Concepteur et ddéveloppeur d'application
Projet professionnel
Codehub
Christophe Calmes
```
import React, { useState, useEffect } from 'react'

import { Text, Image, View, ScrollView } from 'react-native'
import { StatusBar } from 'expo-status-bar';
import { SafeAreaView } from 'react-native-safe-area-context';
import ModalConnexion from '../../Composants/Modals/ModalConnexion';
import ModalInscription from '../../Composants/Modals/ModalInscription';
import styles from '../../../assets/styles/styles';
import { useFonts, Iceland_400Regular } from '@expo-google-fonts/iceland';

import ButtonComponent from '../../Composants/Bouton/buttonComponent';

export default function LandingScreen({ navigation }) {

    const [isFormConnexionVisible, setIsFormConnexionVisible] = useState(false)
    const [isFormInscriptionVisible, setIsFormInscriptionVisible] = useState(false)

    const _toggleFormConnexion = () => {
        setIsFormConnexionVisible(!isFormConnexionVisible)
    }
    const _toggleFormInscription = () => {
        setIsFormInscriptionVisible(!isFormInscriptionVisible)
    }

    let [fontsLoaded] = useFonts({
        Iceland_400Regular,
    });
    if (!fontsLoaded) {
        return null;
    }

    return (
        <SafeAreaView st    s.container}>
            <Scrol    >
                <V        {styles.imgContainer}>
                    <Image
                        style={styles.img}
                        source={require('../../../assets/Logo/                e45.webp')}
                    />
                </View>
                <View style={styles.textContainer}
                    <Text style={styles.H1B}>CODEHUB</Text>
                </View>
                <ButtonComponent
                    contButon={styles.contenerCenter}
                    button={styles.butonStyle}
                    txtButton={styles.textButon}
                    text={"Entrer"}
                    onPress={() => navigation.navigate('Articles', { refresh: true })}
                />
                <ButtonComponent
                    contButon={styles.contenerCenter}
                    button={styles.butonStyle}
                    txtButton={styles.textButon}
                    text={"Connexion"}
                    onPress={_toggleFormConnexion}
                />

                <ButtonComponent
                    contButon={styles.contenerCenter}
                    button={styles.butonStyle}
                    txtButton={styles.textButon}
                    text={"Inscription"}
                    onPress={_toggleFormInscription}
                />
                {isFormConnexionVisible && <ModalConnexion onPress={_toggleFormConnexion} onClose={_toggleFormConnexion}
nav={navigation} />}
                {isFormInscriptionVisible && <ModalInscription onPress={_toggleFormInscription} onClose=
{_toggleFormInscription} nav={navigation} />}
                <StatusBar style="light" />
            </ScrollView>
        </SafeAreaView>
    )
}
```

Présentation du projet

Le projet CodeHub repose sur plusieurs objectifs et a été développé par une équipe de quatre développeurs dans le cadre de notre formation en tant que "Concepteurs Développeurs d'Applications". Ce projet représente le point culminant de notre formation et nous a permis d'acquérir une expérience variée dans la mise en place, la conception et la création d'une application.

Code Hub propose un forum permettant aux utilisateurs de publier des articles et de les commenter. Cette fonctionnalité favorise les échanges et les discussions au sein de la communauté.

L'application dispose également d'une interface dédiée aux administrateurs, qui leur permet de gérer les utilisateurs, de modérer, de modifier ou de supprimer les articles, ainsi que de faire de même avec les commentaires. Cette fonctionnalité donne aux administrateurs un contrôle complet sur le contenu de l'application et garantit un environnement sûr et de qualité pour les utilisateurs.

En résumé, le projet Code Hub est le fruit de notre formation en tant que développeur d'applications. Il nous a permis d'acquérir une expérience précieuse dans la mise en place, la conception et la création d'une application complète. Avec des fonctionnalités telles qu'un forum d'articles et une interface d'administration, Code Hub vise à offrir une expérience interactive et enrichissante pour les utilisateurs et les administrateurs.

Apprentissages techniques

Le projet Code hub s'appuie sur plusieurs types de technologies, notamment, les API REST créées à l’aide du Frameworks Symfony et API plateforme, la création d’applications mobiles avec React Native et la librairie Axios.

Ces éléments ont été pour toute l’équipe, une opportunité de monter en compétence sur ces technologies.

Travail en équipes

Pour les membres de notre équipe, nous avons dû nous organiser selon les forces de chacun d’entre nous. Répartir le travail et monter en compétence sur les divers éléments du projet.

Création d’un forum pour application mobile

Le projet Code hub a été développé en tenant compte des normes de sécurité en vigueur, ainsi que la RGPD.

- Le projet Code hub possède plusieurs fonctionnalités :
- Lecture d’article,
- Lecture des commentaires,
- Modération ou suppression des articles,
- Modération ou suppression des commentaires,
- Création d’un compte utilisateur,
- Administration de son compte utilisateur,
- Création d’articles si l’on est connecté en tant qu’utilisateur,
- Création de commentaires sous les articles si l’on est connecté en tant qu’utilisateur,
- Administration de son profil utilisateur,
- Possibilité d'effacer son compte utilisateur

La maquette du projet Code hub

Conception de la maquette

Code hub est un projet commun, il a donc était 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 critère esthétique, veillé à ce que la lisibilité soit bonne, vu que nous avons créer 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.

Réunion préparatoire

La réunion préparatoire a permis d'établir la structure du site, nous étions quatre à apporter des idées et à en débattre. Chacun avec sa sensibilité et son vécu en tant que développeur, nous avons donc établi diverses possibilités. Nous avons définis ensemble un parcours utilisateur ainsi qu'établi pour chaque fonctionnalité de l'application, la forme qu'elle devait prendre.

Le choix d’un nuancier de couleur

Le choix de couleur a été essentiellement définis 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 définis la palette de couleur avec ce site.

En tant que leader sur la création de la maquette, je me suis assuré que mon choix de contraste de couleur était bien lisible avec un outil de adobe color.

Wireframe

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.

Maquette

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

Une vision globale de la maquette.

La landing page

La page d'accueil, également appelée landing page, est la première interface que les utilisateurs rencontrent lorsqu'ils ouvrent l'application. Son objectif principal est de fournir un point central de départ pour faciliter la navigation des utilisateurs. La conception de cette page a été pensée pour offrir une expérience utilisateur simple et intuitive, notamment pour les nouveaux utilisateurs découvrant l'application pour la première fois.

L'ensemble des autres panneaux ont été créés pour correspondre à un forum en ligne.
Nous pouvons retrouver dans les autres panneaux de la maquette, des éléments liés à la publication d'article, la création de commentaires, l'édition d'article et l'administration du site par des utilisateurs avec les droits utilisateurs.

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.

La première option de navigation en React Native est la navigation basée sur des piles (stack navigation). Elle permet d'empiler des écrans les uns sur les autres et de naviguer entre eux en utilisant des boutons de navigation ou des gestes de balayage. Cette méthode est idéale pour les applications qui ont une structure de navigation simple avec une hiérarchie linéaire d'écrans.

La deuxième option de navigation est la navigation basée sur des onglets (tab navigation). Cette méthode permet d'afficher différentes sections de l'application sur des onglets, chacun ayant son propre contenu. Les utilisateurs peuvent naviguer entre les onglets en appuyant sur l'onglet souhaité. Cette méthode est idéale pour les applications qui ont plusieurs sections distinctes qui peuvent être explorées de manière indépendante.

La troisième 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.

Le Drawer Navigation est une fonctionnalité de navigation couramment utilisée dans les applications mobiles pour améliorer l'expérience utilisateur, offrir un design moderne et élégant et faciliter la navigation entre les différentes sections de l'application. Pour l'utiliser, nous avons besoin d'une bibliothèque de navigation qui supporte cette fonctionnalité et nous devons implémenter le menu déroulant et la gestion de l'interaction avec l'utilisateur en utilisant les composants fournis par la bibliothèque.

Organisation en couche de l’application CodeHub

L'application Code Hub est conçue selon une architecture en couches qui permet une organisation claire et efficace de ses fonctionnalités. Chaque couche joue un rôle spécifique dans le fonctionnement global de l'application.

La première couche de l'architecture est la couche API. Elle utilise une ***API plateforme*** pour gérer les interactions entre l'application et les services externes. Cette couche facilite l'intégration de différentes fonctionnalités, telles que l'authentification, l'accès aux données et les fonctionnalités supplémentaires, en se basant sur des normes et des protocoles standardisés. L'utilisation de l'API plateforme permet une meilleure modularité et une évolutivité de l'application.

La deuxième couche est la couche logicielle, qui constitue l'interface utilisateur principale de l'application. Cette couche est développée à l'aide de React Native, un framework de développement d'applications mobiles multiplateformes. Elle permet aux utilisateurs de naviguer facilement à travers les différentes fonctionnalités de l'application, notamment le forum. Grâce à React Native, l'application offre une expérience utilisateur réactive et fluide, quel que soit le système d'exploitation utilisé.

La couche logicielle comprend également la logique métier de l'application, qui gère les interactions entre les différentes fonctionnalités et les données de l'API. Elle permet de gérer les actions des utilisateurs, d'effectuer des calculs ou des traitements spécifiques, et de synchroniser les données avec l'API et la base de données.
Une couche de sécurité créée avec JWT permet une connexion sécurisée à l'API et l'identification de tous les utilisateurs.

L'organisation en couches de l'application Code Hub offre plusieurs avantages. Elle permet une séparation claire des responsabilités, facilitant la maintenance et l'évolutivité de l'application. Chaque couche peut être développée, testée et déployée indépendamment des autres, ce qui favorise la flexibilité et la réutilisation du code. De plus, cette architecture facilite la collaboration entre les développeurs travaillant sur différentes parties de l'application.

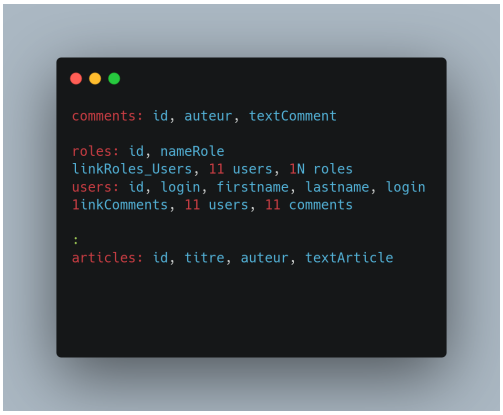
L'application Code Hub est organisée en couches, comprenant une couche API utilisant une API plateforme pour faciliter l'intégration de fonctionnalités, et une couche logicielle permettant aux utilisateurs de naviguer sur le forum grâce à React Native. Cette organisation en couches favorise une meilleure structuration de l'application, améliore l'expérience utilisateur et facilite le développement et la maintenance de l'application.

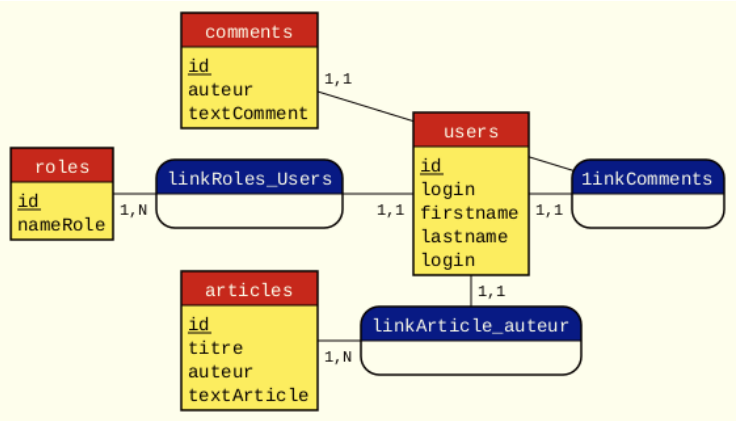
Présentation base de données

La base de données a été le premier élément que nous avons dû conceptualiser. Pour ce faire, nous avons utilisé divers outils pour la mettre en œuvre. Les fonctionnalités de l'application nous ont permis de mieux comprendre la structure des tables et leur interaction dans la future application. Nous avons pris en compte l'ensemble des fonctionnalités pour concevoir les tables de la base de données ainsi que leur interaction. Nous avons ensuite créé le MCD et le MLD de la base de données pour nous assurer que tout ce que nous avions prévu était possible dans notre modèle de données. Ce travail en amont nous a permis de comprendre et anticiper le déroulement du développement de l'application future.

Le MCD

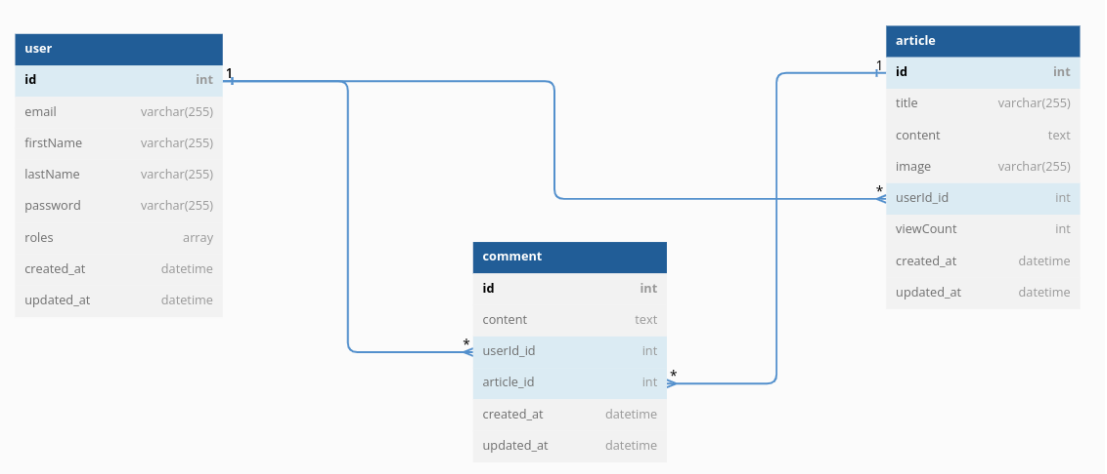
Le MCD (Modèle Conceptuel de données) a été créé avec l'outil en ligne Mocodo qui permet de manière simple, de créer un MCD via quelques lignes de commande et obtenir une représentation graphique claire de ce MCD. Cet outil permet de gagner du temps dans la conception d'une base de données.





Le MLD

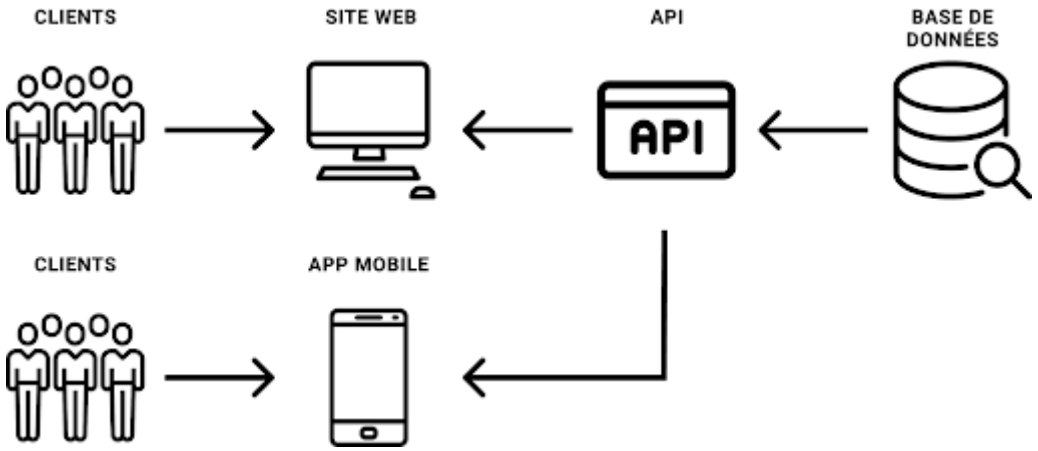
Le MLD (Modèle Logique de données) a été créé en se basant sur le MCD.



Description de l’API REST

Dans cette section, je vais vous présenter notre API réalisée avec les frameworks API Platform et Symfony. Dans ce projet, nous avons utilisé ces technologies pour créer une API REST performante et facile à utiliser.

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)

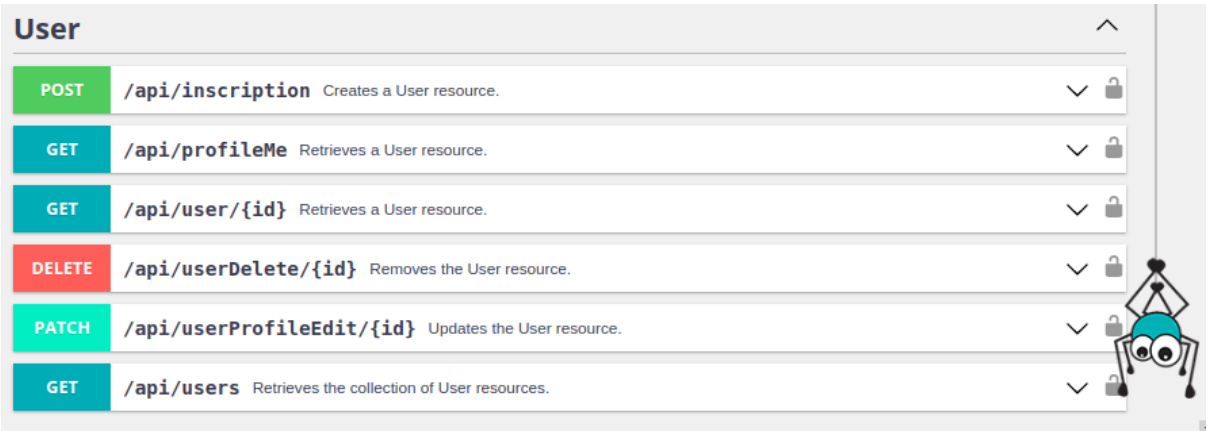


Une API REST (Representational State Transfer) est un style d'architecture qui permet à différents systèmes de communiquer entre eux via des requêtes HTTP. Cette architecture est devenue très populaire ces dernières années grâce à sa flexibilité et à sa capacité à fonctionner avec un large éventail de langages de programmation et de systèmes.

Les API REST fonctionnent en envoyant des requêtes HTTP à des ressources spécifiques, qui peuvent être des données, des fichiers ou d'autres types de ressources. Les réponses renvoyées par l'API sont généralement des données structurées dans des formats tels que JSON ou XML. Il repose sur un certain nombre de principes clés, tels que l'utilisation de verbes HTTP standard tels que GET, POST, PUT, PATCH et DELETE pour effectuer des opérations sur les ressources, l'utilisation de l'URI pour identifier de manière unique les ressources, et l'utilisation de l'état représentationnel pour représenter l'état des ressources.

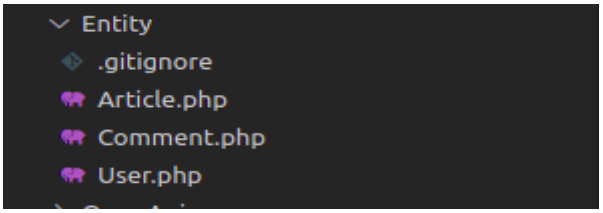
Dans ce projet, nous avons utilisé API Platform, un framework web utilisé pour générer des API REST et GraphQL4, se basant sur le patron de conception MVC, tout en offrant des fonctionnalités avancées telles que la documentation automatique de l'API, la validation des données, la sécurité, etc. La partie serveur du framework

est écrite en PHP et basée sur le framework Symfony, tandis que la partie client est écrite en JavaScript et TypeScript.



Les entités

Notre API est conçue pour servir notre application mobile Code Hub. Elle permet aux utilisateurs de saisir et stocker des informations dans trois entités distinctes : utilisateurs, articles et commentaires. Grâce à cette API, les données collectées peuvent être facilement traitées et utilisées pour alimenter les fonctionnalités de notre application mobile.



Entité user

```php
#[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
        ),
```

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.

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

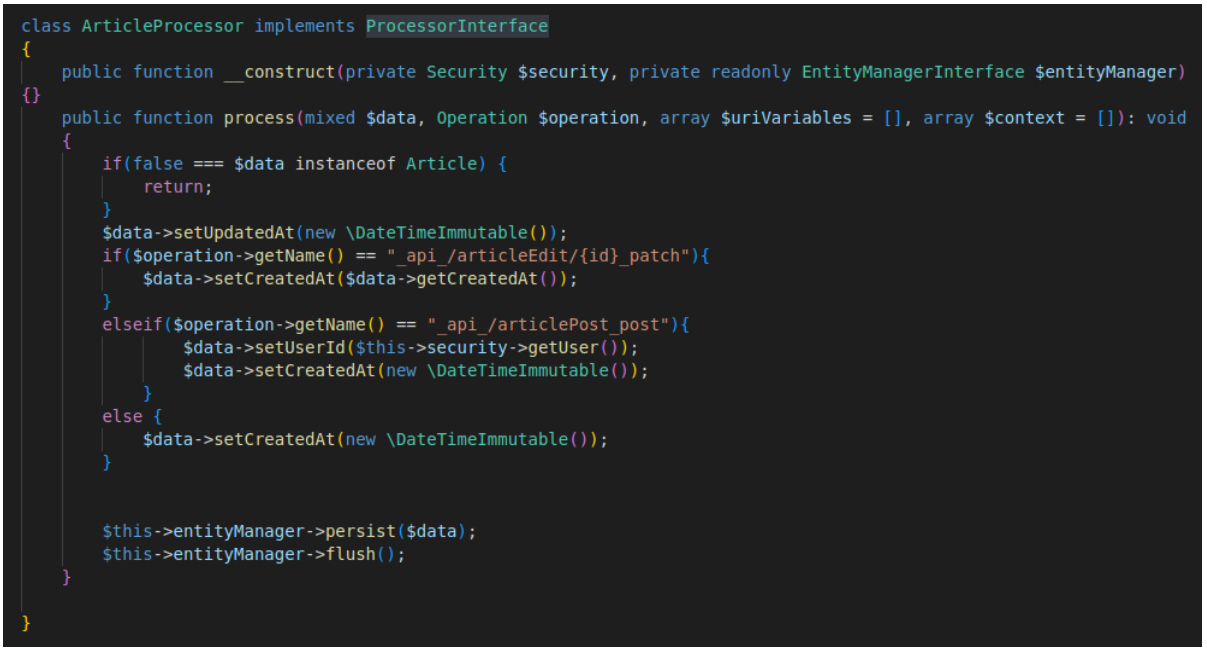
```php
class CommentStateProvider implements ProviderInterface
{
    public function __construct(private readonly ArticleRepository $articleRepository, private Security $security)
    {}

    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 persistances des données

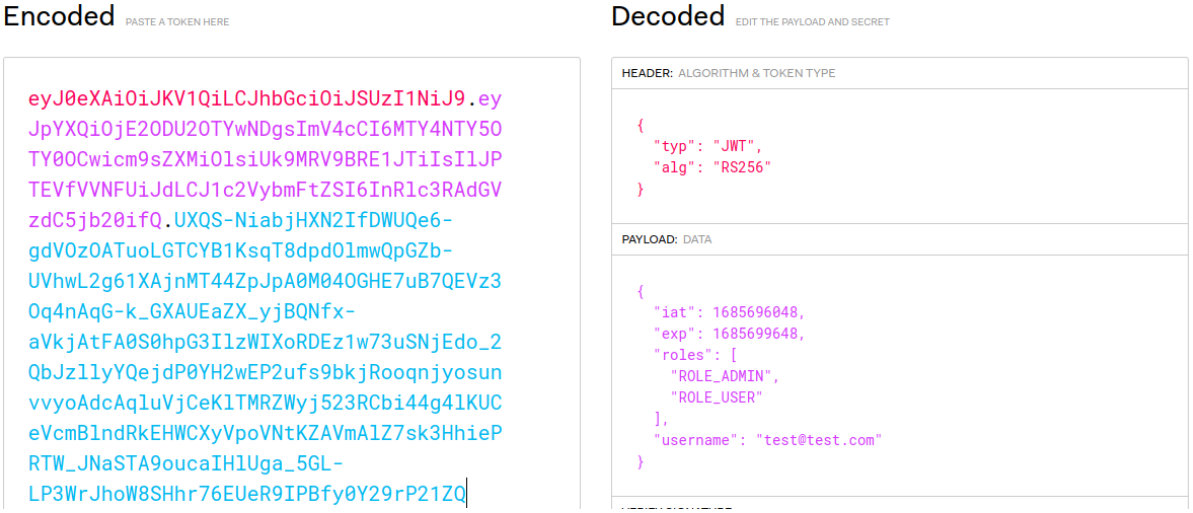


```php
class ArticleProcessor implements ProcessorInterface
{
    public function __construct(private Security $security, private readonly EntityManagerInterface $entityManager)
{}
    public function process(mixed $data, Operation $operation, array $uriVariables = [], array $context = []): void
    {
        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 authentification

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.



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 :

```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

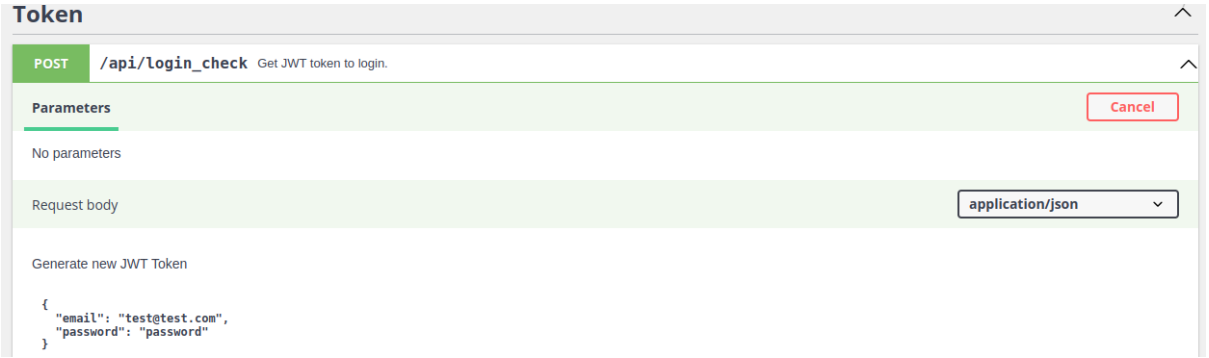
```yaml
controllers:
    resource: ../src/Controller/
    type: attribute
api_login_check:
    path: /api/login_check
    methods: ['POST']
```

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

config > packages > ! api_platform.yaml

```yaml
api_platform:
  collection:
    pagination:
      page_parameter_name: _page
  defaults:
    pagination_items_per_page: 5
  swagger:
    api_keys:
      JWT:
        name: Authorization
        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 :



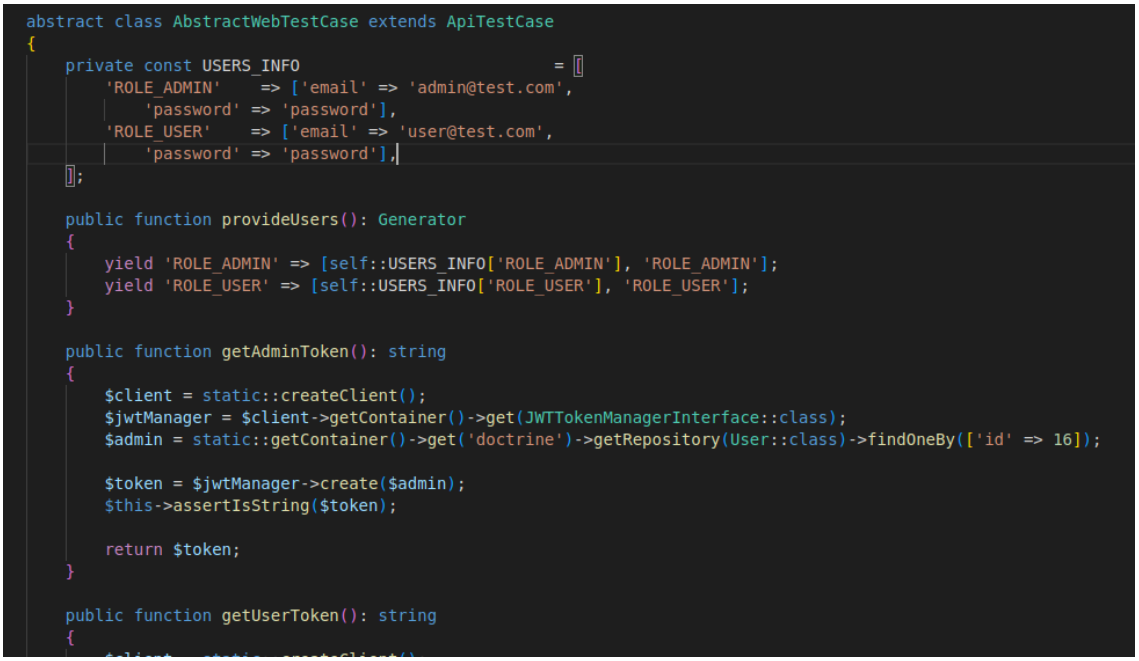
Tests unitaires

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.

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

        $this->user->expects($this->once())->method('setUpdatedAt');
        $this->user->expects($this->once())->method('setCreatedAt');
        $this->user->expects($this->any())->method('getCreatedAt')->willReturn($this->dateTimeImmutable);


        $this->operation->expects($this->any())->method('getName')->willReturnOnConsecutiveCalls('_api_/articleEdit/{id}_patch');

        $this->user->expects($this->any())->method('getPlainPassword')->willReturn('password');
        $this->user->expects($this->once())->method('setPassword');
        $this->userPasswordHasher->expects($this->once())->method('hashPassword')->with($this->user, 'password');
        $this->user->expects($this->any())->method('getPassword');


        $processor = new UserProcessor($this->userPasswordHasher, $this->entityManager);

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

Test 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.



```php
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
    {
```

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

```php
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.

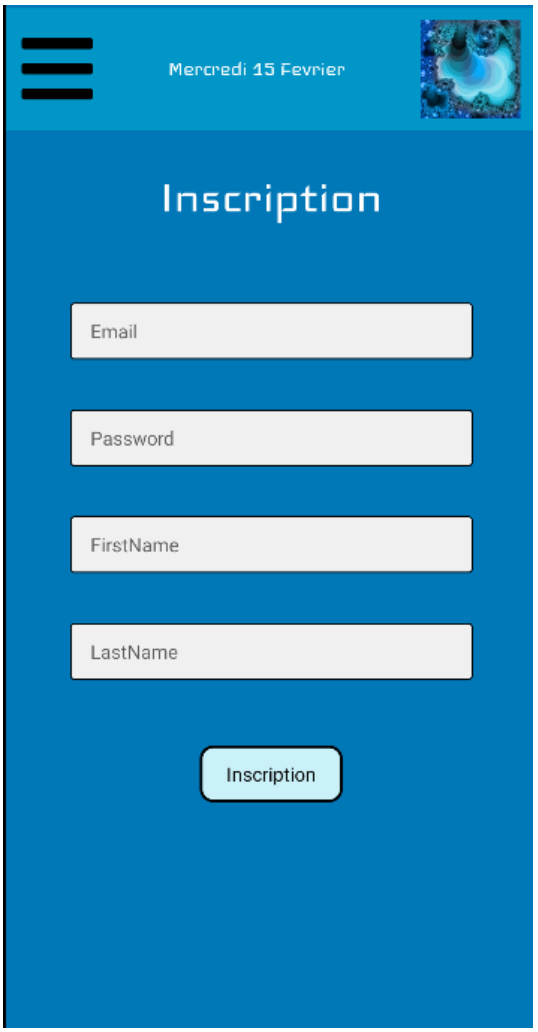
Les fonctionnalités du projet Code hub

Code hub est un forum en version application mobile. Ainsi, il regroupe les fonctionnalités classiques d'un forum que l'on pourrait retrouver sur un site web.

Utilisateur non connecté

Création d'un compte utilisateur

Un utilisateur non connecté peut créer un compte utilisateur. Pour ce faire, il doit renseigner un ***courriel***, un nom et un prénom et définir un mot de passe.



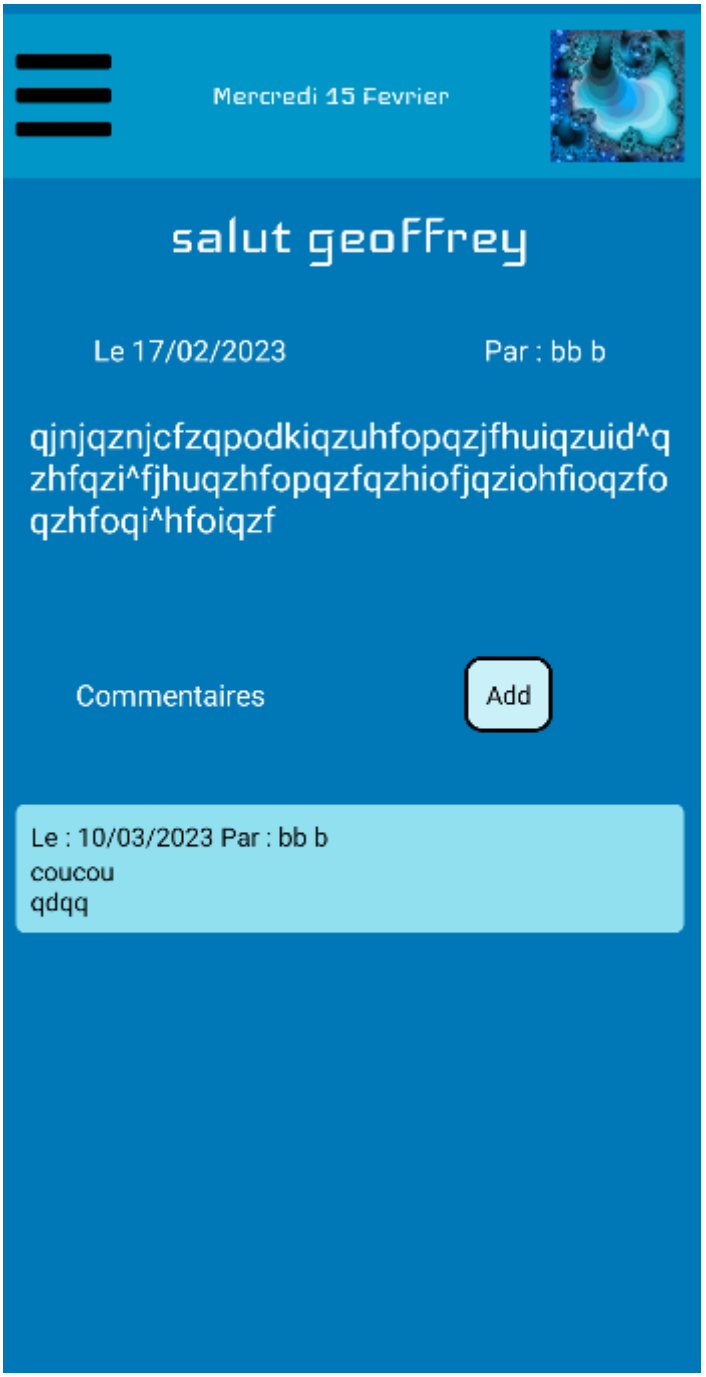
Les articles

Les articles sont publics, c'est-à-dire qu'ils sont accessibles en lecture à toute personne ayant l'application disponible. Un visiteur non identifié peut donc lire les articles librement.

Les utilisateurs connectés à l'application via leur compte peuvent consulter les articles eux aussi.

Les commentaires

Les commentaires sont faits pour enrichir un article. Ainsi, leur lecture est libre pour toute personne ayant l'application Code hub disponible sur son téléphone et même si il n'est pas identifié.Modération ou suppression des articles.
Les utilisateurs connectés à l'application via leur compte peuvent consulter les commentaires eux aussi.

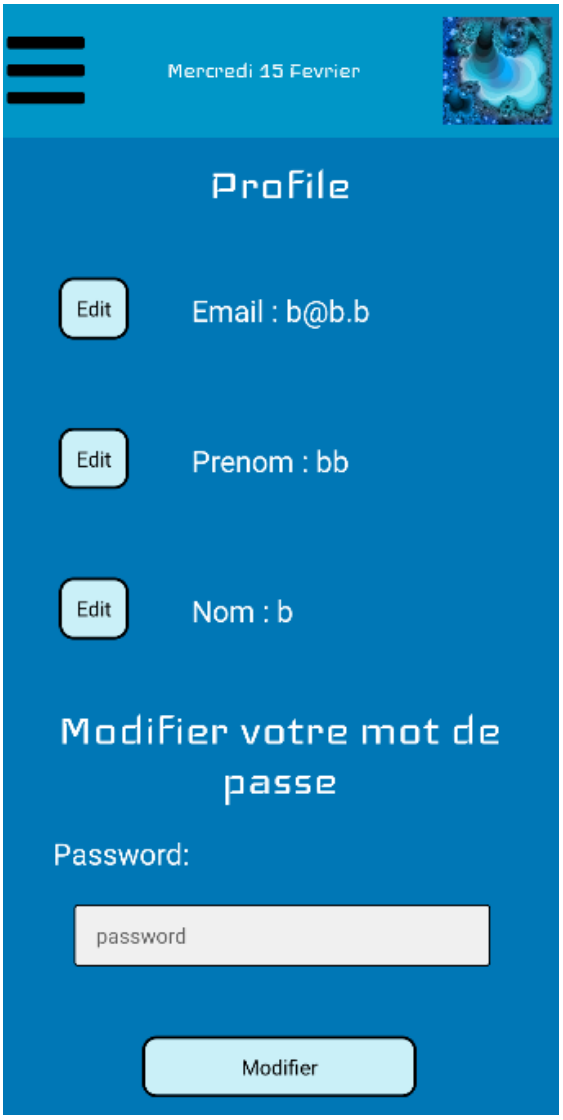


Utilisateur connecté

Administration de son compte utilisateur

Pour administrer son compte utilisateur il suffit de se rendre dans la section profil, disponible dans le menu.
La page permet de :

- Modifier son nom
- Modifier son prénom
- Modifier son mail
- Modifier son mot de passe
- Supprimer son compte

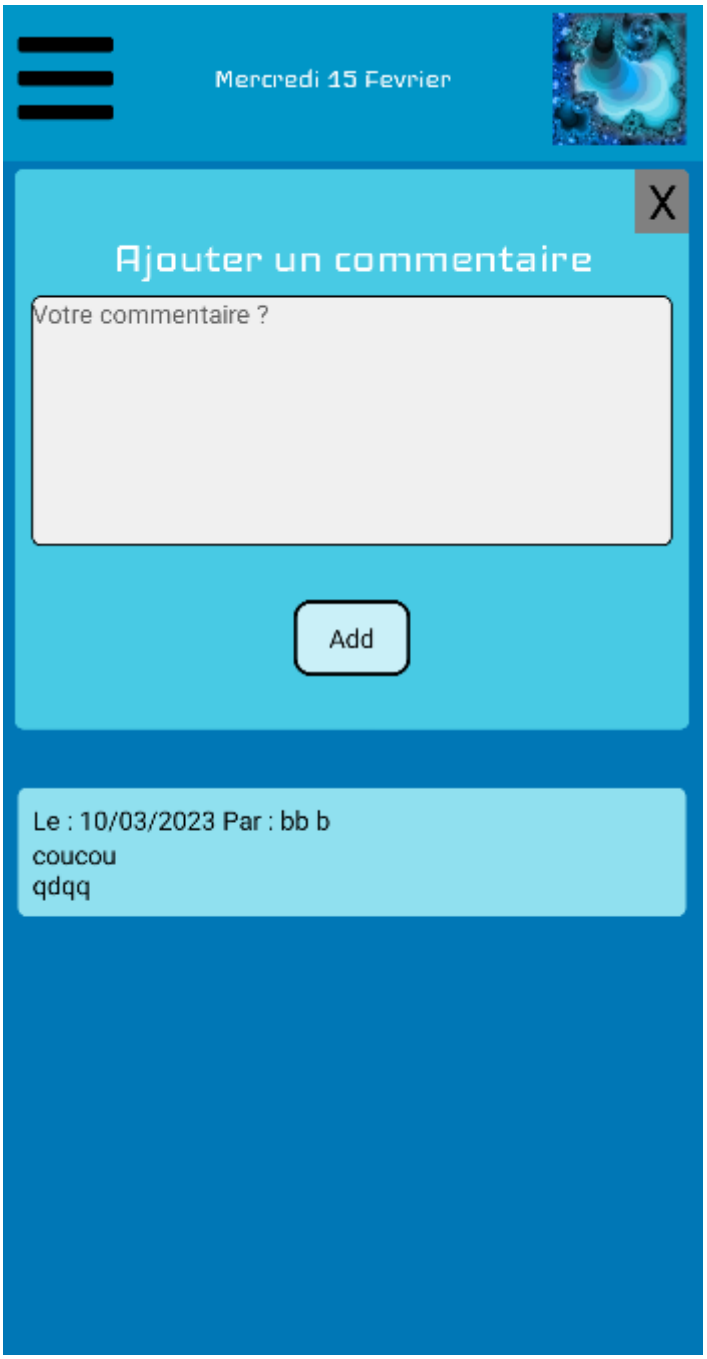


Création d’articles

Créer un article se fait en se rendant sur la page “Création d’article” et l’on peut alors y rédiger un article d’un minimum de 750 caractères. Une fois écrit, il suffit de l’enregistrer en actionnant le bouton “enregistrer” en bas de l’article. Celui-ci est alors immédiatement publié et visible par l’ensemble des utilisateurs.

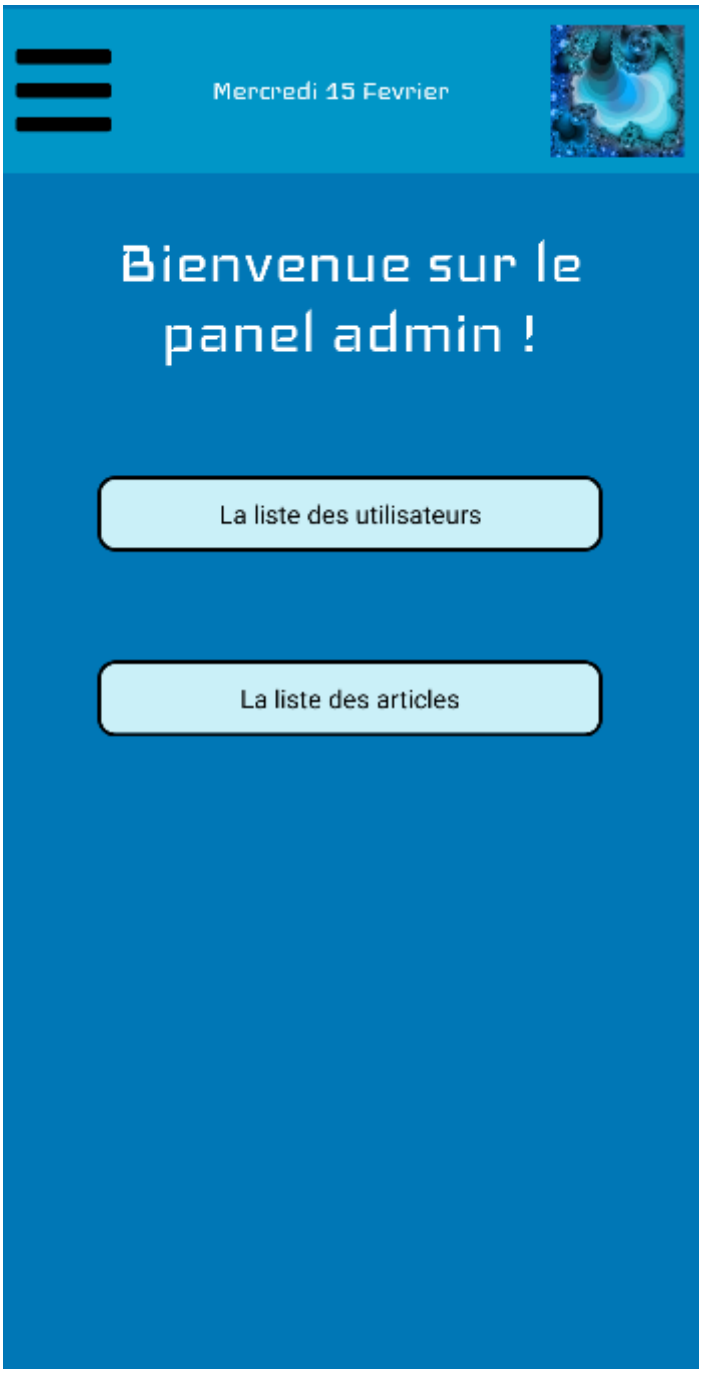
Création d’un commentaire

Pour créer un commentaire, il suffit d’aller sur l’article que l’on veut commenter et ajouter un commentaire via le bouton “ajouter un commentaire” en bas de l’article. Une fois que l’on a rédigé le commentaire, il suffit de cliquer sur le bouton “enregistrer” pour que le commentaire soit visible et disponible sur l’application.



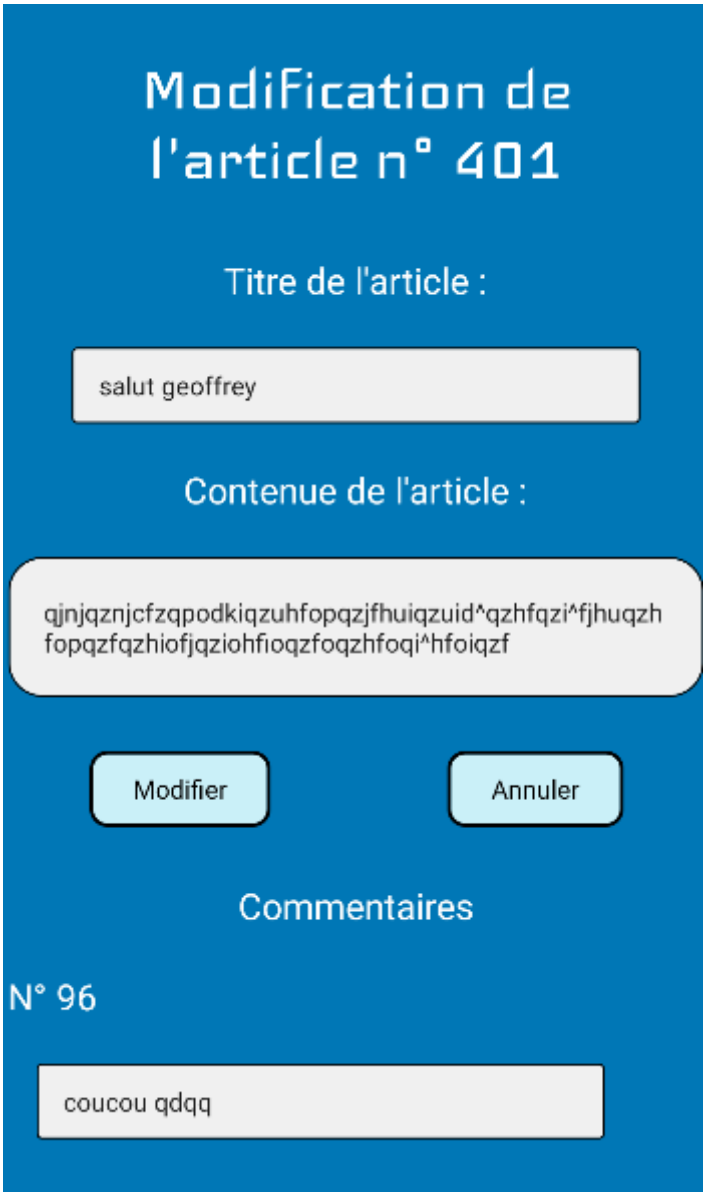
Administrateur du site

L'administration du site se fait directement sur l'application. Les utilisateurs avec un rôle administrateur peuvent accéder aux fonctionnalités décrites ci-dessous. Nous avons voulu faire un panel administrateur intégré directement à l'application pour simplifier l'administration du site et proposer un produit 100% intégré et simple de déploiement.



Modération des articles

La modération d'article se fait via le menu "admin" > La liste des articles.
Cela ouvre une liste des articles paginés et triés dans leur ordre de rédaction.
Chaque article propose 2 choix, la suppression ou la modification.
Pour modérer un article, il faut prendre l'option modification.
L'ouverture de l'article permet de le modifier, puis de l'enregistrer à nouveau. L'article est alors modifié.



Suppression des articles

La suppression d'article se fait simplement en cliquant sur la poubelle, un pop-up vous demande de confirmer la demande de suppression, une fois valider. L'article est totalement supprimé.

Modération des commentaires

La modération des commentaires se fait sur la même page que la modération des articles. Chaque commentaire est accessible pour pouvoir être modifié. Il suffit de modifier l'article directement dans le champ texte et cliquer sur "enregistrer" le commentaire.

Suppression des commentaires

La suppression des commentaires se fait en cliquant sur le bouton "Supprimer" du commentaire que l'on veut supprimer.

Back-end

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 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 si il y en a, et enfin elle renvoie en callback la réponse (ou l'erreur).

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

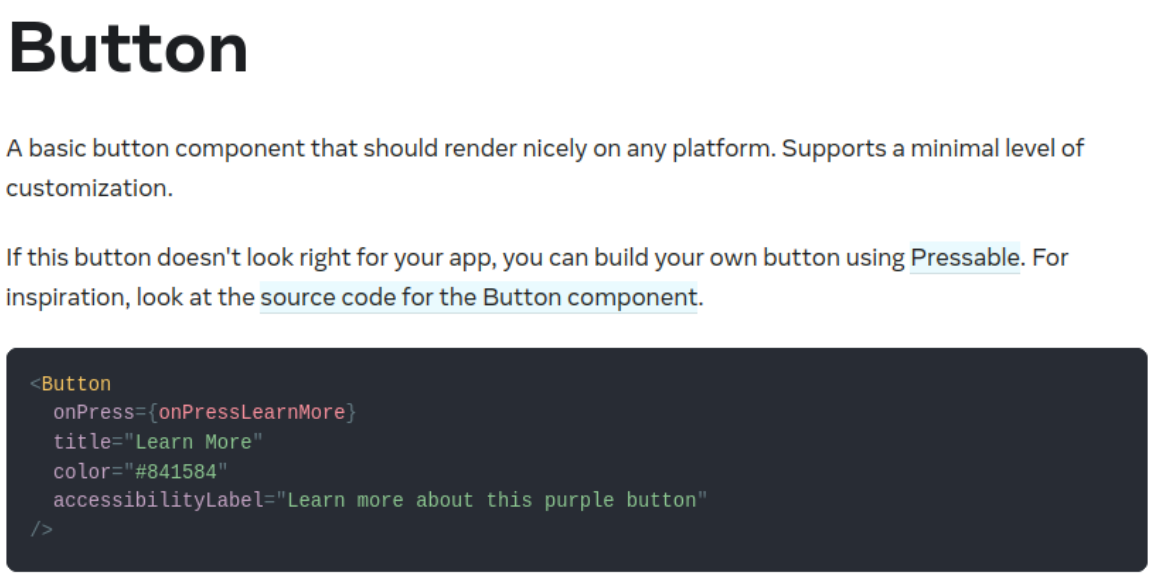
```
import { getArticles } from '../../../api';

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 paramètre il y a, il suffit de les envoyer à la méthode en premier temps, et ensuite récupérer la réponse en fonction fléché anonymes.

Traduction de doc sur react native

Pour ce projet, nous avons consulté la documentation officielle de React Native disponible à cette adresse documentation React Native.

Exemple :



Button

A basic button component that should render nicely on any platform. Supports a minimal level of customization.

If this button doesn't look right for your app, you can build your own button using Pressable. For inspiration, look at the source code for the Button component.

```
<Button
  onPress={onPressLearnMore}
  title="Learn More"
  color="#841584"
  accessibilityLabel="Learn more about this purple button"
/>
```

J'ai traduit cette élément ainsi :

Un composant "bouton" a un rendu "joli" sur n'importe quelle plateforme. Il peut accueillir un nombre minimum de niveaux de personnalisation.

Si le bouton ne convient pas à votre application, vous pouvez créer un bouton "pressable" Vous trouverez l'inspiration en regardant le code pour les "composants de bouton".

Projet d’avenir et prochaines fonctionnalités

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.

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 de 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. Je suis également conscient que ce n'est qu'un galop d'essais et qu'il faut maintenant tenter de produire une autre application sur smartphone. Je sais que certains membres de notre équipe travaillent déjà à cet objectif.
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. L'importance des réunions matinales et de clôture chaque jour, afin de faire le point sur les défis à venir et les progressions de la journée passée.
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 remercie entre autres, le personnel de la plateforme, notamment Ruben Habib notre responsable pédagogique ainsi que Clément Caillat qui a su en tant qu'accompagnateur pédagogique, nous donner les clés pour que nous puissions en autonomie, créer ce projet.
Il me semble aussi important de préciser que sans Rany Alo, Mathieu Ruiez, Geoffrey Meca alias "Le Lead Dev", ce projet n'aurait sans doute pas abouti.