

Rapport de projets

Le titre professionnel Concepteur Développeur d'Application

Rany ALO

Juillet 2023

Sommaire

I. Remerciements	4
II. Projet : Code Hub	5
1. La liste des compétences couvertes par le projet	5
2. Présentation du projet en anglais	6
3. Résumé du projet	7
4. Spécifications fonctionnelles du projet	7
5. Spécifications techniques du projet	8
6. Étapes de réalisation	10
♦ Construire l'équipe et organiser l'environnement de développement	10
♦ L'analyse des besoins	11
♦ UML (Unified Modeling Language)	11
– Diagramme de cas d'utilisation	12
– Diagramme de classes	13
♦ La maquette du projet CodeHub	14
♦ Code Hub est une application multicouche	15
♦ La base de données	17
– Le MCD	17
– Le MLD	18
– Créer la base de données	18
♦ Création de l'API	21
– Création de l'entité User	22
– State Provider	23
– State Processor	24
– JWT authentication	25
– Les tests unitaires	27
– Les tests fonctionnels	28

– Le service generateArticle avec OpenAi	30
– Déploiement de l'API sur PLESK	35
♦ Interface utilisateur mobile	37
– Arborescence	37
– Les composants	38
– Les screens	39
– Ajout et appelle d'une route	40
– Choix de la navigation	41
♦ Interface utilisateur web	43
– Mettre en place la gestion des routes	43
– Présentation du jeu d'essai d'une fonctionnalité	44
♦ Interface utilisateur desktop	47
– Le fichier login.py	47
– Le fichier generator.py	47
– Le fichier main.py	48
♦ La sécurité de l'API	49
♦ Une situation de travail ayant nécessité une recherche à partir de site anglophone	50
7. Conclusion	51
8. Annexes	52

REMERCIEMENTS

Tout d'abord, je tiens à remercier la direction de La Plateforme représentée par Ruben Habib, sa disponibilité et surtout ses judicieux conseils, qui ont contribué à alimenter ma réflexion.

Je désire aussi remercier les formateurs Nicolas Revel, Nicolas Acard, Celine Emptoz-Lacote et Clément CAILLAT, qui m'ont fourni les outils nécessaires à la réussite dans cette formation. Ils étaient toujours disponibles pour répondre à toutes nos questions, même les plus banales.

Je tiens à remercier spécialement Julie Steiner, la Chargée des relations entreprises, m'a apporté une aide précieuse dans ma recherche d'entreprise d'accueil pour mon alternance.

Un grand merci à mes compagnons dans cette formation, leur soutien inconditionnel et leurs encouragements ont été d'une grande aide.

Enfin, je remercie ma famille, ma femme, mes deux enfants Gabriel et Eliana, qui ont vécu au quotidien mes préoccupations tout au long de la période de la formation.

Projet CodeHub

Liste des compétences couvertes par le projet

Le projet CodeHub couvre les compétences suivantes de la partie « Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité : » :

- Maquetter une application
- Développer une interface utilisateur de type desktop
- Développer des composants d'accès aux données
- Développer la partie front-end d'une interface utilisateur web
- Développer la partie back-end d'une interface utilisateur web

Les compétences suivantes de la partie « Concevoir et développer la persistance des données en intégrant les recommandations de sécurité » :

- Concevoir une base de données
- Mettre en place une base de données
- Développer des composants dans le langage d'une base de données

Et les compétences suivantes de la partie « Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité » :

- Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement
- Concevoir une application
- Développer des composants métier
- Construire une application organisée en couches
- Développer une application mobile
- Préparer et exécuter les plans de tests d'une application
- Préparer et exécuter le déploiement d'une application

Présentation du projet en anglais

The Code Hub project is an online discussion forum designed to allow users to share articles and comment on other users' articles. This forum can be adapted and utilized to create specialized discussion communities in various domains such as technology, cooking, sports, and more. This flexibility enables users to engage and exchange ideas on topics that interest them. The specifications for developing this forum were provided by our school, La Plateforme.

I have developed three user interfaces as part of the project: a mobile interface using React Native, collaboratively with the team, a web interface, a solo project, implementing an admin panel using React JS, and a desktop interface, also a solo project, involving an AI article generator utilizing the OpenAI API with Python.

Throughout these development tasks, I gained valuable experience in utilizing modern frameworks and technologies to create user-friendly and efficient interfaces across multiple platforms. Working both collaboratively and independently, I acquired proficiency in React Native, React JS, and Python, expanding my skills as a versatile developer in various application development scenarios.

Résumé du projet

Code Hub est un forum de discussion en ligne conçu pour permettre aux utilisateurs de partager des articles et de commenter les articles des autres. Le cahier des charges pour le développement de ce forum a été fourni par notre école La plateforme.

Spécifications fonctionnelles du projet

L'application permet aux visiteurs sans compte de consulter les articles et les commentaires, mais ils ne peuvent pas en ajouter eux-mêmes. Les utilisateurs avec un compte peuvent ajouter des articles et des commentaires. Ils ont également la possibilité de modifier leur profil et de supprimer leur compte si nécessaire.

Les administrateurs ont les mêmes droits qu'un utilisateur standard, mais disposent également de fonctionnalités supplémentaires. Ils peuvent modifier, supprimer et ajouter des utilisateurs au forum. De plus, ils peuvent supprimer des articles ou des commentaires en cas de violation des règles du forum.

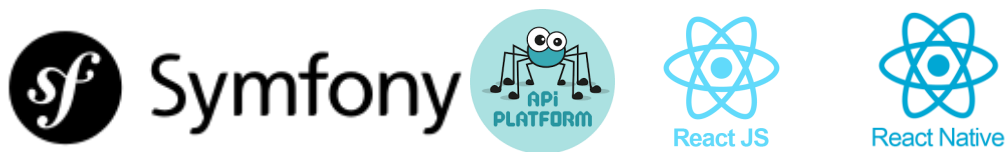
Le projet CodeHub comprend trois parties distinctes : une application mobile, une interface web et une interface desktop.

- L'application mobile est accessible à tous les utilisateurs, qu'ils soient visiteurs, utilisateurs enregistrés ou administrateurs.
- L'interface web est réservée aux administrateurs. Ces derniers disposent de fonctionnalités supplémentaires par rapport aux utilisateurs standards. Ils peuvent gérer les utilisateurs, les articles et les commentaires, notamment en effectuant des ajouts, modifications et suppressions en cas de violation des règles du forum.
- L'interface desktop est également réservée aux administrateurs et aux utilisateurs disposant de comptes. Elle permet de générer des articles à partir de titres grâce à l'outil OpenAI, et de publier ces articles sur le forum.

Spécifications techniques du projet

- Les langages et les Frameworks utilisés dans ce projet :

- Symfony et API Platform : Symfony est un framework PHP open source. API Platform est une extension de Symfony qui permet de créer des API web RESTful. Nous avons utilisé Symfony et API Platform pour créer l'API backend de l'application.
- React JS : C'est une bibliothèque JavaScript open source pour la création d'interfaces utilisateur. Nous avons utilisé React JS pour développer l'interface frontend de l'application.
- React Native est une bibliothèque JavaScript open source développée par Facebook pour créer des applications mobiles multiplateformes pour iOS et Android en utilisant une seule base de code.
- PHP
- JavaScript
- Python
- MYSQL MariaDB



- Les outils utilisés dans ce projet :

- Visual Studio Code (VSCode) est un éditeur de code source gratuit.
- PhpStorm est un environnement de développement intégré (IDE) conçu spécifiquement pour le développement en PHP. J'ai utilisé PhpStorm pour créer l'api symfony car il offre une gamme de fonctionnalités avancées pour faciliter le processus de développement.
- Symfony et API Platform : Symfony est un framework PHP open source. API Platform est une extension de Symfony qui permet de créer des API web RESTful. J'ai utilisé Symfony et API Platform pour créer l'API backend de l'application.
- Figma pour la maquette
- GitHub : C'est une plateforme de développement logiciel qui permet de stocker et de partager du code source.
- PhpMyAdmin : une application web open source pour gérer des bases de données MySQL et MariaDB.
- Doctrine : ORM (Object-Relational Mapping) pour PHP, qui permet de gérer les

interactions avec une base de données de manière objet.

- PhpMyAdmin : une application web open source pour gérer des bases de données MySQL et MariaDB.
- Mocodo : un outil en ligne qui permet de concevoir des modèles de données de manière graphique, on a utilisé Mocodo pour générer le modèle conceptuel de données (MCD).
- DBDiagram est un outil en ligne gratuit permettant de concevoir et de modéliser visuellement des bases de données relationnelles, on a utilisé dbdiagram pour générer le Modèle Logique de Données MLD.
- Trello : C'est un outil de gestion de projet en ligne basé sur la méthode Kanban. Nous avons utilisé Trello pour organiser les différentes tâches à effectuer pour réaliser le projet et pour suivre leur progression.
- Lucidchart : est un outil en ligne de création de diagrammes et de visualisations de données. Nous avons utilisé Lucidchart pour créer les diagrammes UML.

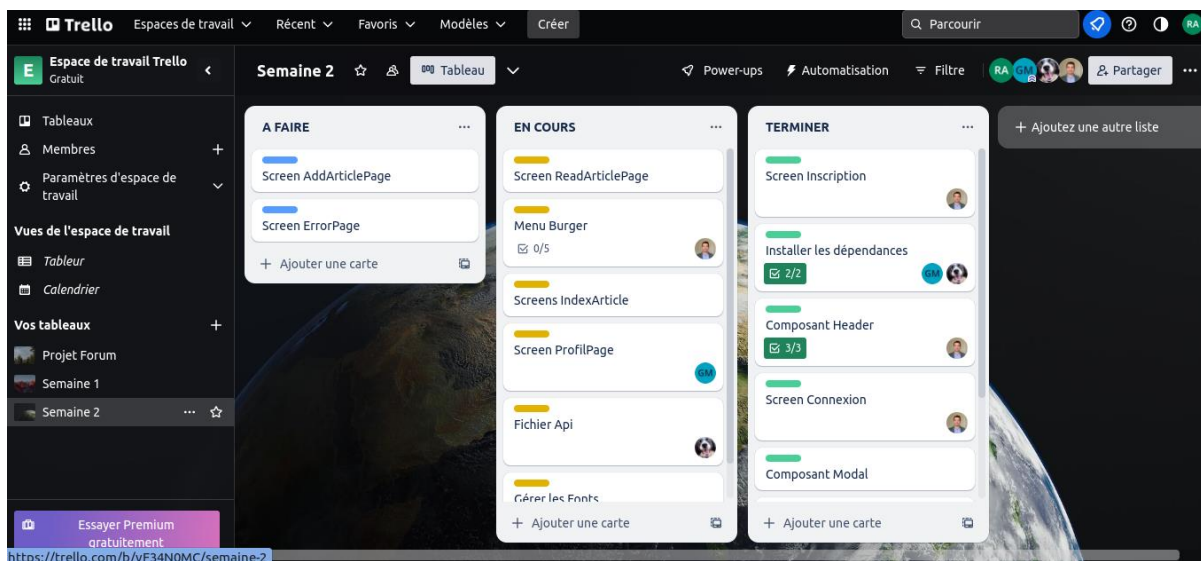
Étapes de réalisation

Construire l'équipe et organiser l'environnement de développement

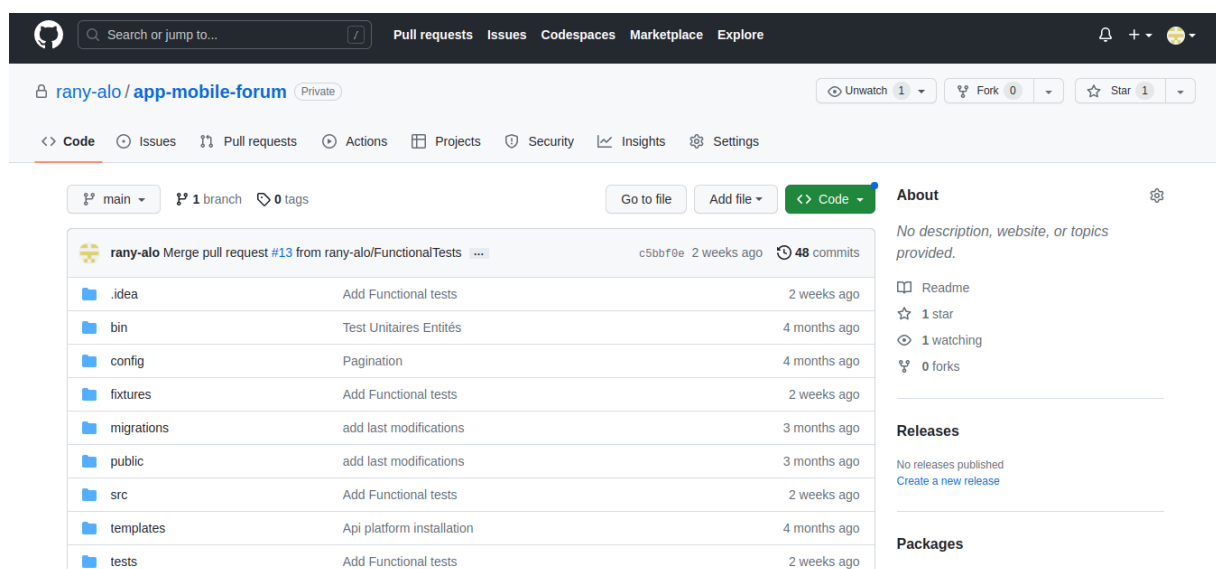
La première étape dans la réalisation d'un projet informatique est de construire une équipe solide de développeurs qui peuvent collaborer efficacement pour atteindre les objectifs du projet, nous avons constitué une équipe de quatre développeurs et utilisé une combinaison de méthodes Scrum et Kanban avec Trello pour suivre les tâches. L'équipe de développeurs de projet CodeHub :

- Christophe CALMES
- Rany ALO
- Geoffrey MECA
- Mathieu RUIZ

Le projet a été divisé en petites tâches pour faciliter le travail et assurer une bonne gestion du temps.



Nous avons également utilisé GitHub pour le suivi des versions et le partage du code.



L'analyse des besoins

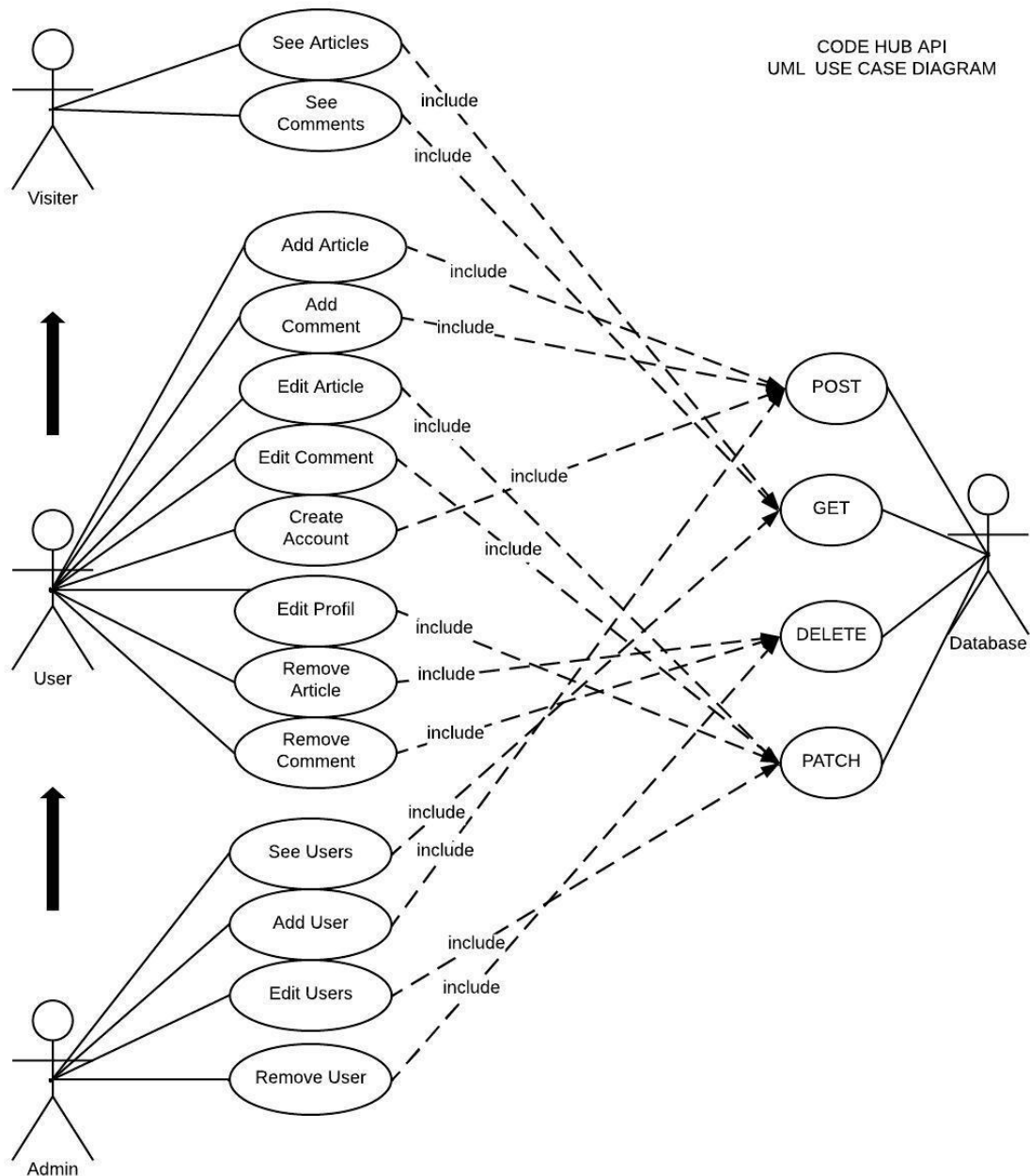
L'analyse des besoins est une étape cruciale dans la conception d'une application, car elle permet de déterminer les exigences et les attentes des utilisateurs. Dans le cadre de notre projet, nous avons reçu un cahier des charges de notre école qui a servi de base pour identifier les fonctionnalités à implémenter. Cette analyse nous a permis de comprendre les besoins des utilisateurs, d'anticiper les problématiques et d'élaborer une application adaptée.

UML (Unified Modeling Language)

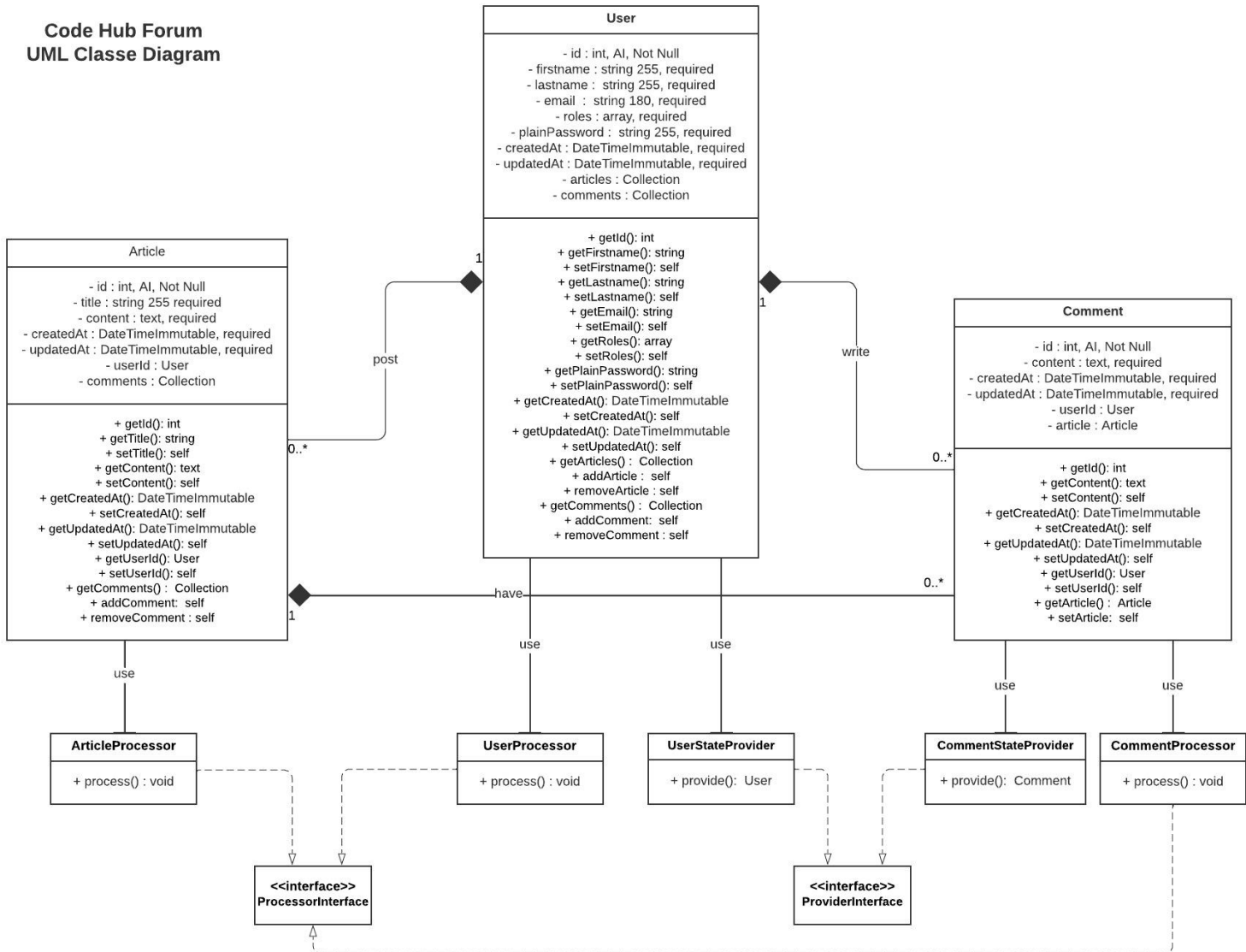
L'UML est utilisé pour modéliser et documenter des systèmes logiciels. Il fournit un langage graphique standardisé qui permet aux développeurs, aux concepteurs et aux architectes de communiquer efficacement entre eux et avec les parties prenantes. L'UML permet de représenter visuellement les différentes parties d'un système, les interactions entre ces parties et les comportements attendus.

Pour la conception de notre projet, nous avons décidé d'utiliser La diagramme de cas d'utilisation et le diagramme de classes :

- **Création de l'UML Diagramme de cas d'utilisation** : permet de visualiser les fonctionnalités de notre application et de faciliter la compréhension des besoins des utilisateurs.



- **Création de l'UML Diagramme de classes** : Le diagramme de classes UML permet de visualiser la structure de notre modèle de données et de faciliter la communication entre les membres de l'équipe de développement.



La maquette du projet CodeHub

CodeHub est un projet commun, il a donc été nécessaire dans un premier temps de créer une maquette de l'application qui réponde à la fois au cahier des charges qui nous a été soumis par La plateforme, mais aussi qu'elle réponde à des critères esthétiques, veillant à ce que la lisibilité soit bonne, vu que nous avons créé un forum, une application permettant de lire des articles, les écritures ou les commenter.

Nous nous sommes donc appuyés sur divers critères pour créer la maquette qui nous a guidé durant tout le développement de notre application.

La maquette a été réalisée sur Figma. Elle est visible [ici](#).

Le choix de couleur a été essentiellement défini pour rendre la lecture des articles lisible, mais proposé une touche d'originalité, le choix du bleu majoritaire et du blanc pour l'écriture a vite été adopté par l'équipe.

J'ai défini la palette de couleur avec ce [site](#).



Je me suis assuré que mon choix de contraste de couleur était bien lisible avec un outil [d'adobe color](#).

Le Wireframe a permis à l'issue de la réunion préparatoire à la création de la maquette de mettre au propre les idées de chacun et de créer un parcours utilisateur simplifié sans s'attacher à une réalisation de maquette très détaillée. Cela a permis aussi de valider le parcours utilisateur de manière intuitive et simplifiée.



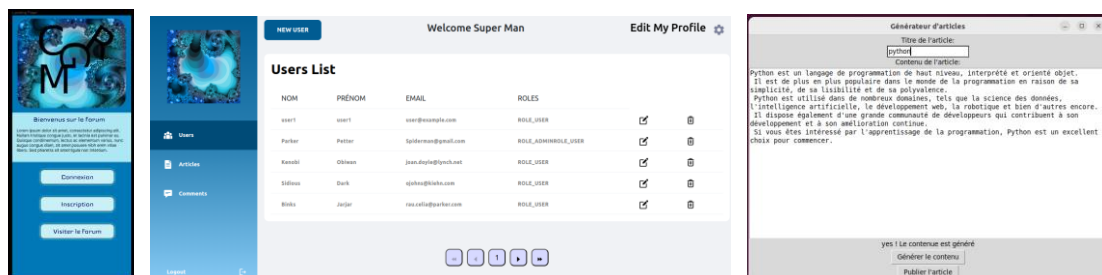
La maquette a été réalisée une fois que le wireframe a été validé par l'ensemble des membres de l'équipe.



Code Hub est une application multicouche

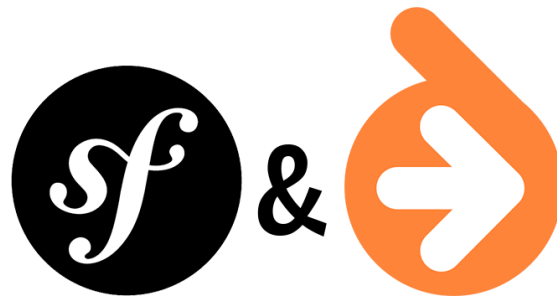
Développement de notre application Code Hub organisée en quatre couches :

1. La couche de présentation : Cette couche comprend toutes les vues de notre application, y compris les écrans que les utilisateurs voient et avec lesquels ils interagissent. Elle est composée des écrans React Native de votre application pour les utilisateurs mobiles, ainsi que d'une interface desktop en Python et d'une interface web en React JS. Ces interfaces permettent aux utilisateurs d'accéder à l'application, de visualiser les articles, les commentaires et les utilisateurs (uniquement pour l'administrateur), de créer de nouveaux articles et commentaires, ainsi que de mettre à jour ou supprimer les articles et les commentaires existants.



2. La couche de logique d'application : Cette couche contient la logique métier de notre application, c'est-à-dire les règles qui déterminent comment les données sont manipulées et gérées. Cette couche est principalement gérée par l'API Platform et Symfony, qui fournissent des fonctionnalités pour gérer les entités (User, Article, Comment) et effectuer des opérations CRUD (créer, lire, mettre à jour, supprimer) sur ces entités. La couche de logique d'application utilise également des services pour effectuer des tâches spécifiques, telles que l'authentification et l'autorisation des utilisateurs.
3. La couche d'accès aux données : Cette couche est responsable de la communication

avec la source de données, dans notre cas la base de données. Cette couche est également gérée par l'API Platform et Symfony, qui utilisent Doctrine pour communiquer avec la base de données et mapper les données en entités PHP.



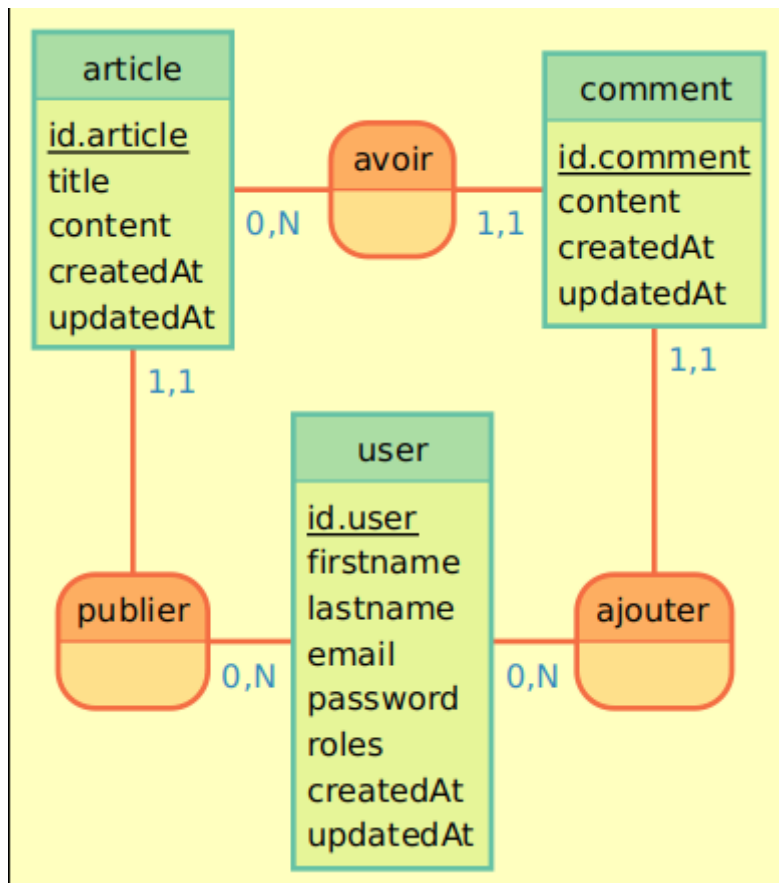
4. La couche de services : Cette couche comprend les différents services de notre application, qui fournissent des fonctionnalités spécifiques à différentes parties de notre application. Dans notre cas, nous avons des services pour gérer l'authentification des utilisateurs et un service qui appelle l'API OpenAi pour générer un article depuis son titre. Ces services sont également gérés par l'API Platform et Symfony.



La base de données

- Le MCD

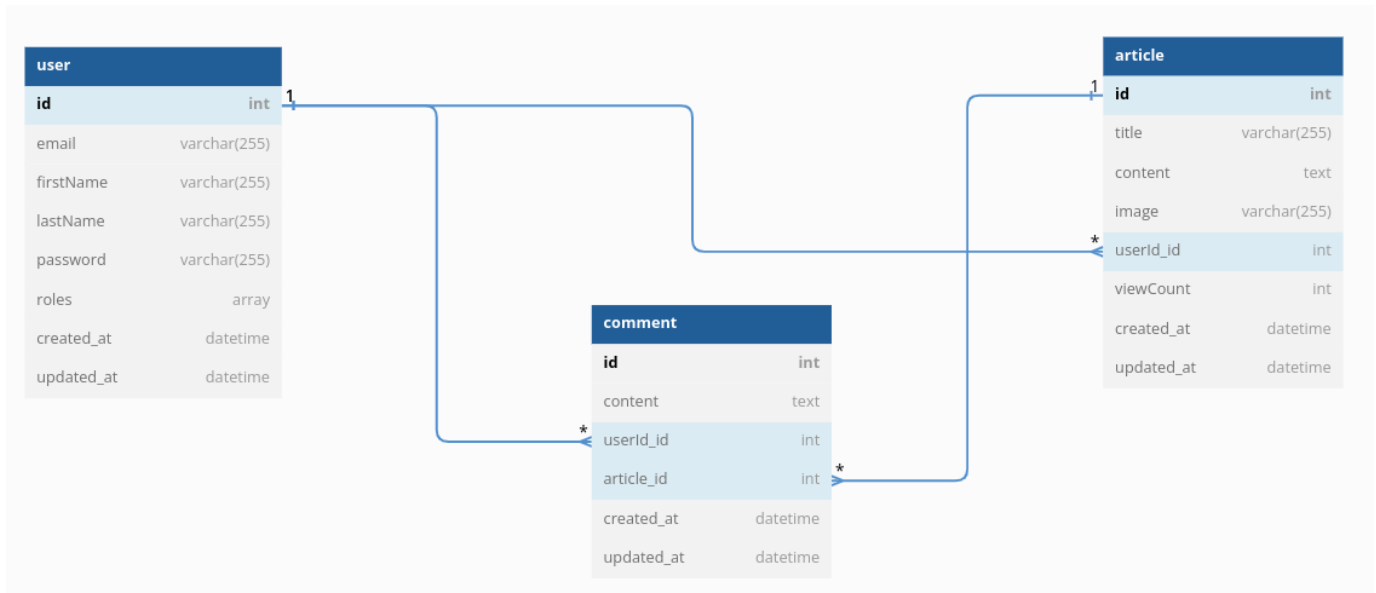
Pour notre application Code Hub, nous avons besoin d'une base de données qui contient trois tables : "user", "article" et "comment". La conception de la base de données a commencé par la création d'un Modèle Conceptuel de Données (MCD) en utilisant la méthodologie Merise à l'aide de l'application web MOCODO. Le MCD représente les entités, les relations et les attributs des données de l'application.



- L'utilisateur peut publier zéro ou plusieurs articles mais l'article ne peut pas avoir qu'un gestionnaire.
- L'utilisateur peut ajouter zéro ou plusieurs commentaires mais le commentaire ne peut pas avoir qu'un gestionnaire.
- Un article peut avoir zéro ou plusieurs commentaires, mais un commentaire ne peut appartenir qu'à un seul article.

- Le MLD :

À partir du MCD, nous avons créé le Modèle Logique de Données (MLD) à l'aide de l'application web dbdiagram, qui décrit la structure de la base de données sous forme de tables, de colonnes et de relations entre les tables :



Les relations entre les tables :

User+article=> 1 to many => users.id sera un Foreign Key dans la table article
User+comment=> 1 to many => users.id sera un Foreign Key dans la table comment
article+comment=> 1 to many => type.id sera un Foreign Key dans la table passage

— Créer la base de données :

Tout d'abord, nous avons installé le support Doctrine via le orm Symfony pack, ainsi que le MakerBundle, qui aidera à générer du code :

```
$ composer require symfony/orm-pack
$ composer require --dev symfony/maker-bundle
```

Configuration de la base de données : Les informations de connexion à la base de données sont stockées dans une variable d'environnement appelée DATABASE_URL. Pour le développement, nous avons personnalisé ceci à l'intérieur de .env

```
#
# DATABASE_URL="sqlite://%kernel.project_dir%/var/data.db"
DATABASE_URL="mysql://api-forum:db_passwor@127.0.0.1:3306/rany_alo_api-forum?serverVersion=8&charset=
# DATABASE_URL="postgresql://app:!ChangeMe!@127.0.0.1:5432/app?serverVersion=14&charset=utf8"
###< doctrine/doctrine-bundle ###
```

Création de la base de données avec cette commande :

```
$ php bin/console doctrine:database:create
```

Création des tables de la base de données :

Nous avons utilisé l'ORM Doctrine de Symfony pour faciliter la création et la gestion des tables de notre base de données. Nous avons commencé par définir les entités de notre application, qui ont été converties en tables dans notre base de données grâce à Doctrine. Nous avons également ajouté des contraintes de clés étrangères pour garantir l'intégrité de nos données et prévenir toute erreur de référence.

```
rany@rany-ThinkPad-T430:~/Images/Captures d'écran/app-mobile-forum$ php bin/console make:entity

Class name of the entity to create or update (e.g. TinyElephant):
> Article

created: src/Entity/Article.php
created: src/Repository/ArticleRepository.php

Entity generated! Now let's add some fields!
You can always add more fields later manually or by re-running this command.

New property name (press <return> to stop adding fields):
> title

Field type (enter ? to see all types) [string]:
> string

Field length [255]:
>

Can this field be null in the database (nullable) (yes/no) [no]:
>

updated: src/Entity/Article.php

Add another property? Enter the property name (or press <return> to stop adding fields):
> 
```

Ensuite nous avons créé le fichier de migration contenant les requêtes SQL qui vont générer les tables :

```
$ php bin/console make:migration
```

Une fois la conception de notre base de données terminée, nous avons migré nos données vers la base de données avec cette commande :

```
$ php bin/console doctrine:migrations:migrate
```

Nous avons également utilisé PhpMyAdmin parce que c'est une application web conviviale pour la gestion de bases de données MySQL. Avec ses fonctionnalités étendues, nous pouvons facilement créer, modifier et gérer les tables, les requêtes et les utilisateurs. Accessible depuis n'importe quel navigateur, phpMyAdmin nous a offert une flexibilité pour gérer notre base de données à partir de n'importe où.

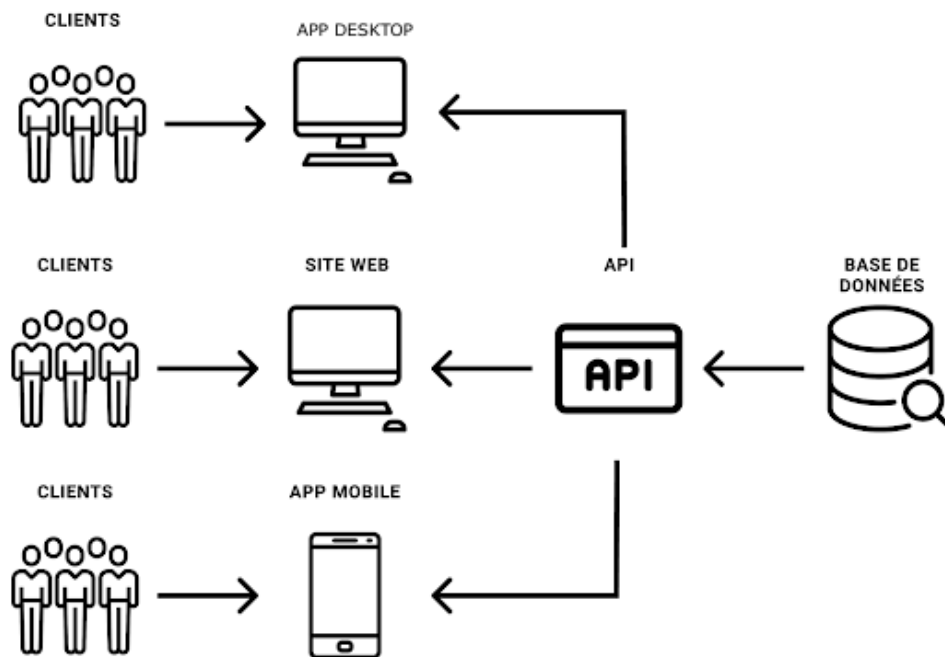
The screenshot shows the phpMyAdmin web interface. On the left is a sidebar with a tree view of databases and tables. The main panel displays the 'Structure' tab for the database 'rany_alo_api-forum'. It lists four tables: 'article', 'comment', 'doctrine_migration_versions', and 'user'. Each table row includes icons for actions like 'Parcourir', 'Structure', 'Rechercher', 'Insérer', 'Vider', and 'Supprimer'. A summary row at the bottom shows '4 tables' with a total of '54' lines and a size of '128,0 kio'.

Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/> article	★ Parcourir Structure Rechercher Insérer Vider Supprimer	20	InnoDB	utf8mb4_unicode_ci	32,0 kio	-
<input type="checkbox"/> comment	★ Parcourir Structure Rechercher Insérer Vider Supprimer	16	InnoDB	utf8mb4_unicode_ci	48,0 kio	-
<input type="checkbox"/> doctrine_migration_versions	★ Parcourir Structure Rechercher Insérer Vider Supprimer	1	InnoDB	utf8mb3_unicode_ci	16,0 kio	-
<input type="checkbox"/> user	★ Parcourir Structure Rechercher Insérer Vider Supprimer	17	InnoDB	utf8mb4_unicode_ci	32,0 kio	-
4 tables		Somme				
			54	InnoDB	utf8mb4_general_ci	128,0 kio 0 o

Création de l'API

Application Programming Interface (API) : Une Application Programming Interface (API) est une façade par laquelle un logiciel fournit des services à une autre application, la plupart du temps via une interface web.

- Permet d'exposer des ressources vers l'extérieur.
- Un seul back, plusieurs front (clients)



Nous avons mis en place un backend pour notre application de forum en créant une API. Pour cela, nous avons choisi d'utiliser Api Platform avec Symfony, qui est un outil très puissant pour la création d'API RESTful en PHP. Api Platform fournit des fonctionnalités avancées telles que la documentation de l'API, la gestion des erreurs, la validation des données, etc.

User				
POST	/api/inscription	Creates a User resource.	✓	🔒
GET	/api/profileMe	Retrieves a User resource.	✓	🔒
GET	/api/user/{id}	Retrieves a User resource.	✓	🔒
DELETE	/api/userDelete/{id}	Removes the User resource.	✓	🔒
PATCH	/api/userProfileEdit/{id}	Updates the User resource.	✓	🔒
GET	/api/users	Retrieves the collection of User resources.	✓	🔒

Nous avons créé des endpoints qui permettent de récupérer, ajouter, modifier et supprimer des données à partir de la base de données de l'application. Ces endpoints sont appelés par

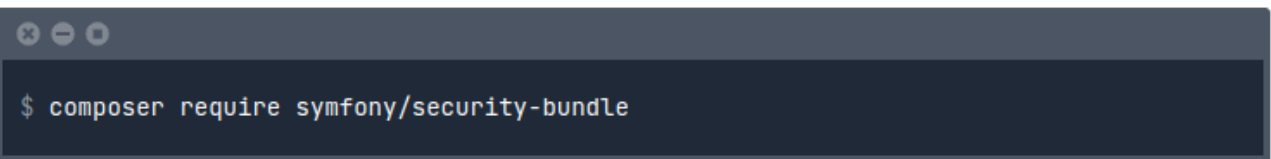
l'application frontend pour effectuer des opérations sur les données, telles que l'ajout d'un nouvel article, la suppression d'un commentaire, la modification d'un profil utilisateur, etc.

Pour faciliter l'accès aux données dans l'application, nous avons utilisé l'ORM Doctrine de Symfony. L'ORM (Object-Relational Mapping) est une technique de programmation qui permet de faire correspondre les objets d'une application avec les tables d'une base de données relationnelle. Avec Doctrine, nous avons créé des entités qui représentent les différents types de données dans l'application (par exemple, les articles, les commentaires, les utilisateurs, etc.) et nous avons défini les relations entre ces entités.

- Création de l'entité User :

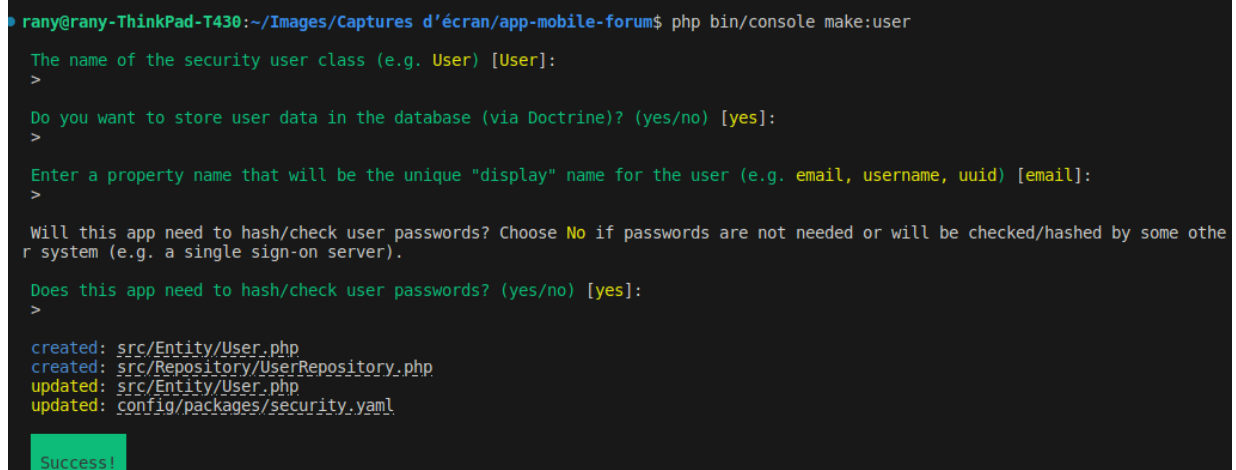
Les autorisations dans Symfony sont toujours liées à un objet User. Si vous avez besoin de sécuriser (des parties de) votre application, vous devez créer une classe User. Il s'agit d'une classe qui implémente `UserInterface`. Il s'agit souvent d'une entité Doctrine, mais vous pouvez également utiliser une classe User dédiée à la sécurité.

Nous devons d'abord installer le `SecurityBundle` avec la commande :



```
$ composer require symfony/security-bundle
```

Par la suite, nous avons exécuté la commande `make:user` afin de générer l'entité User, entraînant ainsi la création du champ password et la mise en place d'un mécanisme de hachage des mots de passe.



```
rany@rany-ThinkPad-T430:~/Images/Captures d'écran/app-mobile-forum$ php bin/console make:user
The name of the security user class (e.g. User) [User]:
>
Do you want to store user data in the database (via Doctrine)? (yes/no) [yes]:
>
Enter a property name that will be the unique "display" name for the user (e.g. email, username, uuid) [email]:
>
Will this app need to hash/check user passwords? Choose No if passwords are not needed or will be checked/hashed by some other system (e.g. a single sign-on server).
Does this app need to hash/check user passwords? (yes/no) [yes]:
>
created: src/Entity/User.php
created: src/Repository/UserRepository.php
updated: src/Entity/User.php
updated: config/packages/security.yaml

Success!
```

L'entité utilisateur (User) est définie en utilisant l'ORM Doctrine de Symfony. Elle est également exposée en tant que ressource API à travers ApiPlatform.

```
#[ORM\Entity(repositoryClass: UserRepository::class)]
#[ORM\Table(name: 'user')]
#[ApiResource(
    operations: array(
        new GetCollection(
            uriTemplate: '/users',
            normalizationContext: ['groups' => 'users.read'],
            security: "is_granted('ROLE_ADMIN')"
        ),
        new Post(
            uriTemplate: '/inscription',
            denormalizationContext: array('groups' => 'user.write'),
            processor: UserProcessor::class
        ),
        new Get(
            uriTemplate: '/profileMe',
            normalizationContext: array('groups' => 'user.profile.me'),
            security: "is_granted('ROLE_ADMIN') or object == user",
            provider: UserStateProvider::class
        ),
    ),
)]
```

La ressource API User a plusieurs opérations définies, notamment GetCollection pour récupérer une collection d'utilisateurs, Post pour créer un nouvel utilisateur, Get pour récupérer un utilisateur spécifique et Patch pour mettre à jour un utilisateur spécifique.

Chaque opération dispose d'un uriTemplate qui définit l'URL correspondante pour accéder à l'opération, ainsi que d'autres contextes de normalisation, de dénormalisation et de sécurité qui sont appliqués lors de l'accès à l'opération.

- State Provider :

Le State Provider est un concept clé dans API Platform qui permet de gérer l'état des ressources exposées par une API RESTful. En d'autres termes, le StateProvider est responsable de :

- ◆ Gère la récupération des données dans API Platform
- ◆ Permet de modifier la récupération des données
- ◆ Permet de récupérer les données depuis un autre service et de les intégrer à API Platform

```
class CommentStateProvider implements ProviderInterface
{
    public function __construct(private readonly ArticleRepository $articleRepository, private Security $security)
    {}

    public function provide(Operation $operation, array $uriVariables = [], array $context = []): object|array|null
    {
        $comment = new Comment();
        $comment->setUserId($this->security->getUser());
        $article = $this->articleRepository->find($uriVariables['id']);
        $comment->setArticle($article);

        return $comment;
    }
}
```

L'image ci-dessus est un exemple dans notre API, CommentStateProvider est un StateProvider personnalisé utilisé pour fournir une instance de la classe Comment lorsqu'une demande de création de ressource est effectuée. Il récupère l'article correspondant à l'ID spécifié dans l'URI à partir d'un ArticleRepository, crée une nouvelle instance de la classe Comment, y ajoute l'utilisateur actuel à partir de la classe Security, et définit l'article récupéré. La méthode provide renvoie ensuite cette nouvelle instance de Comment pour qu'elle soit persistée dans la source de données.

- State Processor :

Le State Processor est un concept clé dans API Platform qui permet de manipuler les données de la ressource avant ou après son enregistrement en base de données, ou avant ou après l'affichage des données lorsqu'une demande est effectuée. Le State Processor est responsable de :

- ◆ Gère la persistance des données dans API Platform
- ◆ Permet de modifier la persistance des données
- ◆ Permet d'effectuer des opérations spécifiques lors de la persistance des données


```

class ArticleProcessor implements ProcessorInterface
{
    public function __construct(private Security $security, private readonly EntityManagerInterface $entityManager)
    {}

    public function process(mixed $data, Operation $operation, array $uriVariables = [], array $context = []): void
    {
        if(false === $data instanceof Article) {
            return;
        }
        $data->setUpdatedAt(new \DateTimeImmutable());
        if($operation->getName() == "_api_/articleEdit/{id}_patch"){
            $data->setCreatedAt($data->getCreatedAt());
        }
        elseif($operation->getName() == "_api_/articlePost_post"){
            $data->setUserId($this->security->getUser());
            $data->setCreatedAt(new \DateTimeImmutable());
        }
        else {
            $data->setCreatedAt(new \DateTimeImmutable());
        }

        $this->entityManager->persist($data);
        $this->entityManager->flush();
    }
}

```

L'image ci-dessus est un exemple dans notre API, ArticleProcessor est un State Processor utilisée pour manipuler les données de l'entité Article avant et après son enregistrement dans la base de données, la méthode vérifie que les données sont bien une instance de la classe Article, puis modifie les propriétés updatedAt et createdAt en fonction de l'opération demandée. Si l'opération est une mise à jour de l'article, la date de création ne sera pas modifiée. Si l'opération est une création de l'article, la date de création sera définie à la date et heure courantes, et l'utilisateur actuel sera également défini comme créateur de l'article.

- JWT authentication

Un JSON Web Token (JWT) est un type de token sécurisé utilisé pour stocker des informations d'identification et d'autorisation d'un utilisateur de manière compacte. Il est constitué de trois parties : l'en-tête, qui contient des informations sur le type de token ; les revendications, qui sont les informations elles-mêmes (comme le nom de l'utilisateur) ; et la signature, qui est utilisée pour vérifier l'authenticité du token. Les JWT permettent de transmettre en toute sécurité des informations entre différentes parties d'une application sans avoir besoin de les stocker côté serveur.

Encoded PASTE A TOKEN HERE

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpYXQiOiJlY200ODUyNDgsImV4cCI6MTY4NTY5OTY0Oiwicm9sZXMiOiIsiUk9MRV9BRE1JTlIsIlJPTEVfVWVFNUIjLCJ1c2VybmFtZSI6InRlc3RAZGVzdC5jb20ifQ.UXQS-NiabjHXN2IfDWUQe6-gdV0z0ATuoLGTcYB1KsqT8dpd0lmwQpGZb-UVhwL2g61XAjnMT44ZpJpA0M040GHE7uB7QEVz3Oq4nAqG-k_GXAUeAZX-yjBQNfx-aVkjAtFA0S0hpG3I1zWIXoRDEz1w73uSNjEdo_2QbJz1lyYQejdP0YH2wEP2ufs9bkjRooqnjyosunvvyoAdcAqluVjCeK1TMRZWYj523RCbi44g41KUCeVcmBlndRkEHWCXyVpoVNTKZAVmAlZ7sk3HhiePRTW_JNaSTA9oucaIHLUga_5GL-LP3WrJhoW8SHhr76EuEr9IPBfy0Y29rP21ZQ
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "typ": "JWT",
  "alg": "RS256"
}
```

PAYLOAD: DATA

```
{
  "iat": 1685696048,
  "exp": 1685699648,
  "roles": [
    "ROLE_ADMIN",
    "ROLE_USER"
  ],
  "username": "test@test.com"
}
```

API Platform permet d'ajouter facilement une authentification basée sur JWT à notre API à l'aide de LexikJWTAuthenticationBundle. Nous avons commencé par installer le bundle avec la commande :

```
composer require lexik/jwt-authentication-bundle
```

Nous avons configuré la sécurité dans le fichier security.yaml :

```
providers:
    # used to reload user from session & other features (e.g. switch_user)
    app_user_provider:
        entity:
            class: App\Entity\User
            property: email
    # used to reload user from session & other features (e.g. switch_user)
firewalls:
    login:
        pattern: ^/api/login
        stateless: true
        json_login:
            check_path: /api/login_check
            username_path: email
            password_path: password
            success_handler: lexik_jwt_authentication.handler.authentication_success
            failure_handler: lexik_jwt_authentication.handler.authentication_failure

    api:
        pattern: ^/api
        stateless: true
        provider: app_user_provider
        jwt: ~
```

Nous avons également déclaré le chemin de la route :

```

config > ! routes.yaml
1 controllers:
2   resource: ../src/Controller/
3   type: attribute
4 api_login_check:
5   path: /api/login check
6   methods: ['POST']

```

Pour documenter le mécanisme d'authentification avec Swagger/Open API, nous avons dû configurer le fichier `api_platform.yaml` :

```

config > packages > ! api_platform.yaml
1 api_platform:
2   collection:
3     pagination:
4       page_parameter_name: _page
5   defaults:
6     pagination_items_per_page: 5
7   swagger:
8     api_keys:
9       JWT:
10        name: Authorization
11        type: header

```

Nous avons créé un décorateur personnalisé pour notre endpoint de login `JwtDecorator.php`. Nous pouvons maintenant tester la route de notre API protégée par l'authentification JWT :

```

    password: password
}

```

- Les tests unitaires :

Les tests unitaires sont des tests automatisés qui permettent de vérifier le bon fonctionnement d'une portion de code, souvent une méthode ou une classe, isolée du reste de l'application. J'ai utilisé le framework PHPUnit et les Mocks qui sont des objets qui simulent le comportement d'un autre objet pour les besoins d'un test unitaire.

Dans le contexte d'API Platform, les tests unitaires peuvent être utilisés pour s'assurer du bon fonctionnement des classes comme les Processors et les Providers. Par exemple, un test unitaire pourrait vérifier si la méthode `provide` d'un `StateProvider` retourne bien les données attendues en fonction des paramètres d'entrée. De même, un test unitaire pourrait vérifier si la méthode `process` d'un `State Processor` modifie correctement les données de la ressource en fonction de l'opération demandée.

```

public function setUp(): void
{
    $this->userPasswordHasher = $this->createMock(UserPasswordHasherInterface::class);
    $this->entityManager = $this->createMock(EntityManagerInterface::class);
    $this->user = $this->createMock(User::class);
    $this->operation = $this->createMock(Operation::class);
    $this->dateTimeImmutable = new DateTimeImmutable();
    $this->processor = new UserProcessor($this->userPasswordHasher, $this->entityManager);
}

public function testProcessWithUserInstancePatchOperationWithChangingPassword(): void
{
    $this->user->expects($this->once())->method('setUpdatedAt');
    $this->user->expects($this->once())->method('setCreatedAt');
    $this->user->expects($this->once())->method('getCreatedAt')->willReturn($this->dateTimeImmutable);
    $this->operation->expects($this->once())->method('getName')->willReturn('_api_/userProfileEdit/{id}_patch');
    $this->user->expects($this->exactly(2))->method('getPlainPassword')->willReturn('plain password');
    $this->userPasswordHasher->expects($this->once())->method('hashPassword')
        ->with($this->user, 'plain password');
    $this->user->expects($this->once())->method('setPassword');
    $this->entityManager->expects($this->once())->method('persist')->with($this->user);
    $this->entityManager->expects($this->once())->method('flush');

    $this->processor->process($this->user, $this->operation);
}

```

- Les tests fonctionnels :

Les tests fonctionnels, également appelés tests d'intégration, sont des tests automatisés qui permettent de vérifier le bon fonctionnement d'une application dans son ensemble, en testant plusieurs composants ensemble, tels que des API, des bases de données, des services tiers, etc.

Dans le contexte d'API Platform, les tests fonctionnels peuvent être utilisés pour tester l'API REST dans son ensemble, en vérifiant que les différentes ressources sont correctement exposées et que les opérations CRUD (Create, Read, Update, Delete) fonctionnent correctement.

Dans le cadre de notre API, nous avons utilisé le framework PHPUnit pour la mise en place de tests fonctionnels. Pour faciliter la rédaction de ces tests, nous avons créé une classe abstraite nommée "AbstractWebTestCase", qui contient les requêtes HTTP et les méthodes les plus fréquemment utilisées pour accéder à nos différents endpoints.

```

abstract class AbstractWebTestCase extends ApiTestCase
{
    private const USERS_INFO = [
        'ROLE_ADMIN' => ['email' => 'admin@test.com',
            'password' => 'password'],
        'ROLE_USER' => ['email' => 'user@test.com',
            'password' => 'password'],
    ];

    public function provideUsers(): Generator
    {
        yield 'ROLE_ADMIN' => [self::USERS_INFO['ROLE_ADMIN'], 'ROLE_ADMIN'];
        yield 'ROLE_USER' => [self::USERS_INFO['ROLE_USER'], 'ROLE_USER'];
    }

    public function getAdminToken(): string
    {
        $client = static::createClient();
        $jwtManager = $client->getContainer()->get(JWTTokenManagerInterface::class);
        $admin = static::getContainer()->get('doctrine')->getRepository(User::class)->findOneBy(['id' => 16]);

        $token = $jwtManager->create($admin);
        $this->assertIsString($token);

        return $token;
    }

    public function getUserToken(): string
    {
        $client = static::createClient();
    }
}

```

Nous avons créé des classes de test pour chaque entité de notre API, telles que "ArticleTest", "UserTest" et "CommentTest".

```

class UserTest extends AbstractWebTestCase
{
    /**
     * @dataProvider provideUsers
     * @throws TransportExceptionInterface
     * @throws ServerExceptionInterface
     * @throws RedirectionExceptionInterface
     * @throws ClientExceptionInterface
     */
    public function testAsUserOrAdminICanLogin($user): void
    {
        $client = static::createClient();
        $response = $client->request('POST', '/api/login_check', ['json' => $user]);

        $this->assertEquals(200, $response->getStatusCode());
        $token = json_decode($response->getContent(), true);
        $this->assertArrayHasKey('token', $token);
        $this->assertIsString($token['token']);
    }

    /**
     * @throws RedirectionExceptionInterface
     * @throws ClientExceptionInterface
     * @throws TransportExceptionInterface
     * @throws ServerExceptionInterface
     * @throws \Exception
     */
    public function testAsAdminICanGetUsers(): void
    {
    }
}

```

L'utilisation de cette stratégie nous a permis de réduire considérablement le temps et l'effort nécessaire pour écrire des tests pour notre API. Nous avons pu réutiliser la classe "AbstractWebTestCase" dans plusieurs classes de tests et réduire la duplication de code, ce qui a également rendu nos tests plus cohérents et plus faciles à maintenir.

- Le service generateArticle avec OpenAi :

Ce service était l'une des tâches que j'ai réalisées au sein de mon entreprise d'accueil, Smart Tribune. L'objectif était de concevoir un service permettant à nos clients de générer des réponses à partir de titres de questions. J'ai appliqué le même principe à notre application et j'ai développé un service offrant aux utilisateurs la possibilité de générer des articles IA à partir de titres, en utilisant l'API OpenAI.

- ♦ Le DTO (Data Transfer Object) : est un modèle couramment utilisé en développement d'applications Web avec Symfony, Il est utilisé pour transférer des données entre les différentes couches de l'application, telles que les contrôleurs, les services. J'ai créé le DTO RequestGenerateDto qui contient trois propriétés "title" le titre de l'article, "language" la langue de l'article généré et le "prompt" l'entrée textuelle que nous fournissons à l'API pour générer des réponses. Le DTO fournit des méthodes pour obtenir et définir la valeur de ces propriétés.

```
class RequestGenerateDto
{
    public const FRENCH = 'French';
    public const ENGLISH = 'English';

    public const AVAILABLE_LANGUAGE = [
        self::FRENCH,
        self::ENGLISH,
    ];
    #[Assert\NotBlank]
    #[Assert\Type('string')]
    #[Assert\Length(min: 2)]
    private ?string $title = null;

    private ?string $prompt = null;

    #[Assert\NotBlank]
    #[Assert\Choice(choices: self::AVAILABLE_LANGUAGE)]
    private ?string $language = null;
```

- ♦ Ensuite, j'ai créé le formulaire "RequestGenerateType" qui sert à créer un formulaire associé au DTO "RequestGenerateDto". Il contient le champ "title" de type texte, et

le champ "language" de type choix qui affiche une liste d'options pour la langue (AVAILABLE_LANGUAGE) que l'utilisateur peut sélectionner pour la génération du texte. Les données soumises par le formulaire seront liées à une instance de RequestGenerateDto.

```
class RequestGenerateType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('title', TextType::class)
            ->add('language', ChoiceType::class,
                ['choices' => RequestGenerateDto::AVAILABLE_LANGUAGE])
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => RequestGenerateDto::class,
        ]);
    }
}
```

- ◆ L'interface PromptGenerateInterface définit un contrat que les classes doivent respecter si elles souhaitent être utilisées pour générer du texte à partir d'un objet RequestGenerateDto. L'interface fournit deux méthodes clés : supports() pour vérifier si une implémentation prend en charge une demande particulière et generate() pour effectuer le traitement de la demande et générer le texte approprié.

```
namespace App\OpenAi\Interface;

use App\OpenAi\Data\RequestGenerateDto;

interface PromptGenerateInterface
{
    public function supports(RequestGenerateDto $requestGenerateDto): bool;
    public function generate(RequestGenerateDto $requestGenerateDto): string;
}
```

- ◆ La classe principale de notre service OpenAiGpt3Dot5TurboArticleService qui génère des articles à l'aide de l'API OpenAI Chat Completions.

```

public function generate(RequestGenerateDto $requestGenerateDto, PromptHandler $promptHandler): string
{
    $title = $requestGenerateDto->getTitle();
    $generatePrompt = $promptHandler->generatePrompt($requestGenerateDto);
    $openaiPromptBase = $generatePrompt. "The title which you have to use to generate the Article is : ".$title;
    $payload = [
        'model' => self::MODEL,
        'messages' => [
            ['role' => 'system', 'content' => $openaiPromptBase],
            ['role' => 'user', 'content' => $title],
        ],
        'temperature' => self::TEMPERATURE,
        'max_tokens' => self::MAX_TOKENS,
        'top_p' => self::TOP_P,
        'n' => self::N,
        'stop' => self::STOP,
        'frequency_penalty' => self::FREQUENCY_PENALTY,
        'presence_penalty' => self::PRESENCE_PENALTY,
    ];
    $options = [
        'headers' => [
            'Authorization' => sprintf('Bearer %s', self::OPENAI_API_KEY),
            'OpenAI-Organization' => self::OPENAI_ORGANIZATION_ID,
        ], 'json' => $payload,
    ];
    $response = $this->client->request('POST', self::OPENAI_CHAT_COMPLETIONS_URL, $options);
    $statusCode = $response->getStatusCode();
    if (200 !== $statusCode) {
        throw new Exception(sprintf('Bad Response (%d) : %s', $statusCode, $response->getContent()));
    }
    return $response->toArray()['choices'][0]['message']['content'];
}

```

La méthode "generate" prend un objet "RequestGenerateDto" contenant le titre de l'article et l'objet "promptHandler" qui contenant le prompt et utilise ces objets pour construire la requête vers l'API OpenAI. Le code envoie une requête POST à l'URL de l'API OpenAI Chat Completions avec les paramètres appropriés, y compris le prompt de génération basé sur le titre fourni. Il gère les exceptions potentielles liées à la requête HTTP et vérifie que le code de réponse est valide (200), comme indiqué dans la documentation OpenAI. Finalement, il extrait le contenu de l'article généré à partir de la réponse de l'API et le retourne.

- ◆ Les handlers : j'ai créé trois handlers, deux handlers pour générer les prompts pour la création d'articles en français et en anglais. Ces handlers sont nommés "EnglishPromptHandler" et "FrenchPromptHandler", et ils implémentent l'interface "PromptGenerateInterface". Cette implémentation signifie qu'ils doivent contenir les deux méthodes suivantes : La méthode "supports()", qui retourne "true" si l'utilisateur choisit la langue définie pour cet handler. La méthode "generate()", qui retourne le prompt défini dans cet handler. En utilisant cette structure, nous pouvons aisément ajouter de nouveaux handlers pour d'autres langues ou types de demandes, tout en respectant le contrat défini par l'interface.


```

class EnglishPromptHandler implements PromptGenerateInterface
{
    final public const ENGLISH_PROMPT = "I am an intelligent articles-generating s
    I know how to generate article from his title with only five sentences.
    I write the article in English.
    If you ask me a question that is nonsense, trickery, or has no clear answer, I
    I know the regulations and laws of France.";
    public function supports(RequestGenerateDto $requestGenerateDto): bool
    {
        return $requestGenerateDto->getLanguage() == RequestGenerateDto::ENGLISH;
    }
    public function generate(RequestGenerateDto $requestGenerateDto): string
    {
        $requestGenerateDto->setPrompt(self::ENGLISH_PROMPT);
        return self::ENGLISH_PROMPT;
    }
}

```

```

class FrenchPromptHandler implements PromptGenerateInterface
{
    final public const FRENCH_PROMPT = "I am an intelligent articles-generating s
    I know how to generate article from his title with only five sentences.
    I write the article in French.
    If you ask me a question that is nonsense, trickery, or has no clear answer,
    I know the regulations and laws of France.";
    public function supports(RequestGenerateDto $requestGenerateDto): bool
    {
        return $requestGenerateDto->getLanguage() == RequestGenerateDto::FRENCH;
    }
    public function generate(RequestGenerateDto $requestGenerateDto): string
    {
        $requestGenerateDto->setPrompt(self::FRENCH_PROMPT);
        return self::FRENCH_PROMPT;
    }
}

```

Le troisième handler `PromptHandler` est conçue pour gérer une collection de gestionnaires de génération de texte, chacun implémentant l'interface `PromptGenerateInterface`. Lorsqu'un objet `RequestGenerateDto` est passé à la méthode `generatePrompt()`, elle recherche le gestionnaire approprié pour traiter la demande spécifiée et génère le texte correspondant en utilisant ce gestionnaire. Si aucun gestionnaire n'est disponible pour la demande, une exception est levée pour indiquer que le traitement a échoué.

```

class PromptHandler
{
    public function __construct(private readonly iterable $handlers)
    {
    }

    public function generatePrompt(RequestGenerateDto $requestGenerateDto): string
    {
        /**
         * @var $handler PromptGenerateInterface
         */
        foreach ($this->handlers as $handler) {
            if ($handler->supports($requestGenerateDto)) {
                return $handler->generate($requestGenerateDto);
            }
        }
        throw new LogicException('[PromptHandler] : No handler available');
    }
}

```

Et j'ai ajouté une configuration sur le fichier service.yaml pour définir et lier les services liés à l'interface PromptGenerateInterface et à la classe PromptHandler.

```

instanceof:
    App\OpenAi\Interface\PromptGenerateInterface:
        tags: ['app.prompt.handler']
    App\OpenAi\PromptHandler\PromptHandler:
        arguments:
            - !tagged_iterator 'app.prompt.handler'

```

Comme ça nous avons identifié les services qui implémentent l'interface PromptGenerateInterface et vous les taguez avec app.prompt.handler. Puis, nous avons configuré le service PromptHandler pour qu'il prenne en argument une collection de ces services tagués, ce qui permettra au PromptHandler d'accéder aux différents gestionnaires de génération pour traiter les demandes spécifiques à l'interface PromptGenerateInterface. Cela permet de rendre le PromptHandler extensible, car de nouveaux gestionnaires de génération pourraient être ajoutés en implémentant l'interface et en étant automatiquement détectés par le système de services de Symfony.

- ◆ En fin le contrôleur GenerateOpenAiArticleController reçoit une requête POST à l'URL '/api/ai-generate'. Il utilise l'API OpenAI pour générer des articles. Les données de la requête sont validées et extraites. Le contrôleur crée un formulaire à partir des données et le soumet. Si le formulaire est valide, le contrôleur appelle le service de génération d'articles OpenAI pour obtenir l'article généré. Le résultat est renvoyé en tant que réponse JSON.

```

#[Route(path: '/api/ai-generate', methods: ['POST'])]
public function generateAction(
    Request $request,
    OpenAiGpt3Dot5TurboArticleService $openAiGpt3Dot5TurboArticleService,
    PromptHandler $promptHandler
): JsonResponse {
    $requestDto = new RequestGenerateDto();
    $requestData = $this->decodeRequestContent($request);
    $form = $this->createForm(
        RequestGenerateType::class,
        $requestDto
    );
    $form->submit($requestData);
    if (!$form->isValid()) {
        return $this->json($this->getFormErrors($form));
    }
    $generatedArticle = $openAiGpt3Dot5TurboArticleService->generate($requestDto, $promptHandler);
    return $this->json($generatedArticle);
}

```

- Déploiement de l'API sur PLESK

Notre école a désigné des espaces pour chaque étudiant sur Plesk pour publier nos applications. Pour déployer notre API Symfony sur Plesk j'ai suivi ces étapes :

1. J'ai généré une version prête à être déployée de votre application Symfony, en utilisant la commande **composer install --no-dev --optimize-autoloader** pour installer les dépendances et optimiser l'autoloader. Cela va générer automatiquement le fichier **.htaccess** dans le répertoire racine de notre API Symfony avec les règles de réécriture appropriées pour diriger toutes les demandes vers le fichier **public/index.php** de Symfony.
2. J'ai accédé au gestionnaire de fichiers Plesk et téléchargé nos fichiers de l'API Symfony dans le répertoire httpdocs.
3. Dans l'onglet php composer, j'ai installé tous les packages de dépendances.
4. J'ai créé une nouvelle base de données sur Plesk, puis j'ai exporté ma base de données locale en utilisant PhpMyAdmin et l'ai importée dans ma nouvelle base de données sur Plesk. Ensuite, j'ai configuré le fichier **.env** en utilisant les paramètres de la base de données.

Files >

File Manager for rany-alo.students-laplateforme.io

Home directory

- > .composer
- > .nodenv
- > .npm
- > .phpenv
- > .pki
- > .revisium_antivirus_cache
- > .ssh
- > .trash
- > error_docs
- > httpdocs
- > logs

+

Copy

Move

Archive

More

Remove

Search in filename

Home directory > httpdocs > app-mobile-forum >

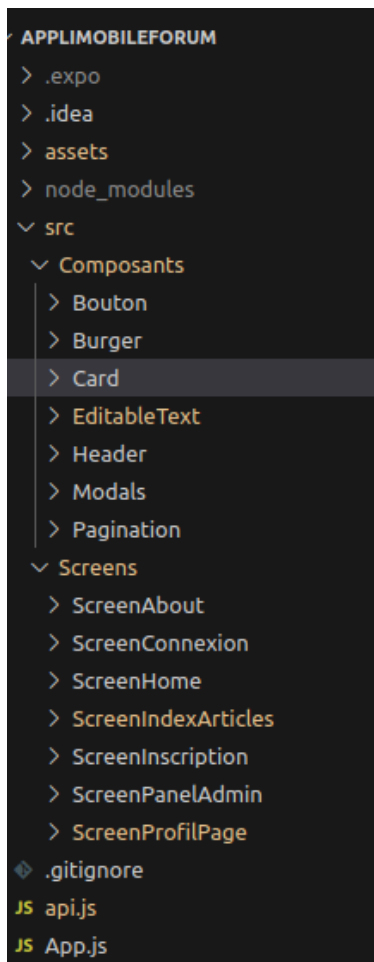
<input type="checkbox"/>	Name ↑	Modified	Size	Permissions	User	Group	
	↑ ..	May 23, 2023 05:15 PM		rwX r-x ---	rany.alo	psaserv	
<input type="checkbox"/>	.git	Jan 10, 2023 08:24 PM		rwX rwX r-x	rany.alo	psacIn	⋮
<input type="checkbox"/>	bin	Jan 10, 2023 08:24 PM		rwX rwX r-x	rany.alo	psacIn	⋮
<input type="checkbox"/>	config	Jan 10, 2023 08:24 PM		rwX rwX r-x	rany.alo	psacIn	⋮
<input type="checkbox"/>	fixtures	Jan 10, 2023 08:24 PM		rwX rwX r-x	rany.alo	psacIn	⋮
<input type="checkbox"/>	migrations	Jan 10, 2023 08:24 PM		rwX rwX r-x	rany.alo	psacIn	⋮
<input type="checkbox"/>	public	Jan 10, 2023 09:55 PM		rwX r-x r-x	rany.alo	psacIn	⋮
<input type="checkbox"/>	src	Jan 10, 2023 08:24 PM		rwX rwX r-x	rany.alo	psacIn	⋮

Interface utilisateur mobile

Nous avons utilisé la plateforme Expo qui facilite le développement d'applications mobiles en fournissant des outils, des fonctionnalités et des services prêts à l'emploi. Il simplifie le processus de développement, de test et de déploiement, Expo est construit sur React Native, une bibliothèque populaire pour le développement d'applications mobiles. Et nous avons installé Expo Go sur nos téléphones android pour tester et de prévisualiser notre application développée avec Expo.



- Arborescence



- Dans le dossier src du projet on y trouve tous les dossiers essentiels au projet.
- Dans le dossier composant on y trouve tous les composants réutilisables.
- Dans screens, se trouve tous les écrans de mon application mobile.
- Le fichier **api.js** regroupe l'ensemble des méthodes utilisant la bibliothèque Axios pour effectuer des requêtes vers notre API Symfony. Cette structure centralisée facilite la gestion des appels réseau et permet une intégration cohérente et professionnelle des fonctionnalités de communication avec notre backend.
- Le dossier "assets" contient toutes les ressources visuelles, telles que les images, utilisées dans notre projet. De plus, nous avons un dossier nommé "styles" qui abrite notre fichier "style.js" pour la gestion de la mise en page de nos écrans. Cette organisation assure une structure cohérente et facilite la gestion des ressources visuelles et du style dans notre application.

- Les composants

Les composants en React Native sont des éléments réutilisables qui permettent de construire l'interface utilisateur de l'application. Ils encapsulent une partie de la logique et de la mise en page d'un élément de l'interface. Les composants peuvent être personnalisés en utilisant les "props", qui sont des propriétés passées aux composants pour modifier leur apparence ou leur comportement. Les props permettent de transmettre des données et des fonctions entre les composants parent et enfant.

```
src > Composants > Bouton > JS buttonComponent.js > ...
1  import React from 'react'
2  import { Pressable, Text, View } from 'react-native'
3  export default function ButtonComponent(props) {
4    return (
5      <View style={props.contBouton}>
6        <Pressable style={props.button} onPress={props.onPress}>
7          <Text style={props.txtBouton}>{props.text}</Text>
8        </Pressable>
9      </View>
10    )
11  }
```

Ce composant, utilisé par tous dans notre code, est une encapsulation réutilisable pour un bouton dans notre application. Il importe les composants React et React Native nécessaires, et expose une fonction "ButtonComponent" qui accepte des props. Ces props permettent de personnaliser l'apparence et le comportement du bouton, y compris les styles, le texte affiché et l'action à exécuter lorsqu'il est pressé. En l'incluant dans notre code, nous pouvons facilement créer et utiliser des boutons cohérents et interactifs dans notre application.

Le Button Publier est un composant ButtonComponent.



The screenshot shows a mobile application interface with a blue background. At the top, there is a header bar with a hamburger menu icon on the left, the date "Mercredi 15 Février" in the center, and a small profile picture on the right. Below the header, the title "Ajouter un article" is displayed in white. Underneath, there is a form with two main sections. The first section is labeled "Titre de l'article :" and contains a text input field with the placeholder text "Salut". The second section is labeled "Texte de votre article :" and contains a larger text area with the placeholder text "Votre Article". At the bottom of the form, there is a button labeled "Publiez".

- Les screens

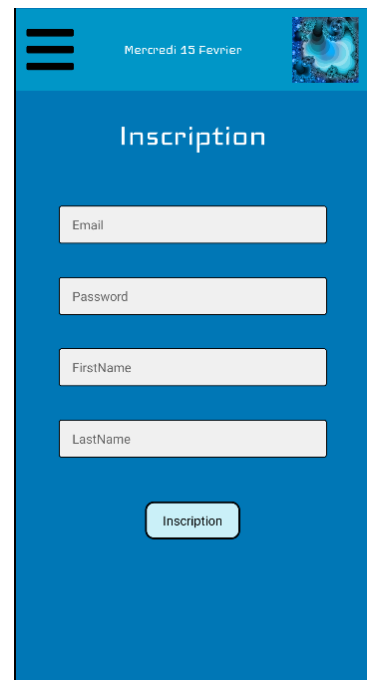
Les screens, également appelés vues, sont des interfaces utilisateur qui s'affichent aux utilisateurs dans le cadre de leur expérience avec une application. Ils représentent les différentes pages, composants ou états de l'application qui fournissent des informations, des fonctionnalités et des interactions spécifiques.

```
10
11 export default function InscriptionScreen({ navigation }) {
12   const emailRegex = /^\\S+@\\S+\\.\\S+$/;
13   const [user, setUser] = useState('')
14
15   const handleSubmit = () => {
16     if (emailRegex.test(user.email)) {
17       postUser(user.email, user.firstName, user.lastName, user.password, (res => {
18         if (res.status !== 201) {
19           Alert.alert('Impossible de créer l'utilisateur', `${res.data.violations[0].message}`, [{
20             style: 'cancel'
21           }])
22         }
23         else {
24           login(user.email, user.password, (res => {
25             if (res.status !== 200) {
26               Alert.alert('Connexion automatique échoué, veuillez vous connecter', `${res.data.message}`, [{
27                 style: 'cancel'
28               }])
29             }
30             else {
31               setUser('')
32               Alert.alert('Votre inscription a bien été prise en compte', 'Bienvenue sur le Forum !')
33               navigation.navigate('Articles', { refresh: true })
34             }
35           })
36         }
37       }));
38     }
39     else {
40       Alert.alert('E-mail invalide', 'Veuillez entrer un e-mail valide pour continuer')
```

Le code ci-dessus représente la vue d'écran d'inscription dans une application mobile.

Il gère l'état de l'utilisateur et valide l'e-mail avant de créer un nouvel utilisateur via une API.

Si l'inscription est réussie, l'utilisateur est automatiquement connecté et redirigé vers une autre page. L'écran comprend des composants tels que des en-têtes, des champs de texte pour l'e-mail, le mot de passe, le prénom et le nom de famille, ainsi qu'un bouton d'inscription.



- Ajout et appelle d'une route

Pour ajouter et appeler une route dans le back-end, il vous suffit de vous rendre sur le fichier api.js qui se situe à la racine du projet.

```
export const getArticles = (page, callback) => {
  request("get", `/articles?_page=${page}`, null, (res) => {
    return callback(res)
  });
}

export const getArticleById = (id, callback) => {
  request("get", `/article/${id}`, null, (res) => {
    return callback(res)
  });
}

export const postArticle = (title, content, callback) => {
  request("post", `/articlePost`, { title, content }, (res) => {
    return callback(res)
  });
}

export const patchArticle = (id, title, content, callback) => {
  request("patch", `/articleEdit/${id}`, { title, content }, (res) => {
    return callback(res)
  });
}

export const deleteArticle = (id, callback) => {
  request("delete", `/articleDelete/${id}`, null, (res) => {
    return callback(res)
  });
}
```

Nous utilisons Axios pour effectuer une requête vers l'API, et une méthode request a été créée qui gère les réponses des requêtes.

```
const request = async (method, url, data, callback) => {
  await getJwtToken();
  header = { 'Content-type': 'application/json', }
  if (method == "patch") {
    header = { 'Content-type': 'application/merge-patch+json' }
  }

  return api({
    method: method,
    url: url,
    data: data,
    headers: header
  }).then(res => {
    return callback(res);
  })
  .catch(error => {
    console.log(error.message)
    console.log('Erreur ${error.response.data.code}')
    console.log(error.response.data.message)
    if (error.response.data.message == 'Expired JWT Token') {
      SecureStore.deleteItemAsync('jwt').then(() => {
      })
      Alert.alert('Votre session a expiré', 'Veuillez vous re-connecter pour continuer', [
        { text: 'OK', onPress: () => { } }
      ])
      return null
    }
    return callback(error.response)
  });
};
```


La fonction request ne requiert que 3 paramètres, la première consiste à définir la méthode (get/post/patch/delete), la seconde la route, la troisième les paramètres de la requête s'il y en a, et enfin elle renvoie en callback la réponse (ou l'erreur).

Pour ensuite appeler votre requête dans n'importe quel fichier, il suffit d'importer le fichier api.js ainsi que la méthode que vous avez définie

```
import { getArticles } from '../../api';

export default function IndexArticleScreen({ navigation }) {
  const route = useRoute();
  const refresh = route.params.refresh;
  console.log(refresh)

  const [articles, setArticles] = useState([]);
  const [loading, setLoading] = useState(false);
  const [page, setPage] = useState(1);
  const [totalItems, setTotalItems] = useState(0);

  const fetchData = () => {
    setLoading(true);
    getArticles(page, (res) => {
      setArticles(prevArticles => [...prevArticles, ...res.data['hydra:member']]);
      setTotalItems(res.data['hydra:totalItems']);
      setLoading(false);
    });
  };
}
```

Si des paramètres sont présents, il suffit de les envoyer à la méthode en premier lieu, puis de récupérer la réponse en utilisant une fonction fléchée anonyme.

- Choix de la navigation

La navigation est un aspect important dans le développement des applications mobiles car elle permet aux utilisateurs de naviguer facilement entre les différentes pages et fonctionnalités de l'application. React Native offre plusieurs options pour gérer la navigation, chacune ayant ses avantages et inconvénients.

Nous avons utilisé deux choix de navigation dans notre projet :

- 1- Nous avons largement utilisé la méthode navigate fournie par l'objet navigation de la bibliothèque React Navigation, dans les boutons et liens pour rediriger l'utilisateur à la page cible.

```
<ButtonComponent
  contButon={styles.containerCenter}
  button={styles.buttonStyleLarge}
  txtButton={styles.textButon}
  text={"La liste des utilisateurs"}
  onPress={() => navigation.navigate('Users', { refresh: true })}
/>
```

- 2- La deuxième option de navigation est la navigation basée sur un menu latéral (drawer navigation). Cette méthode permet d'afficher un menu latéral sur le côté de l'écran, qui peut être ouvert ou fermé pour afficher différentes sections de l'application. Cette méthode est idéale pour les applications qui ont plusieurs sections différentes qui ne sont pas facilement accessibles depuis l'écran principal.

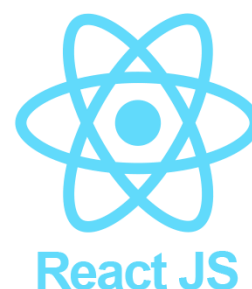
```
<DrawerItem label={'Articles'} onPress={() => props.navigation.navigate('Articles', { refresh: true })} />
  {isLog && <DrawerItem label={'Ajouter un article'} onPress={() => props.navigation.navigate('AddArticle', { refresh: true })} />
  {isLog && <DrawerItem label={'Home'} onPress={() => props.navigation.navigate('Home', { refresh: true })} />} />
  {isLog && <DrawerItem label={'Profil'} onPress={() => props.navigation.navigate('Profil', { refresh: true })} />
  {isLog && <DrawerItem label={'Connexion'} onPress={() => props.navigation.navigate('Connexion')} />} />
  {isLog && <DrawerItem label={'Inscription'} onPress={() => props.navigation.navigate('Inscription')} />
  {Admin && <DrawerItem label={'Admin'} onPress={() => props.navigation.navigate('AdminScreen')} />} />
<DrawerItem label={'About'} onPress={() => props.navigation.navigate('About')} />
```

Ce composant **CustomDrawer** est utilisé pour afficher un tiroir de navigation personnalisé dans notre application, offrant des liens vers différentes sections de l'application en fonction de l'état de connexion de l'utilisateur.

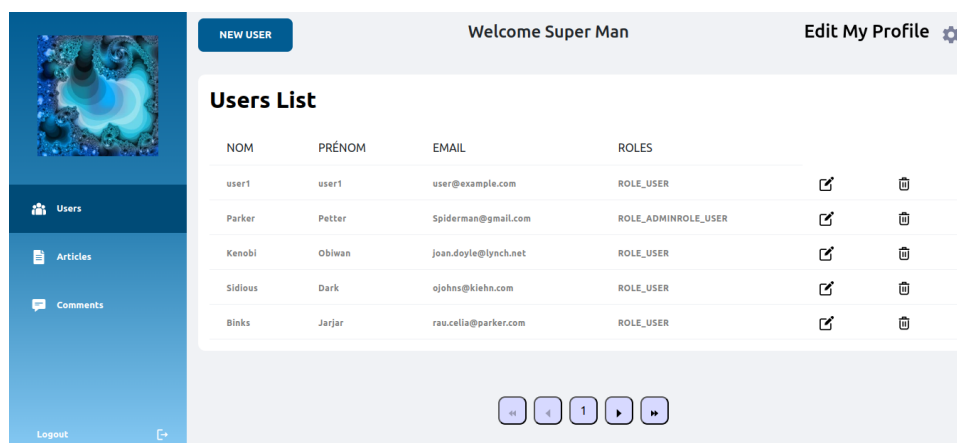
Interface utilisateur web

Dans le cadre du développement de l'interface web, j'ai pris la décision de concevoir un panneau d'administration qui offre aux utilisateurs disposant du rôle d'administrateur l'accès et la possibilité de gérer efficacement les données relatives aux utilisateurs, aux articles et aux commentaires. Il offre une interface conviviale et sécurisée, avec des fonctionnalités avancées de gestion, telles que la création, la modification et la suppression d'utilisateurs, d'articles et de commentaires.

La décision d'utiliser le Framework React.js a été motivée par ses avantages en termes de modularité, de réutilisabilité du code et de performances. Grâce à la structure basée sur des composants de React.js, j'ai pu créer des interfaces interactives et dynamiques, offrant ainsi une expérience utilisateur améliorée.



Pour la mise en page de mon application, j'ai utilisé la combinaison de CSS native et du Framework Bootstrap. Cette approche m'a permis de bénéficier des fonctionnalités avancées de mise en page et de design fournies par Bootstrap, tout en personnalisant l'apparence et le style spécifiques de mon application grâce à CSS natif.



- Mettre en place la gestion des routes

L'utilisation des routes permet de créer une application réactive où le contenu et l'interface utilisateur sont dynamiquement mis à jour en fonction de l'URL courante. Cela offre une

expérience de navigation fluide et permet de gérer efficacement différentes sections de l'application.

Pour ce faire j'ai installé la react-router-dom, dans composant racine App j'ai défini mes différentes routes, Les routes imbriquées dans la balise **<Routes>** définissent les composants à afficher en fonction de l'URL. Par exemple, la route **/users** affiche le composant **Users**, **/users/add** affiche **UserAdd**, etc.

```
<Routes>
  <Route path="/" element={< Login/>} />
  <Route exact path="/dashboard/*" element={

    <div className='dashboard-container'>
      <SideBar menu={sidebar_menu} />
      <div className='dashboard-body'>
        <Routes>
          <Route path="*" element={<div></div>} />
          <Route exact path="/" element={<div></div>} />
          <Route exact path="/users" element={< Users/>} />
          <Route path="/users/add" element={< UserAdd/>} />
          <Route path="/users/:id" element={< UserEdit/>} />
          <Route exact path="/articles" element={< Articles/>} />
          <Route path="/articles/add" element={< ArticleAdd/>} />
          <Route path="/articles/:id" element={< ArticleEdit/>} />
          <Route exact path="/comments" element={< Comments/>} />
          <Route path="/comments/add" element={< CommentAdd/>} />
          <Route path="/comments/:id" element={< CommentEdit/>} />
        </Routes>
      </div>
    </div>
  } />
</Routes>
```

- Présentation du jeu d'essai d'une fonctionnalité

J'ai décidé de tester la fonctionnalité d'affichage des articles, qui est implémentée dans la fonction `fetchData`.

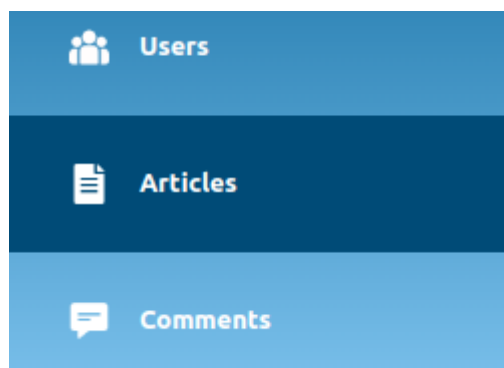
```
const fetchData = async () => {
  try {
    const {data} = await axios.get(`/articles?_page=${page}`);
    setArticles(data);
    setTotalItems(data['hydra:totalItems']);
  } catch (error) {
    window.alert(error.response.data['hydra:description']);
  }
}
```

La fonction **fetchData** récupère des données à partir d'une API en utilisant une requête GET avec Axios. Elle utilise l'URL `"/articles"` avec un paramètre de pagination `"_page"` égal à la valeur de la variable **page**. Les données récupérées sont ensuite stockées dans le state **articles**, et le total des éléments est stocké dans le state **totalItems**. En cas d'erreur, un

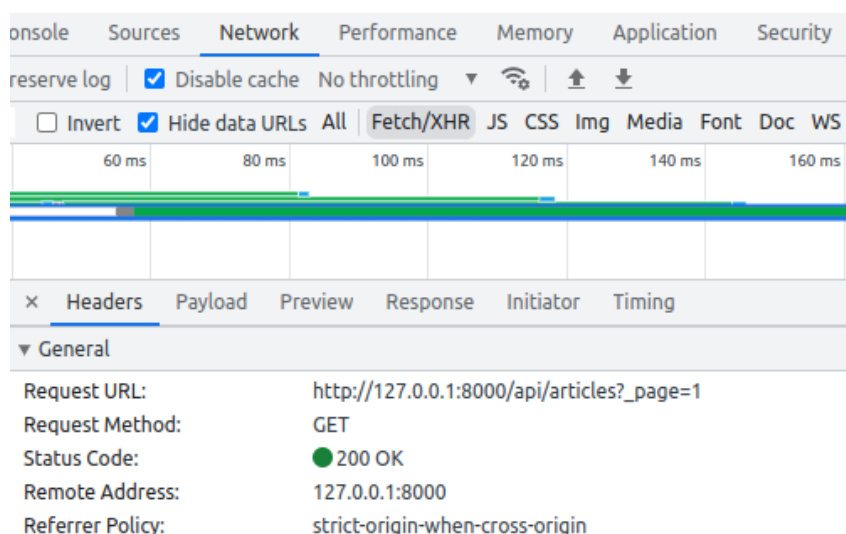
message d'alerte est affiché avec la description de l'erreur provenant de la réponse de la requête.

Les étapes du test :

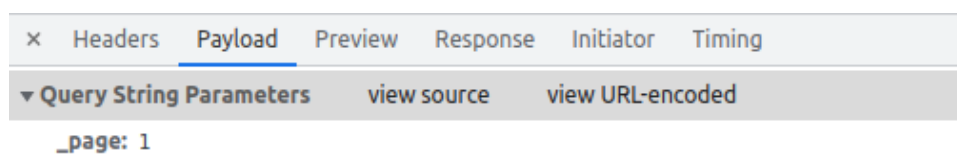
→ Pour initier l'affichage des articles, nous commençons par cliquer sur l'onglet Articles située dans le sidebar au gauche de notre Dashboard.



→ Nous pouvons voir toutes les informations de la requête envoyée à notre API en ouvrant l'inspecteur de notre navigateur (chrome), dans l'onglet Network, on choisit Headers, on peut voir la requête exécutée par le serveur.



→ Les données envoyées comme query params seront dans l'onglet payload, il s'agit du numéro de page.



→ Les données attendues seront le résultat de traitement de la requête préparée par le serveur en format json, pour contacter à base de données et renvoyer l'information demandée. On peut vérifier cela par ouvrir l'onglet Response.

```
Headers Payload Preview Response Initiator Timing
1 {"@context":"\\api\\contexts\\Article","@id":"\\api\\articles","@type":"hydra:Collection","hydra:member":[{"@id":"\\api\\article\\27","@type":"Article","id":27,"tit
```

→ Les données obtenues seront les données les premières 5 articles comme une réponse en format JSON. Ensuite on peut afficher ces données dans la page articles dans un tableau HTML en utilisant la méthode **map**. Chaque article est représenté par une ligne (**<tr>**) avec différentes colonnes contenant les informations du titre, du contenu, de la date de création et de la date de mise à jour de l'article.

The screenshot shows a web application dashboard. The top navigation bar includes a 'NEW ARTICLE' button and a welcome message 'Welcome admin admin'. The main content area is titled 'Articles List' and contains a table with the following data:

Title	Content	Created At	Updated At
Docker	Docker est une plateforme de conteneurisation qui permet de créer, déployer et exécuter des applications dans des conteneurs isolés. Cette technologie est de plus en plus populaire dans le monde de l'informatique car elle permet de simplifier le processus de développement et de déploiement des applications. Les conteneurs Docker sont légers, portables et peuvent être exécutés sur n'importe quel système d'exploitation. Cependant, l'utilisation de Docker peut également poser des défis en termes de sécurité et de gestion des conteneurs. Il est donc important de bien comprendre les avantages et les inconvénients de cette technologie avant de l'adopter dans votre entreprise.	03/06/2023 17:26:16	10/06/2023 17:35:21
React Native	React Native est un framework open-source développé par Facebook pour créer des applications mobiles multiplateformes. Il permet aux développeurs d'utiliser le langage de programmation JavaScript pour créer des applications pour iOS, Android et d'autres plateformes. React Native utilise une approche de développement basée sur des composants, ce qui facilite la réutilisation de code et la maintenance de l'application. Il offre également des performances élevées grâce à son architecture de rendu natif. Avec React Native, les développeurs peuvent créer des applications mobiles de haute qualité avec une expérience utilisateur fluide et cohérente.	02/06/2023 16:33:59	02/06/2023 20:53:56
Api platform	API Platform est une plateforme de développement d'API open source qui permet aux développeurs de créer des API REST et GraphQL de manière rapide et efficace. Elle offre une multitude de fonctionnalités telles que la documentation automatique, la validation des données, la pagination, la gestion des erreurs, la sécurité et bien plus encore. API Platform est basée sur Symfony et utilise des technologies modernes telles que Doctrine ORM, Swagger UI et ReactJS pour offrir une expérience de développement fluide et agréable. Avec API Platform, les développeurs peuvent se concentrer sur la création de fonctionnalités plutôt que sur la mise en place de l'infrastructure de leur API.	02/06/2023 16:52:56	02/06/2023 16:52:56
Symfony	Symfony est un framework PHP open source qui permet de développer des applications web de manière rapide et efficace. Il offre une architecture modulaire et flexible, ainsi qu'une grande variété de composants réutilisables pour faciliter le développement. Symfony est utilisé par de nombreuses entreprises et développeurs à travers le monde pour créer des applications web robustes et évolutives. Parmi les avantages de Symfony, on peut citer sa documentation complète, sa communauté active et son support à long terme. Si vous cherchez un framework PHP pour votre prochain projet, Symfony est certainement une option à considérer.	30/05/2023 10:46:10	30/05/2023 10:46:10
AI Technos	AI Technos, or Artificial Intelligence Technologies, is a rapidly growing field that is revolutionizing the way we live and work. From self-driving cars to virtual assistants, AI Technos is changing the way we interact with technology and each other. However, with this rapid growth comes new challenges and concerns, such as the ethical implications of AI and the potential for job displacement. Despite these challenges, the potential benefits of AI Technos are vast, including increased efficiency, improved healthcare, and enhanced security. As we continue to develop and integrate AI Technos into our lives, it is important to carefully consider the implications and work towards responsible and ethical implementation.	24/05/2023 20:00:48	24/05/2023 20:00:48

Interface utilisateur desktop

Cette interface est une application Python qui utilise une interface graphique tkinter et le module requests pour se connecter à notre API Symfony. L'API Symfony, à son tour, effectue des appels vers l'API OpenAI pour générer un article à partir d'un titre donné.

Le code est réparti en trois fichiers : "login.py", "generator.py" et "main.py".

- Le fichier login.py

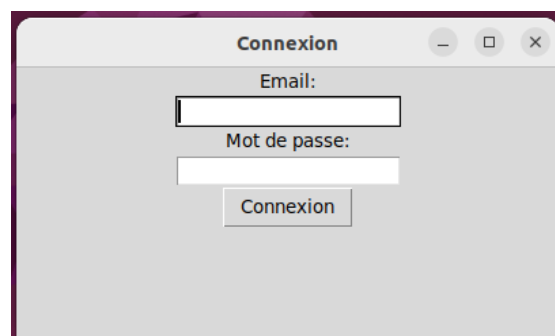
Le fichier "login.py" contient la classe "LoginWindow" qui représente la fenêtre de connexion de l'application. L'utilisateur peut entrer son email et son mot de passe, et ensuite l'application envoie une requête POST à l'API Symfony pour vérifier les informations de connexion. Si l'authentification réussit (code de statut 200), la fenêtre de génération d'articles est affichée.

```
def login(self):
    email = self.email_entry.get()
    password = self.password_entry.get()

    url = "http://127.0.0.1:8000/api/login_check"
    headers = {"Accept": "application/json", "Content-Type": "application/json"}
    payload = {"email": email, "password": password}

    response = requests.post(url, headers=headers, data=json.dumps(payload))

    if response.status_code == 200:
        # Authentification réussie, passer à la fenêtre de génération d'article
        token = response.json().get('token')
        self.destroy()
        self.generator_window = GeneratorWindow(token)
    else:
        # Authentification échouée, afficher un message d'erreur
        self.error_label.config(text="Identifiants incorrects")
```



- Le fichier generator.py

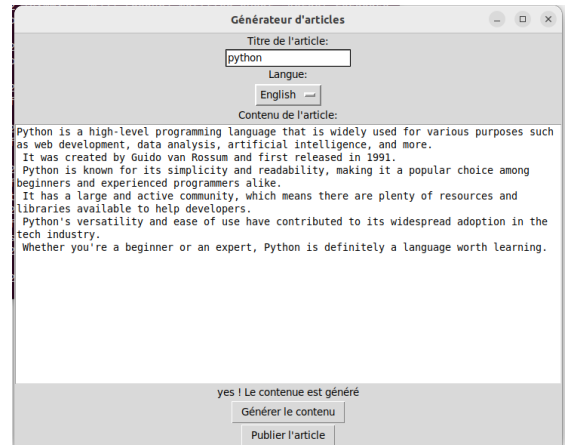
Le fichier "generator.py" contient la classe "GeneratorWindow" qui représente la fenêtre de génération d'articles. L'utilisateur peut saisir le titre de l'article et choisir la langue dans laquelle il souhaite que l'article soit créé, puis l'application envoie une requête POST à l'API Symfony avec le titre et la langue pour générer le contenu de l'article à partir de l'API OpenAI. Le contenu généré est affiché dans une zone de texte. L'utilisateur a également la possibilité de publier l'article en envoyant une requête POST à l'API Symfony avec le titre et le contenu.

```
def generate_content(self):
    title = self.title_entry.get()
    language = self.language_var.get()
    url = "http://127.0.0.1:8000/api/ai-generate"
    headers = {"Authorization": "Bearer " + self.token, "Content-Type": "application/json"}
    payload = {"title": title, "language": language}

    self.disable_widgets()
    self.loading_label.config(text="Chargement en cours...")
    self.loading_label.update()

    response = requests.post(url, headers=headers, json=payload)

    if response.status_code == 200:
        self.enable_widgets()
        decoded_text = co (variable) formatted_text: str ode_escape')
        formatted_text = formatted_text.strip('')
        self.content_text.delete("1.0", tk.END)
        self.content_text.insert(tk.END, formatted_text)
        self.loading_label.config(text="yes ! Le contenu est généré")
    else:
        self.enable_widgets()
        self.content_text.delete("1.0", tk.END)
        self.content_text.insert(tk.END, "Erreur lors de la génération du contenu")
```



- Le fichier main.py

Le fichier "main.py" est le point d'entrée principal du programme. Il crée une instance de la classe "LoginWindow" et la lance.

```
generator.py login.py main.py X
main.py > ...
1 from login import LoginWindow
2
3 def main():
4     login_window = LoginWindow()
5     login_window.mainloop()
6
7 if __name__ == "__main__":
8     main()
9
```

Bibliothèques utilisées : tkinter pour l'interface graphique, requests pour effectuer des requêtes HTTP, json pour la manipulation de données JSON, codecs pour décoder le texte de réponse, et re pour le traitement des expressions régulières.

La sécurité de l'API

L'utilisation du Framework Symfony pour construire notre API présente plusieurs avantages en termes de sécurité. Voici quelques-uns des avantages les plus notables :

- 1- Architecture sécurisée : Symfony suit une architecture MVC (Modèle-Vue-Contrôleur) qui favorise la séparation des préoccupations et l'organisation du code. Cela permet de mettre en œuvre des pratiques de sécurité solides, telles que l'application de règles d'accès aux ressources, la validation des données et la protection contre les attaques courantes.
- 2- Composants de sécurité : Symfony propose un ensemble de composants dédiés à la sécurité, tels que le composant Security, le composant Validator et le composant CSRF (Cross-Site Request Forgery). Ces composants facilitent la mise en place de fonctionnalités de sécurité avancées, telles que l'authentification, l'autorisation, la validation des données et la protection contre les attaques CSRF.
- 3- Gestion des erreurs et des exceptions : Symfony offre un système de gestion des erreurs et des exceptions qui permet de traiter de manière sécurisée les erreurs et les exceptions qui se produisent pendant l'exécution de l'API. Cela aide à prévenir les fuites d'informations sensibles et à fournir des messages d'erreur appropriés sans révéler d'informations sensibles aux utilisateurs finaux.
- 4- Protection contre les injections SQL : Symfony intègre un ORM (Object-Relational Mapping) appelé Doctrine, qui facilite l'interaction avec la base de données. L'utilisation de Doctrine permet d'automatiquement échapper et paramétrer les requêtes SQL, ce qui réduit considérablement les risques d'injections SQL.

Exemple sur l'injection SQL : dans le champ d'email du formulaire d'inscription, si on a la requête d'insertion d'utilisateur suivante :

```
$rqt = "INSERT INTO users (id, firstname, lastname, password ,email) VALUES  
($id, $firstname, $lastname, $password, $email)";
```

L'utilisateur a le droit d'écrire dans le champ email cela :

```
'aaa@gmail.com'),('a','b','c','d','e'
```

Une situation de travail ayant nécessité une recherche à partir de site anglophone

Lorsque j'ai pris la décision de mettre en place un service de génération d'articles IA en utilisant OpenAI, j'ai réalisé que je n'avais aucune connaissance préalable sur ce sujet, en particulier sur le format du payload de la requête envoyée au serveur OpenAI. Par conséquent, j'ai immédiatement consulté le site officiel d'OpenAI afin d'obtenir les informations nécessaires et de comprendre les exigences techniques liées à l'intégration de leur API.

Voici le lien vers l'article que j'ai trouvé :

<https://platform.openai.com/docs/api-reference/chat/create>

§ Extrait du site anglophone§

Create chat completion Beta

POST <https://api.openai.com/v1/chat/completions>

Creates a model response for the given chat conversation.

Request body

model string Required

ID of the model to use. See the [model endpoint compatibility](#) table for details on which models work with the Chat API.

messages array Required

A list of messages describing the conversation so far. [Example Python code](#).

role string Required

The role of the author of this message. One of `system`, `user`, or `assistant`.

content string Required

The contents of the message.

name string Optional

The name of the author of this message. May contain a-z, A-Z, 0-9, and underscores, with a maximum length of 64 characters.

temperature number Optional Defaults to 1

What sampling temperature to use, between 0 and 2. Higher values like 0.8 will make the output more random, while lower values like 0.2 will make it more focused and deterministic.

We generally recommend altering this or `top_p` but not both.

top_p number Optional Defaults to 1

An alternative to sampling with temperature, called nucleus sampling, where the model considers the results of the tokens with top_p probability mass. So 0.1 means only the tokens comprising the top 10% probability mass are considered.

We generally recommend altering this or `temperature` but not both.

Example request

curl Copy

```
1 curl https://api.openai.com/v1/chat/completions \
2 -H "Content-Type: application/json" \
3 -H "Authorization: Bearer $OPENAI_API_KEY" \
4 -d '{
5   "model": "gpt-3.5-turbo",
6   "messages": [{"role": "user", "content": "Hello!"}
7 ]'
```

Parameters

Copy

```
1 {
2   "model": "gpt-3.5-turbo",
3   "messages": [{"role": "user", "content": "Hello!"}]
4 }
```

Response

Copy

```
1 {
2   "id": "chatcmpl-123",
3   "object": "chat.completion",
4   "created": 1677652288,
5   "choices": [{
6     "index": 0,
7     "message": {
8       "role": "assistant",
9       "content": "\n\nHello there, how may I assist
10    },
11    "finish_reason": "stop"
12  }],
13   "usage": {
14     "prompt_tokens": 9,
15     "completion_tokens": 12,
16     "total_tokens": 21
17  }
18 }
```

Conclusion

Le projet Codehub avait pour visée de produire en équipe une application pour smartphone de type forum de discussion. J'ai personnellement pris ce projet avec un grand professionnalisme et veillé à ce que chacun dans l'équipe puisse exercer différents types de responsabilités dans celle-ci.

Le choix de Symfony était logique au vu de la forme que prend un forum, en effet, inutile d'avoir une API REST d'une grande célérité pour faire fonctionner un forum, nous avions surtout besoin d'une architecture relativement simple à déployer, sécurisée et fiable, ce qu'offre un projet Symfony, l'utilisation de la bibliothèque API Plateforme possible sur le projet Symfony était aussi une bonne option pour gagner en productivité et en temps de développement de l'API.

Le choix de coder l'application dans le framework React Native réside dans sa simplicité de déploiement, qu'il utilise les librairies React ainsi que le langage JavaScript. Il permet aussi de "fabriquer" rapidement une application sur portable de manière fluide, on peut ajouter à cela une communauté de développeurs assez riche, permettant d'obtenir de nombreux tutoriels sur le net, ainsi que des espaces d'échange pour poser des questions en cas de blocage.

L'utilisation de Expo Go résidait surtout dans sa simplicité de déploiement et notamment la prise en charge du Build l'exportation très simple de l'application sur un téléphone portable pour voir quasi en direct les actions que nous menions sur le développement de l'application. L'un des inconvénients d'Expo Go était surtout dans la publication de notre application, que nous n'avons pas encore réussi à finaliser correctement.

Globalement, j'avais lors de ce projet, l'objectif de créer une application pour Smartphone, ce qui était pour moi même et notre équipe, une nouveauté. En quelques semaines d'intenses travaux, sans compter nos heures à développer, lire de la documentation, expérimenté des nouvelles technologies et approche de production, nous sommes arrivés à nos fins.

N'ayant jamais travaillé en équipe à un projet commun, j'ai appris plusieurs choses très importantes pour la suite de ma carrière professionnelle. La force de la délégation de certaines tâches à des personnes bien plus spécialisé que moi dans certains domaines. La puissance de l'intelligence collective, permettant d'aborder les problématiques avec plusieurs angles d'attaque. L'organisation en pôle de responsabilité afin d'optimiser au mieux notre temps de travail, l'utilisation d'outils de gestion du temps et du planning de chacun.

Cette expérience fut, pour moi, très enrichissante. Enfin, l'outil Github, qui nous a permis de travailler de concert sur un même projet sans jamais nous perdre sur des branches inconnues et hasardeuses.

Je vois dans les prochaines fonctionnalités à prévoir dans codehub, divers éléments :

- ◆ Le nombre de vues des articles.
- ◆ Classement des articles par nombre de vues pour promouvoir les contenus les plus populaires.
- ◆ Ajouter des images d'illustrations dans les articles.

Annexes

- ❑ Le lien vers l'API :
<https://rany-alo.students-laplateforme.io/app-mobile-forum/public/api>
- ❑ Le lien vers la maquette du projet :
<https://www.figma.com/file/roTKUxyD5fAzghKQxiX30q/FormReactNative?type=design&node-id=6-18&t=Y9zBkl8745eydAha-0>
- ❑ Le lien vers le GIT de l'API :
<https://github.com/rany-alo/app-mobile-forum>
- ❑ Le lien vers l'interface web (panel admin) :
<https://rany-alo.students-laplateforme.io/panel-admin/build/>
- ❑ Lien vers le MLD du projet :
<https://dbdiagram.io/d/6141c179825b5b0146031a0c>
- ❑ Le lien vers le GIT du projet panel admin :
<https://github.com/rany-alo/panel-admin>
- ❑ Le lien vers le GIT de l'interface desktop :
<https://github.com/rany-alo/articleGenerator>