

Dossier de projet professionnel:

Application mobile Junior

**Titre de concepteur développeur
d'applications de niveau 6**

Sommaire:

Introduction:

Présentation personnelle	04
Présentation du projet en anglais	04
Compétences couvertes par le projet	04
Compétences couvertes par des projets annexes	05

Cahier des charges:

Analyse de l'existant	05
User story	05
Les objectifs de l'applications	06
Les livrables	07

Conception du projet et outils:

Langages	08
Frameworks et librairie	08
Maquette	09
Test des route	09
Environnement de développement	09
Environnement de la base de données	10
Maquette de la base de données	10
Versioning	10
Paquets et dépendances	11
Hébergement de l'API	11

Gestion et collaboration du projet:

Trello	12
Git et Github	13
Google chat	14

Conception du front-end:

Pourquoi React Native	15
Charte Graphique	15
Maquette	15

Conception du back-end:

Introduction	18
MCD	18
MLD	19
MPD	20

Développement du front-end:

Installation	21
J'ai créé des composants	21
J'ai mis en place la navigation	22
J'ai consommé mon API	23
J'ai utilisé des hooks	23
J'ai mis en place une architecture à trois niveaux	24

Développement du back-end:

Introduction	25
API-Platform	25
L'architecture de l'API REST	25
Fichiers présents dans mon API	27
Sécurité	28
Hachage de mot passe	29
TOKEN JWT	30
Cas pratique sur POSTMAN	31
Firewall	32

Déploiement de l'API sur PLESK:

Introduction	33
Cas pratique	33

Introduction:

Présentation personnelle:

Je suis Toréa Patissier, et j'ai 26 ans. Ancien légionnaire parachutiste, je suis en reconversion à La plateforme depuis 2020. Actuellement en alternance chez Orange au sein de l'équipe DIST j'y développe un outil de gestion des accès en parallèle des projets d'école. Aujourd'hui je suis en coding school 2 afin de préparer mon titre de concepteur/ développeur d'applications web.

Présentation du projet en anglais:

We started from the simple fact that access to the first job after the training course, whether classic or not, was a real problem.

As a result, we decided to create a mobile application aimed at connecting companies looking for young talents, and conversely young developers looking for their first job. For juniors, it is possible to consult job offers and apply if the offer is interesting. For the business side, on the home page, it will be possible to view the different profiles and post new ads.

Compétences couvertes par le projet:

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

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

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
- Préparer et exécuter le déploiement d'une application

Compétences couvertes par des projets annexes:

4. Gestbadge (projet d'entreprise) : voir dossier de projet (dossier rose)

- Développer la partie front-end d'une interface utilisateur web
- Développer la partie back-end d'une interface utilisateur web

5. Snake : voir dossier de projet (dossier rose)

- Développer une interface utilisateur de type desktop

Cahier des charges:

Analyse de l'existant:

Il existe de nombreuses applications concernant la recherche d'emploi, cependant la volonté de l'application mobile « Junior » est de mettre en avant les profils ayant le plus de difficultés à trouver des postes, à savoir, les juniors.

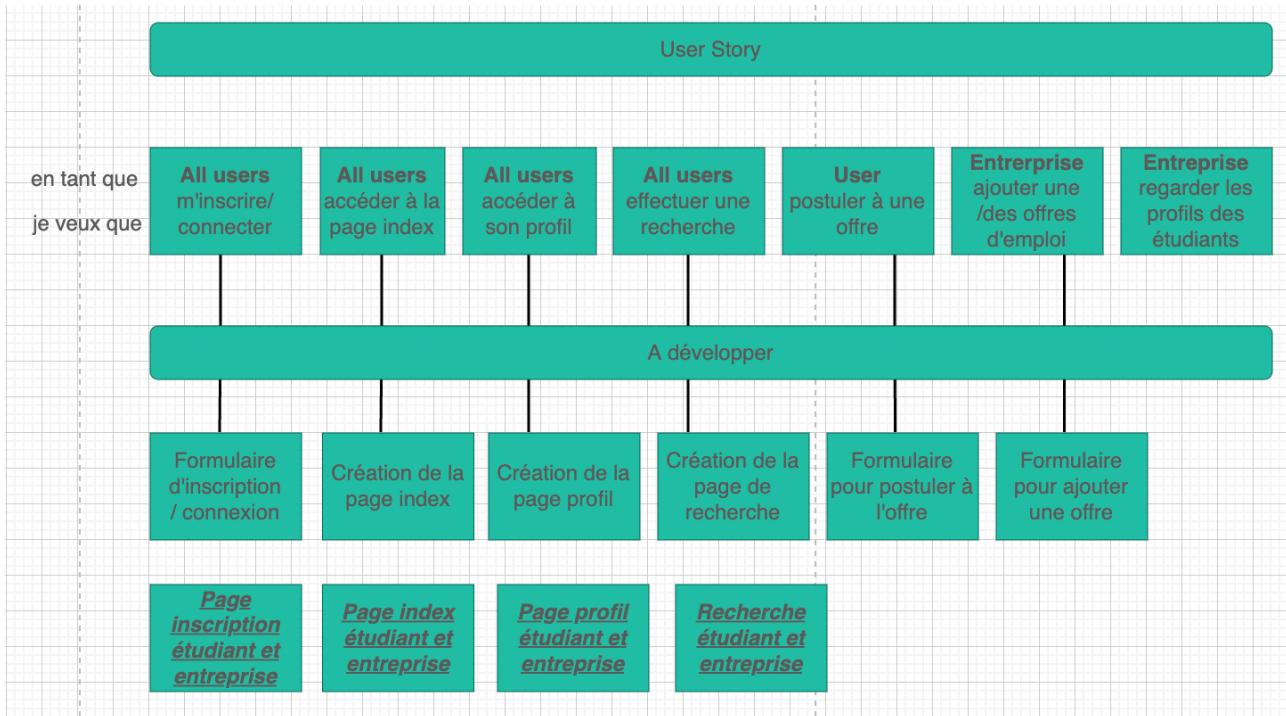
Ainsi, en axant notre démarche en ce sens, nous permettons aux entreprises de bénéficier de jeunes talents, mais aussi au fait de les introduire dans le marché du travail.

User story:

Dans la méthode agile Scrum dont elle est issue, la User Story est censée illustrer un besoin fonctionnel exprimé par les types d'utilisateurs. Elle vise ainsi à assurer que l'on délivre bien de la valeur à l'utilisateur.

Mettre en place une user story est pertinent car:

- elle permet de coller au maximum au besoin et à la vision de l'utilisateur (car elles sont exprimées de manière simple, en langage courant)
- elle engendre un gain de temps considérable pour l'équipe de développement dans la compréhension de la fonctionnalité à développer (toujours grâce à sa forme synthétique)



Ci-dessus la User Story que j'ai réalisé sur Draw.io

Les objectifs de l'applications:

- Faciliter l'insertion des jeunes dans le marché du travail
- Attirer de nouveaux prospects
- Obtenir des partenariats avec des entreprises
- Gagner en notoriété
- Développer notre communauté
- Augmenter notre visibilité

Les livrables :

- **L'écran de chargement :**

- C'est le premier écran sur lequel l'utilisateur arrive quand il ouvre l'application, il contient le logo de « Junior ».

- **L'écran choix inscription en tant qu'entreprise ou junior:**

- Ici l'utilisateur peut choisir de s'inscrire en tant qu'entreprise ou junior.

- **L'écran inscription:**

- Ici différents champs devront être renseigné selon le type d'inscription choisi auparavant.

- **L'écran connexion:**

- Ici l'utilisateur va se connecter en renseignant son adresse e-mail et son mot de passe.

- **L'écran profil:**

- Ici l'utilisateur pourra renseigner ses informations (nom, prénom, ou encore nom de la société si c'est une entreprise).

- **L'écran fil d'actualité entreprise:**

- Ici les entreprises auront accès à tous les profils juniors de l'application et auront la possibilité de consulter des profils en particulier.

- **L'écran fil d'actualité junior:**

- A contrario les Juniors pourront consulter le fil d'actualités des offres d'emploi disponible sur l'application, et consulter une offre en particulier.

- **L'écran consultation d'une offre:**

- Ici il y'aura toutes les informations concernant une offre d'emploi

- **L'écran consultation d'un profil:**

- Ici il y'aura toutes les informations concernant un profil junior

Conception du projet et outils:

Langages:



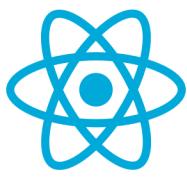
J'ai utilisé **PHP** via le framework **Symfony** pour la conception de mon API.

J'ai utilisé **Javascript** via **React Native** pour le développement de l'application mobile.

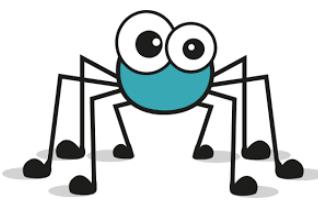
Frameworks et librairie:



Symfony



React Native



Pour le back-end, j'ai utilisé **Symfony** et **API Platform** pour le développement de l'API. Symfony est l'un des frameworks PHP les plus riches en fonctionnalités. Les deux avantages technologiques les plus remarquables de Symfony sont les bundles et les composants.

API-Platform permet non-seulement de créer rapidement une **API REST**, mais également sa documentation au format OpenAPI (ex Swagger).

Pour le front-end, j'ai utilisé React Native. Un des **avantages** dans un projet de développement d'application typique nécessite des équipes de développement Android et iOS. Un tel arrangement peut entraîner des incohérences. Dans certains cas, l'apparence et les fonctions de l'application ne seraient pas les mêmes sur les deux plateformes. Cependant, l'utilisation de **React Native offre l'avantage de construire des applications Android et IOS avec une seule équipe**. Il est donc bénéfique d'avoir un développeur natif expérimenté parmi les membres de l'équipe. Deux équipes de développement différentes ne sont plus nécessaires.

Un des **inconvénients** est le fait que React Native ait été **développé par Facebook**. Cela constitue une des lacunes de la plateforme. **Supposons que Facebook cesse de fournir une sauvegarde** de la plateforme, alors tout s'effondrera.

Maquette:



J'ai utilisé figma pour créer la maquette de l'application en passant par les différentes étapes de conception, du zoning à la maquette (pas de prototype). **L'avantage** de figma est son panel de fonctionnalité, et le fait que ce soit un outil totalement **gratuit**.

Test des routes:



J'ai utilisé postman pour tester les différentes routes de mon API via les **requêtes HTTP**. Cette application gratuite permet de créer des dossiers avec les routes, puis de sauvegarder les **URLs** entrés ainsi que le **JSON** reçu en tant que réponse. Postman permet également d'exécuter automatiquement tous les appels d'une collection les un après les autres, d'où l'importance de définir l'ordre. Les requêtes HTTP peuvent également comporter des **Request Headers**. Il s'agit de données transmises en en-tête de chaque requête ayant différentes utilités. Dans certains cas, il faut spécifier certains *headers*, comme un **Token d'authentification**.

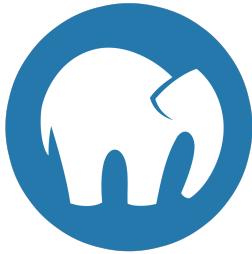
Environnement de développement:



J'ai utilisé **VS Code** comme éditeur de code. Ces fonctionnalités incluent la prise en charge du **débogage**, la mise en évidence de la syntaxe, la **complétion intelligente** du code, les **snippets**, la **refactorisation** du code et **Git intégré**.

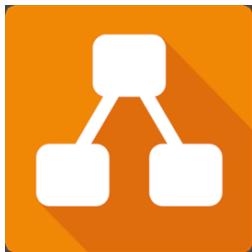
J'ai utilisé **Xcode** pour avoir accès au simulator et voir ainsi mon projet en cours de développement. Xcode est un environnement de développement pour macOS.

Environnement de la base de données:



Concernant l'environnement de la base de données j'ai utilisé MAMP. Sur la version local de mon API j'ai utilisé MySQL puis MariaDB sur la version en production. Concernant le SGBDR j'ai utilisé PhpMyAdmin.

Maquette de la base de données:



Concernant la conception du MCD,MLD,MPD ainsi que le User Story j'ai utilisé [draw.io](#)

Versioning:



Git est un logiciel **VCS (Version Control System) local** qui permet de sauvegarder l'historique de modifications de leurs projets.

Quant à **GitHub**, il s'agit d'une plateforme web qui intègre les fonctionnalités de contrôle de versions de Git.

Il permet ainsi de les utiliser en collaboration et d'instaurer un système de codage social. J'ai eu recours à Git et GitHub pour chaque projet.

Le grand avantage de GitHub, c'est qu'il **facilite la collaboration** à une échelle mondiale sur les projets. Tous les développeurs dans le monde vont pouvoir récupérer des projets et y contribuer si le propriétaire du projet le permet. Il constitue également un portfolio qui est souvent demandé par les recruteurs. Même si la différence entre Git et GitHub est un peu confuse, il est clair que ce sont deux outils inestimables pour le développement.

Paquets et dépendances:



Dependency Manager for PHP

J'ai utilisé NPM pour **l'installation des différents paquets** de mon application via **React Native**. npm est le gestionnaire de paquets par défaut pour l'environnement d'exécution JavaScript Node.js.

J'ai utilisé composer concernant **l'installation des différentes dépendances** du mon **API** via **Symfony**.

Composer est un gestionnaire de dépendances pour PHP. Il s'inspire du système de package npm utilisé par node.js. L'exécutable Composer est un script PHP qui est capable de télécharger et installer les librairies nécessaires à un projet.

Hébergement de l'API:



Ayant un accès **gratuit à Plesk**, je l'ai utilisé pour **déployer** mon **API** afin qu'elle soit en ligne et puisse être utilisé par les autres membres de mon équipe.

Plesk permet aux clients d'hébergement Web d'exécuter de nombreuses tâches depuis une interface robuste et simple d'utilisation. Cette solution inclut tous les outils de création et de gestion des sites Web, boîtes mail, bases de données, etc.

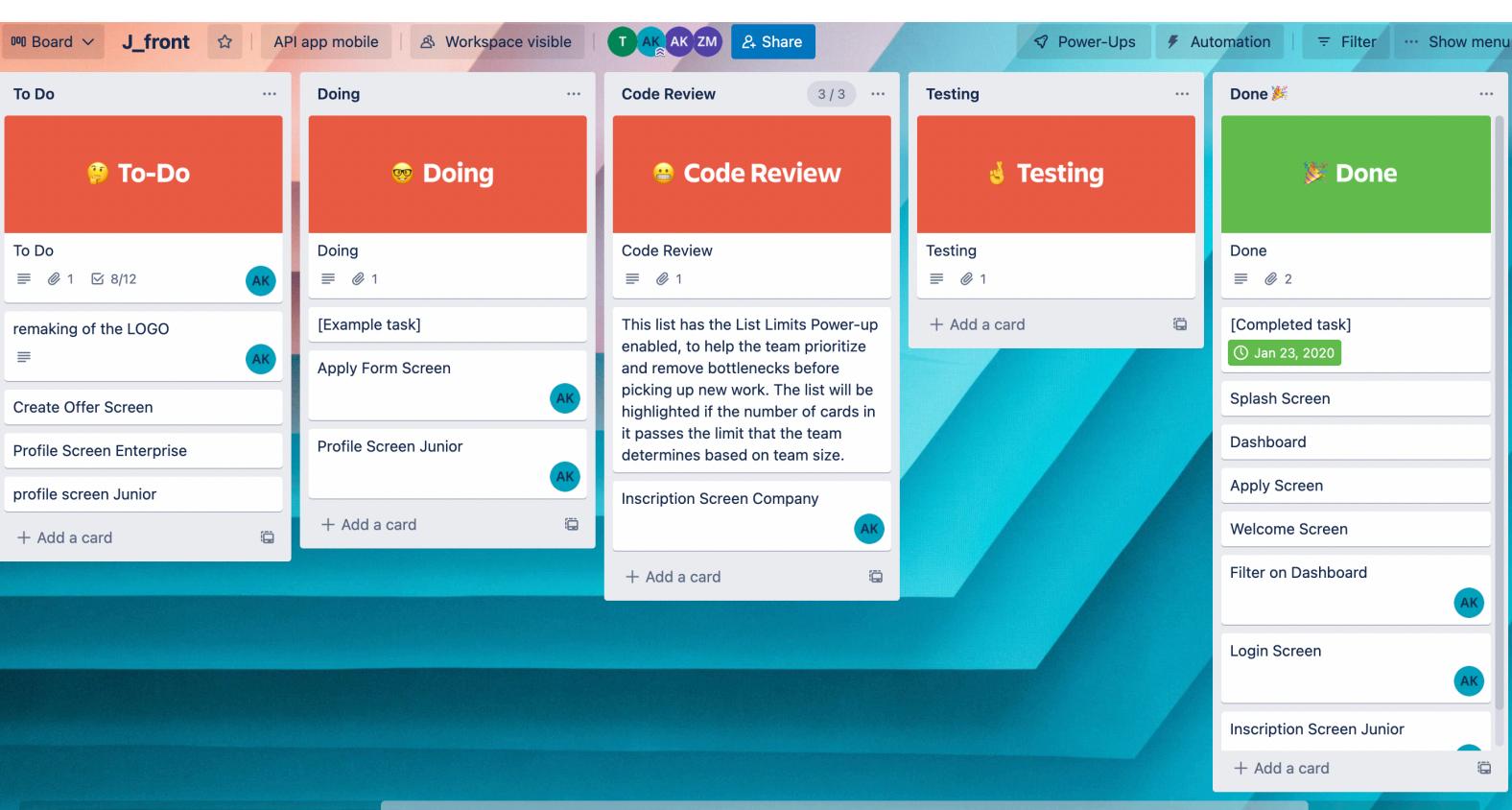
J'ai pu y rattacher mon **dépôt git** afin de pouvoir lier mon API à un nom de domaine et faire en sorte que chaque modification apporté en cours de production mette à jour la version en ligne de mon API

Gestion et collaboration du projet:

Trello:

Trello est un outil de gestion de projet en ligne, lancé en septembre 2011 et inspiré par la méthode Kanban. Il repose sur une **organisation des projets** en planches listant des cartes, chacune représentant des tâches. Cet outil nous m'a permis de **collaborer** avec mon **équipe** concernant le développement de l'application mobile.

Dans un premier temps, nous avons organisé une réunion pour lister les différentes tâches à faire pour le projet dans la colonne « To-Do » .



Ci-dessus le Trello que j'ai utilisé durant l'année pour la gestion des différentes tâches concernant le développement de l'application mobile

Si j'ai une nouvelle tâche à faire, je vais l'ajouter dans la carte To-do et elle passera par les différentes étapes avant d'arriver à "Done". De cette façon, je peux consulter l'avancée des différentes tâches du projet et leurs statuts.

Git et Github:

Git est un logiciel **VCS (Version Control System) local** qui permet de sauvegarder l'historique de modifications de leurs projets.

Quant à **GitHub**, il s'agit d'une plateforme web qui intègre les fonctionnalités de contrôle de versions de Git.

Il permet ainsi de les utiliser en collaboration et d'instaurer un système de codage social. J'ai eu recours à Git et Github pour chaque projet.

The screenshot shows a GitHub repository page for 'torea-patissier/API_junior'. The repository is public and contains 37 commits across 8 branches and 10 tags. The 'Code' tab is selected. On the right, there's a sidebar with project details:

- About:** API pour l'application mobile Junior. It has 0 stars, 1 watching, and 1 fork.
- Releases:** 10 tags. A link to 'Create a new release' is available.
- Packages:** No packages published. A link to 'Publish your first package' is available.
- Contributors:** 2 contributors: torea-patissier (Toréa) and abdulrahman-kamara (kamara Abdulr...).

Ci-dessus le répertoire Github concernant mon API, on peut y retrouver différentes informations:

- Le nom de répertoire (**API_Junior**)
- Le nom de la branche sur laquelle je suis (**ToreaAPI5**)
- Le nombres de commits effectués (**37**)
- Le nombres total de branches (**8**)
- Le nombres de tags (**10**)

Google chat:

Google Chat est un logiciel de communication développé par Google conçu pour les équipes qui fournit des messages directs et des salles de chat d'équipe, similaires aux concurrents Slack et Microsoft Teams, ainsi qu'une fonction de messagerie de groupe qui permet le partage de contenu sur Google Drive.

Zoheir, Abdulrahman, Ahcene ▾

4 membres

 Torea PATISSIER 9 juin, 07:39

<https://github.com/torea-patissier/myReactNativeAppJunior>

torea-patissier/
myReactNativeAppJunior

V1 test de l'application mobile Junior en react native

A 1 Contributor I 0 Issues ⭐ 0 Stars F 0 Forks

GitHub - torea-patissier/myReactNativeAppJu...
github.com

V1 du login
Il manque une route pour log out
Sur l'API
Mais j'ai fais une condition IF qui affiche HOME si j'ai un Token SINON affiche Register/Login
Vous pouvez jeter un coup d'oeil
Si tu peux m'expliquer comment ça fonctionne la route /me et à quoi ça sert Ahcene stp
Bonne journée l'équipe

 Zoheir MAATALLA 9 juin, 09:44

Salut l'équipe
J'espère pere vous allez bien
J'ai ai un petit problème sur react native que j'ouvre le projet et je meet npm start me sort ça

↓ Aller au bas des messages

Historique activé

A ☺ GIF ↑ ↻ ➤

Ci-dessus le groupe google chat sur lequel j'ai collaboré durant l'année pour debugger ou partager l'avancé de mes tâches concernant l'application mobile.

Conception du front-end:

Pourquoi React Native?

Concernant le front-end de l'application mobile, j'ai choisis d'utiliser React Native. Le gros **avantage** est le fait de pouvoir développer une application **multiplateformes**, c'est à dire qu'avec le même code je peux développer et une application **iOS**, mais aussi **Android**.

Charte Graphique:

Couleur primaire:  #0070FF

Couleur secondaire:  #FFFFFF

Maquette:

Le maquettage pendant la phase de conception avec une forte implication de l'utilisateur final, actuel ou futur, et dans une démarche projet. Les itérations peuvent donner lieu à des ateliers de présentation aux utilisateurs.

Dans le cadre de mon application mobile, j'ai réalisé une maquette durant la phase de conception. J'ai utilisé **figma** pour créer les différentes étapes de la maquette

La maquette que j'ai réalisé se décompose en différentes parties:

1- Le zoning:

Le zoning est la schématisation grossière de ce que sera la future application mobile. J'ai utilisé des blocs pour représenter l'emplacement des différents éléments d'un écran de l'application mobile (bouton, formulaire..).

2 - Le wireframe:

Le wireframe est la suite logique du zoning. Chaque bloc que j'ai réalisé lors de l'étape précédente peut être à présent une image, du texte, ou représenter un bouton.

3 - Le mockup:

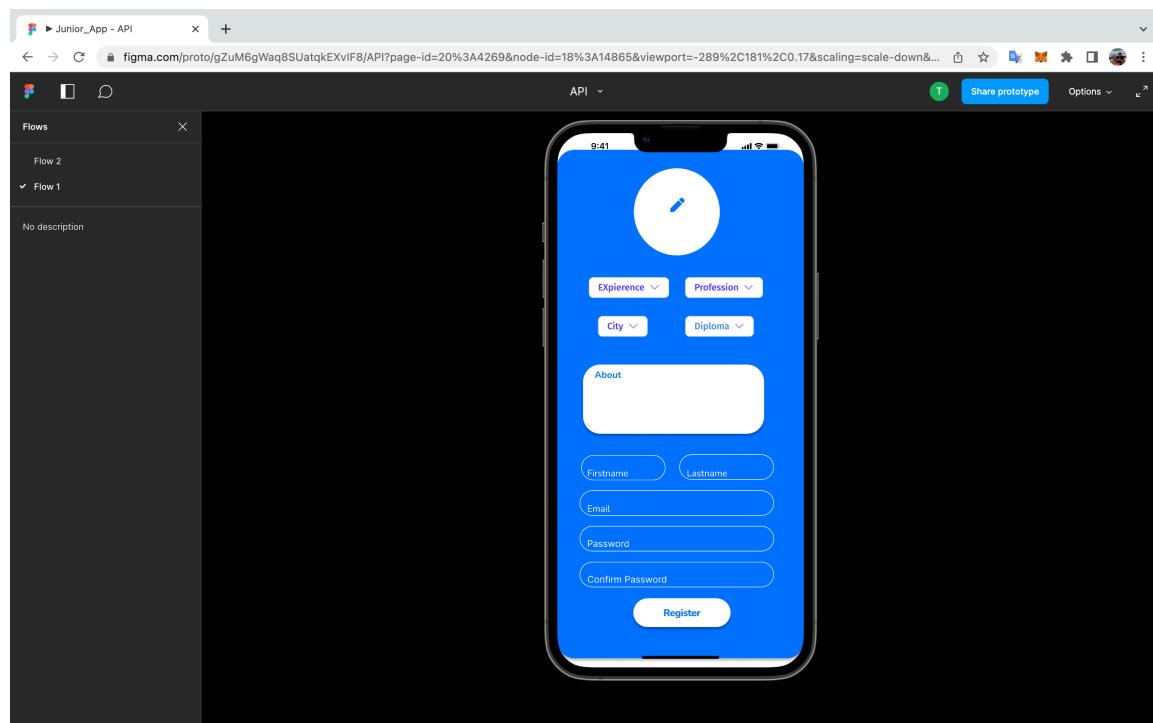
Le mockup est une maquette de la future interface mobile. C'est un outil de conception graphique, d'évaluation et de communication visuelle. De plus, il sert à améliorer l'ergonomie de l'interface et l'expérience utilisateur avant de créer un prototype.

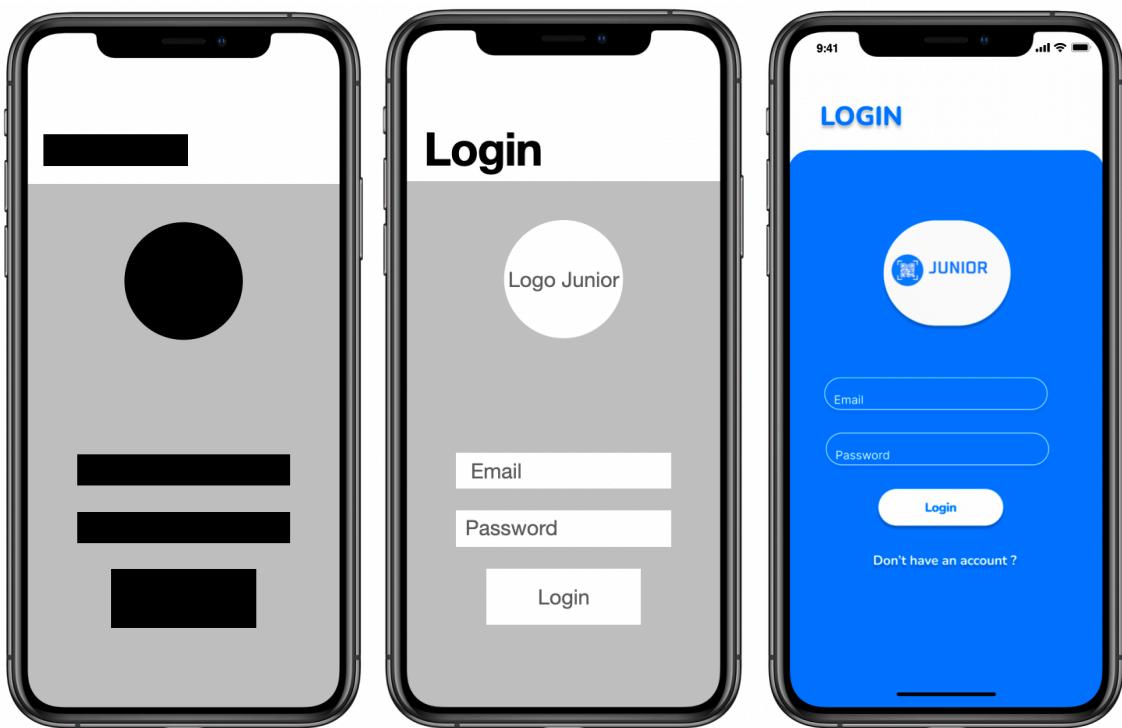
4 - Le prototype:

Contrairement aux maquettes et schémas qui sont statiques, le prototype est interactif.

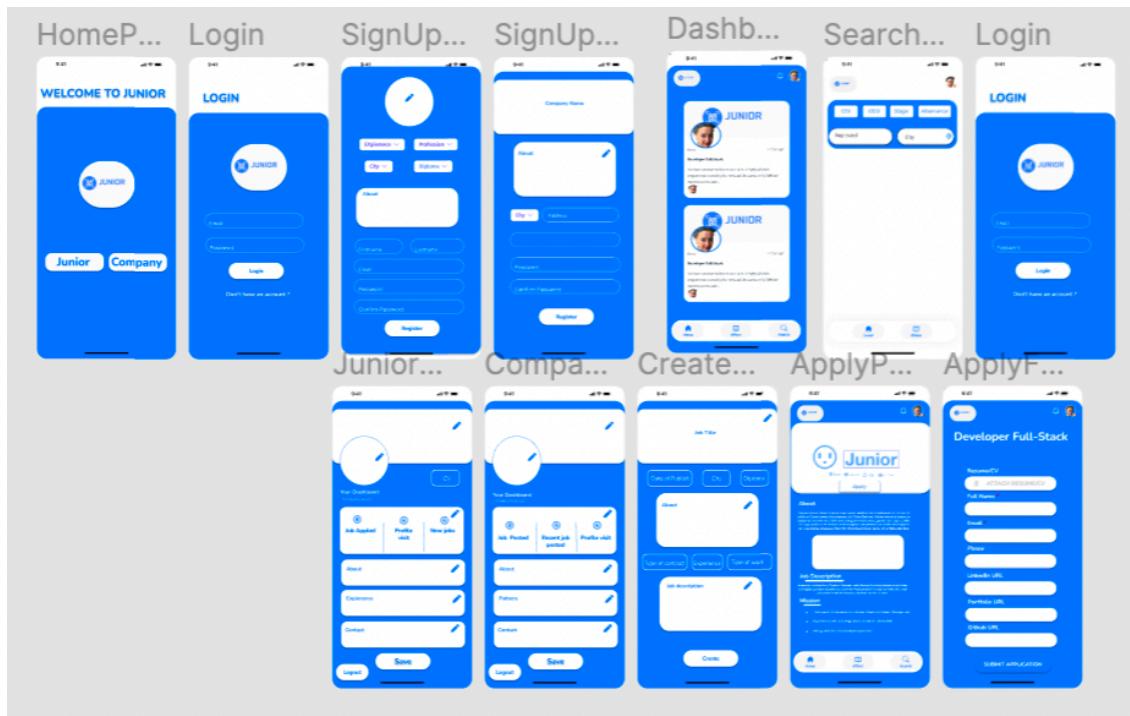
Le prototype peut avoir un but uniquement expérimental sans réutilisation dans le projet réel ou être réalisé comme une première version du projet en cours de production.

<https://www.figma.com/proto/gZuM6gWaq8SUatqkEXvIF8/API?page-id=20%3A4269&node-id=18%3A14865&viewport=-289%2C181%2C0.17&scaling=scale-down&starting-point-node-id=18%3A12227&show-proto-sidebar=1>





Ci-dessus, un exemple de la décomposition des 3 étapes pour la création de l'écran login
dans l'ordre: zoning, wireframe, mockup (ou maquette)



Ci-dessus différents écrans de figma

Conception du back-end:

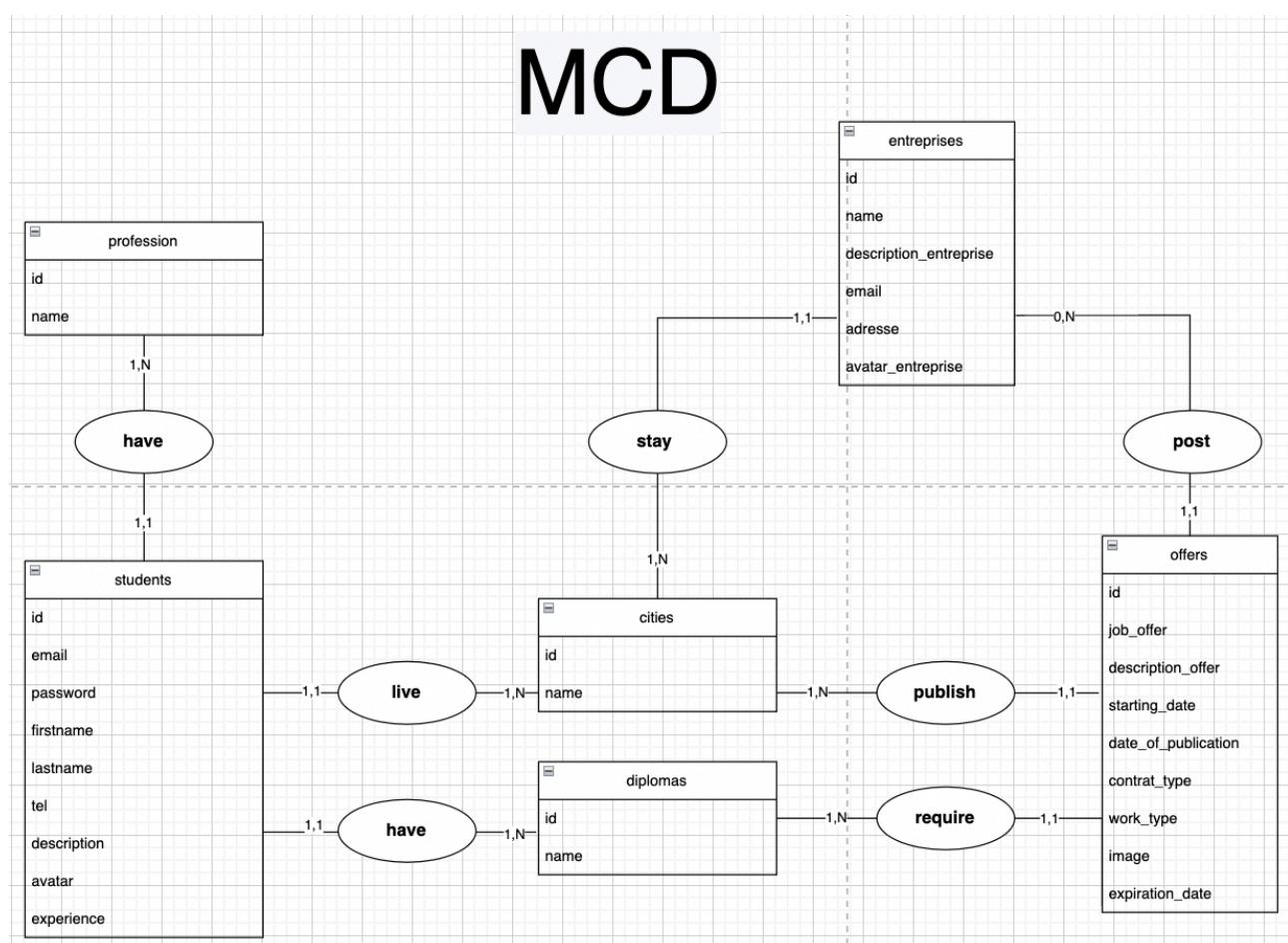
Introduction:

La partie étude et conception est une étape primordiale dans la réalisation d'une base de données. Pour cela j'ai eu recours à la méthode Merise.

Merise est une méthode d'analyse, de conception et de gestion de projet informatique.

MCD:

Dans une premier temps j'ai créé un MCD. Le MCD est une représentation graphique de haut niveau qui permet facilement et simplement de comprendre comment les différents éléments sont liés entre eux.

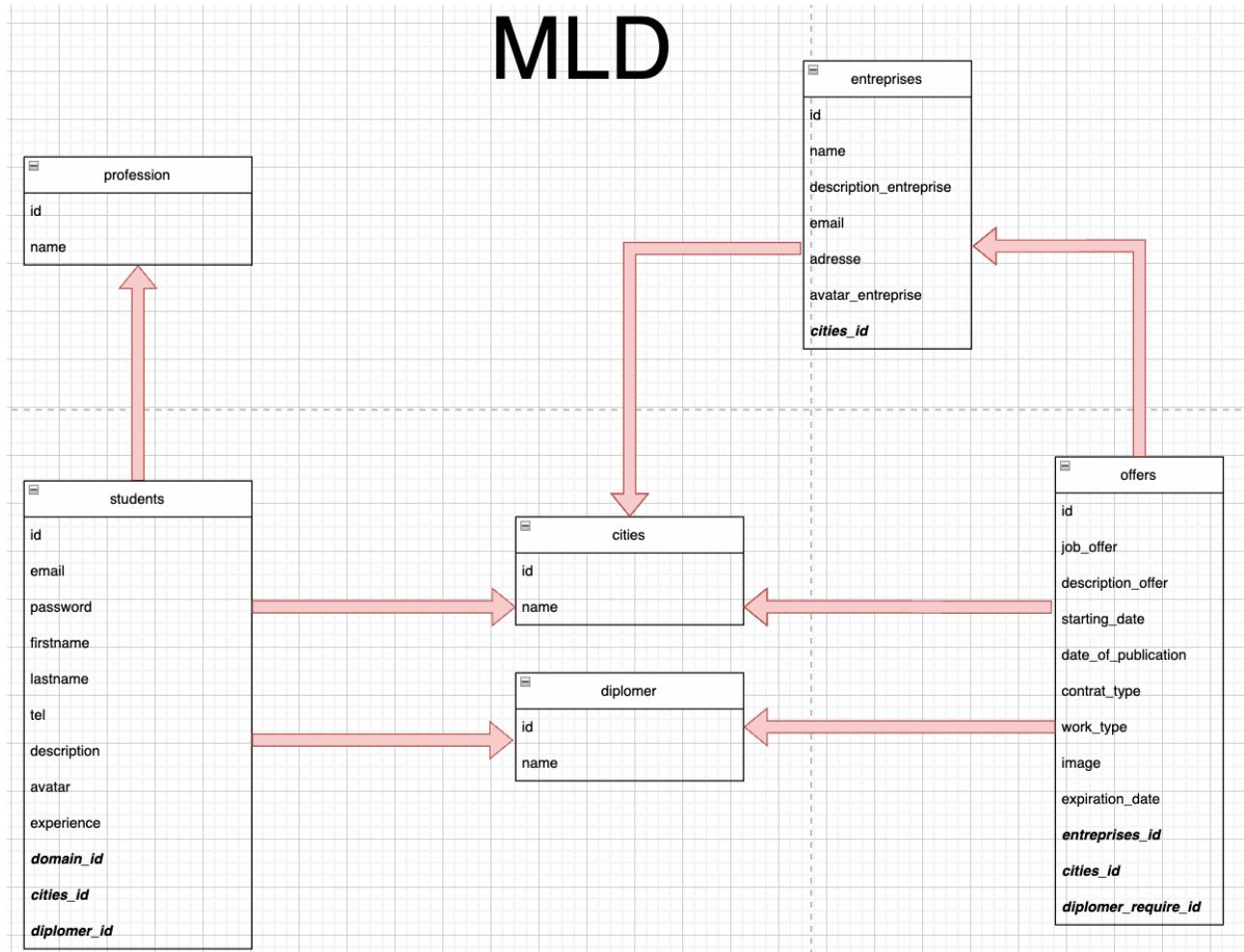


Les informations sont représentées logiquement en utilisant un ensemble de règles et de diagrammes codifiés :

- Les **entités** (1 rectangle = 1 objet) ;
- Les **propriétés** (la liste des données de l'entité) ;
- Les **cardinalités** (les petits chiffres au dessus des « pattes »).
- Les **relations** qui expliquent et précisent comment les entités sont reliées entre elles (les ovales avec leurs « pattes » qui se rattachent aux entités) ;

MLD:

Pour la deuxième étape, pas de travail poussé à réaliser, il s'agit juste d'appliquer quelques règles toutes simples.

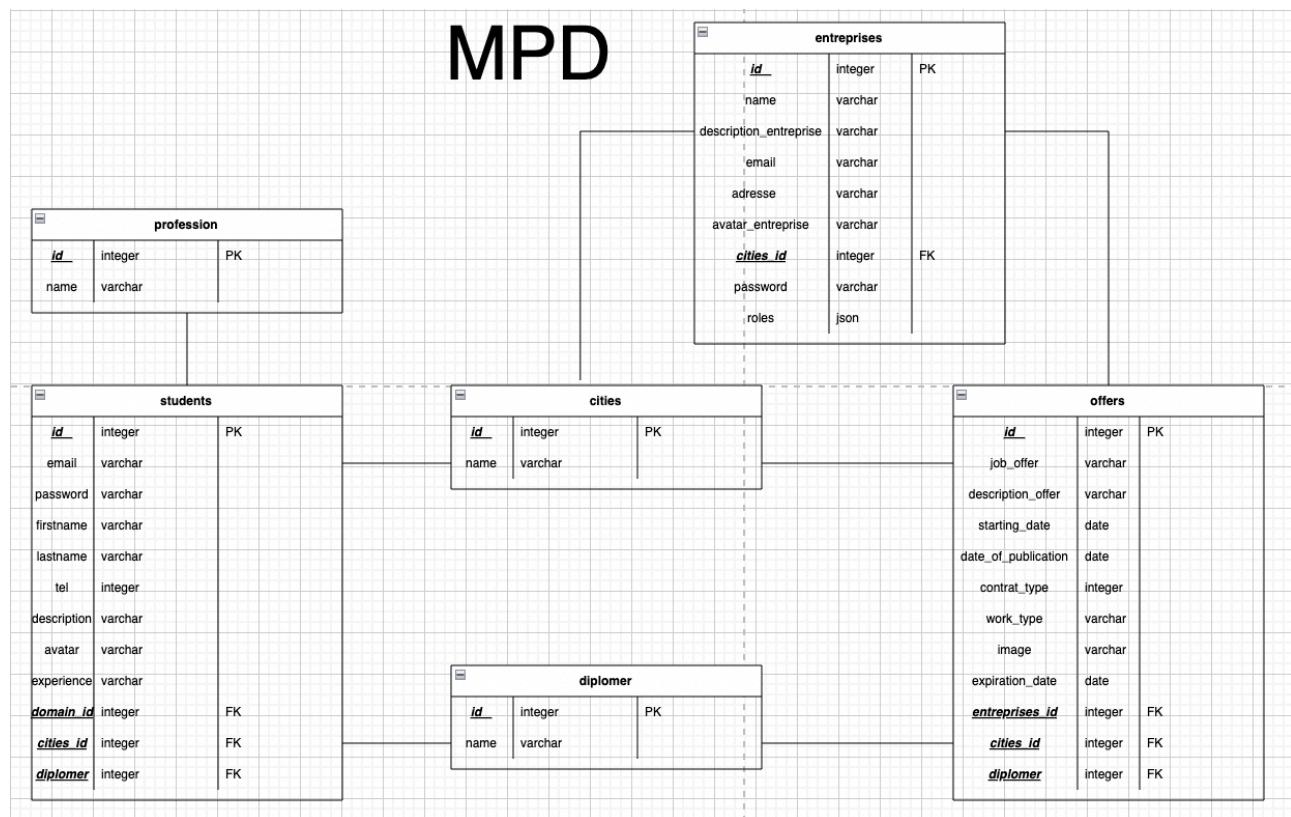


- Il n'y a **plus de cardinalité**
- Pour une **relation N,1** l'entité N donne sa **clé primaire** pour l'entité 1. L'entité 1 aura donc pour **clé étrangère N**
- Une **relation N,N** aurait donné un table de liaison pour récupérer les clés primaires des 2 entités N.

MPD:

Cette étape m'a permis de construire la structure finale de la base de données avec les différents liens entre les éléments qui la composent. Pour la peine, j'ai changé aussi de vocabulaire :

- Les **entités** se transforment en **tables**
- Les **propriétés** se transforment en **champs** (ou **attributs**)



Compétence du REAC validé:
Concevoir une base de données

Développement du front-end:

Installation:

D'abord j'ai installé **homebrew** et **node**: (<https://changelog.com/posts/install-node-js-with-homebrew-on-os-x>)

j'ai installé **EXPO** : npm install --global expo-cli

j'ai installé **react native** : npm install -g react-native-cli

j'ai installé **react navigation** : npm install @react-navigation/native

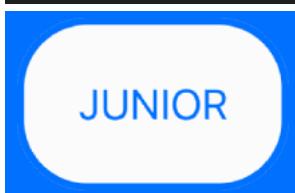
j'ai installé **Xcode** pour avoir le simulator: (<https://developer.apple.com/documentation/xcode/running-your-app-in-the-simulator-or-on-a-device>)

J'ai lancé le projet: expo init my-app

J'ai créé des composants:

Les **composants** permettent de **découper** l'interface utilisateur en **éléments indépendants et réutilisables**, permettant ainsi de considérer chaque élément de manière isolée.

```
<View style={styles.buttonContainer}>
  <ButtonComponent activeOpacity={0.6}>
    <View style={styles.button}>
      <Button
        style={{ ...styles.buttonText, ...styles.props }}
        title="JUNIOR"
        onPress={() => navigation.push("LoginScreen")}
      />
    </View>
  </ButtonComponent>
</View>
```



J'ai mis en place la navigation:

Stack Navigator permet à mon application de passer d'un écran à l'autre, chaque nouvel écran étant placé au-dessus d'une pile.

Par défaut, le navigateur de pile est configuré pour avoir l'apparence familière d'iOS. De nouveaux écrans glissent depuis la droite. Cependant j'ai également mis en place d'autres types de navigations, comme la **BottomTabNavigation**, le **DrawerNavigation**

Stack Navigation

Installer la librairie « React Navigation » pour gérer la navigation des différents écrans.



```
export default StackNavigation = () => {
  return (
    <Stack.Navigator headerInside={false}>
      <Stack.Screen name="Welcome" component={WelcomeScreen} />
      <Stack.Screen name="Login" component={LoginScreen} />
      <Stack.Screen name="Profile" component={SplashScreen} />
      <Stack.Screen name="ProfileContent" component={ProfileContent} />
      <Stack.Screen name="Register" component={RegisterScreen} />
      <Stack.Screen name="JuniorLoginScreen" component={JuniorLoginScreen} />
    </Stack.Navigator>
    // <Stack.Screen name="Dashboard" component={DashboardScreen} />
  );
};
```

Ci-dessus la pile d'écrans, on peut voir que le 1er étant "Welcome" ce sera celui-là qui sera affiché en 1er.

J'ai consommé mon API:

```
//Inscription Junior
const Junior = async (firstname, lastname, email, password, navigation) => {
  setIsLoading(true);
  axios
    .post(` ${BASE_URL}/api/register_user`, {
      firstname,
      lastname,
      email,
      password,
      roles,
    })
    .then((res) => {
      let userInfo = res.data;
      console.log("REGISTER JR OK, INFO = ", userInfo);
    })
    .catch((err) => {
      console.log(roles)
      console.log(`ERREUR REGISTER JR : ${err}`);
    });
  setIsLoading(false);
};
//Inscription Junior
```

J'ai fait une fonction qui va faire appel à mon API et faire une requête sur la route /api/register_user pour inscrire un utilisateur en base de données.

J'ai utilisé des hooks:

Un Hook est une fonction qui permet de « se brancher » sur des fonctionnalités React.

```
const [firstname, setFirstname] = useState('');
<TextInput
  style={styles.TextInput}
  placeholder="firstname"
  value={firstname}
  onChangeText={(text) => setFirstname(text)}
/>
```

Ci-dessus j'ai utilisé le hook useState.

firstname est la **valeur initial** de mon hook, elle pour valeur ‘‘

setFirstname est la **fonction qui va mettre à jour firstname**

Quand je vais rentrer du texte dans mon input alors la valeur de firstname changera. L'objectif de ce hook était de pouvoir récupérer le prénom de l'utilisateur pour l'envoyer en base de données via l'API lors de son inscription

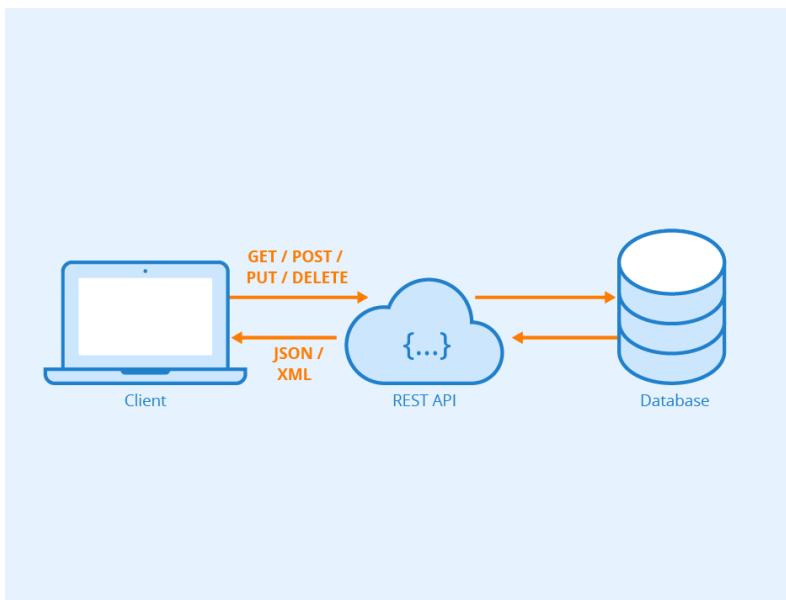
J'ai mis en place une architecture à trois niveaux:

L'architecture à trois niveaux est une architecture d'application logicielle bien établie qui organise les applications en trois niveaux:

Le niveau Présentation est l'interface utilisateur et la couche de communication de l'application, où l'utilisateur final interagit avec l'application. Sa principale fonction est d'afficher des informations à l'attention de l'utilisateur et d'en collecter de ce dernier.

Le niveau Application, également appelé niveau logique ou niveau intermédiaire, est le cœur de l'application. Dans ce niveau, les informations collectées dans le niveau Présentation sont traitées.

Le niveau Données est l'endroit où les informations traitées par l'application sont stockées et gérées. Dans mon cas, il s'agit d'un système de gestion de base de données relationnelle tel que MySQL.



Développement du back-end:

Introduction:

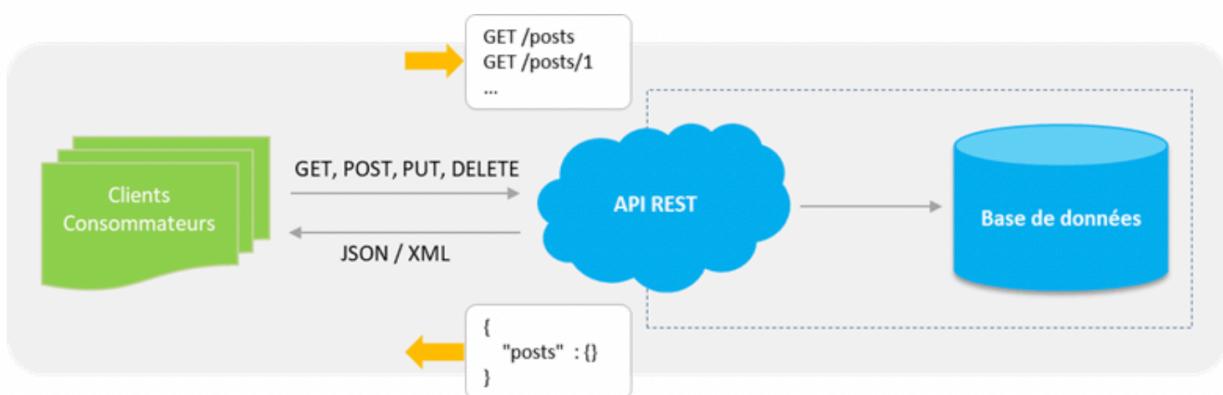
Pour le back-end de mon application mobile, j'ai décidé de créer une API REST. Ayant déjà plusieurs projets en PHP et en Symfony, framework de celui-ci. J'ai décidé d'utiliser API-Platform pour mettre en place mon API REST.

API-Platform:

API-Platform est une distribution Symfony qui permet de créer rapidement et simplement de puissantes API REST.

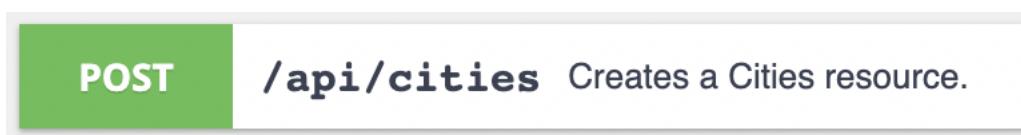
Une API REST est entièrement basé sur le **protocole HTTP** et met à profit l'utilisation des verbes HTTP (GET, POST, PUT, DELETE etc.) qui permettent une compréhension aisée des différentes actions possibles sur une API. API-Platform permet non-seulement de créer rapidement une API REST, mais également sa documentation (**Swagger**).

L'architecture de l'API REST:



L'architecture REST utilise les spécifications originelles du protocole HTTP,

- L'URI comme identifiant des ressources
- Les verbes HTTP comme identifiant des opérations
- Les réponses HTTP comme représentation des ressources
- Un paramètre comme jeton d'authentification



Selon la requête envoyé par le client, l'API va analyser l'URL et la méthode utilisé (GET,POST..) pour ensuite faire appel au contrôleur, qui lui communique avec le modèle pour traiter la requête. Le résultat sera envoyé au format JSON mais il pourrait être également au format XML ou YAML avec un code statut.

Curl

```
curl -X 'GET' \
'https://api.torea-patissier.students-laplateforme.io/api/cities?page=1' \
-H 'accept: application/ld+json'
```

Ci-dessus une requête faite à mon API, on peut y voir différentes informations

- La verbe HTTP utilisé
- Le nom de domaine de l'API
- Le chemin vers la ressource

401

Undocumented Error: response status is 401

Response body

```
{  
  "code": 401,  
  "message": "JWT Token not found"  
}
```

Ci-dessus la réponse reçue au format JSON ainsi que le code status, on peut voir l'absence du Token JWT mais je vais y revenir

Fichiers présents dans mon API:

Entity:

Il représente les entité que j'ai en base de données. API Platform est capable d'exposer automatiquement les entités mappées en tant que **#APIResource** prenant en charge les opérations CRUD.

```
#[ApiResource(
    collectionOperations: ['me' => [ ...
    ],
    itemOperations: [ ...
    ],
    normalizationContext: ['groups' => ['item']])
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[Groups(["item"])]
    #[ORM\Column(type: 'integer')]
    private $id;
```

Ci-dessus une partie du code de mon entité User

Controller:

```
class MeController extends AbstractController
{
    public function __construct(private Security $security)
    {

    }

    public function __invoke()
    {
        $user = $this->security->getUser();
        return $user;
    }
}
```

Ci-dessus, le MeController qui va retourner les informations de l'utilisateur connecté sur la route /api/me de mon API

Sécurité :

L'inconvénient des **API Web publiques** est qu'elles présentent des **risques**. De par leur conception, elles permettent à des **personnes extérieures** d'accéder aux données. Derrière chaque API se cache un endpoint, à savoir le serveur (et ses bases de données) qui répond aux requêtes. Les attaques les plus courantes, et les mesures associées d'atténuation des risques, sont les suivantes (liste non exhaustive):

Failles	Description
Injection SQL :	C'est une injection code malveillants dans un programme. Elles permettent à l'attaquant de prendre le contrôle d'une base de données SQL.
Attaque DDoS (Distributed Denial of Service):	Saturation du réseau en générant un quantité de requêtes massive
MitM (Man in the Middle):	MitM se produit lorsqu'un cybercriminel intercepte le trafic entre deux systèmes communicants et se fait passer pour l'autre auprès de chacun d'eux, agissant ainsi comme un intermédiaire invisible. les attaques MitM peuvent survenir entre le client (application) et l'API.
Credential stuffing :	Utilisation d'informations d'identification volées sur les endpoints d'authentification des API pour obtenir un accès non autorisé.

Afin de contrer ces attaques j'ai pris les mesures suivante:

Hachage de mot passe:

```
$ composer require symfony/password-hasher
```

J'ai eu recours à la `UserPasswordHasherInterface` de `symfony` pour hacher le mot de passe et ainsi sécuriser les mots de passes des utilisateurs en base de données.

J'ai configuré mon fichier `security.yaml` comme indiqué ci-dessous

```
password_hashers:
    Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
    App\Entity\User:
        algorithm: auto
```

```
interface UserPasswordHasherInterface
{
    /**
     * Hashes the plain password for the given user.
     */
    public function hashPassword(PasswordAuthenticatedUserInterface $user, string $plainPassword): string;

    /**
     * Checks if the plaintext password matches the user's password.
     */
    public function isPasswordValid(PasswordAuthenticatedUserInterface $user, string $plainPassword): bool;

    /**
     * Checks if an encoded password would benefit from rehashing.
     */
    public function needsRehash(PasswordAuthenticatedUserInterface $user): bool;
}
```

Ci-dessus le `UserPasswordHasherInterface`

```
if ($data->getPassword()) {
    $data->setPassword(
        $this->userPasswordEncoderInterface->hashPassword($data, $data->getPassword())
    );
}

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

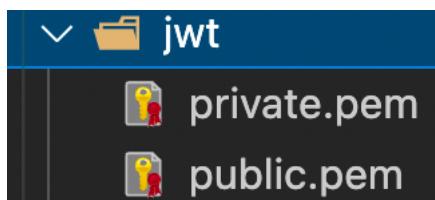
Ci-dessus l'utilisation de la méthode `hashPassword`

TOKEN JWT:

JWT ou JSON Web Token définit un moyen compact et autonome pour transmettre en toute sécurité des informations entre les parties en tant qu'objet JSON. Il permet d'authentifier chaque utilisateurs enregistré dans la base de données de l'API afin de prouver leur légitimité et ainsi leurs permettre de consommer l'API.

Les étapes réalisé:

- **Installation** de LexikJWTAuthenticationBundle (**composer require jwt-auth**)
- j'ai généré les **clés publique et privé** pour signer les jetons JWT



- j'ai **configuré** mon fichier **security.yaml**

```
app_entreprise_provider:
    entity:
        class: App\Entity\Entreprises
        property: email

app_user_provider:
    entity:
        class: App\Entity\User
        property: email
all_users: # Permet d'utiliser 2 User provider sur une même route pour le token JWT
    chain:
        providers: ['app_user_provider', 'app_entreprise_provider']
```

Ci-dessous /authentication_token sera la route pour se connecter et générer le TOKEN

```
main:
    json_login:
        check_path: /authentication_token
        provider: all_users
        username_path: email
        password_path: password
        success_handler: lexik_jwt_authentication.handler.authentication_success
        failure_handler: lexik_jwt_authentication.handler.authentication_failure
```

Ayant une date d'expiration j'ai mis en place le **Refresh Token**, il me permet d'avoir un nouveau token.

Pour cela:

- j'ai exécuté la commande **composer require gesdinet/jwt-refresh-token-bundle**
- j'ai configuré mon fichier **route.yaml**

```
gesdinet_jwt_refresh_token:  
    path: /api/token/refresh
```

- j'ai configuré mon fichier **security.yaml**

```
api_token_refresh:  
    pattern: ^/api/token/refresh  
    stateless: true
```

Cas pratique sur POSTMAN:

The screenshot shows the POSTMAN interface with the following details:

- Method:** POST
- URL:** https://127.0.0.1:8000/authentication_token
- Body:** (JSON)
{"email": "Lina.Orn49@hotmail.com", "password": "12345678"}
- Response Status:** 200 OK
- Response Body:** (Pretty)
{"token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpYXQiOjE2NTYzNTM4MDAsImV4cCI6MTY1NjM1NzQwMCwicm9sZXMiOlisiUk9MRV9VU0VSI10sInVz", "refreshToken": "1c37d238c4a9f119e667fc98ca78aed8493026063e4fe53bd6865d5a76d993b5075d0d8255ae895e52a100c6222895dd876308cf4ea9"}

En rentrant des **identifiants** et un **mot de passe existant** en base de données on voit qu'en retour je reçois en **réponse** un JSON contenant mon **token** et **refreshToken** et un **code statut HTTP 200**.

Firewall:

Le pare-feu est au cœur de la sécurisation l'API.

Chaque demande dans le pare-feu est vérifiée si elle nécessite un utilisateur authentifié ou non. Concernant les routes ou le **roles** est **PUBLIC_ACCESS** alors il est inutile d'être authentifié pour y avoir accès. En revanche pour les routes ou le **roles** est **IS_AUTHENTICATED_FULLY** alors l'utilisateur devra être authentifié, donc connecté.

```
access_control: #important
- { path: ^/docs, roles: PUBLIC_ACCESS } # Allows accessing API documentations and Swagger UI
- { path: ^/authentication_token, roles: PUBLIC_ACCESS }
- { path: ^/api/token/refresh, roles: PUBLIC_ACCESS }
- { path: ^/api/users, method: POST, roles: PUBLIC_ACCESS }
- { path: ^/api/, roles: IS_AUTHENTICATED_FULLY }
```

*Ci-dessus une partie du fichier **security.yaml***

Compétences du REAC validés:

Mettre en place une base de données

Développer des composants dans le langage d'une base de données

Déploiement de l'API sur PLESK:

Introduction:

Après le développement de l'API, il fallait pouvoir l'héberger afin qu'il soit disponible et utilisable par tous. J'ai donc décidé d'utiliser Plesk pour répondre à ce besoin.

Plesk est une interface de gestion de serveur payante, mais j'ai pu avoir un accès gratuit en tant qu'étudiant à La Plateforme.

Mais une API c'est quoi?

En informatique, **API** est l'acronyme d'Application Programming Interface, ou en français, interface de programmation applicative. L'API est une solution informatique qui permet à des applications de communiquer entre elles et de s'échanger mutuellement des services ou des données.

Dans le cas de l'application junior elle me permettra de m'inscrire en tant qu'utilisateur ou encore consulter les offres d'emploi

Cas pratique:

Pour cela j'ai procédé comme suit:

- J'ai ajouté mon **dépôt git** sur plesk

Code location

Remote repository
Your code is hosted online (a cloud service like GitHub, GitLab, or Bitbucket, or your own server).
Plesk will pull code from there.

Local repository
Your code is on the computer to which Plesk wouldn't be able to connect. For example it's your laptop or some server unavailable outside of your LAN.
You will push code to the server with Plesk yourself.

Repository URL *

git@github.com:torea-patissier/API_junior.git

Both HTTP(S) and SSH protocols are supported

SSH public key

```
ssh-rsa
AAAAB3NzaC1yc2EAAAQABAAQACQC3uOq8WyMPDckExgifCwmbEwy
Can5s6qcM4US9Tl+bWj11nnQ8NreDKUzRkKh3HRMKL7ZIS9+ccGFBK3wXn
Dhm2XCEw9G8NyHVlj+KUpSseCpKcqFDns3NNoZT2Etfo4jfgNuPODEINpp
```

This is the public part of the SSH key used for authorization in the remote repository. It must be added to the remote service.

Repository name *

API_junior.git

Specify a name that is unique within a domain.

- J'ai choisi le mode de **déploiement automatique**, cela signifie que toutes les modifications qui ont lieu sur mon dépôt git sont déployées automatiquement.
- Pour la connexion au dépôt git distant j'ai créé une **clé publique SSH**

Le **SSH**, pour **Secure Shell**, désigne à la fois un protocole de communication et un programme informatique. Il permet la connexion d'une machine distante (serveur) via une liaison sécurisée dans le but de transférer des fichiers ou des commandes en toute sécurité.

Deployment settings

Deployment mode *

Automatic Manual Disabled

Files will be deployed to the production site as soon as they are available in the Plesk repository.

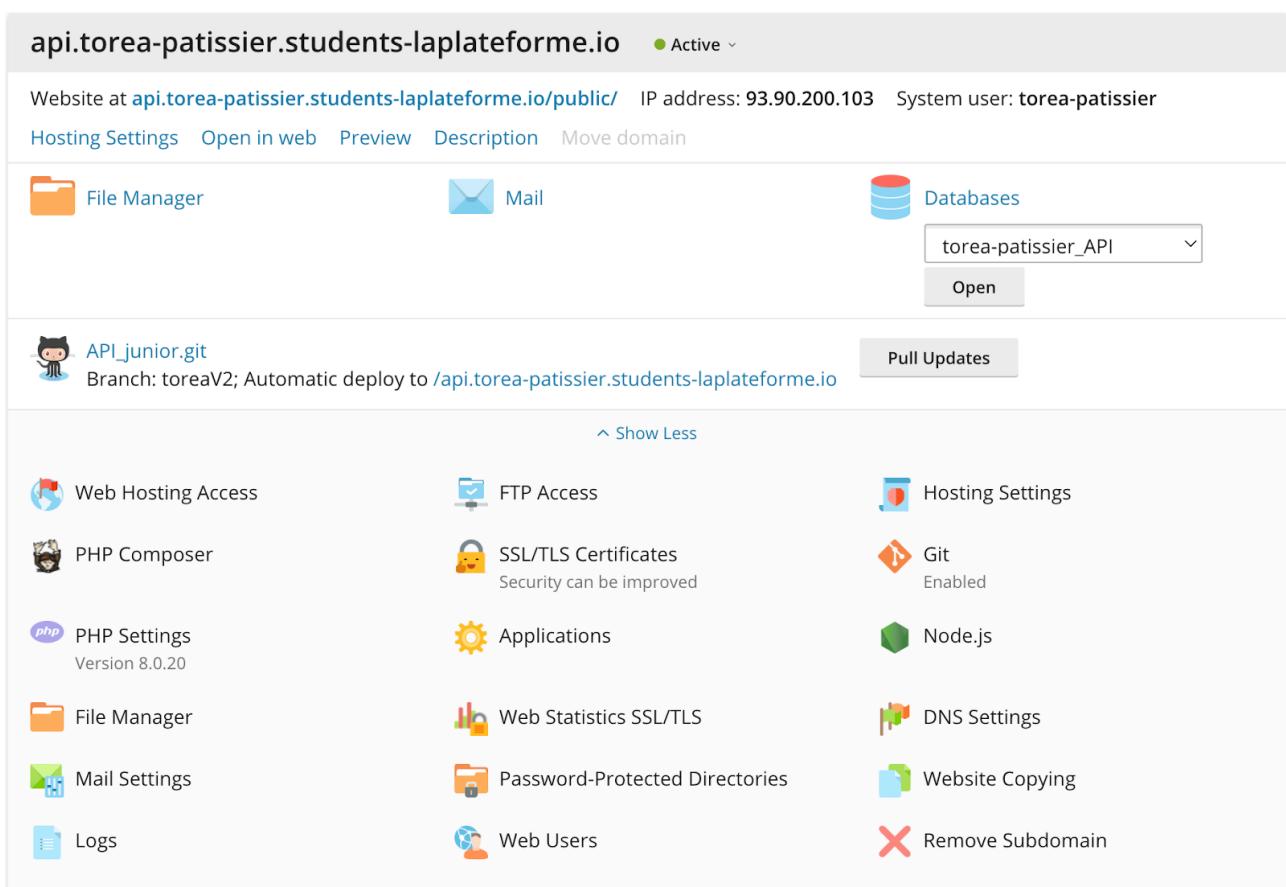
Server path *

/httpdocs 

Directory on the server where files will be deployed.

Enable additional deployment actions

Specify shell commands to be run every time upon deployment.



Website at api.torea-patissier.students-laplateforme.io/public/ IP address: 93.90.200.103 System user: torea-patissier

Hosting Settings Open in web Preview Description Move domain

File Manager **Mail** **Databases** (torea-patissier_API) **Open**

API_junior.git Branch: toreaV2; Automatic deploy to /api.torea-patissier.students-laplateforme.io **Pull Updates**

Show Less

 Web Hosting Access	 FTP Access	 Hosting Settings
 PHP Composer	 SSL/TLS Certificates Security can be improved	 Git Enabled
 PHP Settings Version 8.0.20	 Applications	 Node.js
 File Manager	 Web Statistics SSL/TLS	 DNS Settings
 Mail Settings	 Password-Protected Directories	 Website Copying
 Logs	 Web Users	 Remove Subdomain

Ci-dessus les différentes informations concernant mon API, déployé sur PLESK:

- L'URL de l'API
- Le nom de la Database
- Déploiement automatique
- Le nom de la branch sur laquelle l'API est déployé (toreaV2)

API_junior.git

URL

Branch

Latest commits

- 2022-06-21 17:14 BUG
- 2022-06-21 16:31 me

[: show more](#)

Deployment

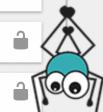
toreaV2 branch automatically to </api.torea-patissier.students-laplateforme.io>



Ci-dessus plus d'informations concernant le dépôt git de mon API:

- Le dernier commit
- La branche sur laquelle l'API est déployé (je peux la modifier)

Entrepris		
GET	/api/entreprises	Retrieves the collection of Entreprises resources.
POST	/api/entreprises	Creates a Entreprises resource.
GET	/api/entreprises/{id}	Retrieves a Entreprises resource.
PUT	/api/entreprises/{id}	Replaces the Entreprises resource.
DELETE	/api/entreprises/{id}	Removes the Entreprises resource.
PATCH	/api/entreprises/{id}	Updates the Entreprises resource.
User		
GET	/api/me	Retrieves the collection of User resources.
POST	/api/register_user	Creates a User resource.
GET	/api/users/{id}	hidden



Ci-dessus mon API déployé et les routes pour les entités Entreprises et User
<https://api.torea-patissier.students-laplateforme.io/api>

Compétence du REAC validé:
Préparer et exécuter le déploiement d'une application