

TITRE RNCP NIV VI

# CONCEPTEUR.ICE DÉVELOPPEUR.SE D'APPLICATIONS

---

## DOSSIER DE PROJET

## PROFESSIONNEL

Application mobile



# Buddy Up

*Partage ton expérience !*

Plateforme de mise en relation entre voyageurs et locaux

**Equipe** : Trois petites chattes (Alicia CORDIAL, Huong-Mây NGUYEN-PHUOC, Aïcha OUATTARA)

**Ecole** : La Plateforme\_

juillet 2022

# TABLE DES MATIÈRES

<b>1/ Introduction</b>	<b>5</b>
1 - Présentation de l'équipe	5
2 - Présentation personnelle	5
3 - Présentation du projet en anglais	6
4 - Compétences couvertes par le projet	6
<b>2/ Contexte du projet</b>	<b>9</b>
1 - Objectifs de l'application	9
2 - Glossaire de l'application	9
a - Buddy up	9
b - Local Buddy	10
c - Nomad Buddy	10
d - Expérience	10
e - Bucket List	10
f - Experience Now	10
g - Connections	10
3 - Utilisateurs cible	10
4 - Périmètre du projet	11
5 - Choix technologiques	11
a- Langages de programmation	11
b - Frameworks	11
c - Logiciels et outils utilisés	12
<b>3/ Graphisme</b>	<b>13</b>
1 - Sitemap	13
2 - Wireframe	14
3 - Identité visuelle	15
a - Moodboard	15
b - Logo	15
c - Charte graphique	17
4 - Maquette	17
<b>4/ Spécifications fonctionnelles et techniques</b>	<b>20</b>
1 - Spécifications fonctionnelles	20
a - Périmètre fonctionnel	20
b - Fonctionnalités détaillées	20
2 - Spécifications techniques	23

a - Contraintes technologiques	23
b - Services tiers	23
c - Accessibilité	24
i - Compatibilité	24
ii - Adaptabilité et responsivité	24
2 - Droits des utilisateurs	24
3 - Sécurité de l'application	25
<b>5/ Gestion de projet</b>	<b>26</b>
1 - Méthode de gestion de projet utilisée	26
2 - Mode de communication / outils utilisés	26
a - Diagramme de Gantt	27
b - Trello	27
<b>6/ Développement du back-end de l'application</b>	<b>29</b>
1 - Conception et mise en place de la base de données	29
a - Modèle Conceptuel de Données (MCD)	29
b - Modèle Logique de données (MLD)	31
c - Modèle Physique de données (MPD)	32
2 - Développement de l'API	32
a - API Platform	33
b - Mise en place de l'API	33
c - Test d'une route	35
d - Statuts de requête HTTP	37
d - Méthodes HTTP utilisées dans ce projet	37
3 - Middleware	38
4 - Sécurité et JWT	39
a - Hachage du mot de passe	39
b - Authentification par JWT	40
c - Test du login	41
<b>7/ Développement du front-end de l'application</b>	<b>43</b>
1 - React Native	43
2 - Arborescence du repository	43
a - A la racine	43
b - Le dossier src	44
3 - Ecrans et composants	44
a - Exemple de composant	45
b - Exemple d'utilisation dans un écran	45
c - Rendus UX	46

4 - Navigation	47
a - Composant Nav.js	47
b - Ecran ProtectedScreen.js	48
5 - Jeu d'essai	49
a - Screenshots de l'interface	50
b - Code	50
i - Clic sur une ville	50
ii - Préremplissage du filtre sur l'écran Search	50
iii - Envoi de la requête	51
iv - Traitement de la requête	51
v - Aperçu sur le swagger	52
<b>8/ Tests</b>	<b>52</b>
Tests route	53
i - Thunder Client	53
ii - Swagger UI	54
Tests unitaires	54
i - Jest	54
<b>9/ Recherche en anglais</b>	<b>55</b>
1 - Présentation du problème	55
2 - Recherche	55
3 - Solution	55

# 1/ Introduction

## 1 - Présentation de l'équipe

Nous sommes Aïcha, Alicia et Mây, trois élèves de l'école **La Plateforme** en deuxième année de cursus Coding School, qui se déroule en alternance.

Afin de valider le titre RNCP de concepteur.ice développeur.se d'applications, nous avons réalisé une **application mobile complète** (front-end et back-end) qui intègre les recommandations de sécurité.

## 2 - Présentation personnelle

Je m'appelle **Huong-Mây Nguyen-Phuoc**. J'ai 27 ans. Après un master 1 en Philosophie et quelques années à alterner jobs alimentaires et voyages, j'ai intégré la formation de développement web de la Plateforme. J'ai obtenu mon titre de développeur web et web mobile l'année dernière. Aujourd'hui, je suis en Coding School 2 et je prépare le titre de concepteur.ice développeur.ice d'application.

## 3 - Présentation du projet

**Buddy Up** est une application mobile destinée aux **voyageurs** en France. Elle permet aux utilisateurs de rentrer en **contact** et de **découvrir** une **ville** au travers des yeux d'une personne locale.

L'utilisateur local peut ajouter des **expériences** et proposer différents types d'activités auxquelles il participe au quotidien : promenades, jeux, boire un verre... Cela peut être quelque chose qu'il fait souvent ou régulièrement, ou un événement ponctuel.

Comme tout réseau social, l'utilisateur voyageur peut **parcourir** ces expériences, ajouter un "**j'aime**" à celles qui **l'intéressent** et **contacter** l'utilisateur local s'il souhaite y participer.

Le nom de notre application est *Buddy up*, qui signifie "**devenir ami** avec quelqu'un". Cette expression nous semblait appropriée : elle est informelle, facile à comprendre et elle souligne l'aspect social de notre application.

En effet, l'objectif de Buddy up est d'encourager les amitiés et les **échanges** entre personnes locales et voyageurs. C'est **gratuit** et basé sur un principe de **communauté**. Ce qu'on tire de l'expérience est de nouveaux liens, de bons souvenirs et un moment unique hors des sentiers battus.

#### 4 - Compétences couvertes par le projet

Le projet couvre **une partie des compétences** définies par le REAC (Référentiel Emploi Activités Compétences).

Pour l'activité type 1, "**concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité**" :

- Maquetter une application
- Développer des composants d'accès aux données

Pour l'activité type 2, "**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

Pour l'activité type 3, "**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

## 2/ Contexte du projet

### 1 - Objectifs de l'application

Notre application a pour but de mettre en **relation** des **utilisateurs de passage** avec des **utilisateurs locaux qui** proposent des **expériences dans leurs villes**. Nous souhaitons créer une **communauté soudée et altruiste**.

Il ne s'agit pas de consommer un produit touristique mais de découvrir de nouveaux lieux ou activités avec quelqu'un qui habite là.

Les expériences ne sont pas nécessairement grandioses, mais elles sont toujours un **bon moment partagé**. La dimension humaine est au cœur du fonctionnement de l'application.

Un utilisateur propose une expérience car il souhaite **la faire vivre** à un voyageur ; un utilisateur choisit de participer à une expérience car elle l'intrigue, ou par affinité avec son organisateur.

Buddy Up vous permet de sortir des sentiers battus et de **découvrir une ville** au travers des yeux d'un de ses habitants. Une fois de retour chez vous, peut-être voudriez-vous à votre tour **accueillir des voyageurs** et donner de votre temps à d'autres membres de la communauté.

### 2 - Glossaire de l'application

Buddy up emploie un vocabulaire spécifique pour ses concepts les plus importants.

#### a - Buddy up

Expression américaine signifiant "devenir amis" avec quelqu'un. Tous les utilisateurs sont des buddies.

### b - Local Buddy

Utilisateur qui propose une expérience dans sa ville.

### c - Nomad Buddy

Utilisateur de passage dans une ville.

### d - Experience

Activité proposée par un local buddy. Il peut s'agir de faire découvrir un coin qu'on apprécie, faire une sortie sport, partager un repas traditionnel.

### e - Bucket List

Liste des expériences qu'on a ajoutées en favoris et qu'on aimerait réaliser un jour.

### f - Experience Now

Liste des expériences qu'on souhaite réaliser très prochainement et dont on a contacté le local buddy.

### g - Connections

Demandes d'experience now reçues par un local buddy. Elles sont accompagnées d'un message.

## 3 - Utilisateurs cible

L'application est destinée aux personnes qui cherchent à vivre / faire vivre **une expérience personnelle**, que ce soit dans leur propre ville que dans une ville où elles sont de passage. Elles apprécient autant apprendre des autres que de partager ce qu'elles aiment et connaissent autour d'elles.

Ce sont des personnes habituées des services et modes de vie **alternatifs et gratuits** : couchsurfing, autostop, wwoofing, freeganisme.

L'application est entièrement **gratuite** et les buddies ne sont pas rémunérés pour l'activité qu'ils animent.

## 4 - Périmètre du projet

L'application est un **projet d'école** développé tout **au long de l'année**, en parallèle de notre alternance en entreprise. Plusieurs technologies ont été recommandées par les encadrants pédagogiques.

Buddy up nous a permis d'appréhender le développement d'une application mobile dans son **intégralité**, de renforcer nos compétences techniques et graphiques et surtout de **gérer un projet** avec des méthodes professionnelles sur une longue période.

## 5 - Choix technologiques

### a- Langages de programmation

<b>Front End</b>	Javascript
<b>Back End</b>	PHP

### b - Frameworks

<b>Front End</b>	React Native
<b>Back End</b>	Symfony
<b>API</b>	Api Platform

### c - Logiciels et outils utilisés

Développement	
<b>IDE</b>	Visual Studio Code
<b>Serveur local</b>	Wamp (windows), Mamp (mac)
<b>Base de données</b>	Mysql
<b>Gestion de versions</b>	Git
<b>Hébergement et partage du repo</b>	GitHub
<b>Emulation</b>	Android Studio
<b>Emulation sur téléphone</b>	Expo Go
<b>Test requêtes API</b>	Thunder Client

Graphisme	
<b>Wireframe, sitemap, maquette</b>	Figma
<b>Modélisation BDD</b>	LucidChart
<b>Logo</b>	Medibang Paint

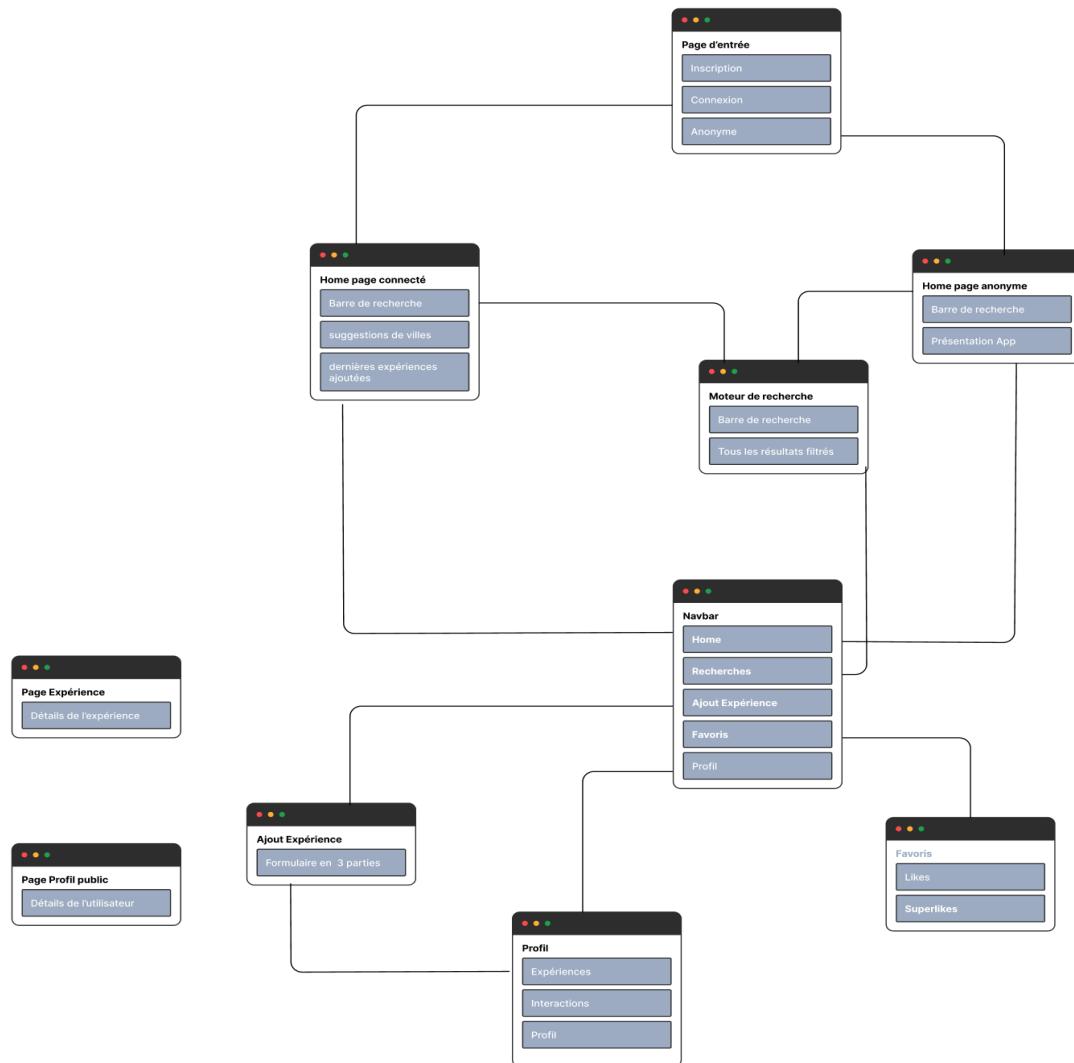
Gestion de projet	
<b>Organisation des tâches</b>	Trello
<b>Communication</b>	Google Chat, Discord
<b>Partage de documents</b>	Google Drive

### 3/ Graphisme

## 1 - Sitemap

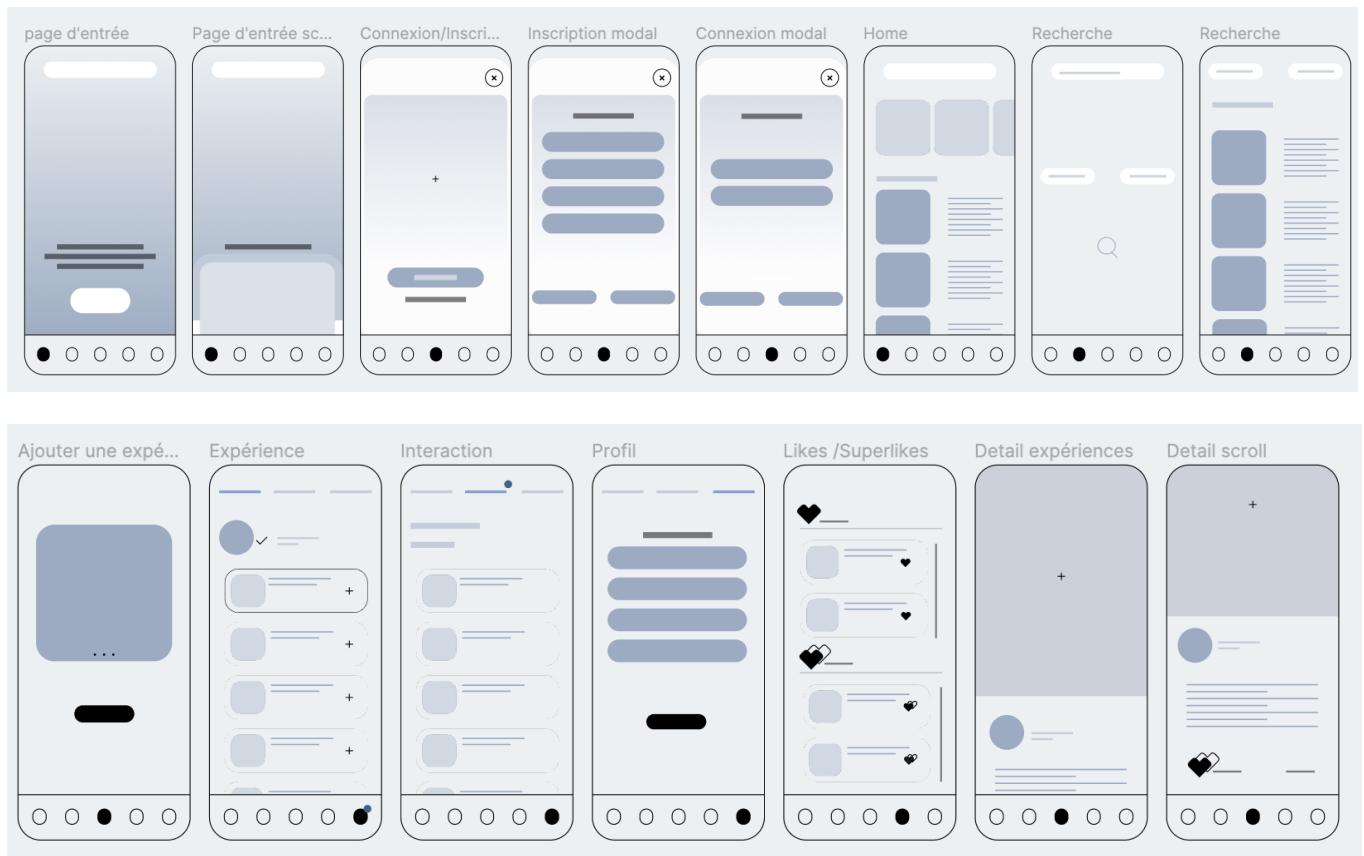
Avant de passer à la réalisation d'un wireframe et d'une maquette, nous avons d'abord voulu avoir une **vue d'ensemble** de notre application. C'est pour cette raison que nous avons commencé par créer un **sitemap**.

Un sitemap permet de **visualiser** l'architecture et les **liens** entre les différents écrans. Evidemment, il a évolué au fur et à mesure du projet.



## 2 - Wireframe

Pour chaque écran de notre application, nous avons ensuite conçu un **wireframe**, aussi appelé maquette fonctionnelle. Cette maquette nous a permis de définir les **zones** et composants qui constituent les **interfaces** utilisateurs.



### 3 - Identité visuelle

Nous avons ensuite discuté du thème, de la palette de couleurs et de l'atmosphère générale de notre application.

Buddy up doit être **coloré, sympathique et illustré**.

#### a - Moodboard

Un ensemble d'images, de polices et de couleurs dont nous nous sommes inspirées.



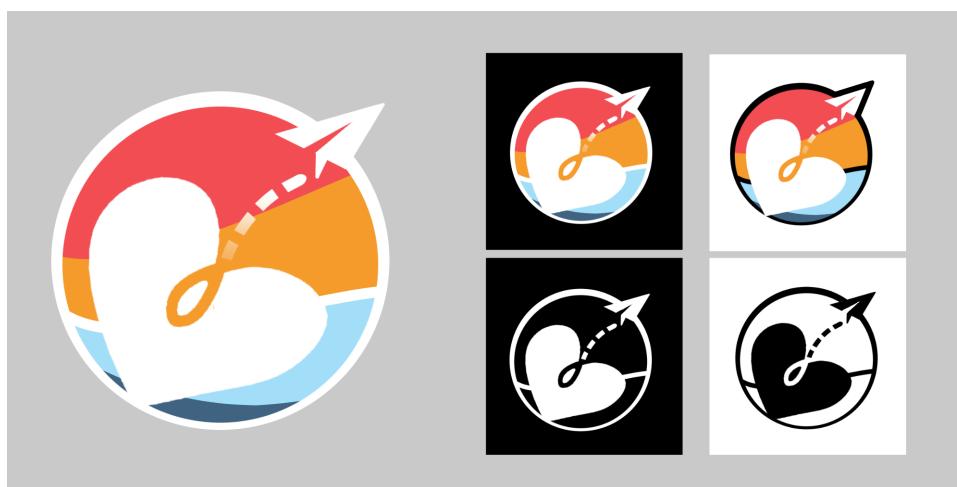
**Poppins**

**Raleway**

**Open Sans**

### b - Logo

Le fond évoque la mer et un coucher de soleil, ce qui fait penser aux vacances. La forme du cœur rappelle le "B" de Buddy up. L'avion en papier représente la communication et le voyage.



## c - Charte graphique

La charte graphique reprend le logo, les polices et la palette de couleurs.



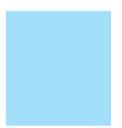
F14D53



F49B2C



FCC5C4



A3DEF8



416382



28130D

Semi-bold 600

**Poppins**  
**AaBbBCcDdEeFf**

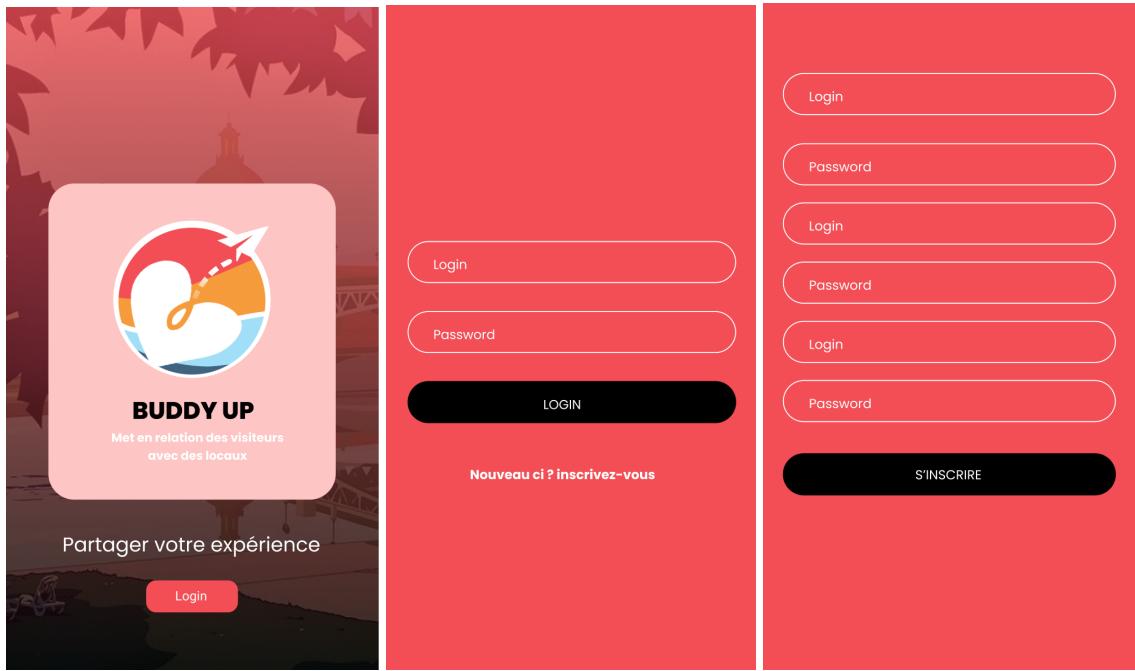
Medium 500

**Raleway**  
**AaBbBCcDdEeFf**

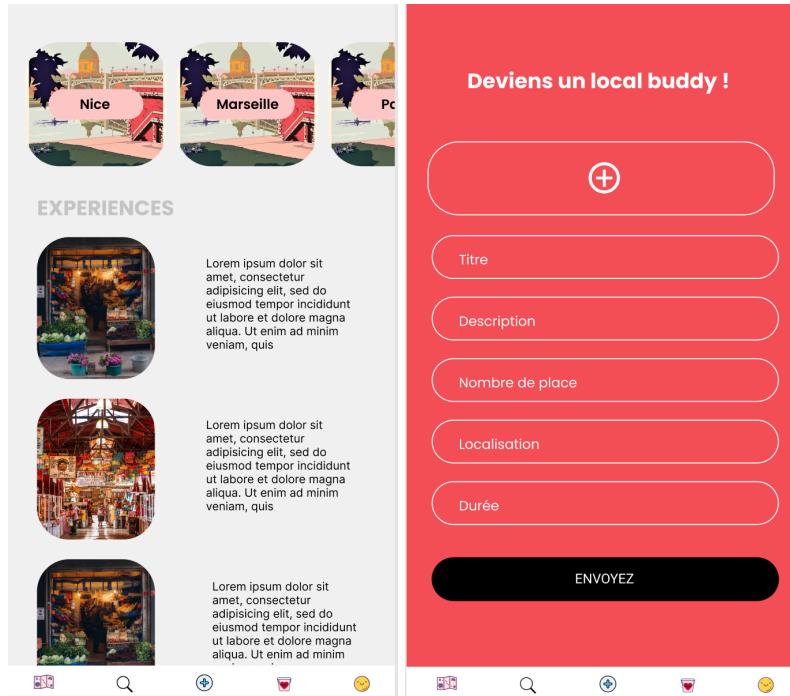
Regular 400

**Open Sans**  
**AaBbCcDdEeFf**

## 4 - Maquette



Page d'accueil - Login - Inscription



Fil d'actualités - Ajout d'une expérience

**Chay**

Experiences Avis Profil

**Bonjour je m'appelle Chay !**

Membre depuis juin 2022

[Modifier le profil](#)

0 Avis

**A propos**

Pas encore de biographie

**Téléphone**  
0678325681

**Mot de passe**  
.....

[Deconnexion](#)

*Profil public - profil privé*

## 4/ Spécifications fonctionnelles et techniques

### 1 - Spécifications fonctionnelles

#### a - Périmètre fonctionnel

L'**objectif** à long terme de cette application est de créer une **communauté de membres** de confiance aux profils différents, impliqués dans le projet et donc forces de proposition.

De nouvelles fonctionnalités seront ainsi développées par la suite pour faciliter l'échange entre les membres et la recherche d'expériences : chat, calendrier, filtres de recherche, catégories d'expériences.

La **fonctionnalité principale** sur laquelle est basée l'application reste **l'ajout et la gestion d'expériences locales**, proposées par les utilisateurs. Le contenu du site est donc en grande partie généré par les utilisateurs eux-mêmes.

Les différents utilisateurs de l'application ont des droits, rôles et besoins **fluides**. Les **fonctionnalités** à implémenter **découlent des actions** que chaque acteur peut et souhaite effectuer.

#### b - Fonctionnalités détaillées

En **bleu clair**, les fonctionnalités qu'il reste à développer.

EXPÉRIENCES - FIL / RECHERCHE	
Fonctionnalités	Détails
Voir les dernières expériences	<ul style="list-style-type: none"><li>• afficher les dernières expériences</li><li>• afficher une sélection d'expériences (par ville)</li></ul>

Rechercher une expérience ou des expériences	<ul style="list-style-type: none"> <li>moteur de recherche (cherche des mots clefs dans le titre ou la description)</li> <li>filtre de localisation</li> <li>affichage d'une liste de résultats</li> </ul>
Consulter le profil public d'un membre	<ul style="list-style-type: none"> <li>pseudonyme</li> <li>avatar</li> <li>voir les avis et la note globale</li> <li>toutes les expériences proposées</li> <li>date d'inscription</li> <li>localisation</li> <li>vérifié ou non</li> </ul>
Consulter la fiche détaillée d'une expérience	<ul style="list-style-type: none"> <li>titre</li> <li>description</li> <li>photo</li> <li>nombre de places</li> <li>localisation</li> <li>date de création</li> <li>date de modification</li> <li>durée</li> <li>accès au profil du membre</li> <li>nombre de nomad buddies intéressés</li> <li>bouton "add to bucket list"</li> <li>bouton "experience now"</li> </ul>
Ajouter une expérience à sa bucket list	<ul style="list-style-type: none"> <li>bouton qui permet de mettre en favori une expérience</li> </ul>
Faire une demande d'experience now	<ul style="list-style-type: none"> <li>le bouton permet d'envoyer une notification à la personne qui a déposé l'expérience et d'entrer en contact avec elle</li> <li>possibilité d'ajouter un message</li> <li>nombre d'experience now par jour limités</li> <li>avertissement automatique des informations personnelles qui sont envoyées à l'autre utilisateur</li> </ul>

PROFIL	
Fonctionnalités	Détails

S'inscrire	<ul style="list-style-type: none"> <li>● email non utilisé</li> <li>● nom d'utilisateur non utilisé</li> <li>● nom et prénom</li> <li>● mot de passe sécurisé</li> <li>● code postal</li> <li>● téléphone</li> </ul>
Se connecter	<ul style="list-style-type: none"> <li>● login ou email existant</li> <li>● mot de passe correspondant</li> <li>● lien vers son profil public</li> </ul>
Modifier son profil	<ul style="list-style-type: none"> <li>● ajouter et modifier son avatar</li> <li>● ajouter et modifier sa présentation personnelle</li> <li>● update ses infos personnelles</li> </ul>
Vérifier son profil	<ul style="list-style-type: none"> <li>● A l'aide d'un SMS ou d'une connexion avec un compte google</li> </ul>
Voir ses notifications	<ul style="list-style-type: none"> <li>● pastille de notification sur l'icône du profil</li> </ul>
Voir ses connections / toutes les demandes d'experience now que l'on a	<ul style="list-style-type: none"> <li>● classés par dates</li> <li>● lien vers le profil public</li> <li>● contact de l'utilisateur (email ou téléphone)</li> <li>● expérience concernée</li> <li>● date de la demande de contact</li> <li>● message personnalisé, si existant</li> <li>● accepter ou refuser l'interaction</li> </ul>

GESTION DES EXPÉRIENCES	
Fonctionnalités	Détails
Ajouter une expérience	<ul style="list-style-type: none"> <li>● Ajouter une photo</li> <li>● Remplir tous les champs</li> <li>● 5 expériences maximum</li> </ul>
Gérer les expériences existantes	<ul style="list-style-type: none"> <li>● modifier les détails</li> <li>● lien vers la fiche publique</li> <li>● marquer comme disponible ou non</li> <li>● supprimer une expérience</li> </ul>

## GESTION DES FAVORIS

Fonctionnalités	Détails
Voir sa bucket list	<ul style="list-style-type: none"> <li>• lien vers les fiches d'expérience</li> <li>• date de la mise en favori</li> <li>• supprimer de sa bucket list</li> </ul>
Voir ses connections	<ul style="list-style-type: none"> <li>• lien vers les fiches d'expérience</li> <li>• date de la demande</li> <li>• message envoyé, si existant</li> <li>• ajouter un avis</li> </ul>
Ajouter un avis sur un autre utilisateur	<ul style="list-style-type: none"> <li>• écrire un avis sur un membre avec qui on a fait une expérience</li> <li>• donner une note</li> <li>• disponible dès que l'interlocuteur a accepté l'interaction</li> </ul>

## 2 - Spécifications techniques

### a - Contraintes technologiques

Solutions retenues	
Besoins	Outils / Technologies
interaction avec les données	API custom avec API Platform
Application cross-platform	React Native
Traitemet des données	Framework Symphony

### b - Services tiers

Solutions utilisées	
Besoins	Services
Vérification de l'identité	SMS / compte google
Stockage des données	Base de données MySQL

### c - Accessibilité

#### i - Compatibilité

- Android
- iOS

#### ii - Adaptabilité et responsivité

- Smartphone
- Tablette

## 2 - Droits des utilisateurs

Nous avons défini **trois rôles d'utilisateurs** avec des droits différents.

Pour cette première version, nous n'avons pas encore mis en place la vérification. Ainsi, il n'y a qu'un rôle : **utilisateur connecté**, qui a accès à toutes les fonctionnalités.

Droits selon le statut	
Statut	Accès
Non identifié	<ul style="list-style-type: none"><li>• consulter la liste des expériences</li><li>• rechercher des expériences</li></ul>
Connecté	<ul style="list-style-type: none"><li>• consulter la fiche détaillée d'une expérience</li><li>• consulter le profil public d'un membre</li><li>• espace personnel</li><li>• ajouter une expérience à sa bucket list</li></ul>
Connecté et vérifié	<ul style="list-style-type: none"><li>• cliquer sur experience now</li><li>• ajouter une expérience</li></ul>

### 3 - Sécurité de l'application

Sécurité	
Objet	Méthode
Mot de passe sécurisé	<ul style="list-style-type: none"><li>• Haché</li><li>• <a href="#">Contrainte de robustesse du mot de passe</a></li></ul>
Vérification du profil	<ul style="list-style-type: none"><li>• <a href="#">Vérification par un service externe (google ou sms) pour confirmer le compte</a></li></ul>
Diffusion des informations personnelles	<ul style="list-style-type: none"><li>• Le contact (téléphone et email) des membres n'est jamais publié publiquement dans l'application.</li><li>• Le nomad buddy est averti des informations transmises au local buddy lorsqu'il lui envoie une demande d'experience now</li></ul>
Limitation du nombre d'expériences publiées par un local buddy	<ul style="list-style-type: none"><li>• 5 annonces maximum</li><li>• système d'archivage d'une expérience</li></ul>
Limitation du nombre d'experience now par jour	<ul style="list-style-type: none"><li>• 5 demandes d'experience now par jour</li></ul>
Absence de réponse à la demande d'experience now	<ul style="list-style-type: none"><li>• Publication d'un avis automatique sur le local buddy après 3 jours</li></ul>

## 5/ Gestion de projet

### 1 - Méthode de gestion de projet utilisée

Nous avons dû réaliser un projet de fin d'année sur une période de dix mois. Pour avoir une organisation optimale du temps de travail et une **répartition** des tâches équitables, nous avons choisi de respecter au maximum les **principes agiles**.

Cette méthodologie de travail nous a permis de fragmenter le projet en plusieurs phases et de nous fixer des **objectifs à court terme**. A l'aide de **sprints** d'un mois, nous avons pu revoir et réajuster les objectifs au fur à mesure, si besoin, et résoudre les bogues et difficultés rencontrés.

Agile met un point d'honneur à renforcer les relations et la **communication** entre les membres de l'équipe projet. C'est pour cette raison que la **flexibilité** et la **souplesse** dans l'organisation sont deux piliers fondamentaux de ces méthodes.

Nous nous sommes appuyées sur la méthode **scrum**, qui repose entre autres sur le principe de **transparence**.

Certaines **informations** doivent être accessibles par tous, comme la tâche en cours de chacun, son état d'avancement et l'objectif actuel de l'équipe. Il est important que ces informations soient visibles en permanence.

Le **tableau scrum** nous permet d'organiser notre **backlog** (liste de tâches à faire, cahier des charges), les tâches à effectuer durant le sprint actuel et leur état d'avancement.

Il nous permet d'identifier facilement les bogues tout au long du projet.

Chaque sprint se termine par une **réunion** dont le but est de partager nos retours d'expérience, de faire le point sur les tâches et le backlog, et de discuter des améliorations possibles lors du prochain sprint.

### 2 - Mode de communication / outils utilisés

Pour gérer notre projet et nous organiser, nous avons principalement utilisé les deux outils suivants : le diagramme de Gantt et Trello.

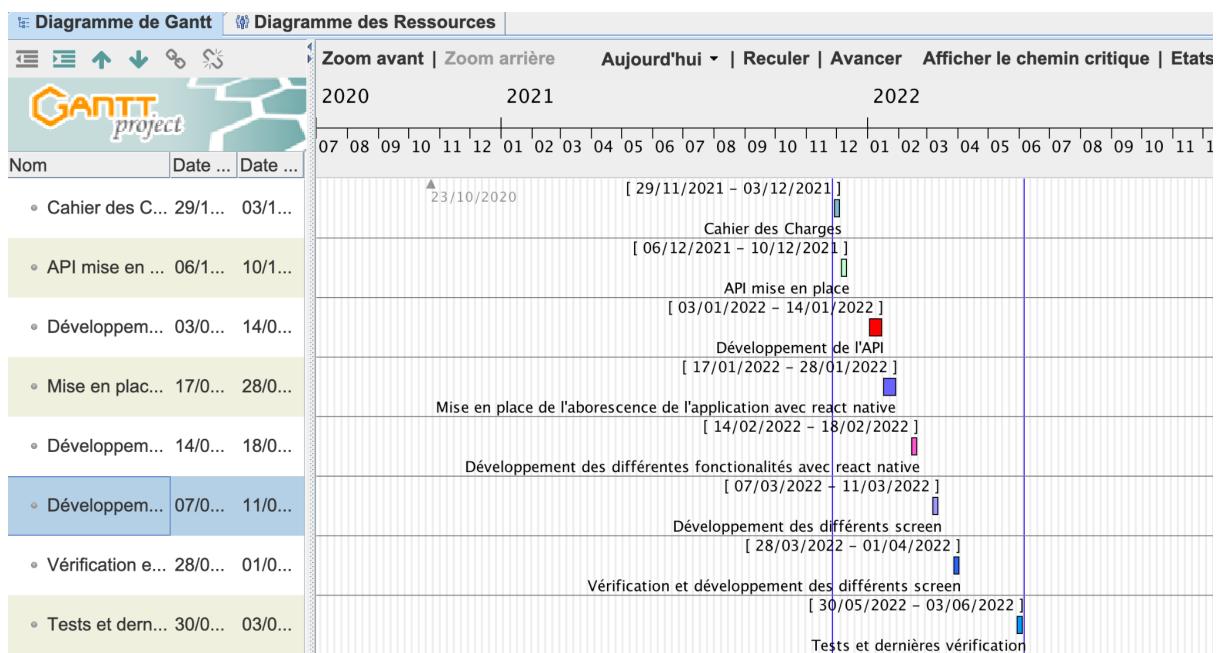
Nous avons aussi créé un salon sur **Google Chat**, qui nous permet de **communiquer** rapidement avec toute l'équipe, même lorsque nous sommes en entreprise.

Lorsque nous sommes à l'école, nous réalisons souvent nos tâches en paire, et toujours dans le même espace de travail. Tous les matins, nous faisons un point rapide sur le programme de la journée.

Nous utilisons également **Google Drive** pour **partager** et **archiver** différents documents qui nous sont utiles tout au long de ce projet.

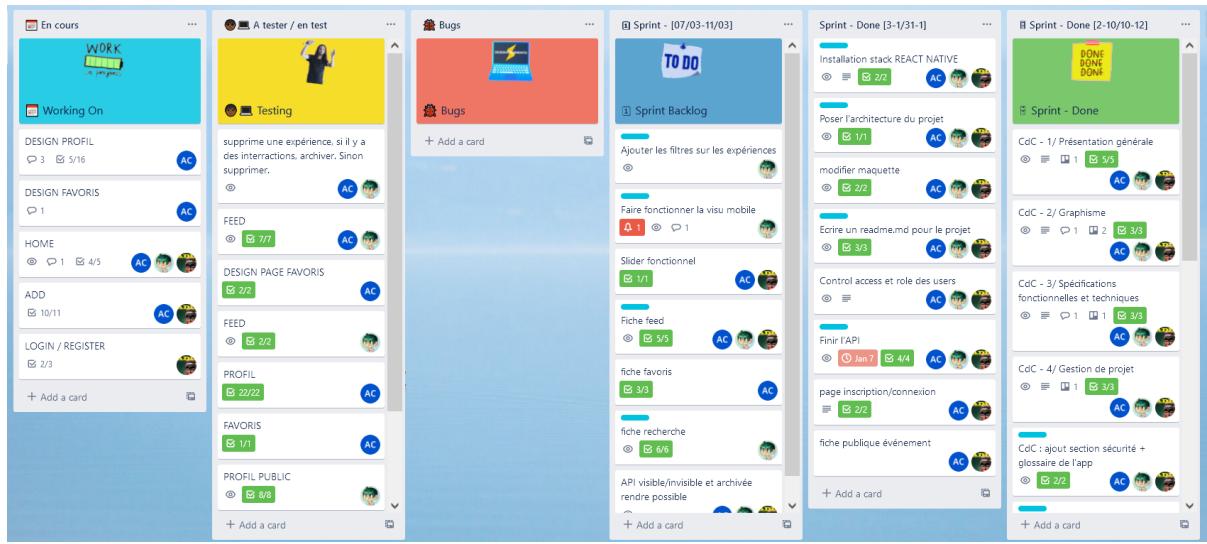
#### a - Diagramme de Gantt

Le **diagramme de Gantt** nous permet de visualiser dans le temps les diverses **phases** de notre projet. Sans être figé dans le marbre, il nous a permis de représenter **graphiquement**, dès le début du projet, toutes les étapes qui le constituent.



## b - Trello

**Trello** a été un outil très utile pour gérer nos **sprints et tâches**. Notre tableau Trello comporte plusieurs listes comme "backlog du sprint actuel", "en cours", "à tester". Les cartes sont déplacées selon le statut de la tâche. Nous le consultons quotidiennement et le tenons à jour du mieux que nous le pouvons.



## 6/ Développement du back-end de l'application

### 1 - Conception et mise en place de la base de données

Pour concevoir notre base de données, nous avons utilisé la méthode Merise.

**Merise** est une méthode de conception de systèmes informatiques créée dans les années 1970. Elle permet de **définir et représenter les données** et leurs relations entre elles avant la réalisation du projet.

#### a - Modèle Conceptuel de Données (MCD)

Le modèle conceptuel des données (MCD) permet de concevoir le système d'information **indépendamment** des choix techniques et de son aspect informatique. C'est une **représentation graphique et facilement compréhensible** des éléments et de leurs liens.

Lorsque l'on conçoit une base de données avec le MCD de Merise, on obtient un **schéma** avec des **entités** (rectangles), **des propriétés** (liste de données d'une entité) **et des associations** (liens).

Pour préciser au mieux les **associations**, on utilise des **cardinalités**. Les cardinalités sont des caractères (0,1, n) qui fonctionnent par **couple** et qui sont présents de chaque côté d'une **association** (sur chaque « patte »).

Elles permettent par la suite (dans un MPD / MLD) de définir :

- les **clés étrangères** dans le cas d'une Contrainte d'Intégrité Fonctionnelle (CIF) : quand il y a une relation "0,1" ou "1,1".
- la création d'une **table intermédiaire** dans le cas d'une Contrainte d'Intégrité Multiple (CIM) : quand il y a une relation "0,n" ou "1,n".

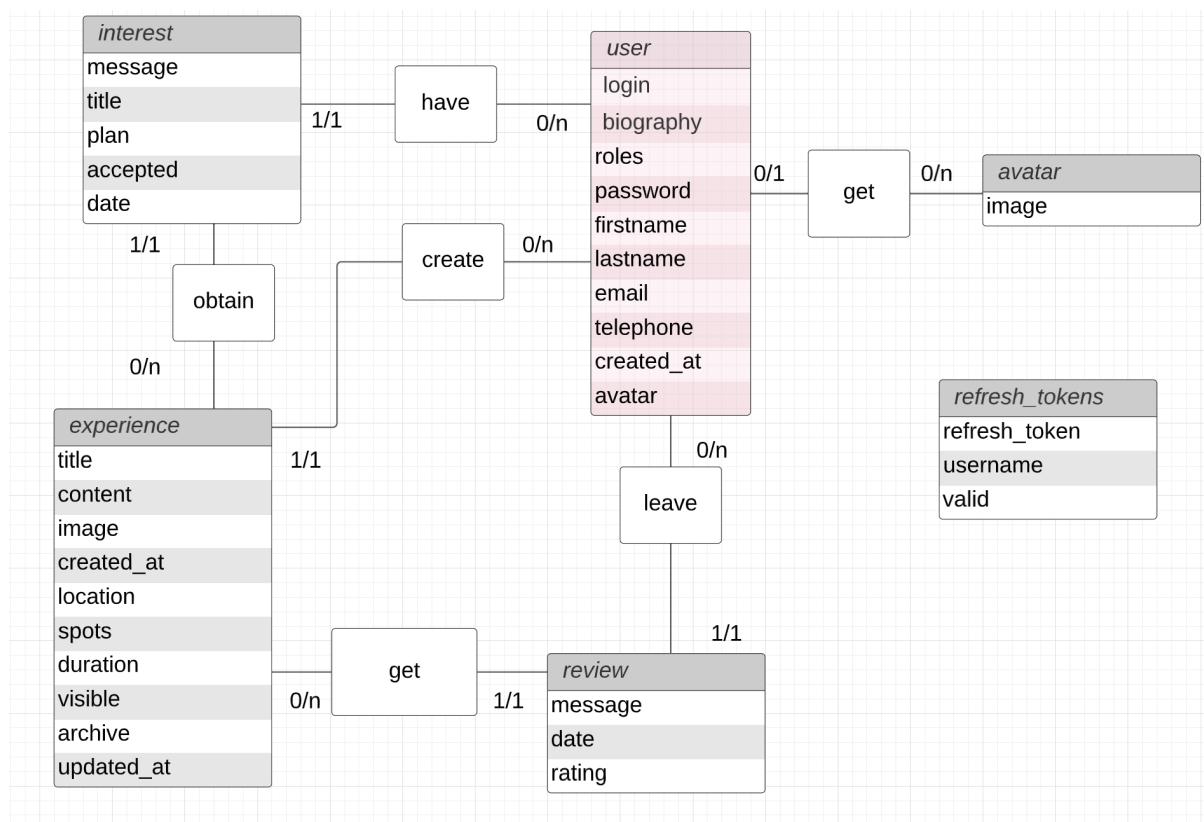
Les cardinalités possibles sont :

- **0,1** : au minimum 0, au maximum 1 seule valeur (CIF)
- **1,1** : au minimum 1, au maximum 1 seule valeur (CIF)
- **0,n** : au minimum 0, au maximum plusieurs valeurs (CIM)
- **1,n** : au minimum 1, au maximum plusieurs valeurs (CIM)

Dans notre MCD, la relation entre les entités "experience" et "user" est, à gauche "1,1" et à droite "0,n".

Il faut lire ces cardinalités ainsi :

- Pour exister, une expérience doit être créée par minimum ET maximum 1 utilisateur. L'entité "experience" est dépendante de "user".
- Un utilisateur peut créer 0 ou plusieurs (n) expériences.



## b - Modèle Logique de données (MLD)

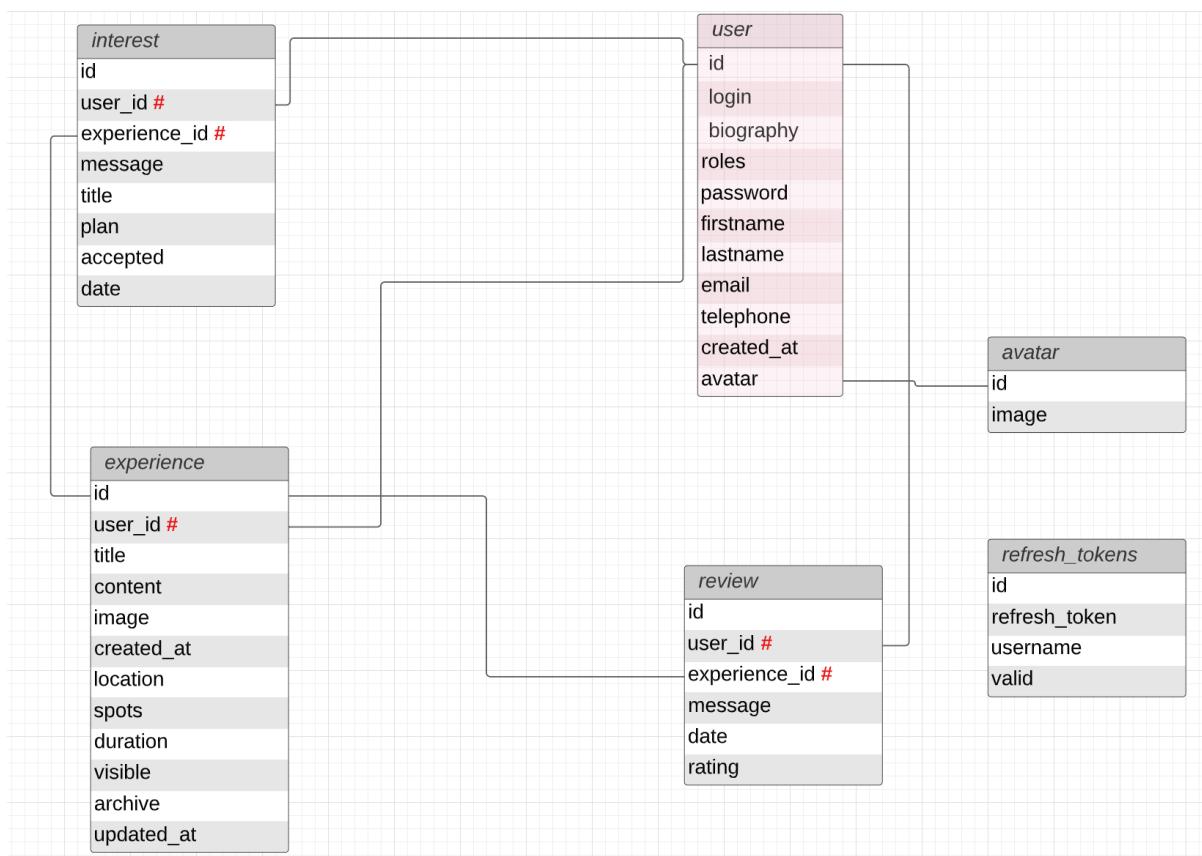
Le Modèle Logique des Données (MLD) se rapproche de la structure réelle d'une base de données.

Il transforme les entités en **tables**.

Il supprime les cardinalités et les verbes de description du MCD et les remplace par des **clés principales et des clés étrangères**. Celles-ci nous permettent de comprendre les liens entre les différentes entités et les **contraintes** qui vont en découler.

Chaque table possède une **clé primaire unique** qui assure **l'intégrité** de la table. Les **contraintes** de clé primaire garantissent des **données uniques**, c'est pourquoi elles sont souvent définies pour une colonne **d'identité**.

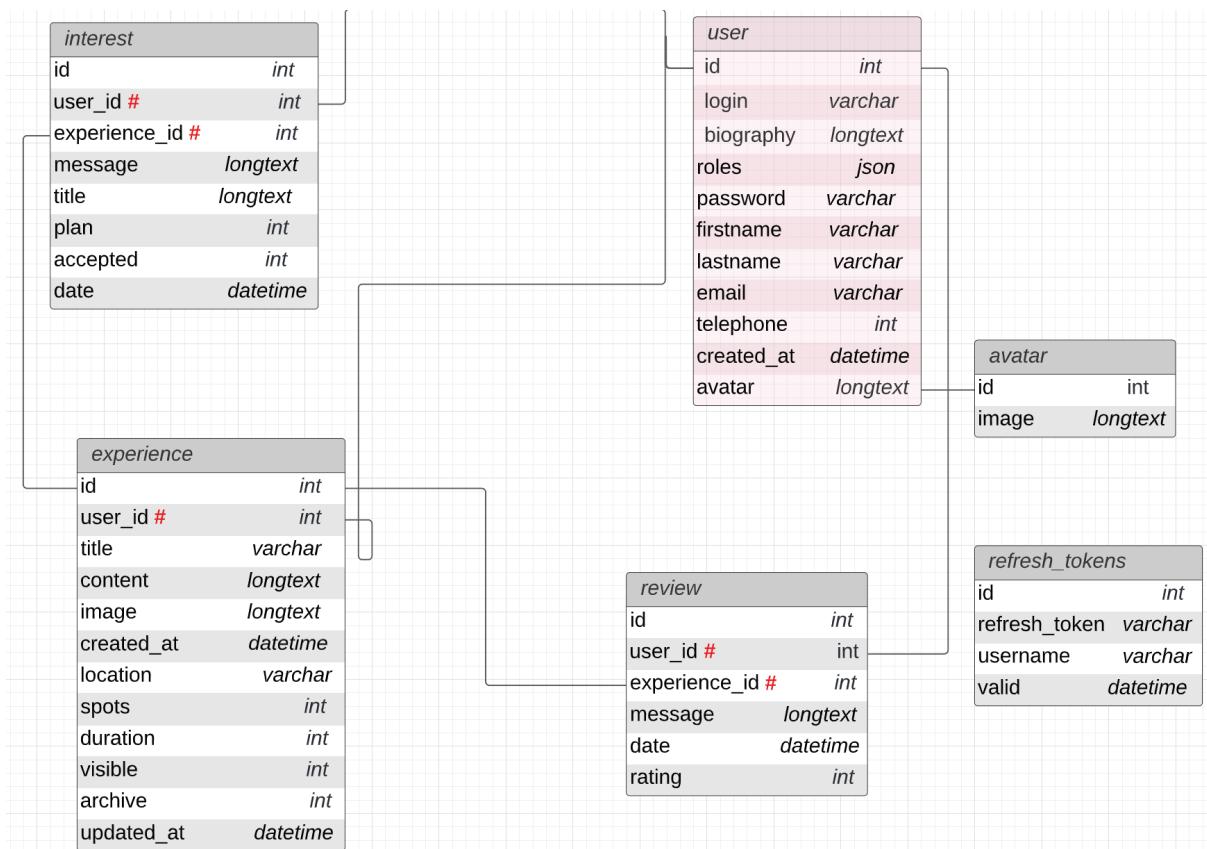
Une **clé étrangère** est une colonne utilisée pour **établir et conserver une liaison** entre les données de deux tables.



### c - Modèle Physique de données (MPD)

Le modèle physique des données (MPD) est encore plus proche de la **structure finale** de la base de données. Il **modélise et décrit** complètement son **architecture**. Il précède son implémentation dans le SGBD.

On répertorie tous **les champs** de toutes les tables, ainsi que **leurs types** (int, varchar etc). Les clés étrangères sont signifiées par un dièse.



## 2 - Développement de l'API

### a - API Platform

Le terme **API** est l'acronyme de Application Programming Interface (interface de programmation d'application). Il s'agit d'une interface qui permet à deux applications de **communiquer** l'une avec l'autre. Une API rend disponible les **ressources et les services** d'une application à une application tierce.

Pour notre projet, nous avons choisi d'utiliser **API Platform**. C'est un **framework PHP** basé sur **Symfony**.

API Platform permet de créer une **API REST et GraphQL**. Il transforme les **modèles** de l'application symfony en **ressources** d'API, avec les **points d'entrées** correspondants.

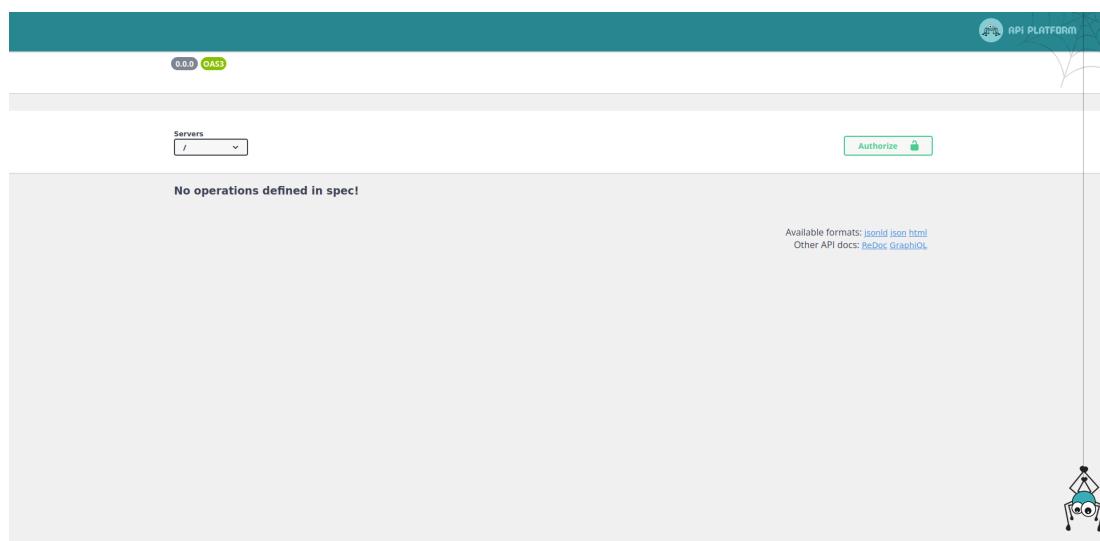
Il génère aussi une **documentation openAPI/Swagger** automatiquement.

### b - Mise en place de l'API

Après avoir créé un **projet symfony**, importé les dépendances, et configuré le fichier .env pour qu'il corresponde à notre version et port de mysql, nous avons **installé** API Platform grâce à composer, le gestionnaire de dépendances de php.

```
composer require api
```

L'interface swagger était désormais disponible au <http://127.0.0.1:8000/api>. Elle montrait une page presque vide.



Il a fallu ensuite **créer nos ressources**/entités basées sur les **tables** définies dans notre **MPD**.

```
symfony console make:entity
```

Cette commande permet aussi d'ajouter toutes les propriétés de l'entité.

On peut ensuite faire les **migrations** qui vont créer la base de données :

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

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

Pour **exposer l'entité** comme une ressource visible sur l'interface swagger, nous devons écrire une **annotation** dans l'entité User.php.

Dans un premier temps, nous allons ajouter cette ligne :

```
use ApiPlatform\Core\Annotation\ApiResource;
```

Puis ensuite nous allons ajouter l'annotation @ApiResource :

```
/**  
 * @ORM\Entity(repositoryClass=UserRepository::class)  
 *  
 * @ApiResource(  
 *     normalizationContext={"groups"={"user:read"}},  
 *     denormalizationContext={"groups"={"user:write"}}  
 * )  
 */
```

Si nous retournons sur l'interface swagger et que nous actualisons la page, nous pouvons voir nos entités :

The screenshot shows the API documentation for the 'User' resource. It lists the following methods:

- GET** /api/users: Retrieves the collection of User resources.
- POST** /api/users: Creates a User resource.
- GET** /api/users/{id}: Retrieves a User resource.
- PUT** /api/users/{id}: Replaces the User resource.
- DELETE** /api/users/{id}: Removes the User resource.
- PATCH** /api/users/{id}: Updates the User resource.

Below the methods, there is a section titled 'Schemas' containing three items:

- Avatar
- Avatar.jsonld
- Experience-experience.read

### c – Test d'une route

Pour ajouter un nouvel utilisateur dans notre base de données, nous pouvons utiliser cette interface graphique :

- On Déroule la section : **Creates a User resource.**
- Puis on clique sur le bouton **try it out**
- On saisi nos **valeurs en json**
- On valide en appuyant sur le bouton **execute**

The screenshot shows the configuration for a POST request to '/api/users' to create a new User resource. The 'Request body' field contains the following JSON schema:

```
{
  "loggin": "text",
  "biographie": "string",
  "firstname": "string",
  "lastname": "string",
  "email": "string",
  "password": "rara",
  "avatar": "string"
}
```

At the bottom, there are 'Execute' and 'Clear' buttons.

Ceci envoie une **requête http POST** à l'url: <http://127.0.0.1:8000/api/users> avec comme body les valeurs en json.

Une fois le formulaire validé, on reçoit une réponse. Si le code est 201, l'utilisateur a bien été ajouté en base de données.

**Responses**

**Curl**

```
curl -X 'POST' \
  'http://127.0.0.1:8000/api/users' \
  -H 'Accept: application/json' \
  -H 'Authorization: bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpYXQiOjE2NTYwNzk0OTksImV4cCI6MTY1NjA4MzA50wi cm9sZXMiOlsoiUK9MRV9VU0VSIL0sInVzZXJuYW1lIjoisWNob3UiLCJpZCIGMyw=' \
  -H 'Content-type: application/json' \
  -d '{
    "login": "rara",
    "biography": "string",
    "firstname": "string",
    "lastname": "string",
    "email": "string",
    "telephone": 0,
    "password": "rara",
    "avatarn": "string"
}'
```

**Request URL**

```
http://127.0.0.1:8000/api/users
```

**Server response**

Code	Details	
201	<b>Response body</b> <pre>{     "@context": "/api-contexts/User",     "@id": "/api/users/5",     "@type": "User",     "id": 5,     "login": "rara",     "biography": "string",     "roles": [         "ROLE_USER"     ],     "firstname": "string",     "lastname": "string",     "email": "string",     "telephone": 0,</pre>	
201	<b>Response body</b> <pre>{     "@context": "/api-contexts/User",     "@id": "/api/users/5",     "@type": "User",     "id": 5,     "login": "rara",     "biography": "string",     "roles": [         "ROLE_USER"     ],     "firstname": "string",     "lastname": "string",     "email": "string",     "telephone": 0,     "created_at": "2022-06-24T14:06:20+00:00",     "experiences": [],     "events": [],     "interests": [],     "avatarn": "string" }</pre>	
	<b>Response headers</b> <pre>cache-control: no-cache,private content-length: 316 content-location: /api/users/5 content-type: application/json; charset=utf-8 date: Fri, 24 Jun 2022 14:06:20 GMT link: &lt;http://127.0.0.1:8000/api/docs.jsonld&gt;; rel="http://www.w3.org/ns/hydra/core#apiDocumentation" location: /api/users/5 vary: Accept x-content-type-options: nosniff x-frame-options: deny x-powered-by: PHP/8.0.19 x-robots-tag: noindex</pre>	
<b>Responses</b>		
<b>Code</b>	<b>Description</b>	<b>Links</b>
201	User resource created	GetUserItem The <b>id</b> value returned in the response can be used as the <b>id</b> .

## d - Statuts de requête HTTP

Les différents statuts utilisés dans ce projet sont :

- 200 : OK

Indique que la requête a réussi

- 201 : **created**

Indique que la requête a réussi et une ressource a été créée

#### - 400 : **bad request**

Indique que le serveur ne peut pas comprendre la requête à cause d'une mauvaise syntaxe, par exemple si le json est mal formaté

#### - 401 : **unauthorized**

Indique que la requête n'a pas abouti, dans notre cas, s'il n'y a pas de jeton JWT.

#### - 500 : **internal server error**

Le serveur ne répond pas.

### d - Méthodes HTTP utilisées dans ce projet

Les **opérations** définies par API Platform sont les méthodes CRUD (Create, Read, Update, Delete)

- **GET** : Pour la récupération de données.

- **POST** : Pour l'envoi de données

- **PUT** : Pour mettre à jour l'intégralité des informations d'une donnée

- **PATCH** : Pour mettre à jour partiellement une donnée

- **DELETE** : Pour supprimer une donnée

## 3 - Middleware

Un middleware est une couche logiciel entre deux couches de logiciels. Pour notre application nous avons utilisé un des middlewares Redux : **Redux-Thunk**.

**Redux Thunk** permet de communiquer de manière asynchrone avec une application externe afin de récupérer ou de sauvegarder des données. Redux Thunk permet de facilement distribuer des actions qui suivent le cycle de vie d'une requête à une application externe.

Redux est **une librairie de state management**, elle permet de gérer l'état global de notre application.

```
const store = configureStore({
  reducer: {
    [authSlice.name]: persistReducer(authPersistConfig, authSlice.reducer),
  },
  middleware: [thunk],
});
```

## 4 - Sécurité et JWT

La sécurité des applications est importante, car les applications d'aujourd'hui sont souvent disponibles sur divers réseaux et connectées au Cloud, ce qui augmente leur vulnérabilité aux menaces et aux violations de sécurité.

Voici des exemples d'attaques qu'on peut avoir :

- Attaques DoS et attaques brute force sur les APIs
- Attaques brute force sur des APIs
- Injections de code
- Injections SQL
- Attaques Man In The Middle

Afin de garantir cette sécurité nous avons mis en place dans notre projet quelques mesures :

- Les mots de passe des utilisateurs sont hachés
- Authentification grâce au JWT
- Faible durée de vie du JWT

### a - Hachage du mot de passe

Le mot de passe est haché dans le **data persister** du User grâce à l'interface `UserPasswordHasherInterface`.

Quand une entité User est créée, s'il existe un **mot de passe non crypté**, on définit l'attribut `password` comme le **retour** du plainPassword **haché**.

Cette interface utilise **l'algorithme** défini dans security.yaml, dans notre cas, il est sur "auto".

```
if ($data->getPlainPassword()) {  
    $data->setPassword(  
        $this->passwordHasher->hashPassword(  
            $data,  
            $data->getPlainPassword()  
        )  
    );  
}
```

#### b - Authentification par JWT

Un token JWT est utilisé pour **authentifier les utilisateurs** et ainsi leur **attribuer des droits**.

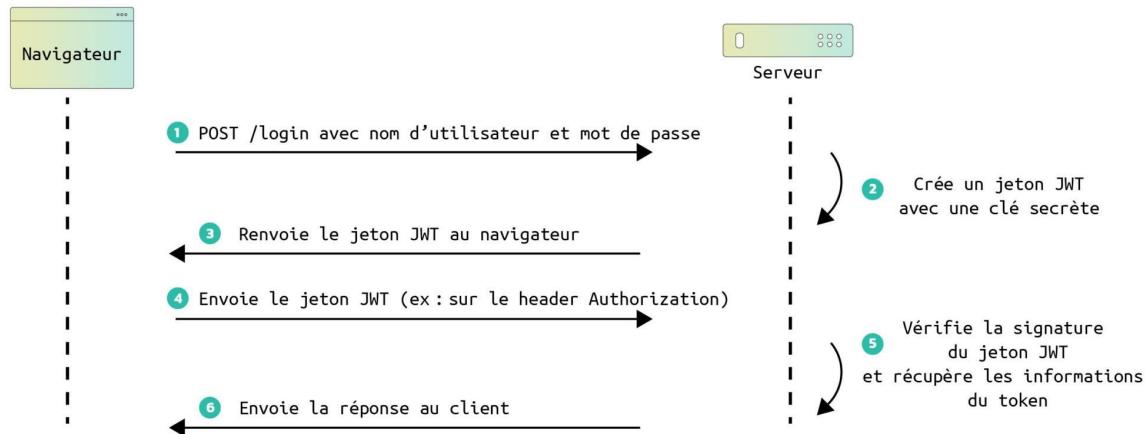
Une mauvaise implémentation peut être source de vulnérabilités. En effet, un utilisateur pourrait par exemple augmenter ses droits (élever ses privilèges et devenir admin par exemple) ou accéder aux données d'un autre utilisateur.

Un token JWT contient :



**Jeton JWT** = Header + payload + signature

vocabulaire



- Une tête « Header », l'algorithme utilisé pour la signature, en JSON encodé en Base64 (généralement HS256)
- Un corps « Payload », les informations du jeton. Il peut s'agir du nom d'utilisateur, du rôle de l'utilisateur ou encore de son email.
- Un cœur « La signature », qui correspond à la concaténation des parties « Header » et « Payload » chiffrée avec la clé privée (détenue par le serveur).

Nous avons utilisé le package **Lexik JWT** (jwt-auth) pour authentifier les utilisateurs.

Toutes les **routes**, exceptées la route login nécessitent d'**envoyer un jeton** en même temps que la **requête** pour être exécutée. S'il n'y a pas de jeton, la requête retourne une erreur **401** avec le message "JWT Token not found".

### c - Test du login

Nous avons exposé une route login sur l'API et sur le swagger qui permet d'envoyer une requête **POST** à la route <http://127.0.0.1:8000/api/login>. En corps, on envoie nos identifiants "test".

#### Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/api/login' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "login": "test",
    "password": "test"
}'
```



#### Request URL

```
http://127.0.0.1:8000/api/login
```



#### Server response

Code	Details
200	<b>Response body</b>

```
{
  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpYXQiOjE2NTYxNjg5NTUsImV4cCI6MTY1NjE3MjU1NSwicm9sZXMiOlIsIUK9MRV9VU0VSII0sInVzZXJuWl1IjoigVzdCIsImIkIjoxlCJ1c2VxSX3pIjoil2FwaS91c2Vycy8xIn0.TDyvJC-SBewELFhVdaPkad
aAlmwia_ntbd7WVYNSSo2bRb0IL6PwZDBH3pv5dYBupnCkmuXUpdpmFdqiQ7s1yAVM47oe_m5kau19UwnJWC4NckQb7nmg5JLFBAq1ipENdH
j_5MYB7_cZL_XmkgtIB8z1KXQ8oIGJILJ96ouDX6CjivrTv5PEWizIcJ5eJsNoUF5ny11gL_XhrAFLIL2k51u3Itxbv6_fVx5boNSvWe1
9zbabSEyJvZa0cgkt-kznqjFVo4Fgz4VCK5Pw5U0vLNisRL1CNUPVg183Y84wJsvyNgH2Nxz9LiULSbN61owcYnxYvrUe0ssfhQuI-60Pp
-AVrWk0-bRm3YUFbzspy81JbnA353v65WP3weCuStf15cMZy16pwYrHfRaoV21ZKCgpqIk5Wjn1En0jtqkkVj3i3L69uf4IuAIwcreM-ZBkAv
B9PYprgmfGM9QzYSrf-VH28tTKZkZnNm1YhDgg99n5C1_zxRrEQjRW9KVAOLA-7AwqdlaV4fz3wmwSH5YVpAGTo2Y1QRy5-30Y5kFXVp7t4T
VTShAQKB1_jNBd8uHMN-jmAJe432Qg77L-y8C43PodmHVLAY5a9X8PXkCqLh61hk5mzoNY9JXjVHo08YeA-ioo01Hue8s0VNzqMGXAPGcxily
Wo3B3Ph9u6T0",
  "refresh_token": "e92f9267577670fd758189a7b91397dc5f30eb50b6a06bedc7a8f9baea5966b0d1d38a0hfaddhh038d9f05ha
5ea4c2a9b1f70cf3f881687f19fd92245f582412"
```



Download

La réponse contient bien un token.

En utilisant <https://jwt.io/>, on peut **décoder** le JWT :

## Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{  
  "iat": 1656168955,  
  "exp": 1656172555,  
  "roles": [  
    "ROLE_USER"  
  ],  
  "username": "test",  
  "id": 1,  
  "userIri": "/api/users/1"  
}
```

Le JWT permet de lire les rôles, l'id et l'identifiant de l'utilisateur. Il est **requis** pour **naviguer** dans l'application. S'il est déjà en mémoire cache, on passe directement de l'écran "login" à l'écran "protected".

```
useEffect(() => {  
  if (isLoggedIn && token) {  
    navigation.navigate("Protected");  
  }  
}, [isLoggedIn, token]);
```

## 7/ Développement du front-end de l'application

### 1 - React Native

React Native est un **framework JS mobile** qui a l'avantage de permettre de développer pour **Android et iOS**. Il est très proche de ReactJS.

Il exploite les **composants natifs** pour les **rendus UX**.

### 2 - Arborescence du repository

```
> .expo
> .expo-shared
> .idea
> .vscode
> .yarn
> android
> assets
> node_modules
< src
|> api
|> components
|> navigation
|> screen
|> store
< .env
< .gitignore
JS .pnp.cjs
JS .pnp.loader.mjs
JS App.js
{} app.json
B babel.config.js
JS index.js
JS metro.config.js
{} package-lock.json
{} package.json
① README.md
JS secrets.js
↳ yarn.lock
```

#### a - A la racine

.env	Contient nos variables globales. Ex: API_URL
App.js	Contient l'application. C'est là qu'on peut définir des provider (pour le store, pour react native paper).

/src	Contient le code principal de l'application.
/assets	Contient toutes les images et icônes de l'application

### b - Le dossier src

/api	Contient les fichiers relatifs aux appels à l'api. Chaque fichier contient une fonction générique pour les différentes méthodes évitant ainsi les répétitions.
/components	Contient des sous-dossiers correspondant à chaque vue. Il contient tous les composants réutilisables dans le projet.
/navigation	Contient la navigation et l'organisation des stacks et écrans.
/screen	Contient tous les écrans de l'application.
/store	Contient les fichiers permettant de gérer l'état global de l'application grâce à Redux.

## 3 - Ecrans et composants

L'architecture en composants permet de découper les écrans en **éléments indépendants et réutilisables**.

Les composants sont des **fonctions** JavaScript. Ils acceptent en entrées ou paramètres des **props** et renvoient **des éléments React** décrivant ce qui doit **apparaître** à l'écran.

### a - Exemple de composant

Le composant “Loading” nous permet de faire patienter l’utilisateur le temps que les datas chargent. Il contient simplement un **return** qui affiche un composant de la librairie de react native: ActivityIndicator.

```
import React from "react";
import { ActivityIndicator, StyleSheet, View } from "react-native";

function Loading() {
  return (
    <View style={styles.mainBody}>
      <ActivityIndicator color="#f14d53" />
    </View>
  );
}

const styles = StyleSheet.create({
  mainBody: {
    flex: 1,
    justifyContent: "center",
    alignContent: "center",
  },
});

export default Loading;
```

### b - Exemple d'utilisation dans un écran

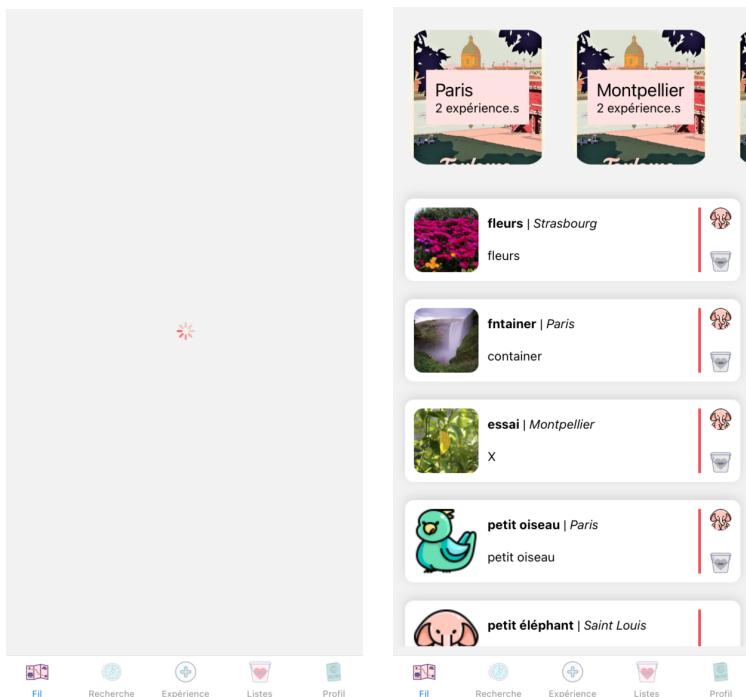
Tant que les datas ne sont pas chargées, le composant “loading” reste affiché. Dès que la variable `isLoading` est passée à false, on affiche les résultats sous forme d’une FlatList.

```

{isLoading ? (
  <Loading />
) : experiences.length > 0 ? (
  <FlatList
    ListHeaderComponent={
      <ContainerCityCarrousel
        experiences={experiences}
        navigation={navigation}
      />
    }
    ListFooterComponent={
      <ContainerFeedExperience
        experiences={experiences}
        navigation={navigation}
      />
    }
  />
)

```

### c - Rendus UX



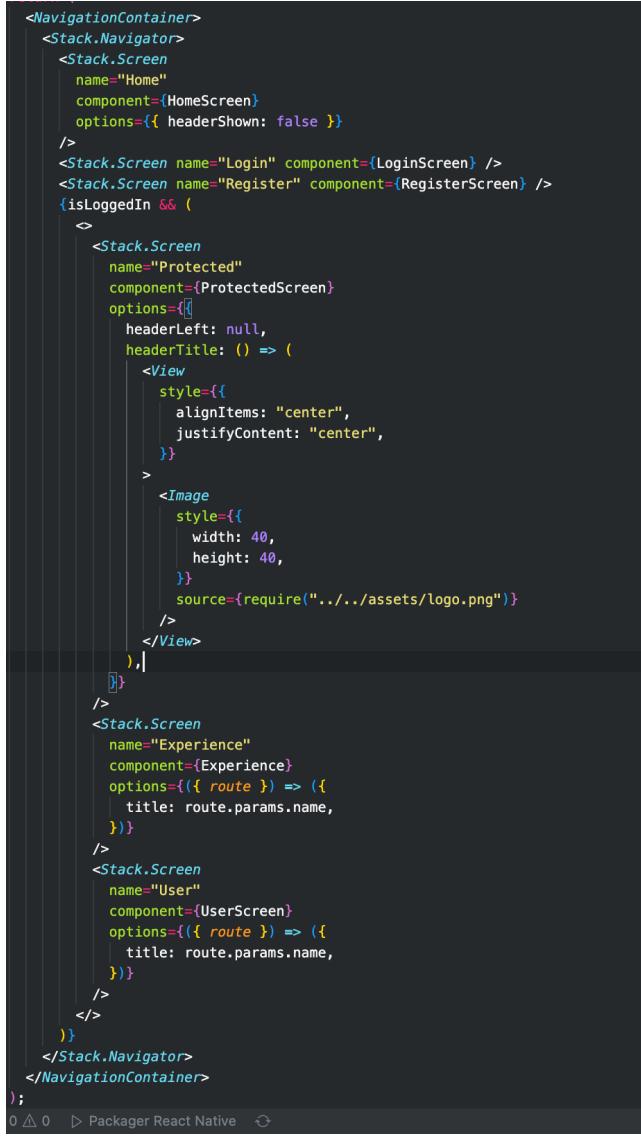
Le *FeedScreen* avant / après chargement des données

## 4 - Navigation

### a - Composant Nav.js

Le composant Nav.js est **chargé de la navigation** dans l'application mobile. Pour qu'elle puisse fonctionner, il faut dans un premier temps importer **react navigation** ainsi que les **différents écrans**.

```
import { NavigationContainer } from "@react-navigation/native";
import { createStackNavigator } from "@react-navigation/native-stack";
```



```
<NavigationContainer>
  <Stack.Navigator>
    <Stack.Screen
      name="Home"
      component={HomeScreen}
      options={{ headerShown: false }}
    />
    <Stack.Screen name="Login" component={LoginScreen} />
    <Stack.Screen name="Register" component={RegisterScreen} />
    {isLoggedIn && (
      <>
        <Stack.Screen
          name="Protected"
          component={ProtectedScreen}
          options={({
            headerLeft: null,
            headerTitle: () => (
              <View
                style={{
                  alignItems: "center",
                  justifyContent: "center",
                }}
              >
                <Image
                  style={{
                    width: 40,
                    height: 40,
                  }}
                  source={require("../assets/logo.png")}
                />
              </View>
            ),|
          )>
        <Stack.Screen
          name="Experience"
          component={Experience}
          options={({ route }) => ({ title: route.params.name, })}
        />
        <Stack.Screen
          name="User"
          component={UserScreen}
          options={({ route }) => ({ title: route.params.name, })}
        />
      </>
    )}
  </Stack.Navigator>
</NavigationContainer>
```

Le **Stack.Navigator** permet le stockage des vues de l'application. L'ordre de celles-ci est important. On voit dans le screenshot ci-dessus les écrans de notre application : "Home", "Login", "Register", "Protected", "Experience", "User".

Les trois premiers sont **accessibles sans login**, les autres requièrent d'être **authentifié**.

### b - Ecran ProtectedScreen.js

L'écran '*ProtectedScreen*' nous permet de créer un **groupe d'écrans** sous forme **d'onglets** (tabs), ainsi que la **navbar** du bas.

Les **icônes** de la navbar, leurs noms, leur comportement au **focus** ainsi que **l'écran** auxquels ils correspondent sont défini dans ce fichier.

```
> function ProtectedScreen() {
  //Permet de créer un groupe de screen avec une navbar en bas
  const Tab = createBottomTabNavigator();

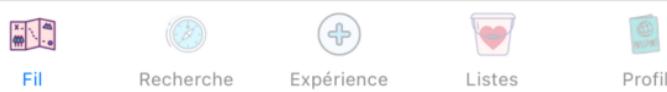
  return (
    <Tab.Navigator screenOptions={{headerShown: false}}>
      <Tab.Screen name="Feed" component={FeedScreen} options={{

        tabBarIcon: ({focused}) => (
          <View>
            <Image
              source={require('../assets/map.png')}
              resizeMode="contain"
              style={{
                width: 25,
                height: 25,
                opacity: focused ? 1 : 0.3,
              }}/>
          </View>
        ), title: "Fil"
      )} />

      <Tab.Screen name="Search" component={SearchScreen}
      options={({ route }) => ({
        tabBarIcon: ({focused}) => (
          <View>
            <Image
              source={require('../assets/compass.png')}
              resizeMode="contain"
              style={{
                width: 25,
                height: 25,
                opacity: focused ? 1 : 0.3,
              }}/>
          </View>
        ), title: "Recherche"
      )})
    />

      <Tab.Screen name="Add" component={AddScreen} options={{

        tabBarIcon: ({focused}) => (
          <View>
            <Image
              source={require('../assets/add.png')}
              resizeMode="contain"
              style={{
                width: 25,
                height: 25,
                opacity: focused ? 1 : 0.3,
              }}/>
          </View>
        ), title: "Expérience"
      )} />
    
```

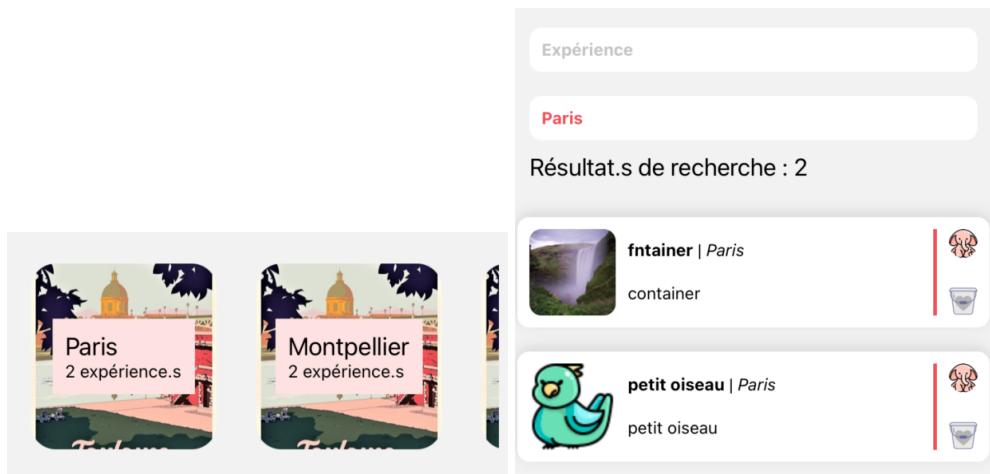


## 5 - Jeu d'essai

Nous allons présenter une fonctionnalité dans son intégralité.

- un utilisateur **clique sur une case “ville”** mise en évidence dans le fil d’actualité
- il est redirigé vers l’écran “**recherche**”
- le **filtre ville** est prérempli
- les **résultats** correspondant se chargent **automatiquement**

### a - Screenshots de l’interface



*Cases dans le fil / Page de recherche*

### b - Code

#### i - Clic sur une ville

Quand on clique sur une case ville, on **navigue** vers l’écran “Search”. On envoie un **paramètre de route** “location”, qui contient le nom de la ville.

```
<TouchableOpacity style={styles.item} onPress={() => {
  navigation.navigate('Search', {
    screen: 'Protected',
    location: item.name,
  })
}}>
```

## *ii - Préremplissage du filtre sur l'écran Search*

Un **hook d'effet** se déclenche dès qu'il y a une modification dans le paramètre de route "location". Il **récupère sa valeur** et l'affecte à la **constante d'état local** "location".

```
//récupère ville sélectionnée dans le feed
useEffect(() => {
  route?.params?.location && setLocation(route.params.location);
  setTitle("");
}, [route?.params?.location]);
```

## *iii - Envoi de la requête*

Un autre **hook d'effet** se déclenche lorsque des **paramètres de recherche** (SearchParamsToString) sont modifiés. Il écoute aussi le changement sur le token.

Si SearchParamsToString n'est pas vide - c'est -à -dire s'il existe des filtres de recherche -, on utilise la méthode 'genericFetchWithToken'.

Celle-ci prend en paramètre :

- la route <http://127.0.0.1:8000/api/experiences?visible=true&location=Paris>
- la méthode GET
- le token

Si la requête est un succès, elle retourne les expériences correspondant aux filtres. Ces résultats sont stockés dans une variable d'état local "experiences".

```

//requête de résultats
useEffect(() => {
  setIsLoading(true);

  if (searchParamsToString.length > 0) {
    console.log("fetch results");
    genericFetchWithToken(
      `${entryPoint}?visible=true&${searchParamsToString}`,
      "GET",
      token
    )
      .then((json) => json.json())
      .then((data) => setExperiences(data))
      .catch((error) => console.error(error))
      .finally(() => setIsLoading(false));
  } else {
    console.log("fetch results fail");
    setExperiences([]);
    setIsLoading(false);
  }
}, [searchParamsToString, token]);

```

#### *iv - Traitement de la requête*

Nous pouvons **filtrer** les expériences grâce à API Platform et à **ApiFilter**. Cette **annotation** permet de créer des filtres sur les attributs d'une ressource.

Nous avons créé deux types de filtres :

- **texte partiel** : title, content, location
- booléen : archive, visible

```

* @ApiFilter(SearchFilter::class, properties={"title" = "partial", "content" = "ipartial", "location" = "partial"})
* @ApiFilter(BooleanFilter::class, properties={"archive", "visible"})

```

#### *v - Aperçu sur le swagger*

Les **filtres** configurés dans l'entité Experience se trouvent bien dans **l'interface**. Quand on déroule l'opération **GET /api/experiences** on trouve les champs : title, location, archive, visible.

**Experience**

**GET /api/experiences** Retrieves the collection of Experience resources.

Retrieves the collection of Experience resources.

**Parameters**

Try it out

Name	Description
page <small>integer (query)</small>	The collection page number <small>Default value : 1</small>
title <small>string (query)</small>	title
Content <small>string (query)</small>	content
location <small>string (query)</small>	location
archive <small>boolean (query)</small>	--
visible <small>boolean (query)</small>	--

**Responses**

## 8/ Tests

### a. Tests route

Dès que nous **modifions l'API** ou la **base de données**, nous testons de nouveau les routes.

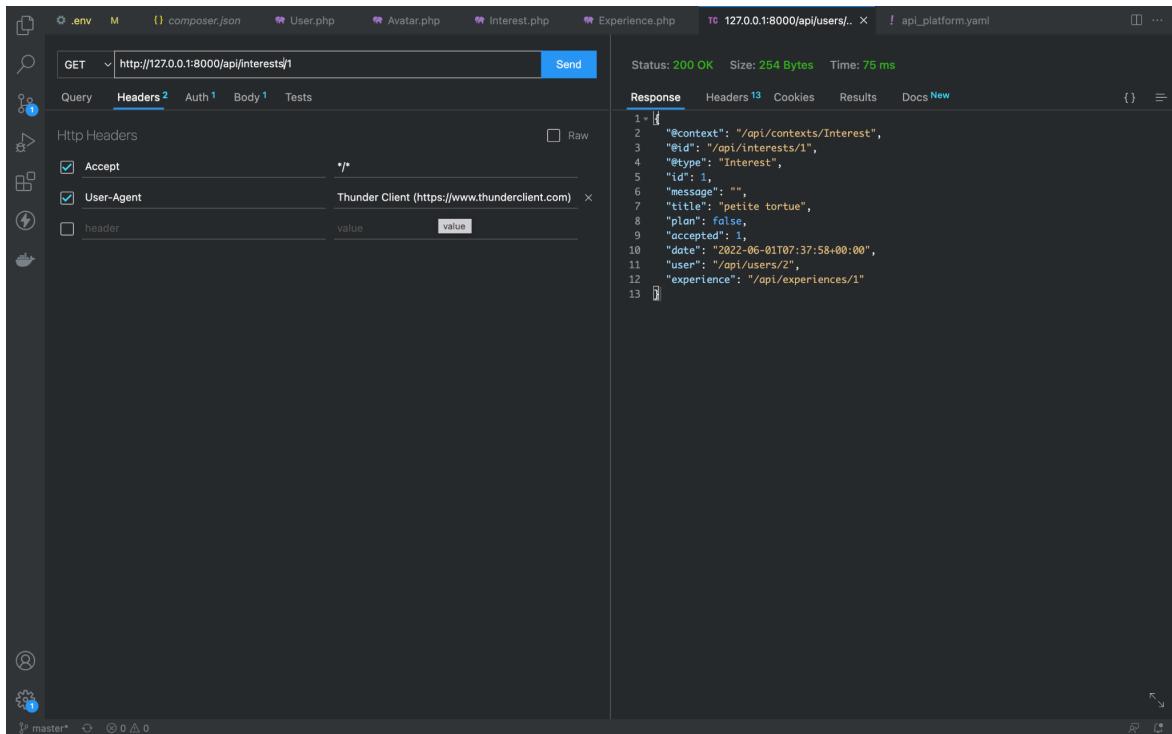
#### i - Thunder Client

Nous avons téléchargé une extension Visual Code Studio, **Thunder Client**, qui nous a permis de tester les différentes routes de notre application.

Exemple : **GET** <http://127.0.0.1:8000/api/interests/1>, on envoie aussi le **token** dans le **body** de la requête.

On peut voir le **statut 200 OK**.

La **réponse** obtenue contient les données de cet intérêt.



## ii - Swagger UI

Swagger UI nous a permis de visualiser et d'interagir avec l'API. Cette interface permet aussi de tester son API et ses routes.

A chaque test, nous vérifions que :

- les **données envoyées** sont bien celles attendues
- le **statut** de la **requête** HTTP est correct
- la réponse correspond à ce que l'on attend

The screenshot shows the Swagger UI interface for a POST request to the endpoint `/api/users`. The request is described as "Creates a User resource". The "Parameters" section shows "No parameters". The "Request body" section is marked as "required" and specifies "application/json" as the content type. The schema for the request body is as follows:

```
{  "login": "rara",  "biography": "string",  "firstname": "string",  "lastname": "string",  "email": "string",  "telephone": 0,  "password": "rara",  "avatars": "string"}
```

At the bottom of the interface are "Execute" and "Clear" buttons.

### b. Tests unitaires

Le **test unitaire** est un moyen de vérifier qu'un extrait de code fonctionne correctement. C'est l'une des procédures mises en œuvre dans le cadre d'une méthodologie de travail agile. Il consiste à isoler une partie du code et à vérifier qu'il fonctionne parfaitement. Il s'agit de **petits tests** qui valident l'attitude d'un objet et la logique du code.

# 9/ Recherche en anglais

## 1 - Présentation du problème

A chaque fois qu'on essayait de se connecter à l'api via **l'ios simulator ou l'android emulator** on rencontrait une **erreur sur le port utilisé**.

## 2 - Recherche

Les mots clés recherchés :

- error react native local api
- query api with react native android studio
- port issue react native api

## 3 - Solution

Nous avons trouvé la solution à notre problème sur **stackoverflow** :

<https://stackoverflow.com/questions/33704130/react-native-android-fetch-failing-on-connection-to-local-api>

Il fallait **utiliser l'adresse ip** de notre ordinateur afin de pouvoir nous connecter à l'api.

Nous avons donc modifié la variable globale API\_URL en conséquence.

API\_URL=http://192.168.137.1:8000/api

The screenshot shows a Stack Overflow question page with the following details:

- Header:** stackoverflow, About, Products, For Teams, Search...
- Left Sidebar:**
  - Home
  - PUBLIC
    - Questions (146)
    - Tags
    - Users
    - Companies
  - COLLECTIVES
    - Explore Collectives
  - TEAMS
    - Stack Overflow for Teams - Start collaborating and sharing organizational knowledge.
    - Create a free Team
    - Why Teams?
- Question Summary:**

14 Answers      Sorted by: Highest score (default)      Trending sort available
- Content:**

You are not able to access your local development server because that port hasn't been forwarded by ADB yet. When you run `react-native run-android`, React Native maps the port 8081 with your mobile device on USB. When you disconnect your USB you won't be able to refresh or hot reload your code anymore. So in this situation you can do 2 things, either map your local server port just like React Native does or use your local IP address.

**1. Mapping Port**

This only works if you are using Android 6.0+. To forward a port using ADB run the following command in your terminal:

```
adb reverse tcp:8163 tcp:8081
```

This will map your local 8163 port to mobile's 8081 port. You'll be able to access your development server this way.

**2. Using local IP address**

You can also use your local IP on React Native development app to reload them without USB. Shake your device or long press the menu button to open developer menu. Open Dev Settings, then tap Debug server host & port for device. Here you can enter your machine's local IP with port number 8081. For ex. if your machine's IP is 192.168.1.100 then you'd enter 192.168.1.100:8081 in here for successful connection. Now we have covered that we can reload the app. After this when you want to use your local machine's development server use the same IP with your server's port number.

You should be good to go with this.
- Answer Details:**

Share Improve this answer Follow

answered Apr 7, 2017 at 12:08 by Jagjit

5,416 ● 1 □ 24 □ 40

1 thank you so much for this! with source and mail at first i thought it was an http url https://www.stackoverflow.com/questions/43730526/react-native-cant-access-local-server-on-android-device-without-usb
- Related Questions:**
  - 618 Hide keyboard in react-native
  - 195 Using an authorization header with Fetch in React Native
  - 550 React Native android build failed. SDK location not found
  - 257 How do I "shake" an Android device within the Android emulator to bring up the dev menu to debug my React Native app
  - 926 What is the difference between React Native and React?
  - 138 Change App Name In React Native
  - 3 React Native Fetch Return Network Request Failed
  - 3 React native TypeError: Network request failed with fetch()
  - 0 React Native Fetch API only works in develop environment
- Hot Network Questions:**
  - Would a solar system with four earths directly opposite to each other have different flora?
  - How to make grep for a regex that appear multiple times in a line
  - The Unaveragables
  - Why would anyone buy a Pony over a Mule?
  - How to join a cube to a cylinder
  - What is the point of term life insurance?
  - Can any Models be "Bagged"?

## 4 - Traduction

Tu ne peux pas accéder à ton serveur de développement local car ce port n'a pas encore été envoyé à ton ADB. Quand tu lances `react-native-run-android`, React Native associe le port 8081 avec ton téléphone portable en USB. Quand tu déconnectes ton USB, tu ne peux plus rafraîchir ou faire un hot reload de ton code. Dans cette situation, tu peux faire deux choses : soit associer le port de ton serveur local comme React Native le fait, soit utiliser ton adresse ip locale.

[...]

### 2. Utiliser ton adresse ip locale

Tu peux aussi utiliser ton adresse ip locale dans une application React Native en développement pour les recharger sans USB.

Agite ton téléphone ou appuie longuement sur le bouton menu pour ouvrir le menu développeur. Ouvre Dev Settings, puis appuie sur Debug server host & port for device. Ici, tu peux entrer l'adresse ip locale de ta machine avec le port 8081. Par exemple, si ton adresse ip est 192.168.1.100, tu dois entrer 192.168.1.100:8081 pour une connexion réussite.