

Documentation Technique

SAE - Qui Fait Quoi Quand

Sommaire

I – Installation du projet	3
A. Mise en place de Docker	3
A.1. Vue d'ensemble du fichier Docker Compose	3
A.2. Détail des Services	3
A.2.1. Service API (Symfony)	3
A.2.2. Service Frontend (VueJS)	3
A.2.3. Service Nginx (Serveur Web)	4
A.2.4. Service Base de données (PostgreSQL)	4
A.3. Variables d'Environnement	4
B. Mise en place des fonctionnalités GitLab	5
B.1. Gestion des branches	5
B. 2. Ajout de <i>pipelines</i> et de <i>runners</i>	7
C. Utilisation de scripts	8
II – API	10
III – Tests	36
IV - Développement Frontend	46
V - Déploiement	48

I – Installation du projet

A. Mise en place de Docker

A.1. Vue d'ensemble du fichier Docker Compose

L'infrastructure comprend 6 services organisés dans une configuration Docker Compose :

- gitlab-runner
- app
- database (PostgreSQL)
- api (Symfony)
- frontend
- nginx

La plupart des services sont connectés via un réseau interne ("*network*"), seuls la base de données et nginx sont exposés sur la machine hôte.

A.2. Détail des Services

A.2.1. Service API (Symfony)

L'API, construite avec Symfony, constitue le backend de notre application. Elle est configurée pour fonctionner dans un environnement PHP :

- L'image est basée sur PHP 8.2-FPM Alpine
- L'environnement inclut toutes les extensions PHP nécessaires (pgsql, pdo_pgsql, intl, zip, opcache)
- Composer est installé pour gérer les dépendances PHP et Symfony
- Le code source est monté depuis le dossier "*/api*"
- Exécute le script "*entrypoint.sh*" à l'initialisation du service

A.2.2. Service Frontend (VueJS)

Le frontend de l'application utilise Node.js et Quasar :

- Basé sur Node.js 18 avec une image Alpine
- Configuration en deux étapes distinctes pour le développement et la production
- Le code source est monté depuis le dossier "*/frontend*"

En développement :

- Hot-reloading activé pour une que la page s'actualise lorsqu'il ya des modifications

En production :

- Construction optimisée avec Quasar CLI
- Déploiement via Nginx

A.2.3. Service Nginx (Serveur Web)

Nginx sert de point d'entrée principal pour l'application, gérant le routage et la sécurité :

- Redirection automatique HTTP vers HTTPS
- Configuration de proxy inverse pour :
 - o Diriger le trafic vers le frontend (racine `/`)
 - o Router les requêtes API vers le backend Symfony (`/api`)
- Gestion SSL complète avec certificats montés depuis le système hôte
- Configuration flexible via variables d'environnement pour les ports
- Optimisation pour la production avec un redémarrage automatique

A.2.4. Service Base de données (PostgreSQL)

La base de données PostgreSQL est configurée pour assurer la persistance et la fiabilité des données :

- Utilisation de PostgreSQL 15 en version Alpine
- Persistance des données assurée par un volume Docker dédié (`postgres_data`)
- Configuration flexible via variables d'environnement pour :
 - o Nom de la base de données
 - o Identifiants de connexion
 - o Port d'exposition

A.3. Variables d'Environnement

Le projet utilise plusieurs fichiers d'environnement :

- `.env` - Configuration actuel
- `.env.dist` - Modèle avec une configuration exemple

Variables principales :

- APP_ENV : Type d'environnement (dev ou prod)
- HTTP_PORT : Port HTTP du serveur nginx (par défaut : 80)
- HTTPS_PORT : Port HTTPS du serveur nginx (par défaut : 443)
- DB_PORT : Port de la base de données (par défaut : 5432)
- DB_USERNAME: Identifiant de la base de données (par défaut : admin)
- DB_PASSWORD : Mot de passe de la base de données (par défaut : admin)
- DB_DATABASE: Nom de la base de données (par défaut : db)

B. Mise en place des fonctionnalités GitLab

B.1. Gestion des branches

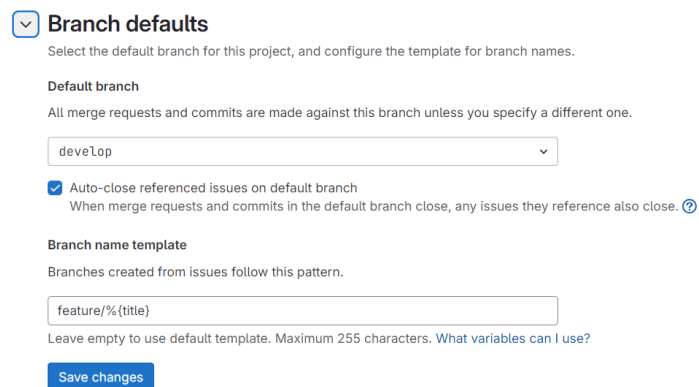
Le projet est stocké sur un dépôt GitLab de l'IUT de Valence, dont le lien est "[Anthony Saillard / SAE S5 - QFQQ · GitLab](#)". Sur ce dépôt, nous avons plusieurs branches afin de suivre le GitFlow.

Pour rappel, le GitFlow est une option pour la gestion des versions, offrant une structure claire et organisée pour le développement. Gitflow repose sur l'utilisation de branches dédiées pour chaque phase du cycle de développement. La branche principale (*main*) est utilisée pour les versions en production, tandis que la branche *develop* sert de base pour toutes les fonctionnalités en cours de développement. Pour chaque nouvelle fonctionnalité ou correctif, une *feature branch* est créée à partir de *develop*, ce qui permet à l'équipe de travailler en parallèle sans perturber le code principal. Une fois la fonctionnalité prête, elle est fusionnée dans *develop* après une révision via une *merge request*. Le modèle inclut également des branches spécifiques pour les releases et les *hotfixes*, ce qui permet de gérer facilement les versions prêtes à être déployées ou les corrections urgentes en production. Ce *workflow* garantit une gestion efficace du code, un suivi précis des versions et une meilleure collaboration au sein de l'équipe.

Pour mettre en place le GitFlow, dans la barre latérale, sélectionnez **Code > Branches**. Dans cet onglet sont visibles les branches ouvertes sur le projet, soit seulement la branche *main* au commencement du projet. En cliquant sur le bouton

"New branch", un nouvel onglet s'affiche pour permettre de choisir le nom de cette nouvelle branche et de choisir quelle branche nous allons copier. Ici, le nom de la nouvelle branche est "*develop*" et elle est créée à partir de *main*.

Pour gérer les droits sur ces deux branches, il faut cliquer sur le bouton "*View branch rules*". Dans *branch defaults*, la branche sélectionnée pour être celle par défaut est *main*. Or, dans notre cas, il faut sélectionner *develop*. De plus, pour que le préfixe "*feature*" s'affiche au début des noms des branches, il faut renseigner le format du nom de la branche, soit "*feature/%{title}*". Enfin, il faut sauvegarder.



Ensuite dans l'onglet "*Branch rules*", les droits peuvent être informés. Il faut d'abord vérifier les droits de la branche *main*, en cliquant sur le lien "*View details*". Le rôle pour "*merge*" et "*push and merge*" doit être seulement "*Maintainers*".

Pour les règles de la branche *develop*, il faut cliquer sur "*Add branch rule*" si les règles de la branche ne sont pas encore créées. Sélectionnez le nom de la branche, soit *develop*, et cliquez sur le bouton "Create protected branch". Editez les rôles de "*merge*" et de "*push and merge*", en décochant "*Maintainers*" et en cochant "*Developers and Maintainers*". Ainsi, si les personnes, ayant accès au projet, ont un rôle de développeur ou de responsable, ils ont l'opportunité de modifier le projet sur cette branche. De plus, nous avons autorisé le "*force push*" sur cette branche, pour éviter d'être bloqués sur une branche à cause d'une erreur qui pourra être réglée dans une deuxième branche.

Le résultat doit être identique à celui ci-dessous.

Branch rules

Define rules for who can push, merge, and the

Branch rules 2		
deveLop	default	protected
<ul style="list-style-type: none">Allowed to force pushAllowed to merge: 1 roleAllowed to push and merge: 2 roles		
main	protected	
<ul style="list-style-type: none">Allowed to merge: 1 roleAllowed to push and merge: 1 role		

B. 2. Ajout de *pipelines* et de *runners*

Nous souhaitons que ce projet contienne un fichier de *pipeline GitLab CI/CD* pour automatiser la gestion du projet, soit le fichier ".gitlab-ci.yml". Ce fichier doit se trouver à la racine du projet et il se constitue d'instructions pour spécifier les étapes, les jobs, les dépendances du *pipeline*.

Dans ce fichier ".gitlab-ci.yml", nous utilisons :

- Une image nommée "docker:dind", permettant une exécution dite "*Docker-in-Docker*". Ainsi, elle permet la construction et la manipulation des conteneurs Docker dans la *pipeline*.
- Trois stages, soit le *stage* "build" pour construire l'image Docker, le *stage* "test" pour effectuer des tests unitaires et de *lint*, et le *stage* "deploy" pour déployer l'application.
- Le service "docker:dind" permettant une interaction entre la *pipeline* et Docker.
- Les deux variables "DOCKER_IMAGE" définissant le registre Docker Hub, et "DOCKER_IMAGE_TAG" correspondant à un tag basé sur l'identifiant unique du *pipeline*.
- Les chemins "vendor/" et "node_modules" mis en cache afin d'accélérer les étapes du *pipeline*.
- Une étape "before_script", comprenant la ligne de commande permettant la connexion à Docker Hub. Elle utilise les variables d'environnement "DOCKER_HUB_PASSWORD" et "CI_REGISTRY_USER", correspondant au *token* servant de mot de passe et du *login* utilisateur.
- Pour le *stage* "build", un script est exécuté. A partir de ce script, une image Docker est construite depuis le dossier nommé "frontend" et le *tag* devient "\$DOCKER_IMAGE:\$DOCKER_IMAGE_TAG", soit l'ensemble des deux variables. Puis, l'image est poussée sur le registre Docker Hub.

- Pour le *stage* "test:lint:", il se base sur le *stage* initial "test" et exécute un script. Ce script exécute le *linter*, qui analyse le code source pour identifier des erreurs de codage ou des problèmes de style.
- Pour le *stage* "test:unit:", il se base sur le *stage* initial "test" et utilise l'image Docker construite dans le *stage* "build". Son script exécute PHPUnit. Il s'agit de l'exécution automatique des tests unitaires et fonctionnels.
- Le *stage* "deploy" déploie l'image Docker en production. Son script télécharge l'image Docker avec le *tag latest*. Il n'est exécuté que pour des *pipelines* déclenchées par des tags.

Pour obtenir un registre Docker Hub, il faut créer un compte sur le site officiel de Docker Hub, soit [Docker Hub Container Image Library | App Containerization](#). Une fois authentifié, il faut aller dans l'onglet "Repositories" et cliquer sur le bouton "Create a repository". Le *namespace* sera le nom de l'utilisateur du compte. Il faut ajouter également un nom de registre, et une description. Il faut choisir un type de visibilité, en sachant que sur un compte gratuit, un seul registre peut être privé. Si le registre est privé, il faut ajouter le *token* et le nom du registre dans les variables du projet Gitlab, qui sont dans les paramètres du projet, dans la section "Variables". Dans cette section, il faut cliquer sur le bouton "Add variable", mettre en "key" le nom sous lequel la variable sera appelée et en "value" soit le *token* ou le nom du registre.

Ensuite, il faut ajouter un *runner*. Nous avons choisi d'en créer un localement. Pour cela, il faut un fichier de configuration du *runner*, soit gitlab-runner/config/config.toml dans notre cas. Il s'agit de la copie du fichier gitlab-runner/config/config.toml.dist. Pour obtenir la bonne *url* et le bon *token*, il faut aller dans l'onglet "Runners" que l'on trouve dans les paramètres du projet, dans "CI/CD". Ici, il faut cliquer sur le bouton "New project runner", cocher "Run untagged jobs", ajouter une description si besoin et cliquer sur le bouton "Create runner". Une fois créé, il faut choisir la plateforme souhaitée (Windows ou Linux) et copier l'*url* et le *token* de l'étape 1 dans le fichier de configuration du *runner*.

Une fois les modifications faites, il ne faut pas oublier de lancer le script "./init.sh" dans un terminal du projet.

C. Utilisation de scripts

Pour simplifier l'utilisation du projet, nous avons créé des scripts.

Le script `./scripts/start.sh`, exécuté dans le *bash*, télécharge les images du fichier `"docker-compose.yml"` via la commande `"docker compose pull --ignore-buildable"`. L'option `"--ignore-buildable"` force le téléchargement des images plutôt que de reconstruire celles marquées avec `"build"` dans le fichier. Puis, le script démarre les services compris dans le `"docker-compose.yml"`, en exécutant les conteneurs en arrière-plan via la commande `"docker compose up -d"`. Enfin, une seconde plus tard, il liste tous les conteneurs gérés par Docker Compose et indique leur statut, soit `"running"` ou `"exited"` par exemples, via la commande `"docker compose ps"`.

Le script `./scripts/stop.sh`, exécuté dans le *bash*, arrête les conteneurs fonctionnels via la commande `"docker compose stop"`.

Le script `./scripts/lint.sh`, exécuté dans le *bash*, vérifie d'abord que le composant PHPStan est installé et accessible dans le conteneur via la commande `"docker compose exec -T api phpstan --version"`. Puis il s'assure de la qualité du code en exécutant le lint dans le conteneur via la commande `"docker compose exec -T api phpstan analyse --level=max src/"`.

Le script `./scripts/pre-commit.sh`, exécuté dans le *bash*, appelle seulement le script exécutant le lint. Pour l'exécuter automatiquement à chaque commit, il faut le copier dans le fichier `".git/hooks/pre-commit"` via la commande `"cp ./scripts/pre-commit.sh .git/hooks/pre-commit"`.

Le script `./scripts/run-tests.sh`, exécuté dans le *bash*, exécute tous les tests unitaires et fonctionnels via la commande `"docker compose exec api vendor/bin/phpunit --configuration /var/www/html/phpunit.xml"`. Il s'agit de lancement automatique de ces tests, situés dans un répertoire `./tests` et dont la configuration se trouve dans le fichier `phpunit.xml`.

Le script `./scripts/build.sh`, exécuté dans le *bash*, construit les images Docker définies dans le fichier `"docker-compose.yml"` en exécutant les étapes dans les directives *build* de chaque service et désactive l'utilisation du cache Docker lors de la construction des images via la commande `"docker compose build --no-cache"`.

Le script `./scripts/uninstall.sh`, exécuté dans le *bash*, arrête les services et les supprime, ainsi que leurs volumes, via la commande `"docker compose down --volumes"`.

Le script `./scripts/migrations.sh`, exécuté dans le *bash*, vérifie et crée d'abord de nouvelles migrations si nécessaire, via la commande `"docker exec -it api php bin/console make:migration"`. Puis, il applique toutes les migrations existantes, via la commande `"doctrine exec -it api php bin/console doctrine:migrations:migrate"`.

Le script `./api/entrypoint.sh`, exécuté dans le *bash*, attend d'abord que le service PostgreSQL soit prêt via la commande `"until pg_isready -h database -p 5432 -q"`. `Pg_isready` vérifie la disponibilité de PostgreSQL sur `database` et le port `5432`. Si PostgreSQL n'est pas prêt, le script réessaie deux secondes plus tard. Puis il installe les dépendances de Symfony comprises dans le fichier `"composer.json"` via la commande `"composer install"`. Ensuite, il s'assure que le répertoire de cache Symfony a les permissions d'être utilisé par le serveur web via les commandes :

```
" chmod -R 775 /var/www/html/var/cache  
  
chown -R www-data:www-data /var/www/html/var/cache"
```

Enfin, si la base de données n'existe pas, il la crée de manière non interactive via la commande `"php bin/console doctrine:database:create --if-not-exists --no-interaction"`, et il applique les migrations si elles existent, une fois le dossier `"migrations"` vérifié, via la commande `"php bin/console doctrine:migrations:migrate --no-interaction"`.

Le script `./init.sh`, exécuté dans le *bash*, stoppe sa propre exécution si une commande échoue via la commande `"set -e"`. En première étape, il vérifie que Docker et Docker Compose soit bien installés via la commande `"command -v docker (ou docker-compose) &> /dev/null"`, et il crée le fichier `".env"`, s'il n'existe pas en copiant le fichier `".env.dist"`, grâce auquel il charge les variables d'environnement qu'il contient pour qu'elles soient disponibles dans le script. À la suite de cela, il arrête les conteneurs existants, reconstruit les images et démarre les conteneurs. Enfin, il attend que les services aient démarré pour afficher les URL locales permettant une visualisation des pages de l'application.

II – API

Authentification

L'authentification se fait par *OAuth*. Quand une personne s'enregistre ou se connecte, il reçoit un *token*, soit *access_token*, lui permettant d'effectuer les requêtes. Ce *token* expire après une heure d'inactivité. Il est passé par le *Header* de la requête, dans *Authorization*.

Endpoint

Method : [POST]

Endpoint: /api/register

Description: Permet d'enregistrer un utilisateur dans l'application

Paramètres :

- Login (string) - Identifiant de l'utilisateur
- Password (string) - Mot de passe de l'utilisateur
- First_name (string) - Prénom de l'utilisateur
- Last_name (string) - Nom de l'utilisateur
- Email (string) - Adresse électronique de l'utilisateur

Exemple de réponse:

HTTP Status Code: [200]

Response Body:

```
{
  "message": "Inscription réussie",
  "access_token": "2227d4f1-c7b5-47ca-a499-18362900dce2",
  "refresh_token": "678e2750-21dd-4fd5-bcf3-97ca0df93153",
  "token_type": "Bearer",
  "expires_in": 3600,
  "user": {
    "id": 2,
    "login": "riottet",
    "email": "theo.riotte@etu.fr",
    "firstName": "theo",
    "lastName": "riotte",
    "role": "ROLE_USER"
  }
}
```

Endpoint

Method: [POST]

Endpoint: /api/login

Description: Permet la connexion à l'application avec ses identifiants

Paramètres :

- Login (string) - Identifiant de l'utilisateur
- Password. (string) - Mot de passe de l'utilisateur

Exemple de réponse:

HTTP Status Code: [200]

Response Body:

```
{
  "access_token": "574798d1-cff5-42b1-aa14-7500cf7ab8f0",
  "refresh_token": "62c6f091-d4e7-4a52-b030-f20b4043b905",
  "token_type": "Bearer",
  "expires_in": 3600,
  "user": {
    "id": 2,
    "login": "riottet",
    "role": "ROLE_USER"
  }
}
```

Endpoint

Method : [GET]

Endpoint: /api/me

Description: Permet la récupération des informations de l'utilisateur connecté.

Paramètres :

- Authorization (header – string) - Récupère le *token* passé dans le header de la requête.

Exemple de réponse:

HTTP Status Code: [200]

Response Body:

```
{
  "user": {
    "id": 2,
    "login": "riottet",
    "email": "theo.riotte@etu.fr",
    "role": "ROLE_USER",
    "firstName": "theo",
  }
}
```

```
        "lastName": "riotte"
    }
}
```

Endpoint

Method : [POST]
Endpoint: /api/logout
Description: Permet la déconnection de l'application

Paramètres :

- Authorization (string) - *Token* de l'utilisateur

Réponse:

HTTP Status Code: [200]

Response Body:

```
{
  "message" : "Logged out successfully"
}
```

Endpoint

Method : [GET]
Endpoint: /users/{id}
Description: Permet la récupération des informations d'une personne à partir d'un id

Paramètres :

- Id (string) - Id de l'utilisateur recherché

Exemple de réponse :

HTTP Status Code: [200]

Response Body:

```
{
  "user": {
    "id": 2,
    "login": "riottet",
    "email": "theo.riotte@etu.fr",
    "role": "ROLE_USER",
```

```
    "firstName": "theo",  
    "lastName": "riotte"  
  }  
}
```

Les types de courses

Endpoint

Méthode : [GET]

Endpoint: /api/course-types

Description : Récupère la liste des types de cours pour une année scolaire spécifique Paramètres :

- School-Year (en-tête - entier, optionnel) - ID de l'année scolaire. Si non fourni, utilise l'année scolaire actuelle

Exemple de réponse :

```
[ { "id": 1, "name": "Nom du type de cours", "hourly_rate": 50.00, "school_year_id": 2,  
  "groups": [1, 2, 3] } ]
```

Endpoint

Méthode : [GET]

Endpoint: /api/course-types/{id}

Description : Récupère les détails d'un type de cours spécifique

Paramètres :

- Id - ID du type de cours

Exemple de réponse :

```
{  
  "id": 1,  
  "name": "Nom du type de cours",  
  "hourly_rate": 50.00,  
  "school_year_id": 2,  
  "groups": [1, 2, 3]  
}
```

Endpoint

Méthode : [POST]

Endpoint: /api/course-types

Description : Crée un nouveau type de cours

Paramètres :

- name (chaîne) - Nom du type de cours
- hourly_rate (nombre) - Taux horaire du cours
- id_school_year (entier) - ID de l'année scolaire associée

Exemple de réponse :

```
{  
  "id": 1,  
  "name": "Nouveau type de cours",  
  "hourly_rate": 50.00, "  
  id_school_year": 2  
}
```

Endpoint

Méthode : [PUT]

Endpoint: /api/course-types/{id}

Description : Met à jour un type de cours existant

Paramètres :

- name (chaîne, optionnel) - Nouveau nom pour le type de cours
- hourly_rate (nombre, optionnel) - Nouveau taux horaire

Exemple de réponse :

```
{  
  • "id": 1,  
  • "name": "Type de cours mis à jour",  
  • "hourly_rate": 60.00  
}
```

Endpoint

Méthode : [DELETE]

Endpoint: /api/course-types/{id}

Description : Supprime un type de cours spécifique

Exemple de réponse :

```
{
  "message": "Type de cours supprimé avec succès"
}
```

Les formations

Endpoint

Méthode : [GET]

Endpoint: /api/formations

Description : Récupère la liste des formations pour une année scolaire spécifique

Paramètres :

- School-Year (en-tête - entier, optionnel) - ID de l'année scolaire. Si non fourni, utilise l'année scolaire actuelle

Exemple de réponse :

```
{
  "id": 1,
  "label": "Nom de la formation",
  "order_number": 1,
  "id_school_year": 2,
  "pedagogical_interruptions": [1, 2],
  "semesters": [1, 2],
  "groups": [1, 3]
}
```

Endpoint

Méthode : [GET]

Endpoint: /api/formations/{id}

Description : Récupère les détails d'une formation spécifique

Paramètres :

- Id (chemin - entier) - ID de la formation

Exemple de réponse :

```
{
  "id": 1,
  "label": "Nom de la formation",
  "order_number": 1,
  "id_school_year": 2,
  "pedagogical_interruptions": [1, 2],
  "semesters": [1, 2],
  "groups": [1, 3]
}
```

Endpoint

Méthode : [POST]

Endpoint: /api/formations

Description : Crée une nouvelle formation

Paramètres :

- label (chaîne, optionnel) - Libellé de la formation
- order_number (entier, optionnel) - Numéro d'ordre de la formation
- id_school_year (entier, optionnel) - ID de l'année scolaire associée

Exemple de réponse :

```
{
  "id": 1,
  "label": "Nouvelle formation",
  "order_number": 1,
  "id_school_year": 2
}
```

Endpoint

Méthode : [PUT]

Endpoint: /api/formations/{id}

Description : Met à jour une formation existante

Paramètres :

- label (chaîne, optionnel) - Nouveau libellé de la formation
- order_number (entier, optionnel) - Nouveau numéro d'ordre
- id_school_year (entier, optionnel) - Nouvel ID d'année scolaire

Exemple de réponse :

```
{ "message": "Formation mise à jour avec succès" }
```

Endpoint

Méthode : [DELETE]

Endpoint: /api/formations/{id}

Description : Supprime une formation spécifique

Exemple de réponse :

```
{ "message": "Formation supprimée avec succès" }
```

Méthode: [GET]

Endpoint: /api/formations/hours/{id}

Description: Récupère les heures associées à une formation, organisées par sous-ressource et type de cours.

Exemple de réponse :

```
{ "id": 1, "label": "Licence Informatique", "sub_resources": [ { "id": 2, "name":  
"Programmation", "course_types_hours": [ { "course_type_id": 1,  
"course_type_name": "Cours magistral", "total_hours": 30.0 }, { "course_type_id": 2,  
"course_type_name": "Travaux dirigés", "total_hours": 20.5 } ] } ] }
```

Les interruptions pédagogiques

Endpoint

Méthode : [GET]

Endpoint: /api/pedagogical-interruptions

Description : Récupère la liste des interruptions pédagogiques pour une année scolaire spécifique

Paramètres :

- School-Year (en-tête - entier, optionnel) - ID de l'année scolaire. Si non fourni, utilise l'année scolaire actuelle

Exemple de réponse :

```
[
  {
    "id": 1,
    "name": "Interruption de Noël",
    "start_date": "2023-12-20",
    "end_date": "2024-01-05",
    "formation_id": 2
  }
]
```

Endpoint

Méthode : [GET]

Endpoint: /api/pedagogical-interruptions/{id}

Description : Récupère les détails d'une interruption pédagogique spécifique

Paramètres :

- Id (chemin - entier) - ID de l'interruption pédagogique

Exemple de réponse :

```
{
  "id": 1,
  "name": "Interruption de Noël",
  "start_date": "2023-12-20",
  "end_date": "2024-01-05"
}
```

Endpoint

Méthode : [POST]

Endpoint: /api/pedagogical-interruptions

Description : Crée une nouvelle interruption pédagogique

Paramètres :

- name (chaîne) - Nom de l'interruption
- start_date (date, optionnel) - Date de début au format YYYY-MM-DD
- end_date (date, optionnel) - Date de fin au format YYYY-MM-DD
- formation_id (entier) - ID de la formation associée
- Exemple de réponse :

```
{
  "id": 1,
  "name": "Nouvelle interruption",
  "start_date": "2024-02-10",
  "end_date": "2024-02-15",
  "formation_id": 2
}
```

Endpoint

Méthode : [PUT]

Endpoint: /api/pedagogical-interruptions/{id}

Description : Met à jour une interruption pédagogique existante

Paramètres :

- name (chaîne, optionnel) - Nouveau nom de l'interruption
- start_date (date, optionnel) - Nouvelle date de début
- end_date (date, optionnel) - Nouvelle date de fin

Exemple de réponse :

```
{
  "id": 1,
  "name": "Interruption mise à jour",
  "start_date": "2024-02-10",
  "end_date": "2024-02-15"
}
```

Les années scolaires

Endpoint

Méthode : [DELETE]

Endpoint: /api/pedagogical-interruptions/{id}

Description : Supprime une interruption pédagogique spécifique

Exemple de réponse :

```
{  
  "message": "Ressource supprimée avec succès"  
}
```

Endpoint

Méthode : [GET]

Endpoint: /api/school-years

Description : Récupère la liste de toutes les années scolaires

Exemple de réponse : [{ "id": 1, "label": "2023-2024", "current_school_year": true }]

Endpoint

Méthode : [GET]

Endpoint: /api/school-years/current

Description : Récupère l'année scolaire en cours

Exemple de réponse :

```
{  
  "id": 1,  
  "label": "2023-2024",  
  "current_school_year": true  
}
```

Endpoint

Méthode : [GET]

Endpoint: /api/school-years/{id}

Description : Récupère les détails d'une année scolaire spécifique

Paramètres :

- Id (chemin - entier) - ID de l'année scolaire
-

Exemple de réponse :

```
{  
  "id": 1,  
  "label": "2023-2024",  
  "current_school_year": true  
}
```

Endpoint

Méthode : [PUT]

Endpoint: /api/school-years/{id}/set-current

Description : Définit une année scolaire comme année courante

Exemple de réponse :

```
{  
  "id": 1,  
  "label": "2023-2024",  
  "current_school_year": true  
}
```

Endpoint

Méthode : [POST]

Endpoint: /api/school-years

Description : Crée une nouvelle année scolaire

Paramètres :

- label (chaîne) - Libellé de l'année scolaire

- `current_school_year` (booléen, optionnel) - Définit si c'est l'année scolaire courante Exemple de réponse : `{ "id": 2, "label": "2024-2025", "current_school_year": false }`

Endpoint

Méthode : [PUT]

Endpoint: `/api/school-years/{id}`

Description : Met à jour une année scolaire existante Paramètres :

- `label` (chaîne) - Nouveau libellé de l'année scolaire
-

Exemple de réponse :

```
{ "message": "Année scolaire mise à jour avec succès" }
```

Endpoint

Méthode : [DELETE]

Endpoint: `/api/school-years/{id}`

Description : Supprime une année scolaire

Exemple de réponse :

```
{  
  "status": "success",  
  "message": "Année scolaire supprimée avec succès"  
}
```

Endpoint

Méthode : [GET]

Endpoint: `/api/users`

Description : Récupère la liste de tous les utilisateurs

Exemple de réponse :

```
[  
  {  
    "id": 1,  
    "login": "utilisateur1",
```

```
    "last_name": "Dupont",  
    "first_name": "Jean",  
    "role": "ROLE_USER",  
    "phone": "0123456789",  
    "email": "jean.dupont@example.com"  
  }  
]
```

Les utilisateurs

Endpoint

Méthode : [GET]

Endpoint: /api/users/{id}

Description : Récupère les détails d'un utilisateur spécifique

Paramètres :

- Id (chemin - entier) - ID de l'utilisateur

Exemple de réponse :

```
{  
  "id": 1,  
  "login": "utilisateur1",  
  "last_name": "Dupont",  
  "first_name": "Jean",  
  "role": "ROLE_USER",  
  "phone": "0123456789",  
  "email": "jean.dupont@example.com"  
}
```

Endpoint

Méthode : [PUT]

Endpoint: /api/users/{id}

Description : Met à jour les informations d'un utilisateur

Paramètres :

- login (chaîne, optionnel) - Nouveau nom de connexion
- password (chaîne, optionnel) - Nouveau mot de passe
- last_name (chaîne, optionnel) - Nouveau nom de famille
- first_name (chaîne, optionnel) - Nouveau prénom
- role (chaîne, optionnel) - Nouveau rôle
- phone (chaîne, optionnel) - Nouveau numéro de téléphone
- email (chaîne, optionnel) - Nouvelle adresse email

Exemple de réponse :

```
{  
  "message": "Utilisateur mis à jour avec succès"  
}
```

Endpoint

Méthode : [DELETE]

Endpoint: /api/users/{id}

Description : Supprime un utilisateur

Exemple de réponse :

```
{  
  "message": "Utilisateur supprimé avec succès"  
}
```

Méthode : [GET]

Endpoint : /api/assignments

Description : Récupère la liste des assignments avec possibilité de filtrage.

Paramètres de requête :

- id_sub_resource (entier, optionnel) - Filtrer par ID de sous-ressource
- id_user (entier, optionnel) - Filtrer par ID d'utilisateur
- id_course_type (entier, optionnel) - Filtrer par ID de type de cours
- id_semester (entier, optionnel) - Filtrer par ID de semestre

Méthode : [POST]

Endpoint : /api/assignments

Description : Crée ou met à jour un assignment si les mêmes critères sont trouvés.
Corps de la requête (JSON) :

- allocated_hours (entier, requis) - Nombre d'heures attribuées
- assignment_date (chaîne, requis) - Date d'affectation (format YYYY-MM-DD)
- annotation (chaîne, optionnel) - Remarque sur l'affectation
- id_sub_resources (entier, requis) - ID de la sous-ressource concernée
- id_users (entier, optionnel) - ID de l'utilisateur concerné
- id_course_type (entier, requis) - ID du type de cours

Exemple de requête :

```
{ "allocated_hours": 15, "assignment_date": "2025-03-21", "annotation": "Cours avancé", "id_sub_resources": 2, "id_users": 5, "id_course_type": 3 }
```

Exemple de réponse :

```
{ "message": "Assignment created successfully", "id": 10, "allocated_hours": 15, "assignment_date": "2025-03-21", "annotation": "Cours avancé", "id_sub_resources": 2, "id_users": 5, "id_course_type": 3 }
```

Les assignments

Méthode : [PUT]

Endpoint : /api/assignments/{id}

Description : Met à jour les informations d'un assignment existant.

Paramètres de route :

- id (entier, requis) - ID de l'assignment à modifier

Corps de la requête (JSON) :

- allocated_hours (entier, requis) - Nouveau nombre d'heures attribuées
- assignment_date (chaîne, requis) - Nouvelle date d'affectation
- annotation (chaîne, optionnel) - Nouvelle annotation
- id_sub_resources (entier, requis) - Nouvel ID de sous-ressource
- id_users (entier, optionnel) - Nouvel ID d'utilisateur
- id_course_type (entier, requis) - Nouvel ID de type de cours

Exemple de requête :

```
{ "allocated_hours": 12, "assignment_date": "2025-03-22", "annotation": "Correction apportée", "id_sub_resources": 3, "id_users": 7, "id_course_type": 2 }
```

Exemple de réponse :

```
{
  "message": "Assignment updated successfully"
}
```

Méthode : [DELETE]

Endpoint : /api/assignments/{id}

Description : Supprime un assignment existant.

Paramètres de route :

- id (entier, requis) - ID de l'assignation à supprimer

Exemple de réponse :

```
{
  "message": "Assignment deleted successfully"
}
```

Les cours des enseignants

Méthode: [GET]

Endpoint: /api/course-teachers

Description: Récupère la liste des enseignants de cours, avec possibilité de filtrage.

Paramètres de requête :

- id_group (int, optionnel) - Filtrer par ID du groupe
- id_sub_resource (int, optionnel) - Filtrer par ID de sous-ressource
- id_user (int, optionnel) - Filtrer par ID de l'utilisateur

Exemple de réponse :

```
[ { "id": 1, "id_sub_resource": 5, "sub_resource_name": "Mathématiques", "id_user": 12, "user_name": "Jean", "id_group": 3, "group_name": "Classe A" } ]
```

Méthode: [GET]

Endpoint: /api/course-teachers/{id}

Description: Récupère les détails d'un enseignant de cours spécifique via son ID.

Exemple de réponse :

```
{ "id": 1, "id_sub_resource": 5, "sub_resource_name": "Mathématiques", "id_user": 12, "user_name": "Jean", "id_group": 3, "group_name": "Classe A" }
```

Méthode: [POST]

Endpoint: /api/course-teachers

Description: Ajoute un nouvel enseignant.

Corps de la requête (JSON) :

```
{ "id_sub_resource": 5, "id_user": 12, "id_group": 3 }
```

Exemple de réponse (succès) :

```
{ "id": 1, "id_sub_resource": 5, "id_user": 12, "id_group": 3 }
```

Codes de réponse :

- 201 Created – Enseignant ajouté avec succès
- 400 Bad Request – Données invalides
- 404 Not Found – L'une des ressources associées (sous-ressource, utilisateur ou groupe) n'existe pas

Méthode: [PUT]

Endpoint: /api/course-teachers/{id}

Description: Modifie un enseignant existant.

Corps de la requête (JSON) :

```
{ "id_sub_resource": 6, "id_user": 15, "id_group": 4 }
```

Exemple de réponse (succès) :

```
{  
  "message": "CourseTeacher updated successfully"  
}
```

Codes de réponse :

- 200 OK – Enseignant mis à jour avec succès
- 400 Bad Request – Données invalides
- 404 Not Found – Enseignant ou l'une des ressources associées non trouvée

Méthode: [DELETE]

Endpoint: /api/course-teachers/{id}

Description: Supprime un enseignant.

Exemple de réponse (succès) :

```
{  
  "message": "CourseTeacher deleted successfully"  
}
```

Codes de réponse :

- 204 No Content – Suppression réussie
- 404 Not Found – Enseignant non trouvé

Les groupes

Méthode : GET

Endpoint : /api/groups

Description : Retourne la liste des groupes avec la possibilité de filtrer par formation et année scolaire.

Paramètres de requête :

- id_formation (*int, optionnel*) – Filtrer par ID de formation.
- id_school_year (*int, optionnel*) – Filtrer par ID d'année scolaire.

Exemple de réponse :

```
[ { "id": 1, "name": "Groupe A", "description": "Groupe avancé", "order_number": 1, "id_parent_group": null, "id_course_types": 2, "id_formation": 5 } ]
```

Méthode : GET

Endpoint : /api/groups/{id}

Description : Retourne les détails d'un groupe en fonction de son ID.

Exemple de réponse :

```
{ "id": 1, "name": "Groupe A", "description": "Groupe avancé", "order_number": 1, "id_parent_group": null, "id_course_types": 2, "id_formation": 5 }
```

Méthode : POST

Endpoint : /api/groups

Description : Crée un nouveau groupe.

Corps de la requête (JSON) :

```
{ "name": "Groupe B", "description": "Groupe intermédiaire", "order_number": 2, "id_parent_group": 1, "id_course_types": 3, "id_formation": 5 }
```

Exemple de réponse :

```
{ "id": 2, "name": "Groupe B", "description": "Groupe intermédiaire", "order_number": 2, "id_parent_group": 1, "id_course_types": 3, "id_formation": 5 }
```

Codes de réponse :

- 201 Created : Groupe créé avec succès.

- 400 Bad Request : Données invalides.

Méthode : PUT

Endpoint : /api/groups/{id}

Description : Met à jour les informations d'un groupe.

Corps de la requête (JSON) :

```
{  
  "name": "Groupe B Modifié",  
  "description": "Mise à jour du groupe",  
  "order_number": 3  
}
```

Exemple de réponse :

```
{  
  "message": "Group updated successfully"  
}
```

Codes de réponse :

- 200 OK : Groupe mis à jour avec succès.
- 400 Bad Request : Données invalides.
- 404 Not Found : Groupe introuvable.

Méthode : DELETE

Endpoint : /api/groups/{id}

Description : Supprime un groupe spécifique.

Exemple de réponse :

```
{  
  "message": "Group deleted successfully"  
}
```

Codes de réponse :

- 204 No Content : Suppression réussie.
- 404 Not Found : Groupe introuvable.

Les ressources

URL : /api/resources

Méthode : GET

Description : Retourne la liste des ressources filtrées en fonction des paramètres fournis.

Paramètres de requête :

- id_user (int) : Filtrer par utilisateur
- id_semester (int) : Filtrer par semestre
- id_formation (int) : Filtrer par formation
- id_school_year (int) : Filtrer par année scolaire (si non fourni, prend l'année scolaire en cours)

Réponse :

200 OK

```
[ { "id": 1, "identifiant": "MAT101", "name": "Mathematics", "description": "Basic Math Course", "id_semesters": 2, "id_formation": 1, "id_school_year": 2023, "id_users": 5 } ]
```

URL : /api/resources/{id}

Méthode : GET

Description : Retourne les détails d'une ressource spécifique.

Réponse :

200 OK

```
{
  "id": 1,
  "identifiant": "MAT101",
  "name": "Mathematics",
  "description": "Basic Math Course",
  "id_semesters": 2,
  "id_users": 5,
  "annotations": [],
  "sub_resources": [],
  "notifications": []
}
```

URL : /api/resources

Méthode : POST

Description : Crée une nouvelle ressource.

Corps de la requête :

```
{  
  "identifiant": "PHY102",  
  "name": "Physics",  
  "description": "Physics Basics",  
  "id_semesters": 2,  
  "id_users": 5  
}
```

Réponse :

201 Created

```
{  
  "id": 2,  
  "identifiant": "PHY102",  
  "name": "Physics",  
  "description": "Physics Basics",  
  "id_semesters": 2,  
  "id_users": 5,  
  "annotations": [],  
  "sub_resources": [],  
  "notifications": []  
}
```

Erreurs possibles :

- 400 Bad Request : JSON invalide ou champ obligatoire manquant (identifiant ou name).
- 400 Bad Request : id_semesters ou id_users invalide.

URL : /api/resources/{id}

Méthode : PUT

Description : Modifie une ressource existante.

Corps de la requête :


```
{  
  "name": "Advanced Physics",  
  "description": "Updated Physics Course"  
}
```

Réponse :
200 OK

```
{  
  "message": "Resource updated successfully"  
}
```

Erreurs possibles :

- 400 Bad Request : JSON invalide.
- 404 Not Found : Ressource inexistante.

URL : /api/resources/{id}
Méthode : DELETE
Description : Supprime une ressource existante.
Réponse :
204 No Content

Les semestres

Méthode : GET
Endpoint : /api/semesters
Description : Récupère la liste des semestres en fonction des filtres appliqués.
Paramètres (query) :

- id_formation (*entier, optionnel*) - Filtrer par ID de formation
- id_school_year (*entier, optionnel*) - Filtrer par ID d'année scolaire

Réponse :

```
[  
  {  
    "id": 1,  
    "name": "Semestre 1",  
    "start_date": "2024-01-01",  
    "end_date": "2024-06-30",  
    "order_number": 1,  
    "id_formation": 10  
  }  
]
```

Méthode : GET

Endpoint : /api/semesters/{id}

Description : Récupère les détails d'un semestre spécifique.

Réponse :

```
{
  "id": 1,
  "name": "Semestre 1",
  "start_date": "2024-01-01",
  "end_date": "2024-06-30",
  "order_number": 1,
  "id_formation": 10,
  "resources": [101, 102, 103]
}
```

Méthode : POST

Endpoint : /api/semesters

Description : Crée un nouveau semestre.

Corps de la requête (JSON) :

```
{
  "name": "Semestre 2",
  "start_date": "2024-07-01",
  "end_date": "2024-12-31",
  "order_number": 2,
  "id_formation": 10
}
```

Réponse :

```
{
  "id": 2,
  "name": "Semestre 2",
  "start_date": "2024-07-01",
  "end_date": "2024-12-31",
  "order_number": 2,
  "id_formation": 10
}
```

Méthode : PUT

Endpoint : /api/semesters/{id}

Description : Met à jour les informations d'un semestre spécifique.

Corps de la requête (JSON) :

```
{
  "name": "Semestre 1 Modifié",
  "start_date": "2024-02-01",
  "end_date": "2024-07-31",
  "order_number": 1
}
```

Réponse :

```
{
  "id": 1,
  "name": "Semestre 1 Modifié",
}
```

```
"start_date": "2024-02-01",  
"end_date": "2024-07-31",  
"order_number": 1  
}
```

Méthode : DELETE

Endpoint : /api/semesters/{id}

Description : Supprime un semestre spécifique.

Réponse :

```
{  
  "message": "Semester deleted successfully"  
}
```

Les sous-ressources

Méthode : GET

Endpoint : /api/sub-resources

Description : Récupère la liste des sous-ressources avec filtres optionnels.

Paramètres (query) :

- id_resource (*entier, optionnel*) - Filtrer par ID de ressource
- id_user (*entier, optionnel*) - Filtrer par ID d'utilisateur

Réponse :

```
[  
  {  
    "id": 1,  
    "name": "Sous-ressource A",  
    "total_hours": 20,  
    "id_resources": 5,  
    "id_users": 2  
  }  
]
```

Méthode : GET

Endpoint : /api/sub-resources/{id}

Description : Récupère les détails d'une sous-ressource spécifique.

Réponse :

```
{  
  "id": 1,  
  "name": "Sous-ressource A",  
  "total_hours": 20,  
  "id_resources": 5,  
  "id_users": 2  
}
```

Méthode : POST

Endpoint : /api/sub-resources

Description : Crée une nouvelle sous-ressource.

Corps de la requête (JSON) :

```
{  
  "name": "Sous-ressource B",  
  "total_hours": 15,  
  "id_resources": 5,  
  "id_users": 3  
}
```

Réponse :

```
{  
  "message": "SubResource created successfully",  
  "id": 2,  
  "name": "Sous-ressource B",  
  "total_hours": 15,  
  "id_resources": 5,  
  "id_users": 3  
}
```

Méthode : PUT

Endpoint : /api/sub-resources/{id}

Description : Met à jour une sous-ressource spécifique.

Corps de la requête (JSON) :

```
{  
  "name": "Sous-ressource B Modifiée",  
  "total_hours": 18  
}
```

Réponse :

```
{  
  "message": "SubResource updated successfully"  
}
```

Méthode : DELETE

Endpoint : /api/sub-resources/{id}

Description : Supprime une sous-ressource spécifique.

Réponse :

```
{  
  "message": "SubResource deleted successfully"  
}
```

III – Tests

Nous avons décidé de tester toutes les entités du projet, ainsi que leurs contrôleurs et leurs *repositories*. De plus, nous testons les commandes, la sécurité via OAuth2Authenticator et le service de *School/Year*. Pour cela, nous avons dû installer *PHPUnit*. À la suite de son installation, plusieurs fichiers sont apparus tels que ".phpunit.result.cache" comportant le cache des tests effectués, ou "phpunit.xml" permettant le lancement du bon répertoire de tests et ressemblant à :

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- https://phpunit.readthedocs.io/en/latest/configuration.html -->
<phpunit xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="vendor/phpunit/phpunit/phpunit.xsd"
  backupGlobals="false"
  colors="true"
  bootstrap="tests/bootstrap.php"
>
  <php>
    <ini name="display_errors" value="1" />
    <ini name="error_reporting" value="-1" />
    <server name="APP_ENV" value="test" force="true" />
    <server name="SHELL_VERBOSITY" value="-1" />
    <server name="SYMFONY_PHPUNIT_REMOVE" value="" />
    <server name="SYMFONY_PHPUNIT_VERSION" value="9.5" />
  </php>

  <testsuites>
    <testsuite name="Project Test Suite">
      <directory suffix="Test.php">tests</directory>
    </testsuite>
  </testsuites>

  <extensions>
  </extensions>
</phpunit>
```

Voici un exemple de tests sur une entité nommée Formation :

```
/* Cette classe de test vérifie que les méthodes de l'entité Formation fonctionnent correctement. */
```

```
class FormationTest extends TestCase
{
```

```
/* Cette instance de Formation est utilisée pour les tests */
private Formation $formation;

/* Méthode exécutée avant chaque test pour initialiser une nouvelle instance
de Formation pour chaque test afin de les isoler. */
protected function setUp(): void
{
    $this->formation = new Formation();
}

/* Test des méthodes getter et setter, pour le champ label dans ce cas, afin
de vérifier que la valeur est récupérée et que la valeur null est gérée. */
public function testSetAndGetLabel(): void
{
    $label = 'Computer Science';
    $this->formation->setLabel($label);
    $this->assertEquals($label, $this->formation->getLabel());
    $this->formation->setLabel(null);
    $this->assertNull($this->formation->getLabel());
}

/* Test des méthodes d'ajout et suppression d'interruptions pédagogiques,
afin de vérifier qu'elles fonctionnent correctement et que les relations sont
maintenues. */
public function testAddAndRemovePedagogicalInterruption(): void
{
    $interruption = new PedagogicalInterruptions();

    // Vérifie que l'interruption est ajoutée à la collection et que la relation est
établie
    $this->formation->addPedagogicalInterruption($interruption);
    $this->assertCount(1, $this->formation->getPedagogicalInterruptions());
    $this->assertSame($this->formation, $interruption->getIldFormation());

    // Vérifie que l'interruption est supprimée de la collection et que la relation
est rompue
    $this->formation->removePedagogicalInterruption($interruption);
    $this->assertCount(0, $this->formation->getPedagogicalInterruptions());
    $this->assertNull($interruption->getIldFormation());
}
}
```

Voici un exemple de tests sur un contrôleur nommée Formation :

/* Cette classe vérifie que les méthodes du contrôleur Formation fonctionnent en utilisant des mocks pour simuler les services. */

```
class FormationControllerTest extends TestCase
{
    private FormationController $controller;
    /** @var MockObject&EntityManagerInterface */
    private MockObject $entityManager;
    /** @var MockObject&FormationRepository */
    private MockObject $repository;
    private ContainerInterface $container;
    private SerializerInterface $serializer;
    /** @var MockObject&SchoolYearService */
    private MockObject $schoolYearService;
    private SchoolYearRepository $schoolYearRepository;
```

/* Mise en place des tests avec PHPUnit */

```
protected function setUp(): void
{
    // Création de mocks pour les dépendances du contrôleur
    $this->entityManager = $this->createMock(EntityManagerInterface::class);
    $this->repository = $this->createMock(FormationRepository::class);
    $this->container = $this->createMock(ContainerInterface::class);
    $this->serializer = $this->createMock(SerializerInterface::class);
    $this->schoolYearRepository =
    $this->createMock(SchoolYearRepository::class);
    $this->schoolYearService = $this->createMock(SchoolYearService::class);

    // Mock du sérialiseur pour retourner une version JSON des données
    $this->serializer->method('serialize')->willReturnCallback(fn ($data) =>
    json_encode($data));

    // Configuration du container pour retourner le sérialiseur
    $this->container->method('has')->willReturn(true);
    $this->container->method('get')->with('serializer')->willReturn($this->serializer);

    // Initialisation du contrôleur avec un service mocké
    $this->controller = new FormationController($this->schoolYearService);
    $this->controller->setContainer($this->container);
```

```
}
```

// Test de la création d'une nouvelle formation avec des données valides

```
public function testCreate(): void
{
    $schoolYearMock = $this->createMock(SchoolYear::class);
    $this->schoolYearService
        ->method('getCurrentSchoolYear')
        ->willReturn($schoolYearMock);

    $data = ['label' => 'New Formation', 'order_number' => 3];

    $json = json_encode($data);
    if ($json === false) {
        throw new RuntimeException('Failed to encode data as JSON');
    }

    $request = new Request([], [], [], [], [], ['CONTENT_TYPE' => 'application/json'],
    $json);

    $response = $this->controller->create($request, $this->entityManager,
    $this->schoolYearRepository, $this->schoolYearService);
    $this->assertIsString($response->getContent());
    $content = json_decode($response->getContent(), true);

    $this->assertEquals(Response::HTTP_CREATED,
    $response->getStatusCode());
    $this->assertEquals('New Formation', $content['label']);
}
```

// Test de la création d'une formation avec des données invalides (JSON malformé)

```
public function testCreateInvalidData(): void
{
    $request = new Request([], [], [], [], [], ['CONTENT_TYPE' => 'application/json'],
    'invalid json');

    $response = $this->controller->create($request, $this->entityManager,
    $this->schoolYearRepository, $this->schoolYearService);
```



```
$this->assertEquals(Response::HTTP_BAD_REQUEST,  
$response->getStatusCode());  
$this->assertIsString($response->getContent());  
$content = json_decode($response->getContent(), true);  
$this->assertArrayHasKey('error', $content);  
}  
}
```

Voici un exemple de tests sur un *repository* nommée FormationRepository :

/* Cette classe de test vérifie que les méthodes du repository FormationRepository fonctionnent correctement. */

```
class FormationRepositoryTest extends KernelTestCase
```

```
{
```

```
    private ?EntityManagerInterface $entityManager;
```

```
    private FormationRepository $repository;
```

```
    // Configuration avant chaque test
```

```
    protected function setUp(): void
```

```
    {
```

```
        $kernel = self::bootKernel();
```

```
        /** @var ManagerRegistry $doctrine */
```

```
        $doctrine = $kernel->getContainer()->get('doctrine');
```

```
        /** @var EntityManagerInterface $entityManager */
```

```
        $entityManager = $doctrine->getManager();
```

```
        $this->entityManager = $entityManager;
```

```
        $this->entityManager->beginTransaction();
```

```
        /** @var FormationRepository repository */
```

```
$repository = $this->entityManager->getRepository(Formation::class);  
$this->repository = $repository;  
}
```

// Teste si le repository est une instance de FormationRepository et de ServiceEntityRepository

```
public function testConstruct(): void  
{  
    $this->assertInstanceOf(FormationRepository::class, $this->repository);  
    $this->assertInstanceOf(ServiceEntityRepository::class, $this->repository);  
}
```

// Teste la méthode getHoursByGroups du repository

```
public function testGetHoursByGroups(): void  
{  
    $mockRepository = $this->getMockBuilder(FormationRepository::class)  
        ->disableOriginalConstructor()  
        ->onlyMethods(['createQueryBuilder'])  
        ->getMock();  
  
    $queryBuilder = $this->createMock(QueryBuilder::class);  
  
    $query = $this->createMock(AbstractQuery::class);  
  
    $expectedResult = [  
        [  
            'course_type_id' => 1,  
            'course_type_name' => 'Lecture',
```

```
'sub_resource_id' => 101,
'sub_resource_name' => 'Mathematics',
'sub_resource_hours' => 45.0,
'total_hours' => 30.5
],
[
  'course_type_id' => 2,
  'course_type_name' => 'Exercise',
  'sub_resource_id' => 101,
  'sub_resource_name' => 'Mathematics',
  'sub_resource_hours' => 45.0,
  'total_hours' => 15.0
]
];
```

```
$mockRepository->expects($this->once())
    ->method('createQueryBuilder')
    ->with('f')
    ->willReturn($queryBuilder);
```

```
$queryBuilder->expects($this->once())
    ->method('select')
    ->with(
        'ct.id as course_type_id',
        'ct.name as course_type_name',
        'sr.id as sub_resource_id',
        'sr.name as sub_resource_name',
        'sr.total_hours as sub_resource_hours',
```

```

        'SUM(a.allocated_hours) as total_hours'
    )
    ->willReturnSelf();

$expectedJoins = [
    ['f.semesters', 's', null, null],
    ['s.resources', 'r', null, null],
    ['r.subResources', 'sr', null, null],
    ['sr.assignments', 'a', null, null]
];

$joinIndex = 0;
$queryBuilder->expects($this->exactly(4))
    ->method('leftJoin')
        ->willReturnCallback(function($join, $alias) use (&$joinIndex,
$expectedJoins, $queryBuilder) {
            $this->assertLessThan(count($expectedJoins), $joinIndex, 'Too many
leftJoin calls');

            $this->assertEquals($expectedJoins[$joinIndex][0], $join, "Join
{$joinIndex} should be {$expectedJoins[$joinIndex][0]}");

            $this->assertEquals($expectedJoins[$joinIndex][1], $alias, "Alias
{$joinIndex} should be {$expectedJoins[$joinIndex][1]}");

            $joinIndex++;

            return $queryBuilder;
        });

$queryBuilder->expects($this->once())
    ->method('innerJoin')
    ->with('a.id_course_types', 'ct')

```

```
->willReturnSelf();
```

```
$queryBuilder->expects($this->once())
```

```
->method('where')
```

```
->with('f.id = :formationId')
```

```
->willReturnSelf();
```

```
$queryBuilder->expects($this->once())
```

```
->method('setParameter')
```

```
->with('formationId', 42)
```

```
->willReturnSelf();
```

```
$queryBuilder->expects($this->once())
```

```
->method('groupBy')
```

```
->with('ct.id', 'ct.name', 'sr.id', 'sr.name')
```

```
->willReturnSelf();
```

```
$queryBuilder->expects($this->once())
```

```
->method('getQuery')
```

```
->willReturn($query);
```

```
$query->expects($this->once())
```

```
->method('getResult')
```

```
->willReturn($expectedResult);
```

```
$result = $mockRepository->getHoursByGroups(42);
```

```
$this->assertEquals($expectedResult, $result);
```

```
$this->assertCount(2, $result);

$this->assertEquals(1, $result[0]['course_type_id']);

$this->assertEquals('Lecture', $result[0]['course_type_name']);

$this->assertEquals(101, $result[0]['sub_resource_id']);

$this->assertEquals('Mathematics', $result[0]['sub_resource_name']);

$this->assertEquals(30.5, $result[0]['total_hours']);

}

protected function tearDown(): void
{
    if ($this->entityManager !== null) {
        $this->entityManager->rollback();
        $this->entityManager->close();
        $this->entityManager = null;
    }

    parent::tearDown();
}
}
```

Pour plus d'information, un mock est un objet simulé. Il est utilisé dans les tests unitaires afin d'imiter le comportement d'une dépendance réelle. Il permet de tester un composant de manière isolée, sans utilisation d'une base de données. Par le biais d'un mock, il est possible de définir les valeurs retournées par les méthodes, tout en vérifiant que ces dernières soient appelées avec les bons paramètres.

IV - Développement Frontend

Pour le développement Frontend, nous avons créé un service d'API pour faire des appels API simplement pour chaque entité de la base de données. Avec pour

chacun où l'on récupère un élément, la liste des éléments, créer un élément, modifier un élément et supprimer un élément.

```
ApiService.pedagogicalInterruptions.fetchPedagogicalInterruption(pild)
```

Il inclut un système facile d'utilisation pour filtrer les éléments récupérés ou rechercher, et aussi pour compléter les données.

Pour l'option de complétion des données 'enrichData', s'il vaut 'true', toutes les données seront récupérées récursivement par rapport aux données non complètes (données ayant seulement leur identifiant de récupérer). Sinon, on peut mettre un tableau comportant les données à récupérer récursivement, voici un exemple :

```
teachers = await ApiService.teachers.fetchCourseTeachers({  
  id_sub_resource: subResource.id  
}, ['user', 'group'])
```

Le premier paramètre étant pour filtrer par sous-ressource et le deuxième paramètre étant pour récupérer les données complètes souhaitées (utilisateur et groupe).

Nous avons aussi créé des composants généraux, par exemple pour les dialogues qui sont très personnalisables avec la possibilité de faire un menu de navigation à l'intérieur.

Pour la gestion des années scolaires, nous avons dans un fichier javascript la récupération et le changement local pour l'administrateur via un store 'SchoolYearStore'.

Nous avons également créé un fichier 'utils.js' qui contient des fonctions utiles pour l'application.

Un fichier 'logger.js' pour logger les erreurs en console lors du développement et en cache en production.

```
logger.error('Error fetching users', error)
```

Un fichier 'notify.js' pour afficher les notifications de succès et d'erreur dans l'application.

```
errorNotify('Veuillez remplir tous les champs obligatoires.')  
successNotify('Modifications enregistrées avec succès')
```

Un fichier 'dialog.js' pour demander l'accord à l'utilisateur avant d'exécuter une action.

```
if (!await confirmDialog('Êtes-vous sûr de vouloir supprimer cette ressource ?')) {  
  return  
}
```

Un fichier 'rules.js' pour la gestion rapide de règles pour les champs de formulaires.

```
:rules="[rules.required, rules.maxLength(80)]"
```

V - Déploiement

Concernant le déploiement, il faut commencer par cloner le projet git, et modifier la valeur de la variable APP_ENV, en la passant de "dev" à "prod". Suite à cela, la commande "./init.sh" doit être exécutée.

A cette étape, pour créer un utilisateur, il est nécessaire d'utiliser la commande "docker exec -it api sh, puis php bin/console create-user" dans le terminal, et suivre les instructions fournies dans ce dernier (en sachant que le rôle à informer doit être "ROLE_ADMIN" dans ce cas).