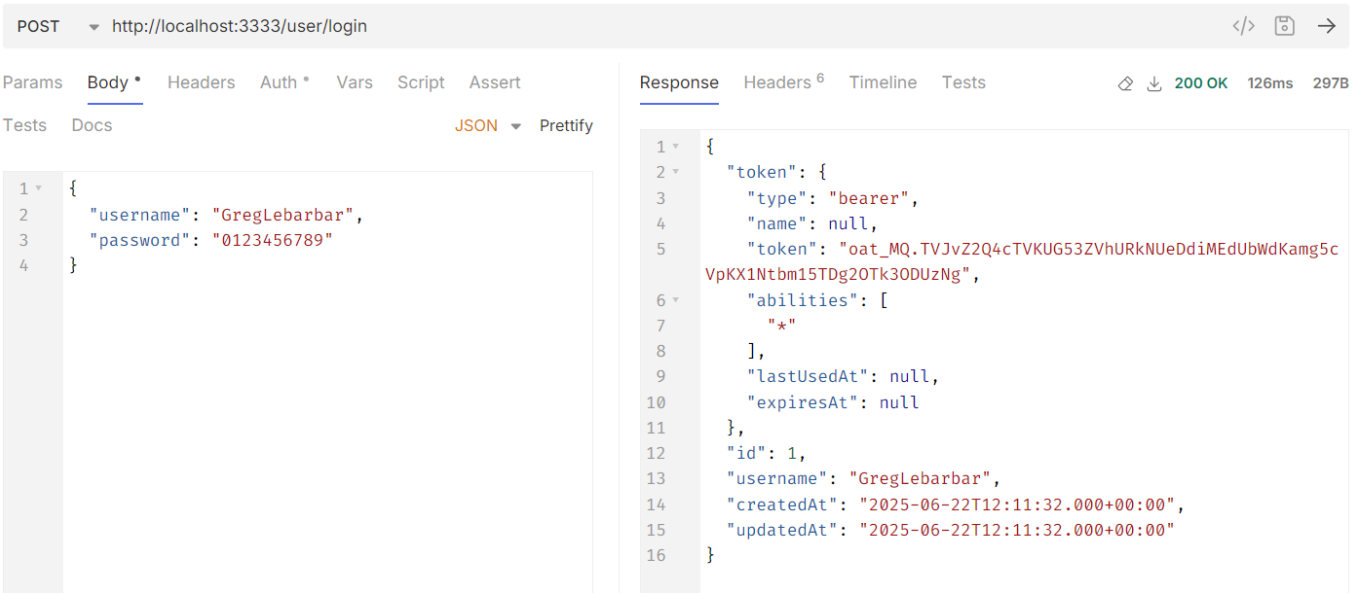
Register

Pour commencer, l'utilisateur doit s'inscrire en précisant son nom d'utilisateur (`username`) et son mot de passe (`password`).
Pour cela, il doit faire un HTTP POST /user/register en fournissant les données en json.



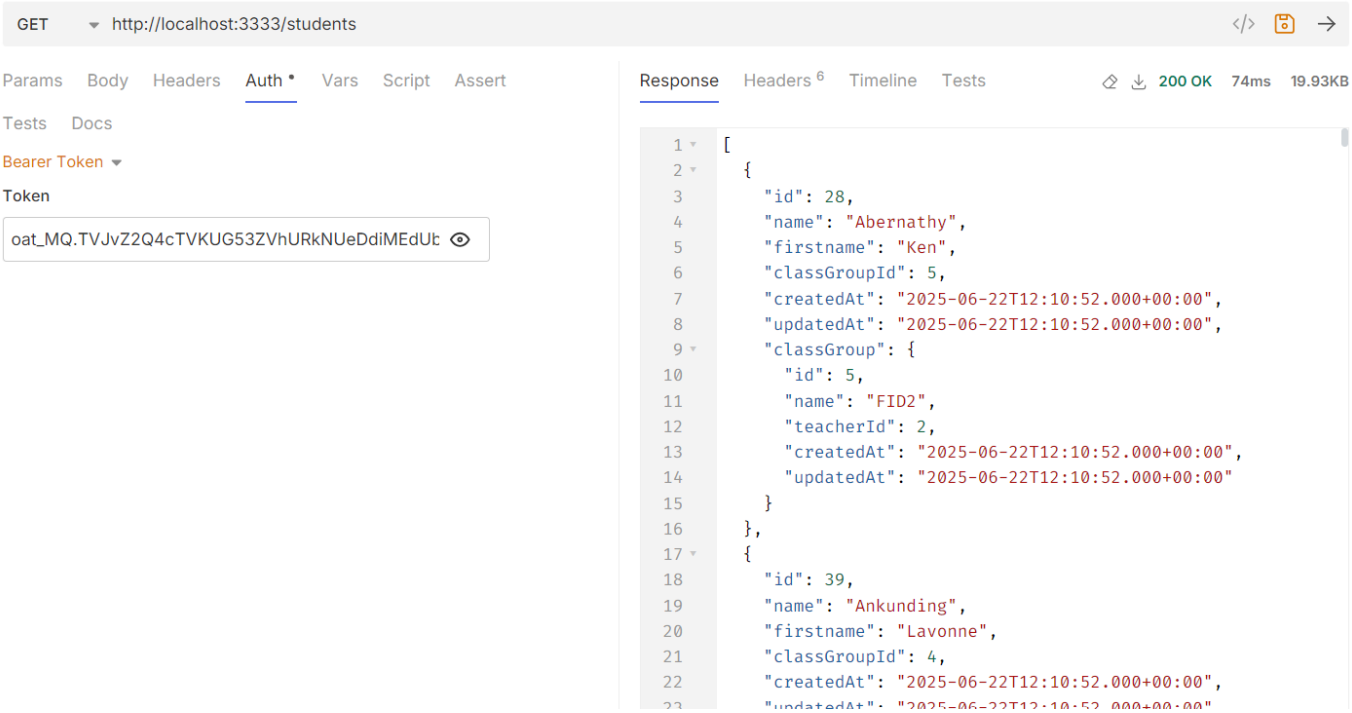
Login

Ensuite, l'utilisateur peut se connecter toujours à l'aide de son nom d'utilisateur et mot de passe.
En retour, il obtiendra son jeton d'authentification ou `token OAT`.



Chaque appel HTTP doit fournir le token

A partir de là, à chaque fois que l'utilisateur souhaite faire un appel à l'API REST, il devra fournir son jeton.



Logout

Il devra pouvoir également se déconnecter et là aussi, il devra utiliser son jeton.

Modèle et migration `User`

Nous avons déjà un modèle `User` et une migration.
Ceci est dû à l'option `--auth-guard=access_tokens`.

Voilà le modèle `User` après nos modifications :

```
import { DateTime } from 'luxon'
import hash from '@adonisjs/core/services/hash'
import { compose } from '@adonisjs/core/helpers'
import { BaseModel, column, hasOne } from '@adonisjs/lucid/orm'
import { withAuthFinder } from '@adonisjs/auth/mixins/lucid'
import { DbAccessTokensProvider } from '@adonisjs/auth/access_tokens'
import Teacher from './teacher.js'
import type { HasOne } from '@adonisjs/lucid/types/relations'
```

```
const AuthFinder = withAuthFinder(() => hash.use('scrypt'), {
  uids: ['username'], // ATTENTION à bien modifier ICI aussi.
  passwordColumnName: 'password',
})

export default class User extends compose(BaseModel, AuthFinder) {
  @column({ isPrimary: true })
  declare id: number

  @column()
  declare username: string

  @column({ serializeAs: null })
  declare password: string

  @column()
  declare role: string // Ce champ sera utilisé dans la prochaine étape pour la
gestion des rôles.

  // Relation : 1 Utilisateur → 1 Enseignant
  @hasOne(() => Teacher)
  declare teacher: HasOne<typeof Teacher>

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

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

  static accessTokens = DbAccessTokensProvider.forModel(User)
}
```

A noter l'utilisation de `@hasOne` pour exprimer la relation 1 - 1 entre `User` et `Teacher`

Voila la migration correspondante :

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

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

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

      table.string('username').notNullable().unique()
      table.string('password').notNullable()
      table.string('role').notNullable().defaultTo('teacher')

      table.timestamp('created_at').notNullable()
      table.timestamp('updated_at').nullable()
```

```
      })
    }

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

Modèle et migration Teacher :

On commence par améliorer le modèle `Teacher`.

Modèle Teacher :

```
import { BaseModel, belongsTo, column, hasMany } from '@adonisjs/lucid/orm'
import ClassGroup from './classgroup.js'
import type { BelongsTo, HasMany } from '@adonisjs/lucid/types/relations'
import Comment from './comment.js'
import { DateTime } from 'luxon'
import User from './user.js'

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

  @column()
  declare name: string

  @column()
  declare firstname: string

  @column()
  declare email: string

  @column()
  declare userId: number

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

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

  // Relation : 1 enseignant → N classes
  @hasMany(() => ClassGroup)
  declare classGroups: HasMany<typeof ClassGroup>

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

```
  // Relation : 1 enseignant → 1 utilisateur
  @belongsTo(() => User)
  declare user: BelongsTo<typeof User>
}
```

A noter la différente entre `@hasOne` et `@belongsTo` :

- `@belongsTo` utilisé dans le modèle `Teacher` car il y a la clé étrangère `userId`.
- `@hasOne` utilisé dans le modèle `User` car il n'y a pas de clé étangère.

Ainsi la relation 1-1 est mise en place.

Migration Teacher :

On passe à la migration :

```
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')
      table.string('name').notNullable()
      table.string('firstname').notNullable()
      table.string('email').notNullable().unique()

      // Relation : 1 enseignant → 1 utilisateur
      table
        .integer('user_id')
        .unsigned()
        .references('id')
        .inTable('users')
        .onDelete('CASCADE')
        .unique() // Un enseignant est lié à un seul utilisateur
        .nullable() // Permettre aux enseignants de ne pas être liés à un
utilisateur (par exemple, pour les anciens enseignants)

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

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

On met à jour les tables MySQL :

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

Modification du contrôleur Teacher et du validateur

On améliore le validateur pour valider le `userId`.

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

const teacherValidator = vine.compile(
  vine.object({
    firstname: vine.string().minLength(2).maxLength(255),
    name: vine.string().minLength(2).maxLength(255),
    email: vine.string().email().maxLength(255),
    userId: vine.number().exists(async (db, value) => {
      const user = await db.from('users').where('id', value).first()
      // user est soit un objet (si trouvé), soit undefined (si non trouvé).
      // Explication du !!user :
      // Si user est un objet → !!user devient true
      // Si user est undefined → !!user devient false
      return !!user
    }),
  })
)

export { teacherValidator }
```

Légère adaptation dans l'utilisation du validateur :

```
/**
 * Créer un nouvel enseignant
 */
async store({ request, response }: HttpContext) {
  // Récupération des données envoyées par le client et validation des données
  const { name, firstname, email, userId } = await
request.validateUsing(teacherValidator)

  // Création d'un nouvel enseignant avec les données validées
  const teacher = await Teacher.create({ name, firstname, email, userId })

  // On utilise `response.created` pour retourner un code HTTP 201 avec les
données de l'enseignant créé
  return response.created(teacher)
}
```

Validateurs

On créé le validateur `auth` via le CLI.

```
node ace make:validator auth
```

Nous avons besoin de 2 validateurs :

- un pour valider les données utilisateurs lors de l'enregistrement d'un utilisateur
- un autre pour valider les données utilisateurs lors du login

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

export const loginValidator = vine.compile(
  vine.object({
    username: vine.string().minLength(3).maxLength(32),
    password: vine.string().minLength(8).maxLength(512),
  })
)

export const registerValidator = vine.compile(
  vine.object({
    username: vine
      .string()
      .minLength(3)
      .maxLength(32)
      .unique(async (query, field) => {
        const user = await query.from('users').where('username', field).first()
        return !user
      }),
    password: vine.string().minLength(8).maxLength(512),
  })
)
```

Contrôleur `AuthController`

On créé le contrôleur `AuthController` via le CLI.

```
 node ace make:controller AuthController
```

Voilà le code du contrôleur `AuthController`.

```
import type { HttpContext } from '@adonisjs/core/http'
import { registerValidator, loginValidator } from '#validators/auth'
import User from '#models/user'

export default class AuthController {
  /**
   * Créer un token OAT après validation des données utilisateurs
   */
```

```
  async login({ request, response }: HttpContext) {
    // Validation du nom d'utilisateur et mot de passe
    const { username, password } = await request.validateUsing(loginValidator)

    // Vérification qu'un utilisateur existe avec ce nom d'utilisateur et ce mot
de passe
    const user = await User.verifyCredentials(username, password)

    // Génération d'un token OAT
    const token = await User.accessTokens.create(user)

    // Retourne le token et les infos utilisateurs
    return response.ok({
      token: token,
      ...user.serialize(),
    })
  }

  /**
   * Enregistre un utilisateur
   */
  async register({ request, response }: HttpContext) {
    // Validation des données utilisateurs
    const payload = await request.validateUsing(registerValidator)

    // Création de l'utilisateur
    const user = await User.create(payload)

    // Retourne les données utilisateurs
    return response.created(user)
  }

  /**
   * Supprime le token OAT de l'utilisateur connecté
   */
  async logout({ auth, response }: HttpContext) {
    // Récupère l'utilisateur connecté/authentifié
    const user = auth.getUserOrFail()

    // Récupère le token de l'utilisateur connecté
    const token = auth.user?.currentAccessToken.identifier

    // Si le token n'existe pas, retourne une erreur HTTP 400
    if (!token) {
      return response.badRequest({ message: 'Token not found' })
    }

    // Supprime le token
    await User.accessTokens.delete(user, token)

    // Confirme à l'utilisateur que le logout est un succès
    return response.ok({ message: 'Logged out' })
  }
}
```

Routes

Certaines routes doivent être protéger en demandant le jeton d'authentification.
Pour cela, on utilise le middleware d'authentification avec `use(middleware.auth())`

C'est le cas des routes pour gérer les élèves, les enseignants, les classes et les commentaires :

```
router
  .group(() => {
    // Routes pour le CRUD /students
    router.resource('students', StudentsController).apiOnly()

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

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

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

Mais également la route permettant à un utilisateur de se déconnecter :

```
// Routes pour l'authentification
router
  .group(() => {
    router.post('register', [AuthController, 'register'])
    router.post('login', [AuthController, 'login'])
    router.post('logout', [AuthController, 'logout']).use(middleware.auth())
  })
  .prefix('user')
```

Création d'un enseignant

Pour créer un enseignant, nous avons maintenant besoin d'un id utilisateur.
Donc nous devons fournir un id d'utilisateur pour créer un enseignant.

```
  /**
   * Créer un nouvel enseignant
```

```
   */
  async store({ request, response }: HttpContext) {
    // Récupération des données envoyées par le client et validation des données
    const { name, firstname, email, userId } = await
request.validateUsing(teacherValidator)

    // Création d'un nouvel enseignant avec les données validées
    const teacher = await Teacher.create({ name, firstname, email, userId })

    // On utilise `response.created` pour retourner un code HTTP 201 avec les
données de l'enseignant créé
    return response.created(teacher)
  }
```

Ajout de commentaire - Utilisateur connecté

Maintenant que l'authentification est mis en place, nous pouvons à l'aide de l'utilisateur connecté, récupérer l'id de l'enseignant afin de créer le commentaire.

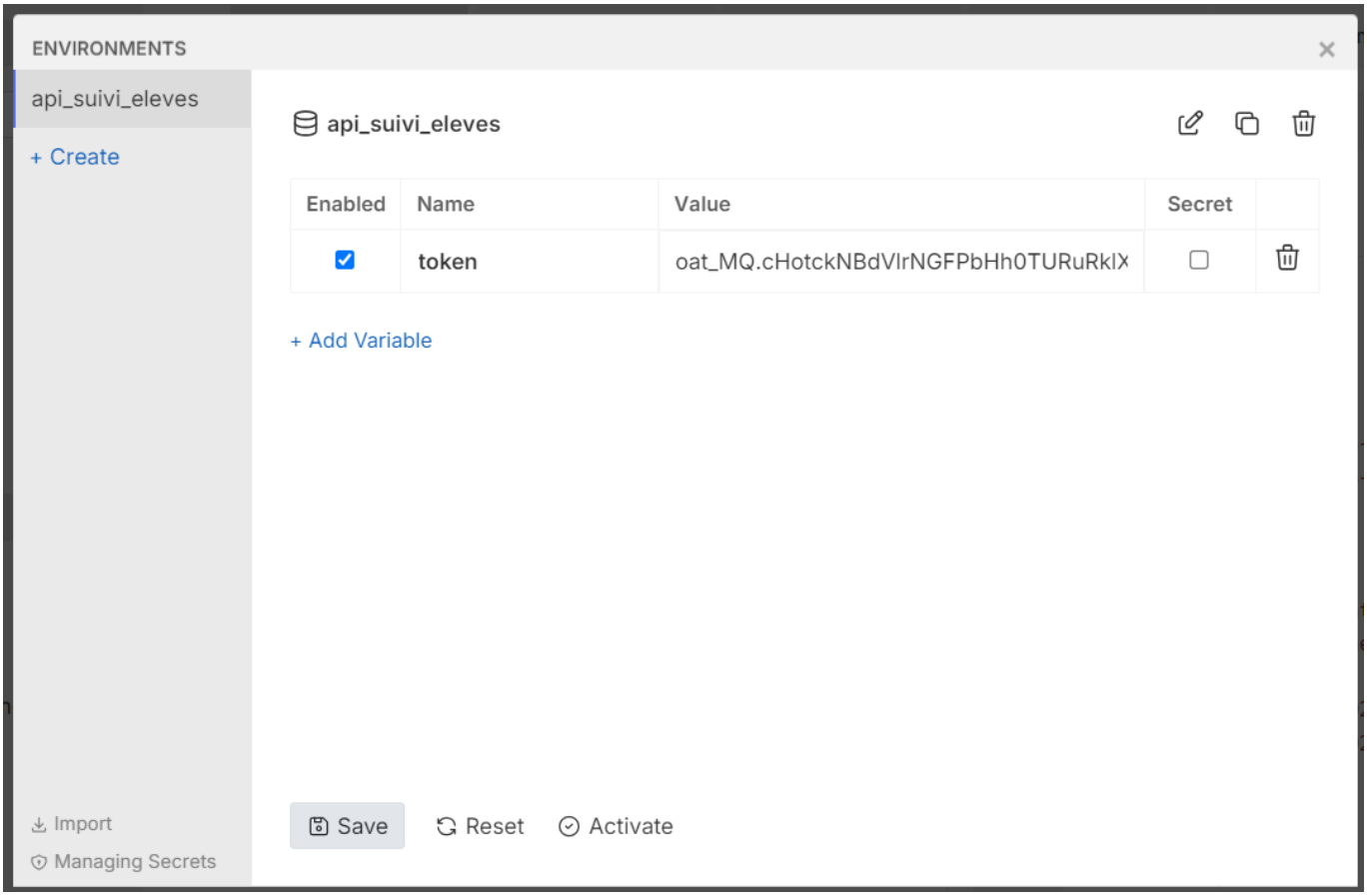
```
  /**
   * Ajouter un nouveau commentaire à l'élève student_id
   */
  async store({ params, request, response, auth }: HttpContext) {
    // Récupération des données envoyées par le client et validation des données
    const { content } = await request.validateUsing(commentValidator)

    // Récupération de l'utilisateur authentifié
    const user = auth.user!

    // Chargement de l'enseignant lié à cet utilisateur
    const teacher = await user.related('teacher').query().first()
    if (!teacher) {
      return response.badRequest({ message: 'Teacher not found' })
    }
    const teacherId = teacher.id

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

    // Réponse HTTP 201 avec le commentaire
    return response.created(comment)
  }
```

Bruno client

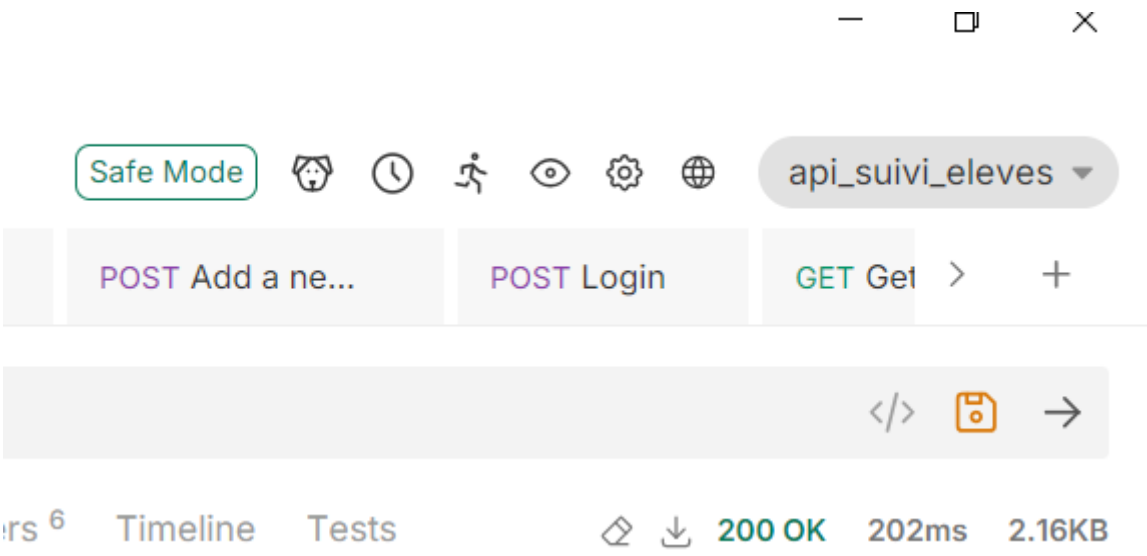
Comment gérer le token OAT pour les tests ?

En effet, nous devons fournir un token à chaque appel de l'API REST. Or, si nous voulons changer d'utilisateur facilement, il faut pouvoir changer de token. Mais cela peut s'avérer contraignant de devoir à chaque fois changer pour toutes les entrées misent en place dans l'outil Bruno.

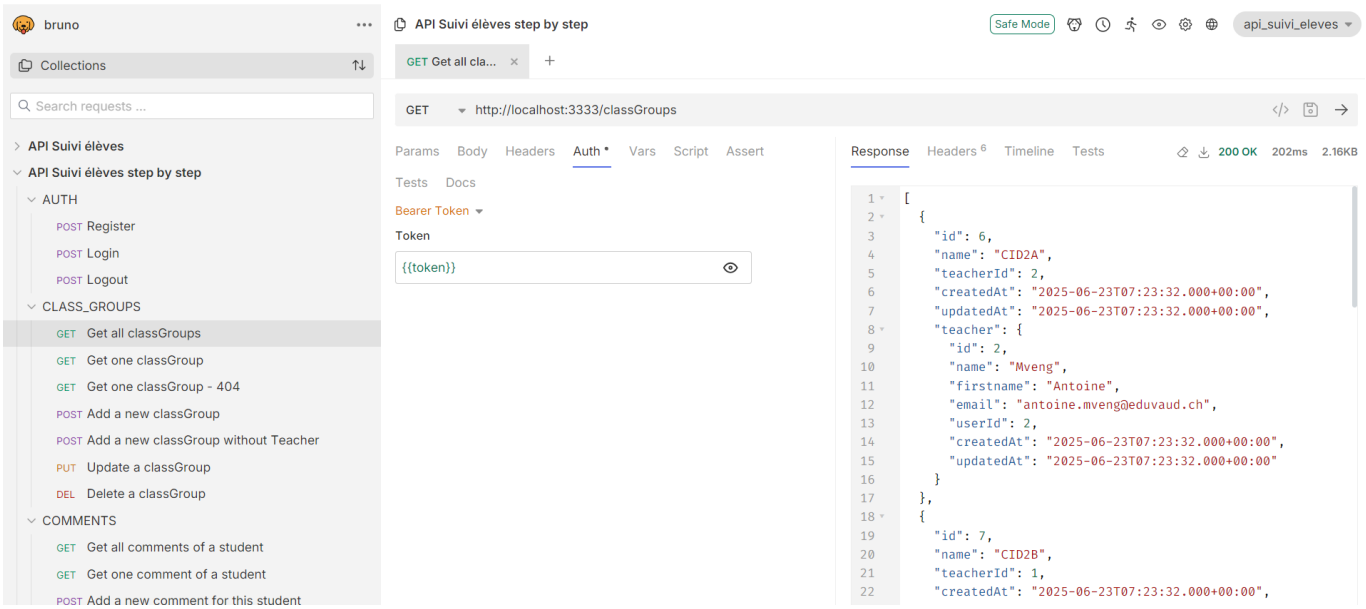
Pour cela, il est possible de créer un environnement (qui a pour nom `api_suivi_eleves`) et créer un ou plusieurs tokens :



Ensuite, nous devons charger cet environnement :



Ce qui nous permet d'utiliser la ou les variable(s) token(s) dans les appels HTTP :


bruno
Collections
Search requests ...
API Suivi élèves
API Suivi élèves step by step
AUTH
POST Register
POST Login
POST Logout
CLASS_GROUPS
GET Get all classGroups
GET Get one classGroup
GET Get one classGroup - 404
POST Add a new classGroup
POST Add a new classGroup without Teacher
PUT Update a classGroup
DEL Delete a classGroup
COMMENTS
GET Get all comments of a student
GET Get one comment of a student
POST Add a new comment for this student
API Suivi élèves step by step
Safe Mode
api_suivi_eleves
GET Get all cla...
GET http://localhost:3333/classGroups
Params Body Headers Auth * Vars Script Assert
Tests Docs
Bearer Token
Token
{{token}}
Response Headers 6 Timeline Tests
200 OK 202ms 2.16KB
[
{
"id": 6,
"name": "CID2A",
"teacherId": 2,
"createdAt": "2025-06-23T07:23:32.000+00:00",
"updatedAt": "2025-06-23T07:23:32.000+00:00",
"teacher": {
"id": 2,
"name": "Mveng",
"firstname": "Antoine",
"email": "antoine.mveng@eduvaud.ch",
"userId": 2,
"createdAt": "2025-06-23T07:23:32.000+00:00",
"updatedAt": "2025-06-23T07:23:32.000+00:00"
}
},
{
"id": 7,
"name": "CID2B",
"teacherId": 1,
"createdAt": "2025-06-23T07:23:32.000+00:00",