

Meca Geoffrey

Concepteur développeur d'applications

Dossier Projet

-

Code hub

2023

Sommaire

Concevoir et développer des composants d'interface utilisateur

Remerciements	4
Présentation du projet	5
Le projet	5
Présentation du projet en anglais	6
Travail en équipes	7
Apprentissages techniques	8
La maquette du projet	8
Conception de la maquette	8
Réunion préparatoire	8
Le choix d'un nuancier de couleur	9
Maquette	10
La landing page	11
L'inscription	10
Liste des articles	11
Article	12
Ajouter un article	13
Choix de la navigation	14
Concevoir et développer la persistance des données	14
Présentation base de données	14
Le MCD	15
Le MLD	15
Description de l'API REST	16
Les entités	16
Entité user	17
State Provider	17
State Processor	18
Tests Unitaire	19
Tests Fonctionnel	21

Concevoir et développer une application multicouche

Les fonctionnalités du projet Code hub	
Back-end	23
Utilisateur non connecté	23
Création d'un compte utilisateur	23
Les articles	24
Les commentaires	25
Utilisateur connecté	26
Administration de son compte utilisateur	27
Création d'articles	28
Création d'un commentaire	29
Administrateur du site	30
Modération des articles	31
Suppression des articles	31
Modération des commentaires	31
Suppression des commentaires	31
Ajout et appelle d'une route	33
Projet d'avenir et prochaines fonctionnalités	33
Conclusion	34

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, Céline 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 Thibault Pattieu, Chargée des relations entreprises, qui 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.

Présentation du projet

Le projet Code hub se base sur plusieurs objectifs. Il a été développé par une équipe de quatre développeurs, dans le cadre de notre formation en tant que “Concepteur développeur d’application”. Ce projet est le point d’orgue de notre formation et nous a apporté diverses expériences dans la mise en place, la conception et la création d’une application.

Le Projet

Le forum Code Hub est une application mobile Android, développée pour permettre aux utilisateurs de publier des articles et de commenter ceux des autres.

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.

Il 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

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.

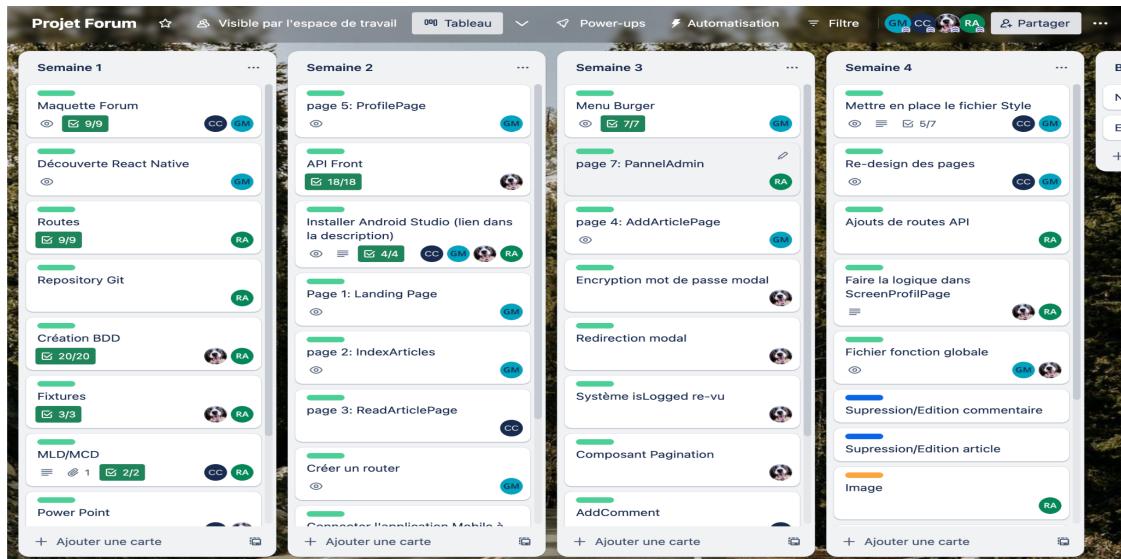
Throughout these development tasks, I gained valuable experience in utilizing modern frameworks and technologies to create user-friendly and efficient interfaces across multiple platforms.

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.

Travail en équipes

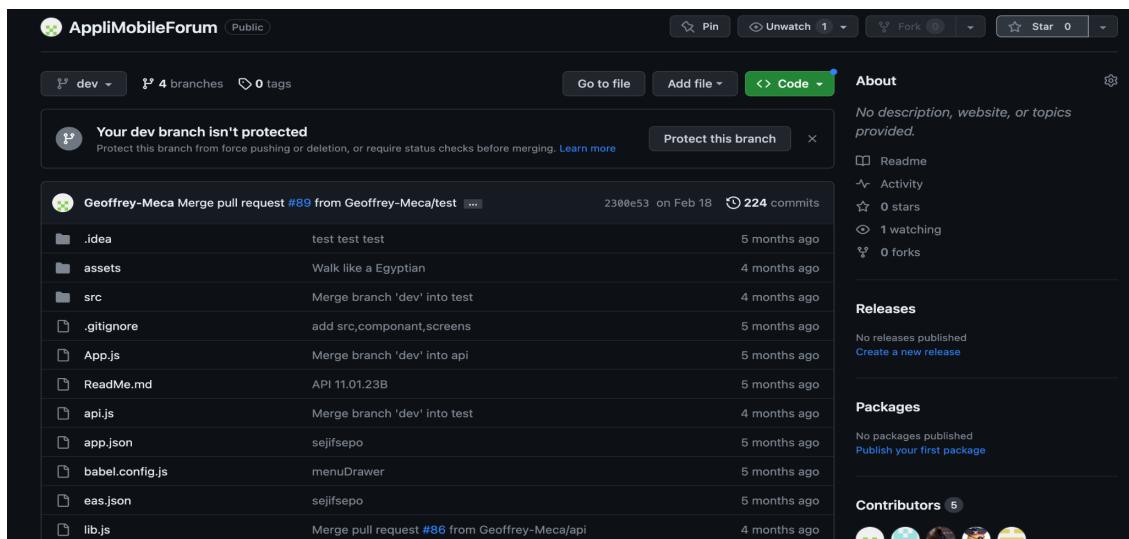
La première étape dans la réalisation d'un projet informatique est de construire une équipe solide de développeurs qui peuvent collaborer efficacement pour atteindre les objectifs du projet.

Nous avons constitué une équipe de quatre développeurs et utilisé les méthodes de **Kanban** et **Trello** pour le suivi des tâches.



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.



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.

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.

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.



Wireframe

Le Wireframe a permis 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.



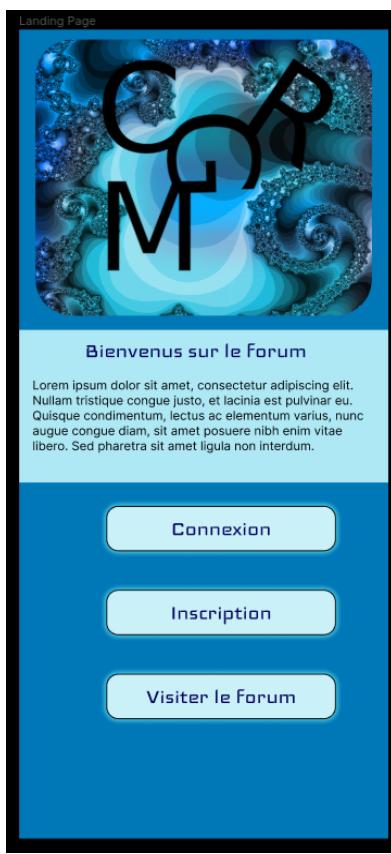
Maquette

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

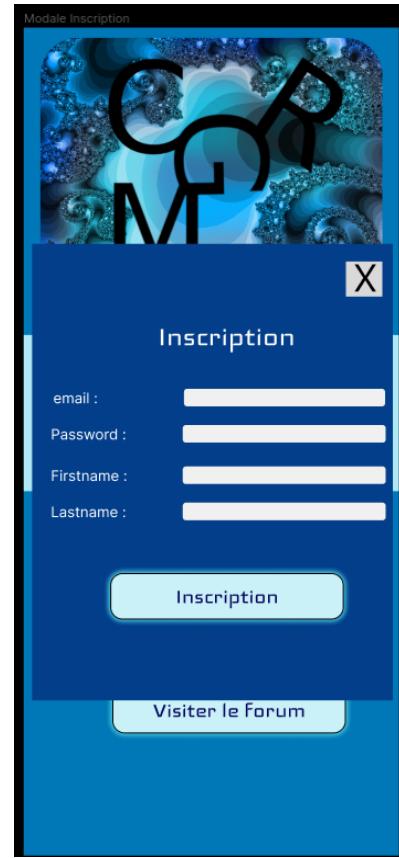
La maquette a été réalisée sur Figma.



La landing page



L'inscription



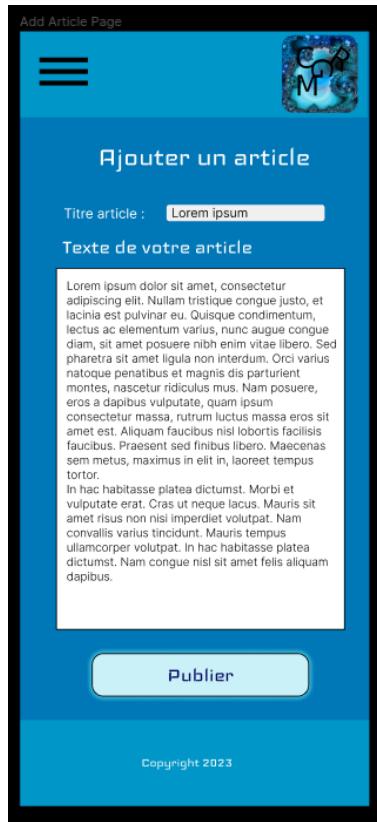
Liste des articles



Article



Ajouter un article

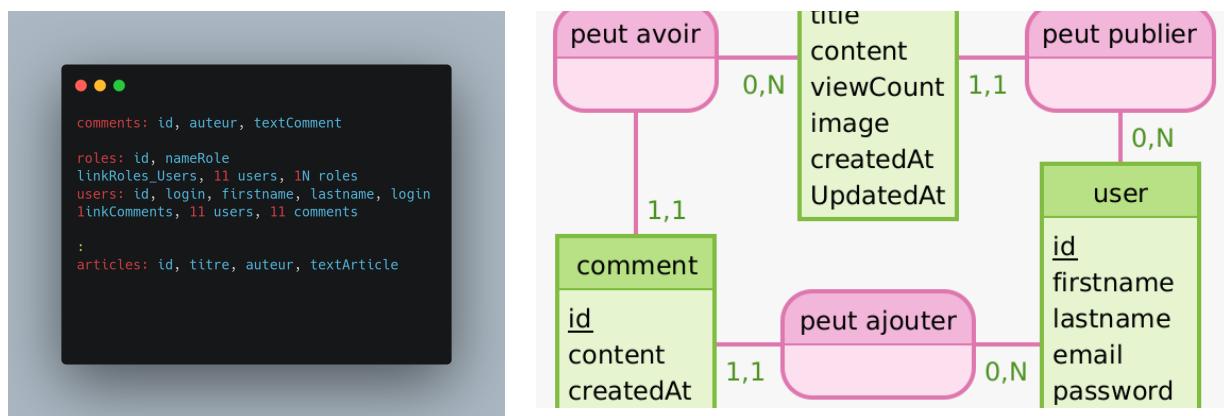


Présentation base de données

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.

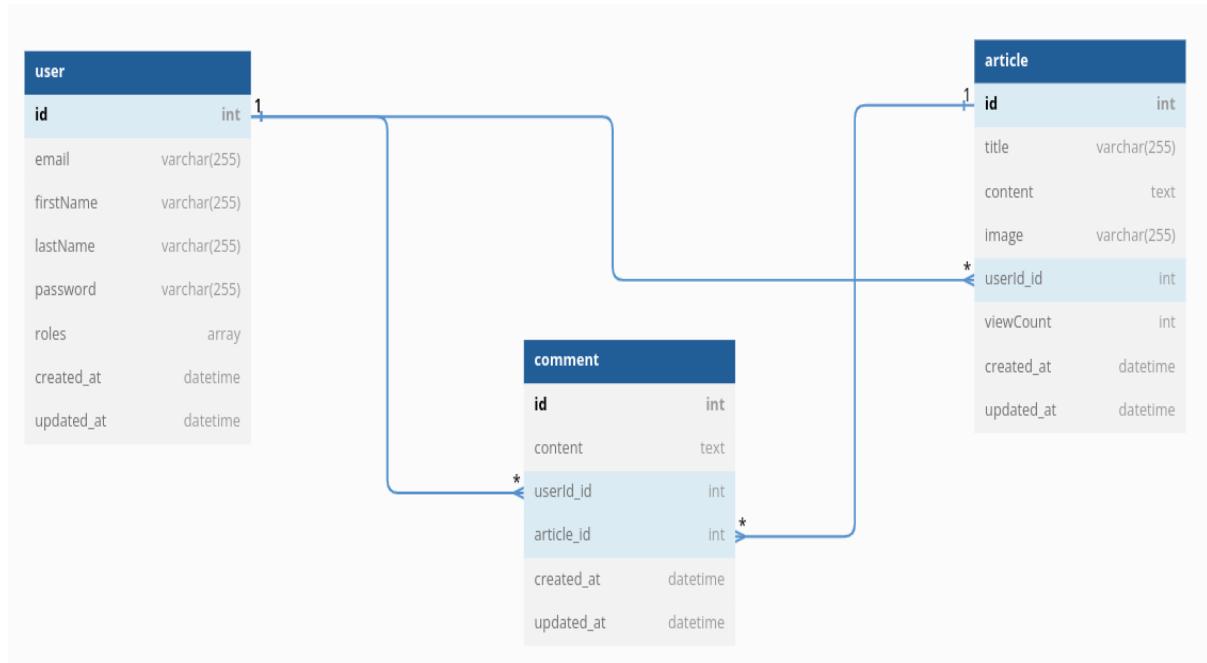
Le MCD

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 Modèle Logique de données a été créé en se basant sur le MCD.

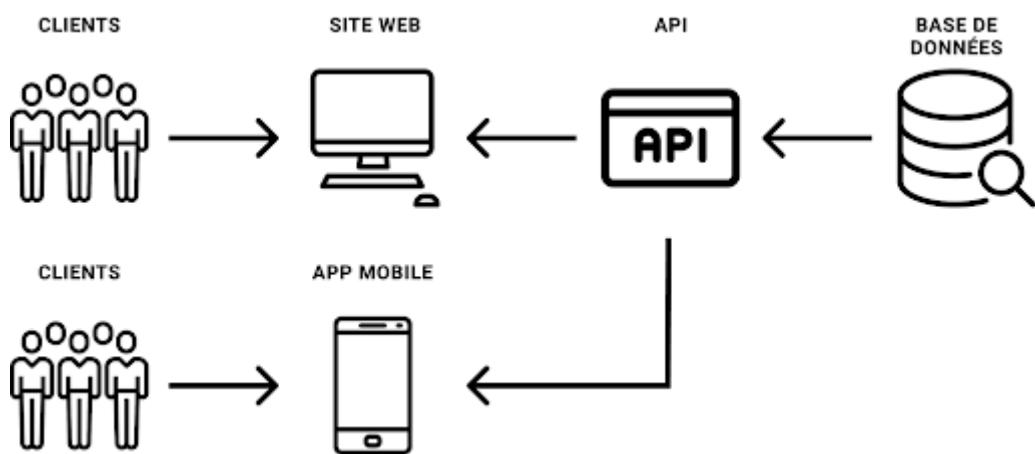


Description de l'API REST

Dans cette section, nous allons 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.

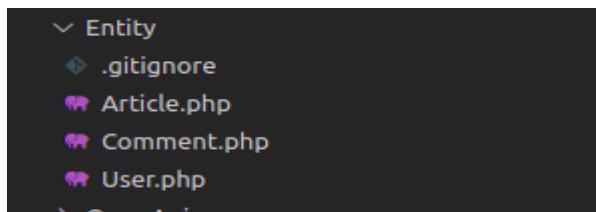
Dans ce projet, nous avons utilisé API Platform, un framework web utilisé pour générer des API REST, se basant sur le patron de conception MVC, tout en offrant des fonctionnalités avancées telles que la documentation automatique de l'API, la validation des données, la sécurité, etc.

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.

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.	▼

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

```
#[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

```
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

```
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 utilisé 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.

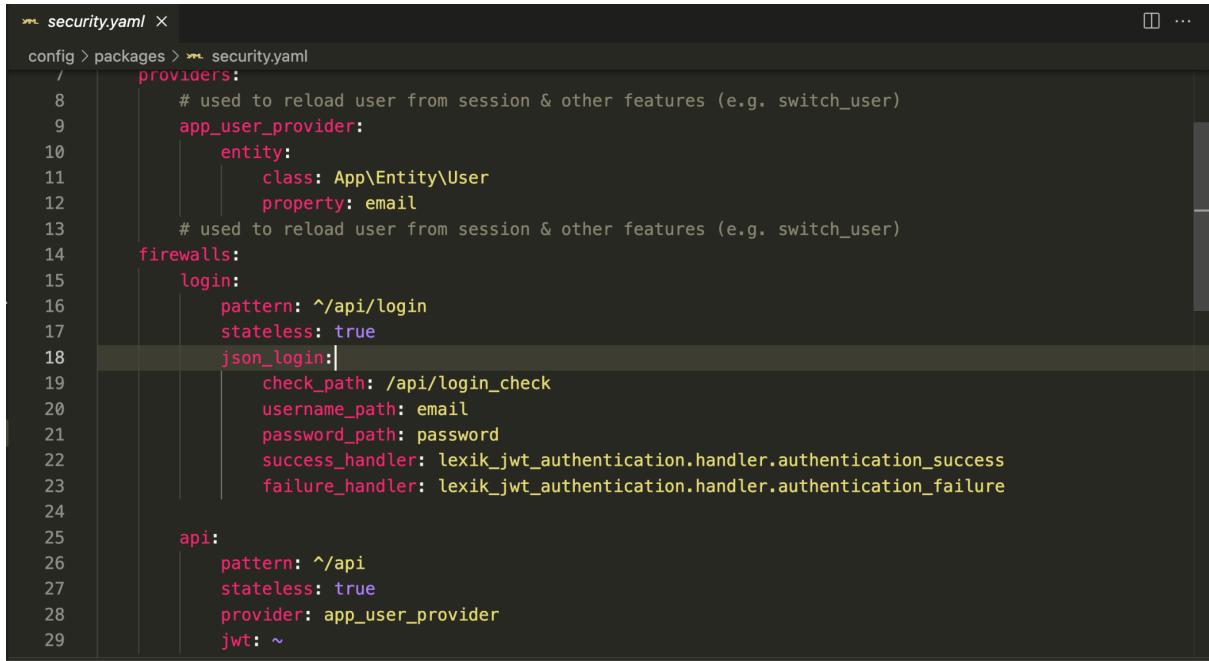
Encoded	Decoded
PASTE A TOKEN HERE	EDIT THE PAYLOAD AND SECRET
<pre>eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpvag4gRG91IiwiaWF0IjoxNTE2MjM5MDIyfQ.Sf1KxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c</pre>	<p>HEADER: ALGORITHM & TOKEN TYPE</p> <pre>{ "alg": "HS256", "typ": "JWT" }</pre> <p>PAYOUT: DATA</p> <pre>"sub": "1234567890", "name": "John Doe", "iat": 1516239022 }</pre> <p>VERIFY SIGNATURE</p> <pre>HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), your-256-bit-secret) □ secret base64 encoded</pre>

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 :

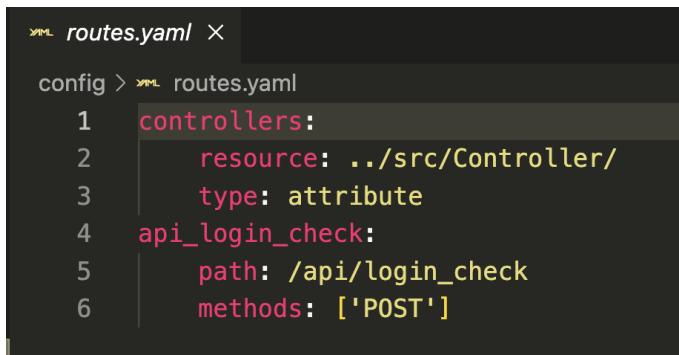
composer require lexik/jwt-authentication-bundle

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



```
yaml security.yaml ×
config > packages > security.yaml
/
  providers:
    app_user_provider:
      entity:
        class: App\Entity\User
        property: email
    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 :



```
yaml routes.yaml ×
config > routes.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 :

```
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
```

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'échapper automatiquement et paramétriser les requêtes SQL, ce qui réduit considérablement les risques d'injection 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')

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->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);
}
}
```

Les 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.

```
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
    {
        // Implementation
    }
}
```

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.

Back-end

- 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.

```
AppliMobileForum > 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.contButton}>
6             <Pressable style={props.button} onPress={props.onPress}>
7                 <Text style={props.txtButton}>{props.text}</Text>
8             </Pressable>
9         </View>
10    )
11 }
12 }
```

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 Inscription est un composant ButtonComponent.



- Les screens

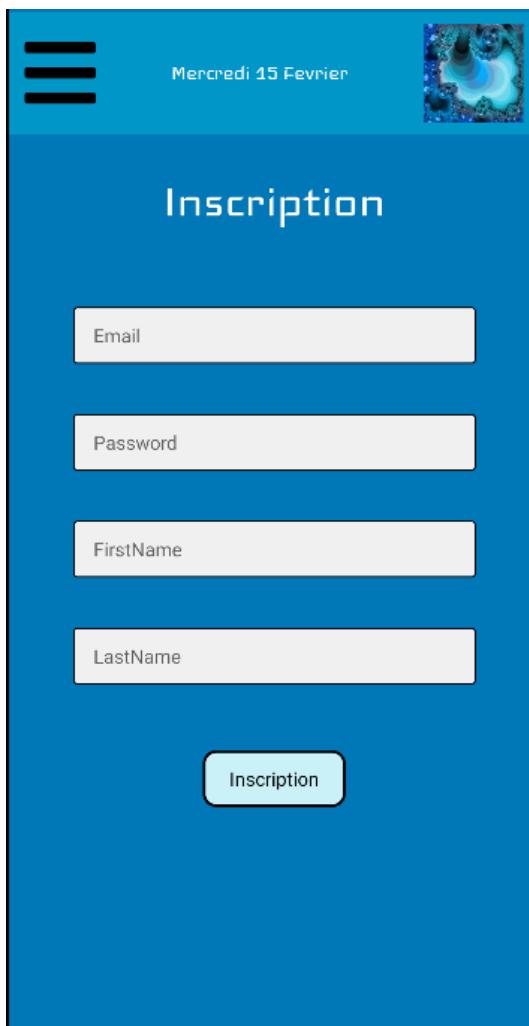
Les screens, également appelés vues, sont des interfaces utilisateur qui s'affichent aux utilisateurs dans le cadre de leur expérience avec une application. Ils représentent les différentes pages, composants ou états de l'application qui fournissent des informations, des fonctionnalités et des interactions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

```
export default function IncriptionScreen({ navigation }) {
  const emailRegex = /^[\S+@\S+\.\S+$/;
  const [user, setUser] = useState('')

  const handleSubmit = () => {
    if (emailRegex.test(user.email)) {
      postUser(user.email, user.firstName, user.lastName, user.password, (res => {
        if (res.status != 201) {
          Alert.alert(`Impossible de crée l'utilisateur`, `${res.data.violations[0].message}`, [{ style: 'cancel' }])
        } else {
          login(user.email, user.password, (res => {
            if (res.status != 200) {
              Alert.alert(`Connexion automatique échoué, veuillez vous connecter`, `${res.data.message}`, [{ style: 'cancel' }])
            } else {
              setUser('')
              Alert.alert('Votre inscription a bien été prise en compte', 'Bienvenue sur le Forum !')
              navigation.navigate('Articles', { refresh: true })
            }
          }))
        }
      })
    }
  }
}
```

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'email, 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 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 getToken();
  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, le premier consiste à définir la méthode (get/post/patch/delete), le second la route, le

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.

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.

```
const Drawer = createDrawerNavigator();

const DrawerNavigator = () => {

  return (
    <Drawer.Navigator
      initialRouteName="Home"
      screenOptions={{ headerShown: false, refresh: true }}
      drawerContent={({ property, name, ...props }) =>
        Route name of this screen. <Drawer.Screen name={name} component={IndexArticleScreen} />
        <Drawer.Screen name="Articles" component={ReadArticle} />
        <Drawer.Screen name="AddArticle" component={AddArticleScreen} />
        <Drawer.Screen name="Profil" component={ProfilScreen} />
        <Drawer.Screen name="Connexion" component={ConnexionScreen} />
        <Drawer.Screen name="Inscription" component={InscriptionScreen} />
        <Drawer.Screen name="AdminScreen" component={AdminScreen} />
        <Drawer.Screen name="Users" component={IndexUsersScreen} />
        <Drawer.Screen name="User" component={UserProfileEditScreen} />
        <Drawer.Screen name="ArticlesAdmin" component={IndexArticlesScreen} />
        <Drawer.Screen name="ArticleAdmin" component={ArticleEditScreen} />
        <Drawer.Screen name="About" component={About} />
      }
    </Drawer.Navigator>
  )
}
```

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.

```

return (
    <View style={{ flex: 1 }}>
      <DrawerContentScrollView {...props}>
        <View style={styles.header}>
          <View>
            {isLog ?
              (<>
                <Text style={styles.headerTxt}>{user.firstname} {user.lastname}</Text>
                <Text style={styles.headerTxt}>{user.email}</Text>
              </>
            ) :
              (<Text style={styles.headerTxt}>Vous n'êtes pas connecté</Text>)
            }
          </View>
          <Image style={styles.imgDrawer} source={require '../../../../../assets/Pictures/320px-Emblème_de_l'Ordre_Jedi..png'} />
        </View>

        <View style={styles.linkText}>
          <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 })} />
            <DrawerItem label={'Connexion'} onPress={() => props.navigation.navigate('Connexion')} />
            <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')} />
            }
          }
        </View>
      </DrawerContentScrollView >
      <TouchableOpacity style={styles.footer} onPress={() => Alert.alert("Vous êtes sur le point de vous déconnecter", "Voulez-vous vraiment vous déconnecter ?"))}>
        ...
      </TouchableOpacity>
    </View>
  )
)

```

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.

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.
- Créer un système de favoris

Conclusion

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.