



Scan me

APP CSE

Présentation et guide du projet

AFPA, Conseil CE,
Cordeiro Maxime

TABLE DES MATIÈRES

I.	<u>Présentation</u>	2
1.	<u>Présentation de l'application</u>	2
2.	<u>Design de l'application</u>	3
2.1.	<u>Pages de connexion</u>	4
2.2.	<u>Infos Exclusives</u>	4
2.3.	<u>Tchat avec les experts</u>	5
2.4.	<u>Webinar/Podcast</u>	5
2.5.	<u>Bon plan</u>	6
3.	<u>Parcours Client</u>	6
II.	<u>Réalisation du projet</u>	7
1.	<u>Présentation des technologies utilisées</u>	7
2.	<u>Présentation, installation et configuration des outils</u>	8
2.1.	<u>Node.js</u>	8
2.2.	<u>Visual Studio Code</u>	9
2.3.	<u>Android Studio</u>	10
2.4.	<u>Expo</u>	12
2.5.	<u>Git et Github Action</u>	13
2.6.	<u>MongoDB</u>	14
3.	<u>Création des parties du projet</u>	15
3.1.	<u>Back Office</u>	15
3.2.	<u>Backend</u>	19
3.3.	<u>Application</u>	21
3.3.a.	<u>Le drawer</u>	22
3.3.b.	<u>Navigation</u>	25
3.3.c.	<u>Le Thème + Traduction</u>	26
3.3.d.	<u>Connexion et Inscription</u>	29
3.3.e.	<u>Validation de formulaire</u>	32
III.	<u>Conclusion et remerciements</u>	35
Annexe 1 :	<u>Pourquoi ce projet ?</u>	36

Dans la version numérique de ce document
vous pouvez retourner à la table des matières
grâce à cette icône que vous retrouverez en
haut à gauche de chaque page : 

I. Présentation

1. Présentation de l'application

Conseil CE est un groupe d'experts des CSE, savant mélange d'anciens élus CE, de spécialistes des loisirs, du droit, de la formation et de la communication.

Depuis 20 ans, ils apportent toute leur expérience afin de répondre aux attentes des CSE. Le groupe s'organise autour de 3 principaux pôles d'activité (loisirs - juridique - communication), qui travaillent en synergie afin de faciliter la gestion de votre comité.

Michel Ruiz, Directeur marketing chez Conseil CE, m'a proposé de développer une application Android et iOS pour les élus CSE.

Cette application devrait proposer une source unique d'informations qualifiées, une formation continue des élus par des webinaires, un support gratuit de réponse en ligne avec des experts et pour finir des offres exclusives sur l'achat de produits ou service en lien avec les CSE.



Lors de la réalisation de ce projet j'ai pu acquérir les compétences suivantes :

- Maquetter une application
- Réaliser une interface utilisateur web statique et adaptable
- Développer une interface utilisateur web dynamique
- Réaliser une interface utilisateur avec une solution de gestion de contenu
- Créer une base de données
- Développer les composants d'accès aux données
- Développer la partie back-end d'une application web ou web mobile

2. Design de l'application

Le processus de design d'une application mobile est découpé en 2 parties : le design UX et le design UI. En quelques mots, une fois l'arborescence et les wireframes (maquette) terminés, nous pouvons commencer la phase de design graphique.

La charte graphique

Tout projet a une identité graphique propre. En fonction des projets, le travail de L'UI designer à cette étape est plus ou moins important.

- Certains projets ont une charte graphique établie à laquelle il faudra se plier.
- D'autres projets ont une charte graphique qui pourra être modifiée, adaptée au support numérique. Ici, le travail du designer consiste à trouver le juste milieu d'évolution pour que l'utilisateur fasse toujours un lien entre la marque et le produit.
- Enfin, certains projets nécessitent de concevoir la charte de A à Z, le travail de l'UI designer sera donc important : logo, couleurs, typographie, pictogramme ...

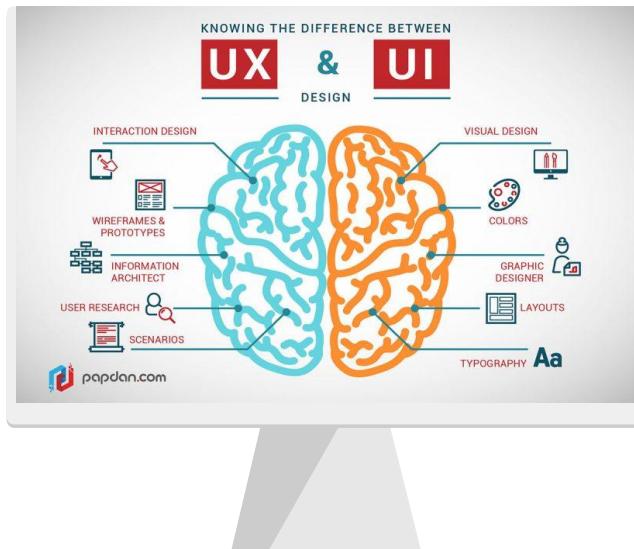
Pour notre projet il nous faudra concevoir la charte graphique, mais en attendant cette charte nous devrons faire la construction d'un prototype architecturé ou preuve du concept (POC), pour montrer la faisabilité de cette application.

Au début du stage j'ai réalisé une maquette de l'application, mais elle n'a pas été retenue car l'aspect de l'application est encore en discussion :

<https://xd.adobe.com/view/cffb96ee-a8b5-4177-7f38-dec5c3376ad4-b9ca/>

Expérience utilisateur (UX)

L'expérience utilisateur est la qualité du vécu de l'utilisateur dans des environnements numériques ou physiques.



Interface utilisateur (UI)

L'interface utilisateur est un dispositif matériel ou logiciel qui permet à un usager d'interagir avec un produit informatique. C'est l'environnement (graphique) dans lequel l'utilisateur interagit.

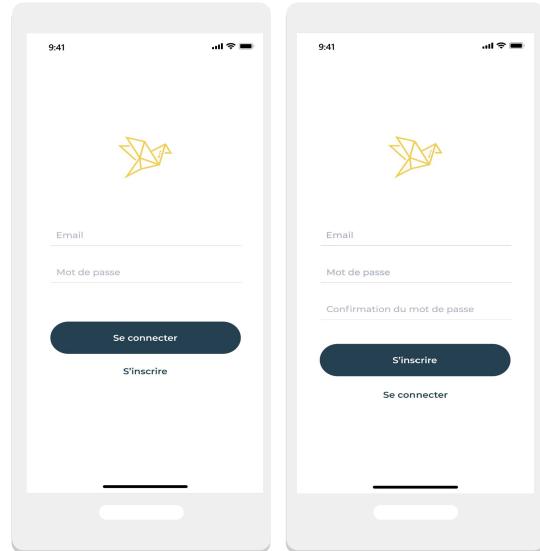
2.1. Pages de connexion

Les pages de connexion sont les premières pages vers l'application, elles représentent le premier contact avec le client. Elles doivent être:

- Belle
- Intuitive
- Facile
- Rapide

De plus deux boutons apparaissent indispensables :

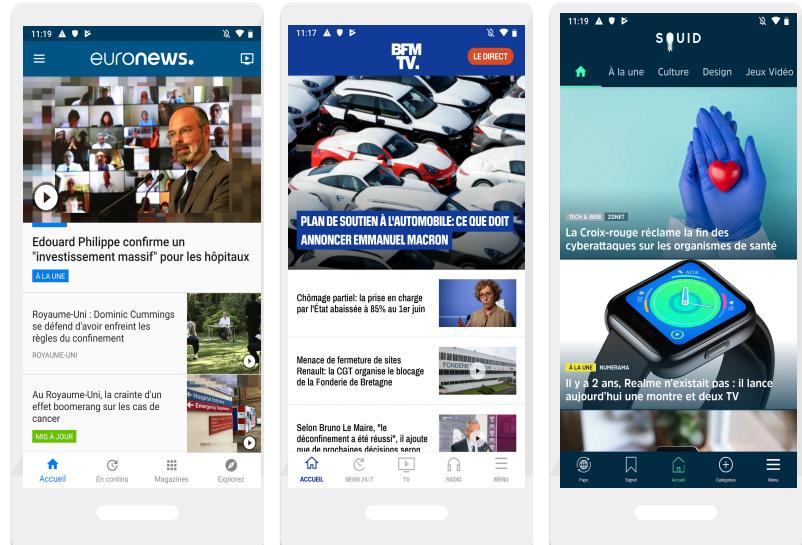
- S'inscrire
- Se connecter



2.2. Info Exclusives

La page Infos Exclusives est la seconde porte de l'application, elle sera la première page parcourue par l'utilisateur.

Les infos faisant office de news ou d'actus elle devra rappeler les différentes applications d'actualités qui existent sur le marché, afin que l'utilisateur puisse être en terrain connu.





2.3. Tchat avec les experts

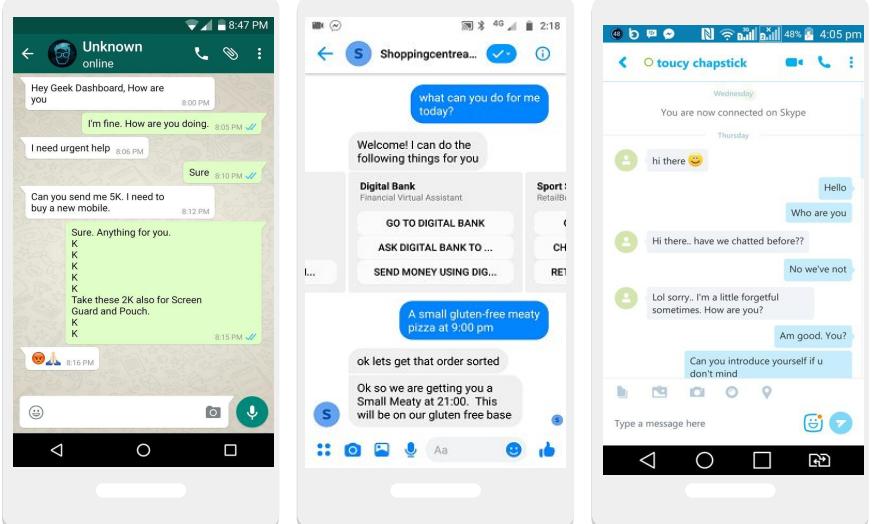
Au niveau du Tchat même contraintes, les systèmes de tchat sont devenus un outil du quotidien.

Le design de ces applications reste souvent, voire toujours, le même.

On peut citer les plus populaires :

- WhatsApp
- Messenger
- Skype

Ces applications ont juste récupéré le design SMS des smartphones pour que les utilisateurs puissent s'en servir instinctivement.



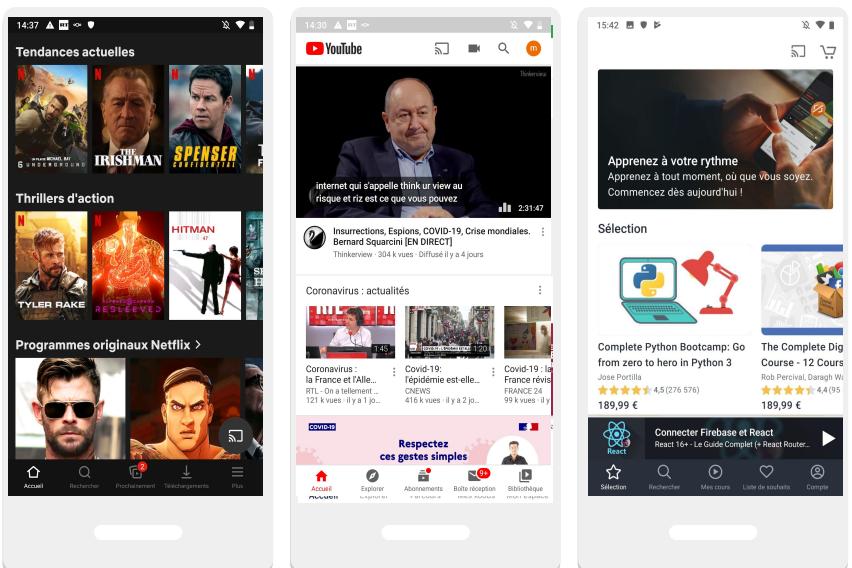
2.4. Webinar/Podcast

La page Webinar/podcast est une page avec du contenu audio et vidéo.

Les applications de ce type sont nombreuses sur le marché et leurs utilisateurs sont donc aussi habitués à un certain schéma de navigation.

On peut retrouver parmi les plus connues :

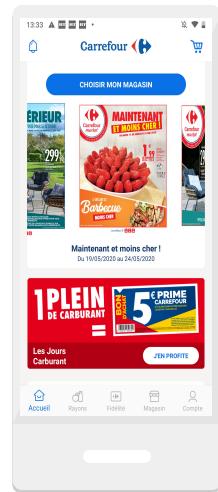
- YoutTube
- Netflix
- Udemy
- Molotov



2.5. Bon plan

Au niveau des Bons Plans, il s'agit ici de publicité, la diversification de la publicité sur téléphone nous permet d'avoir un champ plus grand en matière de design.

On remarquera tout de même qu'une page dédiée avec un design soigné et épuré aura plus d'impact que des publicités intempestives qui pourraient provoquer un effet négatif (irritabilité de l'utilisateur).



3. Parcours Client

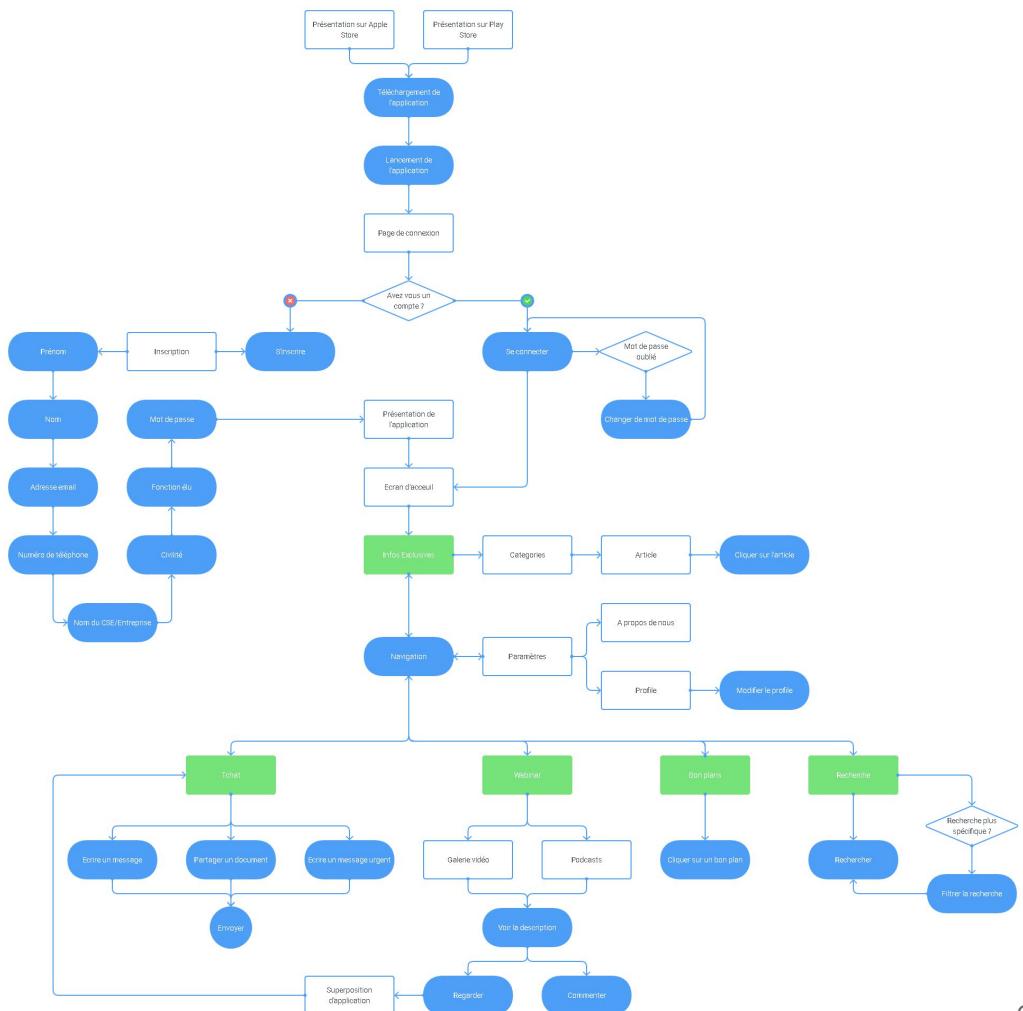
Le parcours client permet d'établir un cheminement dans l'application par ses utilisateurs.

Bien fait, il permet une navigation sereine et intuitive de l'application.

Le parcours client pour applications mobiles doit

lui aussi respecter certaines contraintes :

- Présentation de l'application dans le play store
- Connexion
- Inscription
- Navigation entre les différentes pages



II. Réalisation du projet

1. Présentation des technologies utilisées

LA STACK MERN :

Pour commencer MERN signifie :

- MongoDB,
- Express,
- React,
- Node.js.



La stack MERN est une composition de ces 4 technologies listées ci-dessus. Cette combinaison va permettre aux développeurs de créer des sites web complets (**back-end et front-end**). Avec la MERN stack, on utilise le **JavaScript** côté client et le **Node.js** côté serveur.



React Native est un framework d'applications mobiles open source basé sur **React**, créé par Facebook. Il est utilisé pour développer des applications pour Android, iOS et UWP en permettant aux développeurs réactionnaires d'utiliser React avec les fonctionnalités natives de ces plateformes.



MongoDB est une base de données distribuée, universelle et basée sur des documents, qui a été conçue pour les développeurs d'applications modernes et pour l'ère du Cloud. Aucune autre base de données n'offre une telle productivité.

express

Express est une infrastructure d'applications Web **Node.js** minimaliste et flexible qui fournit un ensemble de fonctionnalités robustes pour les applications Web et mobiles.



React

React est une bibliothèque JavaScript libre développée par Facebook depuis 2013. Le but principal de cette bibliothèque est de faciliter la création d'application web monopage, via la création de composants dépendant d'un état et générant une page HTML à chaque changement d'état.



NodeJS est une plateforme construite sur le moteur JavaScript V8 de Chrome qui permet de développer des applications en utilisant du **JavaScript**. Il se distingue des autres plateformes grâce à une approche non bloquante permettant d'effectuer des entrées/sorties (I/O) de manière asynchrone.

JS

JavaScript est un langage de programmation de scripts principalement employé dans les pages web interactives mais aussi pour les serveurs avec l'utilisation de **Node.js**.



2. Présentation, installation et configuration des outils

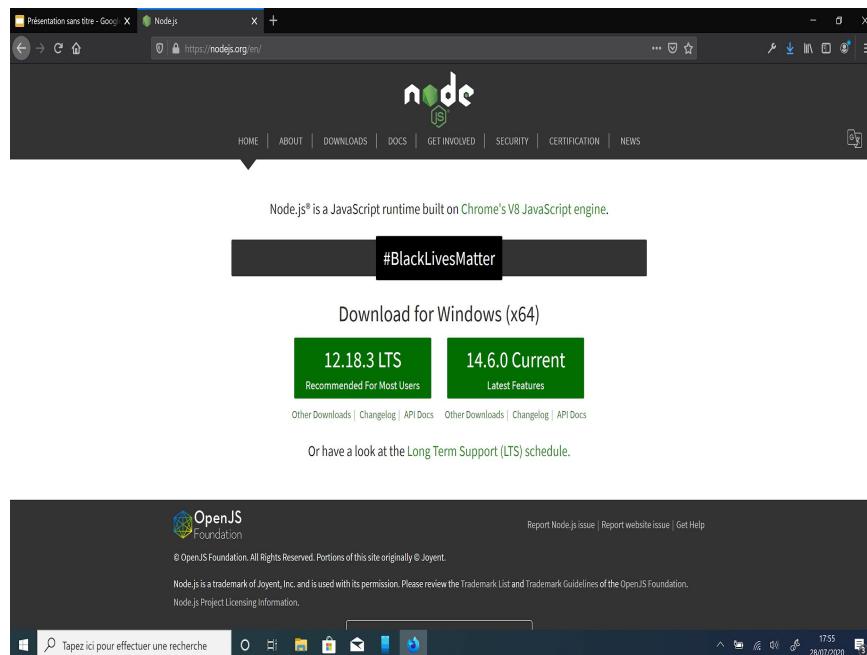
2.1. Node.js

[Node.js](#) est une plateforme de développement Javascript.

Ce n'est pas un serveur, ce n'est pas un framework, c'est juste le langage Javascript avec des bibliothèques

permettant de réaliser des actions comme écrire sur la sortie standard, ouvrir/fermer des connexions réseau ou encore créer un fichier.

Pour l'installer, il suffit d'aller sur le site de [Node.js](#), télécharger l'une des 2 versions proposées (LTS = long-term support) installer en laissant tout par défaut.



2.2. Visual Studio Code

Visual studio code ou VS Code est un éditeur de code développé par Microsoft en 2015.

Il est l'un de ses premiers produits open source et gratuit, et surtout disponible sur tous les systèmes d'exploitation. il est développé avec le framework [Electron](#) et conçu principalement pour développer des projets avec Javascript, Node.js ou encore TypeScript.

Pour l'installer, on se rend sur la page de téléchargement de [VS code](#) puis on installe en laissant les paramètres par défaut.

Nous allons maintenant installer les extensions et configurer VS code pour un meilleur confort d'utilisation.

On ouvre VS code puis on clique sur l'icône des extensions (1).

On va installer la langue on entre "fr" (2) dans la barre de recherche puis on installe. cette extension nécessite un redémarrage de VS code.

De la même façon nous pouvons installer les extensions suivantes :

[StandardJS](#) : Guide de style JavaScript, avec linteur et formateur

[Prettier](#) : permet de mettre en place des règles d'indentation.

[ESLint](#) : Un linteur, il ne s'occupe que de la qualité du code et non du formatage des règles d'indentations.

[Material Theme](#) : Un Thème plus confortable que le thème par défaut.

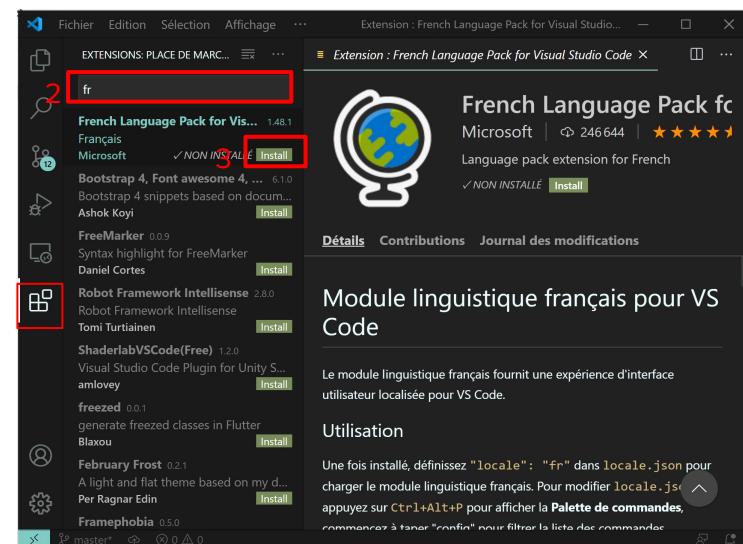
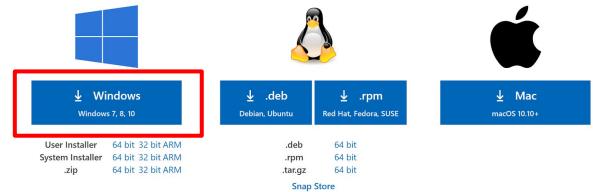
[Material Icon Theme](#) : Un Thème mais pour les icônes.

[Bracket Pair Colorizer](#) : facilite l'identification des crochets.

[ES7 React/Redux/GraphQL/React-Native snippets](#) : permet d'accélérer considérablement le temps d'écriture en générant du code automatiquement à partir d'un modèle prédéfini.

[Wrap Console Log](#) : permet de faire des console.log() depuis un raccourci.

Download Visual Studio Code
Free and built on open source. Integrated Git, debugging and extensions.



Prettier et ESLint nécessite l'installation d'une dépendance dans le projet grâce au gestionnaire de paquets (npm) de Node.js. Vous pourrez trouver une guide de l'installation en annexe.



2.3. Android Studio

Android Studio est un environnement de développement pour développer des applications mobiles Android. Il est basé sur IntelliJ IDEA (un environnement de développement intégré: IDE) et utilise le moteur de production Gradle.

Il propose entre autres des outils pour gérer le développement d'applications multilingues et permet de visualiser rapidement la mise en page des écrans sur des écrans de résolutions variées simultanément. Il intègre par ailleurs un émulateur permettant de faire tourner un système Android virtuel sur un ordinateur.

Pour installer Android studio nous allons nous rendre sur le site :

[Android studio](https://developer.android.com/studio)

durant l'installation laissez tout par défaut

The screenshot shows a web browser window with the URL <https://developer.android.com/studio>. The page is titled "Android Studio". At the top, there are navigation links for "developers", "Platform", "Android Studio" (which is underlined in green), "Google Play", "Jetpack", "Kotlin", "Docs", and "News". There is also a "SEARCH" bar and a "LANGUAGE" dropdown. Below the header, there are sections for "Android Studio", "DOWNLOAD", "WHAT'S NEW", "USER GUIDE", and "PREVIEW". The main content area features the "android studio" logo with a green robot icon. A green button labeled "DOWNLOAD ANDROID STUDIO" is prominently displayed, with a red box drawn around it. Below the button, it says "4.0.1 for Windows 64-bit (871 MB)". At the bottom of the page, there are "DOWNLOAD OPTIONS" and "RELEASE NOTES" sections, along with a Windows taskbar at the very bottom.



2.3. Android Studio

- 1) Tout en bas d'Android Studio, on sélectionne "configure" puis AVD Manager
- 2) On sélectionne un appareil au choix, par défaut nous utiliserons le Pixel 2
- 3) Puis nous allons choisir une version d'android, nous allons télécharger la dernière version d'android,
- 4) Une fois la version d'android téléchargé, on laisse les paramètres par défaut puis on clique sur finish.
- 5) Une fois notre appareil android installé, nous allons pouvoir le lancer depuis l'icône play

The image consists of six screenshots illustrating the process of creating an AVD:

- Screenshot 1:** Welcome to Android Studio screen. The "AVD Manager" option is highlighted in the sidebar.
- Screenshot 2:** Virtual Device Configuration screen. It shows a list of devices under "Choose a device definition". The "Pixel 2" device is selected. A large arrow points from the first screenshot to this one.
- Screenshot 3:** System Image screen. It shows a list of system images under "Select a system image". The "Pixel 2 API 30" image is selected. A large arrow points from the second screenshot to this one.
- Screenshot 4:** SDK Quadrant Installation screen. It displays the "License Agreement" section. The "Accepting the License Agreement" checkbox is checked. A large arrow points from the third screenshot to this one.
- Screenshot 5:** Virtual Device Configuration screen. It shows the "Verify Configuration" section. The "Pixel 2 API 30" device is listed with "Android 10.0+ (Google Play)" selected. A small red box highlights the "Finish" button at the bottom right. A small arrow points from the fourth screenshot to this one.
- Screenshot 6:** Android Virtual Device Manager screen. It shows a table of virtual devices. The "Pixel 2 API 30" device is listed with "Android 10.0+ (Google Play)" selected. A small red box highlights the "Create Virtual Device..." button at the bottom left.



2.4. Expo

Expo: C'est un ensemble d'outils gratuit et open source développé autour de React Native qui permet de gagner du temps dans le développement d'applications iOS et Android.

Il va nous permettre de faire abstraction de la plateforme de développement (XCode pour iOS ou d'Android Studio pour Android), ce qui rend ce projet plus « portable » qu'avec l'ancienne méthode. On a donc un projet pure JS.

En effet expo fournit un SDK qui permet d'accéder aux APIs natives. Il fournit aussi un ensemble de services pour développer ses apps facilement.

Grâce à [Node.js](#) que nous avons installé précédemment nous allons pouvoir utiliser la commande npm, dans un terminal

npm est le gestionnaire de paquets officiel de Node.js, il va nous permettre d'installer des outils "modules ou packages" pour notre projet

Nous allons donc entrer la commande : :

npm install --global expo-cli

explication de la commande :

global : le package est installé de façon globale dans un répertoire accessible dans le répertoire d'installation de Node.js (pour Mac ou Linux doit être installé avec "sudo" devant pour avoir les droits d'administration)

expo-cli : ce package permet d'utiliser Expo dans un terminal

The screenshot shows a Visual Studio Code interface. The left sidebar displays a file tree for a project named 'PROJET_CSE'. The 'test.js' file is open in the editor, containing the following code:

```
function toggle(elemID) {
  var elem = document.getElementById(elemID);

  if (elem.style.display === 'block') {
    elem.style.display = 'none';
  } else {
    elem.style.display = 'block';
  }
  toggle();
}
```

Below the editor, the terminal tab is active, showing the command:

```
C:\Users\max30\Documents\Projets\Projet_CSE>npm install --global expo-cli
```

2.5. Git et Github Action

Git est un système de contrôle de versions, développé en par Linus Torvalds, le créateur bien connu du noyau du système d'exploitation Linux. il fonctionne aussi sur une vaste gamme de systèmes d'exploitations et d'environnements de développement intégrés (IDE).

Par sa structure décentralisée, Git illustre parfaitement ce qu'est un système de contrôle de versions décentralisé (DVCS). Plutôt que de consacrer un seul emplacement pour l'historique complet des versions du logiciel comme c'était souvent le cas dans les systèmes de contrôle de version ayant fait leur temps, dans Git, chaque copie de travail du code est également un dépôt qui contient l'historique complet de tous les changements.

En plus d'être décentralisé, Git a été conçu pour répondre à trois objectifs : performance, sécurité et flexibilité

Pour l'installer on se sur le [site de téléchargement](#), puis on l'installe en laissant tous les paramètres par défaut.



GitHub Actions facilite l'automatisation et permet de créer, tester et déployer votre code directement depuis GitHub.

Pour ça il suffit de créer à la racine du projet un dossier .github → workflows → fichier en .yml

Ici vous pouvez voir un exemple d'un déploiement sur expo depuis GitHub Actions qui publie l'application à l'adresse suivante : <https://expo.io/@mjcc/cse-news> puis build l'application afin de pouvoir la déployer sur le play store.

```
name: Test & Publish on master
on:
  push:
    branches:
      - master
jobs:
  publish:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v1
      - uses: actions/setup-node@v1
      with:
        node-version: 12.x
      - uses: expo/expo-github-action@v5
      with:
        expo-version: 3.x
        expo-username: ${{ secrets.EXPO_CLI_USERNAME }}
        expo-password: ${{ secrets.EXPO_CLI_PASSWORD }}
        expo-packer: npm
      - name: Install and publish
        run: |
          npm ci
          npm test
          expo doctor
          expo publish
          expo build:android -t app-bundle
```



2.6. MongoDB et robo 3T

MongoDB est une base de données créée en 2009 et qui se définit comme une base de données NoSQL. NoSQL parce que vous n'avez pas besoin d'apprendre le langage SQL pour travailler avec MongoDB.

elle adopte une approche différente dans le stockage de données. Contrairement aux autres bases de données comme MySQL, elle stocke les données sous forme de documents avec le format JSON (JavaScript Object Notation)

JSON est un format texte indépendant et qui est facile à lire. Il est déjà utilisé dans la programmation JavaScript.

Avec le format JSON, les données chez MongoDB sont stockées comme des documents.

exemple de fichier json :

```
{
  "espèce": "Dog",
  "race": "Labrador Retriever",
  "couleur": "Yellow",
  "âge": 6
}
```

Ici nous installerons la version MongoDB Community Server, pour ce faire nous allons dans Software ⇒ Community Server ⇒ Download

Robo 3T (anciennement Robomongo) est un outil de gestion multi-plateforme MongoDB libre et open source basé sur un shell.

Contrairement à la plupart des autres outils d'interface utilisateur d'administration MongoDB, Robo 3T incorpore le shell mongo dans une interface à onglets avec l'accès à une ligne de commande shell ainsi qu'à une interaction avec l'interface graphique.

on va installer mongo 3T à la place de "mongoDB Compass" pour gérer la base de données plus facilement.
Pour télécharger l'exécutable il nous demandera une adresse mail, mais nous pouvons entrer une fausse adresse mail.

Simplicity Meets Power

Two products. One download. Double the MongoDB GUI power.

[Download your Double Pack](#)

The most powerful option
Meet Studio 3T, Robo 3T's big brother.

Studio 3T is the professional GUI and IDE for MongoDB available for Windows, Mac, and Linux. Explore and manage your data faster with features like query building, data exploration, aggregation and data comparison, import/export, code generation, and more – with or without the knowledge of the MongoDB query language.

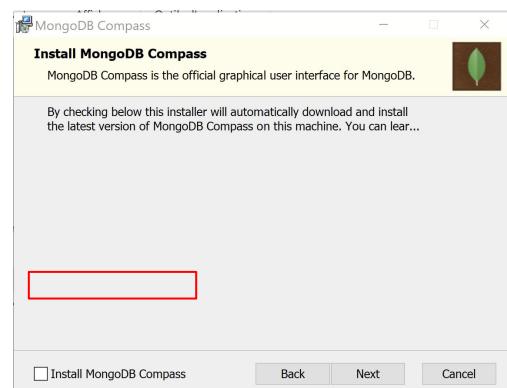
[Download Studio 3T](#)

The latest version
Robo 3T (formerly Robomongo)

Robo 3T 1.8 brings you support for MongoDB 4.0 and SCRAM-SHA-256, an upgraded mongo shell, support for importing from MongoDB SRV connection strings, among many other fixes and improvements.

[Download Robo 3T](#)

puis on installe MongoDB en n'oubliant pas de décocher "Install MongoDB Compass"





3. Crédit des parties du projet

3.1. Back Office

Nous allons créer une application web CRUD (create, read, update et delete) avec React, ce projet est divisé en plusieurs sections:

1. Mise en place du projet
2. Ajout de la table users
3. Ajout d'un utilisateur
4. Suppression d'un utilisateur
5. Modifier un utilisateur
6. récupérer des utilisateurs depuis une API

1. Mise en place du projet

Nous commençons par créer un projet react:
npx create-react-app backoffice

Puis nous allons dans le dossier backoffice et on supprime tout dans le dossier src sauf App.js, index.js et index.css

Pour le css nous utiliserons Skeleton (Skeleton est un framework CSS léger). On ajoute le style (code ci-dessous) dans index.html situé dans le dossier public:

```
<link rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/skeleton/2.0.4/skeleton.min.css">
```

Ensuite, on convertit App.js en composant fonctionnel et on ajoute le code mentionné ci-dessus (en haut à droite) afin d'établir la structure de base :

```
import React from 'react'
const App = () => {
  return (
    <div className="container">
      <h1>Back Office</h1>
      <div className="row">
        <div className="five columns">
          {editing ? (
            <div>
              <h2>Modifier un utilisateur</h2>
            </div>
          ) : (
            <div>
              <h2>Ajouter un utilisateur</h2>
            </div>
          )}
        </div>
        {loading || !users ? (
          <p>Loading...</p>
        ) : (
          <div className="seven columns">
            <h2>Liste des utilisateurs</h2>
          </div>
        )}
      </div>
    );
  export default App;
```

2. Ajout de la table users

Nous créons un dossier appelé tables et ajoutons un fichier UserTable.js. nous allons ajouter un tableau de base qui boucle sur les utilisateurs.

Nous sommes également en train de déstructurer les propriétés de l'objet afin de ne pas avoir à réécrire la propriété. S'il n'y a aucun utilisateur trouvé, nous afficherons une cellule vide avec du texte.

Ne pas oublier d'importer UserTable dans App.js et ajoutez les utilisateurs comme props (propriétés du composant parent) dans UserTable. pour voir le code, il suffit de voir la page ci-dessous.



3.1. Back Office

```
import React from "react";
const UserTable = (props) => {
  return (
    <table>
      <thead>
        <tr>
          <th>ID</th>
          <th>Email</th>
          <th>Password</th>
          <th>Actions</th>
        </tr>
      </thead>
      <tbody> // si users n'est pas vide alors
        {props.users.length > 0 ? ( // on affiche chaque
utilisateur
          props.users.map((user) => {
            const { _id, email, password } = user;
            return (
              <tr key={_id}>
                <td>{_id}</td>
                <td>{email}</td>
                <td>{password}</td>
                <td> // ici Les boutons qui seront lié au
fonction situé dans App.js grâce à props.
                  <button> Supprimer </button>
                  <button> Modifier </button>
                </td>
              </tr>
            );
          })
        : ( // sinon
          <tr>
            <td colSpan={4}>Aucun utilisateur trouvé</td>
          </tr>
        )
      )
    </tbody>
  );
}
export default UserTable;
```

Puis on crée un dossier formulaires avec un fichier appelé AddUserForm.js avec les fonctions handleChange et handleSubmit

```
import React, { useState } from "react";
import axios from "axios";
import url from "../api";
const AddUserForm = (props) => {
  return (
    <form>
      <label>Email</label>
      <input className="u-full-width" type="text"
        value={user.email} name="email"
        onChange={handleChange}>
      />
      <label>Password</label>
      <input className="u-full-width" type="text"
        value={user.password} name="password"
        onChange={handleChange}>
      />
      <button className="button-primary" type="submit"
        onClick={handleSubmit}>
        Ajouter un utilisateur
      </button>
    </form>
```

3. Ajout d'un utilisateur

Ensuite, nous ajouterons la fonction pour ajouter un utilisateur, d'abord en l'ajoutant dans App.js qui reçoit le nouvel utilisateur du composant AddUser.

```
const addUser = (user) => {
  user._id = users.Length + 1;
  setUsers([...users, user]);
};
```

Cette fonction place un objet contenant un nouvel objet utilisateur dans notre tableau users. Nous allons utiliser notre fonction setUsers définie par useState. En utilisant le “spread operator” (qui permet de décomposer notre tableau), nous conservons le tableau actuel auquel on ajoute le nouvel objet. Pour l'ID nous allons simplement définir en fonction du nombre actuel d'utilisateurs plus un.

Ensuite, nous passerons cette fonction à notre composant AddUserForm:

```
<AddUserForm addUser={addUser} />
```

Pour handleChange, nous déstructurons les propriétés de l'objet event.target. Ensuite, nous définissons nos clés d'objet ([name]) en fonction du champ de saisi utilisé:

```
const handleChange = (e) => {
  const { name, value } = e.target;
  setUser({ ...user, [name]: value });
};
```



3.1. Back Office

Dans handleSubmit, nous empêchons l'actualisation de la page par défaut et vérifions si nos user.email et user.password ont réellement été renseignés, puis nous envoyons les données au serveur grâce à axios

```
const handleSubmit = async (e) => {
  e.preventDefault();
  if (user.email && user.password) {
    handleChange(e, props.addUser(user));
    await axios.post(url, {
      _id: user._id,
      email: user.email,
      password: user.password,
    });
  }
};
```

4. Suppression d'un utilisateur

Nous allons maintenant supprimer un utilisateur en filtrant dans notre tableau, l'utilisateur qui a l'ID que nous voulons supprimer.

dans UserTable.js, on lie le bouton Supprimer à la fonction deleteUser grâce au props du composant parent App.js

```
<button onClick={() => props.deleteUser(_id)}>
```

Dans App.js on crée la fonction deleteUser, nous utiliserons setUsers pour mettre à jour le nouvel état des utilisateurs.

```
const deleteUser = async (_id) => {
  setUsers(users.filter((user) => user._id !== _id));
  await axios.delete(url, {
    headers: {
      Authorization: '123',
    },
    data: {
      _id: _id,
    },
  });
};
```

5. Modifier un utilisateur

Nous allons d'abord ajouter EditUserForm.js dans le dossier forms et l'importer dans App.js.

Dans App.js, nous utiliserons useState pour vérifier si l'utilisateur est en train d'être modifié et pour décider quel utilisateur est en cours de modification:

```
const [editing, setEditing] = useState(false);
const initialUser = { _id: null, email: '', password: '' };
const [currentUser, setCurrentUser] = useState(initialUser);
```

Ensuite, nous ajouterons nos fonctions editUser et updateUser dans App.js:

```
const editUser = (_id, user) => {
  setEditing(true);
  setCurrentUser(user);
};

const updateUser = (newUser) => {
  setUsers(
    users.map((user) => (user._id === currentUser._id ? newUser : user))
  );
  setCurrentUser(initialUser);
  setEditing(false);
};
```

puis on met à jour nos composants.

```
<EditUserForm currentUser={currentUser}
setEditing={setEditing} updateUser={updateUser}>
</EditUserForm>

<UserTable users={users} deleteUser={deleteUser}
editUser={editUser}/>
```

6. récupérer des utilisateurs depuis une API

Créez un nouveau dossier appelé hooks avec un fichier useAsyncRequest.js avec le code suivant:

```
import { useState, useEffect } from "react";
import axios from "axios";
import url from "../api";
const useAsyncRequest = () => {
  const [data, setData] = useState();
  useEffect(() => {
    const fetchData = async () => {
      const result = await axios(url);
      setData(result.data);
    };
    fetchData();
  }, []);
  return [data, loading];
};
export default useAsyncRequest;
```



3.1. Back Office

Nous mettons simplement `async` devant notre fonction, puis nous pouvons utiliser `await` pour exécuter uniquement les lignes de code ci-dessus, lorsque cette ligne est terminée. Nous enregistrons le résultat. On place cela dans notre `useEffect` de `App.js` et récupérer les données du composant.

```
const [data, loading] = useAsyncRequest();
```

```
useEffect(() => {
  if (data) {
    const formattedUsers = data.map(obj) => {
      return {
        _id: obj._id,
        email: obj.email,
        password: obj.password,
      };
    });
    setUsers(formattedUsers);
  }
}, [data]);
```

Back Office

Ajouter un utilisateur

Email	<input type="text"/>
Password	<input type="password"/>
AJOUTER UN UTILISATEUR	

Liste des utilisateurs

ID	Email	Password	Actions
1	test@test.fr	azertyuiop	SUPPRIMER MODIFIER
2	test2@test.fr	bfdssh	SUPPRIMER MODIFIER
3	test1@test.fr	azertyuiop	SUPPRIMER MODIFIER

Voici le rendu final avec la récupération des données depuis notre serveur Backend que nous verrons dans la partie suivante.

Ici vous pouvez accéder au Back Office et Backend de l'application que nous créerons par la suite. Ces deux parties ont été publiés grâce à GitHub et Heroku, la base de données est quant à elle sur MongoDB Atlas pour plus de portabilité.

<https://back-office-ce-news.herokuapp.com/>

<https://backend-ce-news.herokuapp.com/>

Ici le code source sur GitHub :

<https://github.com/mjcc30/backoffice-cse>

<https://github.com/mjcc30/backend-cse>

3.2. Backend

On va maintenant procéder à l'installation du serveur Backend qui fera le pont entre l'application et la base de données

Pour cela on va créer un dossier backend puis on lancera la commande “npm init” pour initialiser un projet avec Nodejs.

Durant l'initialisation, on nous demandera diverses informations que vous pouvez remplir ou laisser par défaut puis nous entrerons “yes” pour confirmer.

```

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (backend)
version: (0.0.1)
description: backend pour l'application ce-news
git repository:
keywords:
license: (ISC)
About to write to C:\Users\max30\Documents\Projets\package.json:

{
  "name": "backend",
  "version": "0.0.1",
  "description": "backend pour l'application ce-news",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Maxime",
  "license": "ISC"
}

Is this OK? (yes) yes
  
```

Après cela nous installerons les dépendances dont nous aurons besoin pour notre backend à savoir : Mongodb, express, cors, http et body-parser.

Nous lançons donc la commande :

“npm install mongodb express cors http body-parser”.

Puis on crée un fichier index.js dans lequel on mettra les lignes suivante pour initialiser le serveur.

```

const express = require("express");
const http = require("http");
const app = express();
const server = http.createServer(app);
app.get("/", function (req, res) {
  res.send("Hello World!");
});
server.listen(3001, () => {
  console.log("Server actif @port 3001!");
});
```

On peut installer nodemon pour éviter de redémarrer le serveur à chaque modification avec “npm i -D nodemon” en développement ou en global avec “npm i -g nodemon”.

On a déjà notre base de données installé avec un utilisateur créé, nous pouvons donc récupérer les données depuis le serveur

```

const mongo = require("mongodb").MongoClient;
const url =
  "mongodb://admin:admin@localhost:27017/myDatabase";
mongo.connect(url, (err) => {
  console.log("Connecter à myDatabase!");
});
```

Ensuite on va modifier la fonction get afin de se connecter à la base de données puis récupérer, dans un tableau le contenu de la collection users, où se trouve tout nos utilisateurs:

```

app.get("/", function (req, res, next) {
  mongo.connect(url, function (err, database) {
    const db = database.db("myDatabase");
    const collection = db.collection("users");
    collection.find({}).toArray((x, result) => {
      res.send(result);
    });
  });
});
```

On affiche le résultat grâce à l'url “<http://localhost:3000/>”

▼ 0:	
_id:	1
email:	"test@test.fr"
password:	"azertyuiop"
▼ 1:	
_id:	2
email:	"test2@test.fr"
password:	"bfdfssh"
▼ 2:	
_id:	3
email:	"test1@test.fr"
password:	"azertyuiop"



3.2. Backend

On peut maintenant faire la même chose pour les autres fonctions du CRUD avec post, delete et put.

Pour la fonction post, nous insérons juste les données récupérées (req.body) avec insertOne

```
app.post("/", function (req, res, next) {
  mongo.connect(url, function (err, database) {
    const db = database.db("myDatabase");
    const collection =
db.collection("users");
    let something = {
      _id: req.body._id,
      email: req.body.email,
      password: req.body.password,
    };
    collection.insertOne(something, (x,
result) => {
      res.send(result);
    });
  });
});
```

Pour la fonction delete, elle est identique à la fonction post, sauf que nous avons besoin que de l'id pour trouver l'utilisateur puis nous le supprimons avec deleteOne.

```
app.delete("/", function (req, res, next) {
  mongo.connect(url, function (err, database) {
    const db = database.db("myDatabase");
    const collection =
db.collection("users");
    let something = {
      _id: req.body._id,
    };
    collection.deleteOne(something, (x,
result) => {
      res.send(result);
    });
});
```

Pour la fonction put, comme la fonction delete nous avons besoin de l'id pour trouver l'utilisateur puis nous remplaçons avec findOneAndUpdate par les données récupérées (req.body).

```
app.put("/", function (req, res, next) {
  mongo.connect(url, function (err, database) {
    const db = database.db("myDatabase");
    const collection =
db.collection("users");
    let something = {
      _id: req.body._id,
    };
    collection.findOneAndUpdate(
      something,
      {
        $set: {
          email: req.body.email,
          password: req.body.password,
        },
      },
      (x, result) => {
        res.send(result);
      }
    );
  });
});
```

3.3. Application

nous allons pouvoir initier notre application
 En exécutant la commande:
 “expo init cse-news”

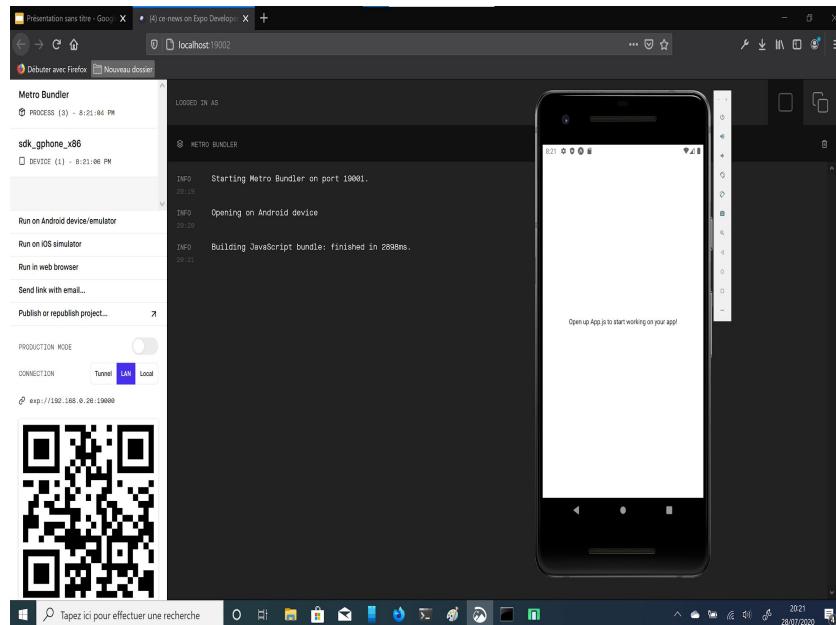
Expo nous propose plusieurs choix, nous voulons créer une simple application vierge
 nous sélectionnons donc le premier “blank”

une fois le projet créé nous pouvons lancer la commande “expo start” pour lancer notre application

une page web s'ouvre avec plusieurs choix.
 nous voulons simuler l'application sur notre ordinateur.

Pour ce faire nous allons utiliser Android studio qui nous permettra d'émuler un appareil android

puis lancer la simulation en sélectionnant “Run on android device/emulator”



Tout d'abord nous allons créer la structure de l'application en commençant par installer [React Navigation](#) en s'aidant de la page Getting started, on se positionne dans le dossier ce-news puis on ouvre un terminal, on lance la commande :

npm install @react-navigation/native
 puis les dépendances pour un projet expo :

expo install react-native-gesture-handler react-native-reanimated react-native-screens
 react-native-safe-area-context @react-native-community/masked-view, on installe la stack
 npm install @react-navigation/stack puis on colle tout le code fourni dans l'exemple de
<https://reactnavigation.org> (ou [ici](#)) et on le remplace dans app.js



3.3. Application

3.3.a. Le drawer

On va ensuite créer le [drawer](#).

Le *navigation drawer* est une méthode de navigation employée couramment par les applications. Elle consiste essentiellement d'un menu latéral caché que l'utilisateur peut invoquer en glissant son doigt du bord gauche de l'écran vers la droite, ou en touchant/cliquant sur une icône dans le coin supérieur droit de son application.

`npm install @react-navigation/drawer`

On importe le drawer dans App.js puis on crée une variable pour utiliser le drawer.

```
import { createDrawerNavigator } from
'@react-navigation/drawer';
const Drawer = createDrawerNavigator()
```

Ensuite on crée un dossier "src" à la racine de l'application puis dans ce dossier, un dossier screens, dans ce dossier on crée les pages suivantes :

- MainTabScreen.js
- SupportScreen.js
- ProfileScreen.js
- SettingScreen.js
- BookmarkScreen.js

Dans App.js on importe les pages de notre drawer

```
import MainTabScreen from './src/screens/MainTabScreen';
import SupportScreen from './src/screens/SupportScreen';
import ProfileScreen from './src/screens/ProfileScreen';
import SettingScreen from './src/screens/SettingScreen';
import BookmarkScreen from './src/screens/BookmarkScreen';
```

Puis on remplace <Stack.Navigator> par notre drawer avec nos pages ci-dessous :

```
<Drawer.Navigator drawerContent={props =>
<DrawerContent {...props} />>
  <Drawer.Screen name="HomeDrawer"
component={MainTabScreen} />
  <Drawer.Screen name="SupportScreen"
component={SupportScreen} />
  <Drawer.Screen name="ProfileScreen"
component={ProfileScreen} />
  <Drawer.Screen name="SettingScreen"
component={SettingScreen} />
  <Drawer.Screen name="BookmarkScreen"
component={BookmarkScreen} />
</Drawer.Navigator>
```

```
import * as React from 'react';
import { View, Text } from 'react-native';
import { NavigationContainer } from
'@react-navigation/native';
import { createStackNavigator } from
'@react-navigation/stack';
function HomeScreen() {
  return (
    <View style={{ flex: 1, alignItems: 'center',
justifyContent: 'center' }}>
      <Text>Home Screen</Text>
    </View>
  );
}
const Stack = createStackNavigator();
function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen name="Home"
component={HomeScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
export default App;
```



3.3. Application

3.3.a. Le drawer

Il nous reste plus qu'à créer la structure de notre drawer.

on aura besoin d'icône, on installe donc react-native-paper avec expo :

```
expo install react-native-paper
```

On crée dans le dossier screens un fichier DrawerContent.js, on colle le code de la page suivante :

```
import React from 'react';
import { View, StyleSheet } from 'react-native';
import { Avatar, Title, Caption, Drawer, Text,
TouchableRipple, Switch } from 'react-native-paper';
import { DrawerContentScrollView, DrawerItem } from '@react-navigation/drawer';
import { MaterialCommunityIcons as Icon } from '@expo/vector-icons';
export function DrawerContent (props) {
  return (
    <View style={{flex:1}}>
      <DrawerContentScrollView {...props}>
        </DrawerContentScrollView >
        <Drawer.Section
style={styles.bottomDrawerSection}>
          <DrawerItem
            icon={({color, size}) => (
              <Icon name="exit-to-app"
color={color} size={size} />
            ) }
            label="Se déconnecter"
          />
        </Drawer.Section >
      </View>
  );
}
```

puis entre <DrawerContentScrollView> : on crée une view

Dans cette view il y aura 3 parties :

- L'avatar
- Les préférences
- Les pages du drawer

L'avatar

```
<View style={styles.userInfoSection}>
  <View style={{ flexDirection: "row", marginTop: 15 }}>
    <Avatar.Image
      source={{ uri:
"https://apiADORABLE.io/avatars/50/abott@adorable.png" }}
      size={50}
    />
    <View style={{ marginLeft: 15, flexDirection: "column" }}>
      <Title style={styles.title}>John Doe</Title>
      <Caption
style={styles.caption}>@j_doe</Caption>
    </View>
  </View>
</View>;
```

Les préférences

```
<Drawer.Section title="Preferences">
  <TouchableRipple>
    <View style={styles.preference}>
      <Text>Dark Theme</Text>
      <View pointerEvents="none">
        <Switch />
      </View>
    </View>
  </TouchableRipple>
</Drawer.Section>;
```



3.3. Application

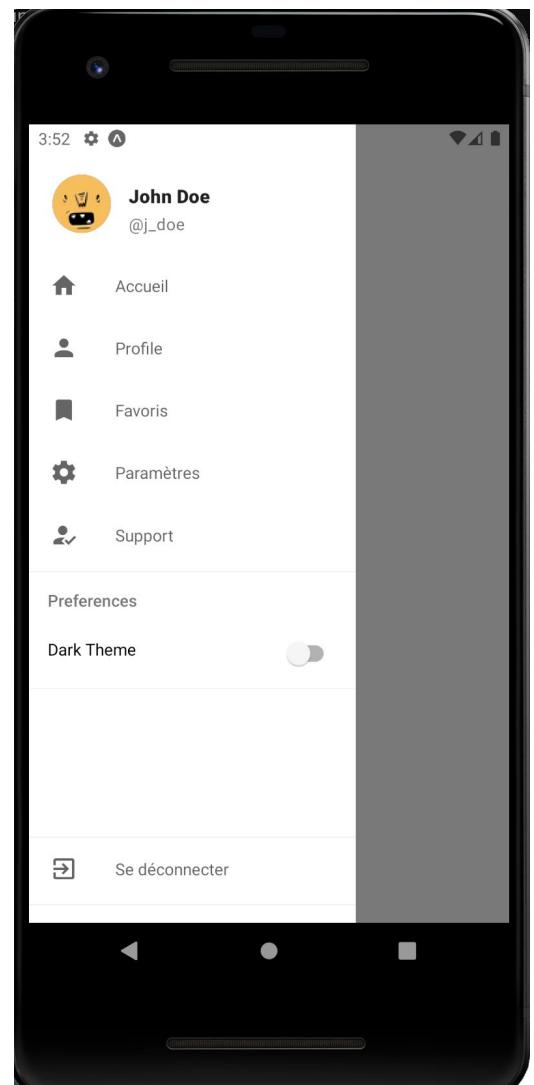
3.3.a. Le drawer

- Les pages du drawer

```
<Drawer.Section style={styles.drawerSection}>
  <DrawerItem
    icon={({ color, size }) => <Icon name="home" color={color} size={size} />}
    label="Accueil"
    onPress={() => {
      props.navigation.navigate("Home");
    }}
  />
  <DrawerItem
    icon={({ color, size }) => (
      <Icon name="account" color={color} size={size} />
    )}
    label="Profile"
    onPress={() => {
      props.navigation.navigate("ProfileScreen");
    }}
  />
  <DrawerItem
    icon={({ color, size }) => (
      <Icon name="bookmark" color={color} size={size} />
    )}
    label="Favoris"
    onPress={() => {
      props.navigation.navigate("BookmarkScreen");
    }}
  />
  <DrawerItem
    icon={({ color, size }) => (
      <Icon name="settings" color={color} size={size} />
    )}
    label="Paramètres"
    onPress={() => {
      props.navigation.navigate("SettingScreen");
    }}
  />
  <DrawerItem
    icon={({ color, size }) => (
      <Icon name="account-check" color={color} size={size} />
    )}
    label="Support"
    onPress={() => {
      props.navigation.navigate("SupportScreen");
    }}
  />
</Drawer.Section>;
```

Ensuite il nous reste plus qu'à lier nos pages dans screens à notre drawer dans onPress.

On applique du style puis on observe le résultat.





3.3. Application

3.3.b. Navigation

Maintenant nous allons créer une barre de navigation en bas de l'application pour ce faire on installe “[BottomTabNavigator](#)” :
 npm install @react-navigation/bottom-tabs

On va créer ensuite nos pages principales :

- HomeScreen
- ChatScreen
- InfoScreen
- VisioScreen

Ensuite dans le fichier MainTabScreen.js, on importe nos 4 pages principales qui seront liées à 4 stacks puis on crée une barre de navigation.

On va donc rentrer par le code suivant.

```
import React from "react";
import { createBottomTabNavigator } from
"@react-navigation/bottom-tabs";
import { createStackNavigator } from
"@react-navigation/stack";
import { Ionicons } from "@expo/vector-icons";
import HomeScreen from "./HomeScreen";
import ChatScreen from "./ChatScreen";
import InfoScreen from "./InfoScreen";
import VisioScreen from "./VisioScreen";
const Home = createStackNavigator();
const Chat = createStackNavigator();
const Visio = createStackNavigator();
const Info = createStackNavigator();
const Tab = createBottomTabNavigator();
```

on crée la fonction MainTabScreen dans laquelle on va mettre la barre de navigation et ses stacks qui seront liées à nos pages.

```
const MainTabScreen = () => (
<Tab.Navigator initialRouteName="Home" activeColor="#fff">
  <Tab.Screen name="Acceuil" component={HomeStack}
    options={{
      tabBarLabel: "Home",
      tabBarColor: "#009387",
      tabBarIcon: ({ color }) => (
        <Ionicons name="ios-home" color={color} size={26} />
      ),
    }}
  />
  <Tab.Screen name="Chat" component={ChatStack}
    options={{
      tabBarLabel: "Chat",
      tabBarColor: "#009387",
      tabBarIcon: ({ color }) => (
        <Ionicons name="ios-chatboxes" color={color} size={26} />
      ),
    }}
  />
  <Tab.Screen name="Visio" component={VisioStack}
    options={{
      tabBarLabel: "Visio",
      tabBarColor: "#009387",
      tabBarIcon: ({ color }) => (
        <Ionicons name="ios-images" color={color} size={26} />
      ),
    }}
  />
  <Tab.Screen name="Infos" component={InfoStack}
    options={{
      tabBarLabel: "Infos",
      tabBarColor: "#009387",
      tabBarIcon: ({ color }) => (
        <Ionicons name="ios-megaphone" color={color} size={26} />
      ),
    }}
  />
</Tab.Navigator>
);
export default MainTabScreen;
```

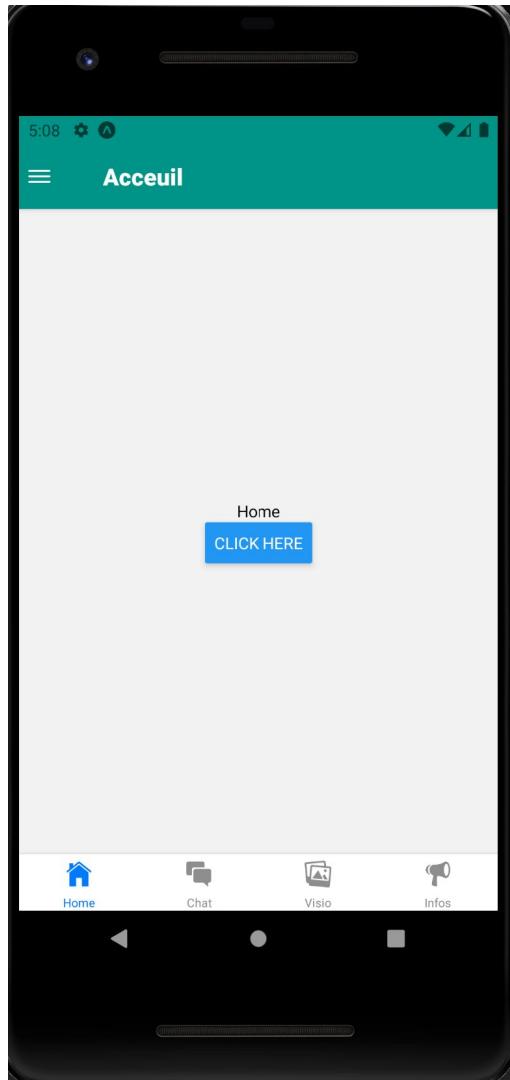


3.3. Application

3.3.b. Navigation

Toujours dans MainTabScreen, on crée nos 4 stacks qui nous permettront d'afficher le bouton du drawer en haut à gauche et le header de la page.

On peut observer le résultat.



```
const HomeStack = ({navigation}) => (
  <Home.Navigator screenOptions={{
    headerStyle: {
      backgroundColor: '#009387',
    }, headerTintColor: '#fff', headerTitleStyle: { fontWeight: 'bold' }
  }}>
  <Home.Screen name="Accueil" component={HomeStackScreen} options={{
    title:'Accueil',
    headerLeft: () => (
      <Ionicons.Button name="ios-menu" size={25}
backgroundColor="#009387" onPress={() => navigation.openDrawer()}>
        </Ionicons.Button>
    )
  }} />
</Home.Navigator>
);
```

3.3.c. Thème

Nous allons maintenant pouvoir appliquer un thème sombre à notre application. pour appliquer l'un thème sur toute l'application nous devrons installer [async-storage](#) : expo install @react-native-community/async-storage

dans App.js on importe async-storage :

```
import AsyncStorage from '@react-native-community/async-storage';
```

Puis nous utiliserons le Contexte (définition de contexte dans la prochaine page, traduction de <https://reactjs.org/docs/context.html>) :

dans le dossier src on crée un dossier components dans lequel on crée un fichier context.js qui contiendra les lignes suivantes :

```
import React from "react";
export const AuthContext = React.createContext();
```



3.3. Application

3.3.c. Thème

Traduction de la Docs de React.

Page : <https://reactjs.org/docs/context.html>

Context provides a way to pass data through the component tree without having to pass props down manually at every level.

In a typical React application, data is passed top-down (parent to child) via props, but this can be cumbersome for certain types of props (e.g. locale preference, UI theme) that are required by many components within an application. Context provides a way to share values like these between components without having to explicitly pass a prop through every level of the tree.

When to Use Context

Context is designed to share data that can be considered “global” for a tree of React components, such as the current authenticated user, theme, or preferred language. For example, in the code below we manually thread through a “theme” prop in order to style the Button component:

```
class App extends React.Component {
  render() {
    return <Toolbar theme="dark" />;
  }
}

function Toolbar(props) {
  <div>
    <ThemedButton theme={props.theme} />
  </div>
}

class ThemedButton extends React.Component {
  render() {
    return <Button theme={this.props.theme} />;
  }
}
```

Le Contexte offre un moyen de faire passer des données à travers l’arborescence du composant sans avoir à passer manuellement les props (propriétés) à chaque niveau.

Dans une application React typique, les données sont passées de haut en bas (du parent à l’enfant) via les props, mais cela peut devenir lourd pour certains types de props (ex. les préférences régionales, le thème de l’interface utilisateur) qui s’avèrent nécessaires pour de nombreux composants au sein d’une application. Le Contexte offre un moyen de partager des valeurs comme celles-ci entre des composants sans avoir à explicitement passer une prop à chaque niveau de l’arborescence.

Quand utiliser le Contexte

Le Contexte est conçu pour partager des données qui peuvent être considérées comme « globales » pour une arborescence de composants React, comme l’utilisateur actuellement authentifié, le thème, ou la préférence de langue. Par exemple, dans le code ci-dessous nous faisons passer manuellement la prop “theme” afin de styliser le composant Button :



3.3. Application

3.3.c. Thème

Reprendons la création de notre thème dans App.js on importe nos 2 le thème sombre et le thème par défaut de react-native-paper

```
import { Provider as PaperProvider, DefaultTheme as
PaperDefaultTheme, DarkTheme as PaperDarkTheme } from
'react-native-paper';
```

Puis toujours dans App.js on import Context qu'on utilisera comme provider (Chaque objet Context est livré avec un composant React Provider qui permet aux composants d'accéder aux mises à jour du contexte.)

```
import { Context } from './src/components/context';
const [isDarkTheme, setIsDarkTheme] =
React.useState(false);
const theme = isDarkTheme ? CustomDarkTheme :
CustomDefaultTheme;
const context = React.useMemo(() => ({
    toggleTheme: () => {
        setIsDarkTheme( isDarkTheme => !isDarkTheme );
    }
}), []);
```

Ensute pour personnaliser nos deux thèmes, on crée deux variables :

- CustomDefaultTheme
- CustomDarkTheme

```
const CustomDefaultTheme = {
  ...NavigationDefaultTheme,
  ...PaperDefaultTheme,
  colors: {
    ...NavigationDefaultTheme.colors,
    ...PaperDefaultTheme.colors,
    background: "#fffffff",
    text: "#333333",
  },
};
```

```
const CustomDarkTheme = {
  ...NavigationDarkTheme,
  ...PaperDarkTheme,
  colors: {
    ...NavigationDarkTheme.colors,
    ...PaperDarkTheme.colors,
    background: "#333333",
    text: "#fffffff",
  },
};
```

Puis on va englober toute notre navigation dans un provider pour appliquer les thèmes :

```
<Context.Provider
value={context}>
  <PaperProvider
theme={theme}>
    <NavigationContainer
theme={theme}>
      <Drawer.Navigator
drawerContent={(props) =>
<DrawerContent {...props} />}>
        </Drawer.Navigator>
      </NavigationContainer>
    </PaperProvider>
  </Context.Provider>;
```



3.3. Application

3.3.c. Thème

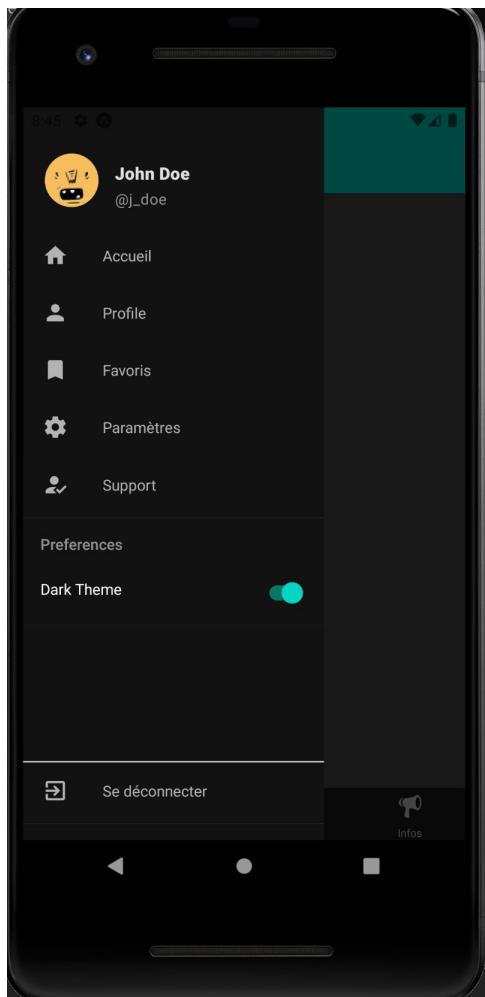
Dans DrawerContent.js, on import useTheme et context, lié à des variables afin de pouvoir les utiliser dans notre composant.

```
import { useTheme, Avatar, Title, Caption, Drawer, Text, TouchableRipple, Switch } from 'react-native-paper';
import { Context } from '../components/context';
const paperTheme = useTheme();
const { toggleTheme } = React.useContext(Context);
```

puis on insère nos deux variables dans le bouton Dark Theme

```
<TouchableRipple onPress={() => {toggleTheme()}}>
<Switch value={paperTheme.dark}/>
```

On observe le résultat :



3.3.d. connexion et d'inscription

Maintenant nous allons créer les pages de connexion et d'inscription.
on commence par créer 4 nouveaux composants dans le dossier screens :

- SplashScreen.js
- SignInScreen.js
- SignUpScreen.js
- RootStackScreen.js



3.3. Application

3.3.d. connexion et d'inscription

Dans RootStackScreen.js, on importe nos trois pages précédentes puis on crée une stack de navigation qui nous permettra de naviguer dans nos trois pages.

```
import React from "react";
import { createStackNavigator } from "@react-navigation/stack";
import SplashScreen from "./SplashScreen";
import SignInScreen from "./SignInScreen";
import SignUpScreen from "./SignUpScreen";
const RootStack = createStackNavigator();
const RootStackScreen = ({ navigation }) => (
  <RootStack.Navigator headerMode="none">
    <RootStack.Screen name="SplashScreen"
      component={SplashScreen} />
    <RootStack.Screen name="SignInScreen"
      component={SignInScreen} />
    <RootStack.Screen name="SignUpScreen"
      component={SignUpScreen} />
  </RootStack.Navigator>
);
export default RootStackScreen;
```

Puis dans App.js on importe donc notre rootStackScreen, on va ensuite mettre nos pages principales en commentaire pour s'occuper des pages de connexion.

```
<NavigationContainer theme={theme}>
  {/*<Drawer.Navigator drawerContent={props =>
    <DrawerContent {...props} />
  </Drawer.Navigator>*/}
  <RootStackScreen />
</NavigationContainer>;
```

On va avoir besoin de deux dépendances supplémentaires :

- react-native-animated
- expo-linear-gradient

npm install react-native-animated
expo-linear-gradient

On crée la fonction SplashScreen dans laquelle on insérera le logo de l'application qui se trouvera dans le dossier assets.

```
const SplashScreen = ({ navigation }) => {
  const { colors } = useTheme();
  return (
    <View style={styles.container}>
      <StatusBar backgroundColor="#009387" barStyle="light-content" />
      <View style={styles.header}>
        <Animatable.Image animation="bounceIn" duraton="1500" source={require("../assets/Logo.png")} style={styles.Logo} resizeMode="stretch" />
      </View>
    </View>
  );
};
```

puis on insère le texte en dessous de l'image avec Animatable.View pour donner un effet au texte.

```
<Animatable.View
  animation="fadeInUpBig"
  style={[styles.footer, { backgroundColor: colors.background }]}
>
  <Text style={[styles.title, { color: colors.text }]}>Bienvenue!</Text>
  <Text style={[styles.text, { color: colors.text }]}>Se connecter</Text>
  <View style={styles.button}>
    <TouchableOpacity onPress={() => navigation.navigate("SignInScreen")}>
      <LinearGradient colors={[ "#08d4c4", "#01ab9d" ]} style={styles.signIn}>
        <Text style={styles.textSign}>Commencer</Text>
        <MaterialIcons name="navigate-next" color="#fff" size={20} />
      </LinearGradient>
    </TouchableOpacity>
  </View>
</Animatable.View>;
```



3.3. Application

3.3.d. connexion et d'inscription

Maintenant on va créer la page de connexion SignInScreen.js avec notre fonction :

Ceci représente la structure de notre page de connexion, il nous reste plus qu'à compléter nos TextInput et ajouter tout le style de notre page.

Pour le code complet voir le code source sur github :
<https://github.com/miccc30/app-cse/blob/master/src/screens/SingInScreen.js>

Pour la page d'inscription "SignUpScreen.js".

Il s'agit du même design on copie tout le contenu de la page SignInScreen.js dans SignUpScreen.js puis on modifie en conséquence en ajoutant un input de confirmation de mot de passe

Dans les prochaines pages, nous verrons la validation du formulaire d'inscription.

```
const SingInScreen = ({ navigation }) => {
  const { colors } = useTheme();
  return (
    <View style={styles.container}>
      <StatusBar backgroundColor="#009387" barStyle="light-content" />
      <View style={styles.header}>
        <Text style={styles.text_header}>Se connecter!</Text>
      </View>
      <Animatable.View animation="fadeInUpBig" style={[styles.footer, { backgroundColor: colors.background }]}>
        <Text style={[styles.text_footer, { color: colors.text }]}>Email</Text>
        <View style={styles.action}>
          <FontAwesome name="user-o" color={colors.text} size={20} />
          <TextInput />
          <Animatable.View animation="bounceIn">
            <Feather name="check-circle" color="green" size={20} />
          </Animatable.View>
        </View>
        <Text style={[styles.text_footer, { color: colors.text, marginTop: 35 }]}>Mot de passe</Text>
        <View style={styles.action}>
          <Feather name="lock" color={colors.text} size={20} />
          <TextInput />
          <TouchableOpacity>
            <Feather name="eye-off" color="grey" size={20} />
          </TouchableOpacity>
        </View>
        <TouchableOpacity>
          <Text style={{ color: "#009387", marginTop: 15 }}>mot de passe oublié?</Text>
        </TouchableOpacity>
        <View style={styles.button}>
          <TouchableOpacity style={styles.signIn} onPress={() => {}}>
            <LinearGradient colors={[ "#08d4c4", "#01ab9d" ]} style={styles.signIn}>
              <Text style={[styles.textSign, { color: "#fff" }]}>Se connecter</Text>
            </LinearGradient>
          </TouchableOpacity>
          <TouchableOpacity onPress={() => navigation.navigate("SignUpScreen")}>
            <Text style={[{ borderColor: "#009387", borderWidth: 1, marginTop: 15 }]}>S'inscrire</Text>
          </TouchableOpacity>
        </View>
      </Animatable.View>
    </View>
  );
};
```



3.3. Application

3.3.e. Validation de formulaire

On importe “useState” et “Alert”.

```
import React, {useState} from 'react';
import { Alert, ScrollView, View, Text, TouchableOpacity, TextInput,
Platform, StyleSheet, StatusBar } from 'react-native';
```

On crée ensuite les 3 variables de nos inputs.

```
const [valEmail, setEmail] = useState('');
const [valPassword, setPassword] = useState('');
const [valConfirmPassword, setConfirmPassword] = useState('');
```

Puis la variable avec nos données de validation.

```
const [data, setData] = useState({
  email: '',
  password: '',
  confirm_password: '',
  check_textInputChange: false,
  secureTextEntry: true,
  isValidUser: true,
  isValidPassword: true,
  isValidConfirmPassword: true,
});
```

On crée ensuite les alertes en cas de champs vides ou mots de passes différents :

```
const onSubmit = (e) => {
  if (valEmail.length == 0 || valConfirmPassword.length == 0) {
    Alert.alert('Attention!', 'email ou mot de passe ne doit pas être vide.', [
      { text: 'OK' },
    ]);
    return;
  }
  if (valConfirmPassword !== valPassword) {
    Alert.alert('Attention!', 'mot de passe doit être le même.', [
      { text: 'OK' },
    ]);
    return;
  }
};
```

On ajoute le paramètre “value” dans les 3 inputs

```
value = { valEmail };
value = { valPassword };
value = { valConfirmPassword };
```

puis la fonction “onSubmit” dans le bouton “s’inscrire” pour tester les alertes

```
<TouchableOpacity style={styles.signIn}
onPress={onSubmit}>
```

On crée 3 fonctions pour remplir les inputs.

```
const textInputChange = (valEmail) => {
  setEmail(valEmail);
};
const handlePasswordChange = (valPassword) => {
  setPassword(valPassword);
};
const handleConfirmPasswordChange =
(valConfirmPassword) => {
  setConfirmPassword(valConfirmPassword);
};
```

Puis on ajoute ces fonctions à chaque input correspondant.

```
onChangeText = { textInputChange };
onChangeText = { handlePasswordChange };
onChangeText = { handleConfirmPasswordChange };
```



3.3. Application

3.3.e. Validation de formulaire

Passons maintenant à la validation de l'adresse mail, on crée un regex :

```
const reg =
/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/;
```

puis dans la fonction `textInputChange` on crée une condition

```
if (reg.test(valEmail) === true) {
  setData({
    ...data,
    checkTextInputChange: true,
    isValidUser: true,
  });
} else {
  setData({
    ...data,
    checkTextInputChange: false,
    isValidUser: false,
  });
}
```

Puis deux conditions pour afficher l'icône de validation et le texte d'erreur

```
{data.checkTextInputChange ? (
  <Animatable.View animation="bounceIn">
    <Feather name="check-circle" color="green" size={20} />
  </Animatable.View>
) : null;
}
{data.isValidUser ? null : (
  <Animatable.View animation="fadeInLeft" duration={500}>
    <Text style={styles.errorMsg}>Email n'est pas valide.</Text>
  </Animatable.View>
)}
```

Une condition pour la validation du mot de passe et de la confirmation, dans les fonctions `handlePasswordChange` et `handleConfirmPasswordChange` :

```
if (valPassword.trim().Length >= 8) {
  setData({
    ...data,
    password: valPassword,
    isValidPassword: true,
  });
  setPassword(valPassword);
} else {
  setData({
    ...data,
    password: valPassword,
    isValidPassword: false,
  });
  setPassword(valPassword);
}
```

Puis pour afficher les messages d'erreur.

```
{data.isValidPassword ? null : (
  <Animatable.View animation="fadeInLeft" duration={500}>
    <Text style={styles.errorMsg}>
      Votre mot de passe doit faire au moins 8 caractères
    </Text>
  </Animatable.View>
)}
```

Il nous manque plus que la sécurité en cachant la saisie des mots de passe et de pouvoir les rendre visibles ou non.

Pour cela on crée deux fonctions

```
const updateSecureTextEntry = () => {
  setData({
    ...data,
    secureTextEntry: !data.secureTextEntry,
  });
}
const updateConfirmSecureTextEntry = () => {
  setData({
    ...data,
    confirm_secureTextEntry: !data.confirm_secureTextEntry,
  });
}
```

3.3. Application

3.3.e. Validation de formulaire

Puis l'affichage des icônes pour les deux inputs, mot de passe et confirmation mot de passe :

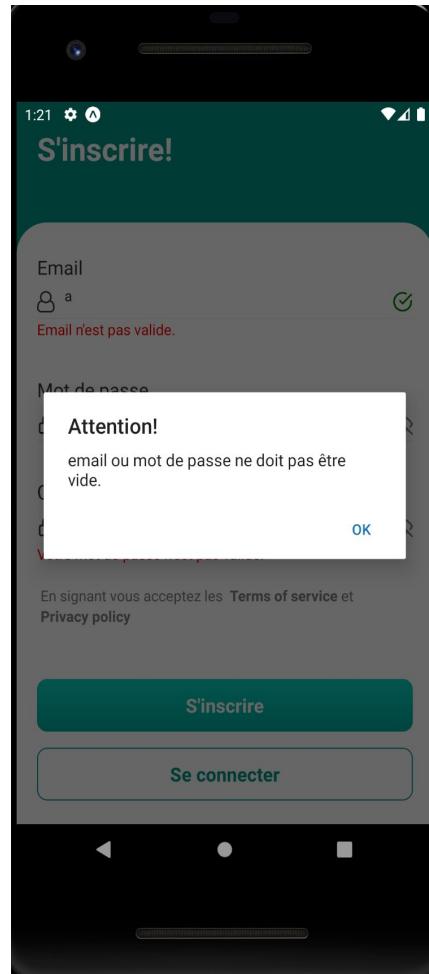
```
<TouchableOpacity onPress={updateSecureTextEntry}>
  {data.secureTextEntry ? (
    <Feather name="eye-off" color="grey" size={20}>
  ) : (
    <Feather name="eye" color="grey" size={20} />
  )}
</TouchableOpacity>;
```

Il ne reste plus qu'à modifier dans les deux TextInput des mots de passes:

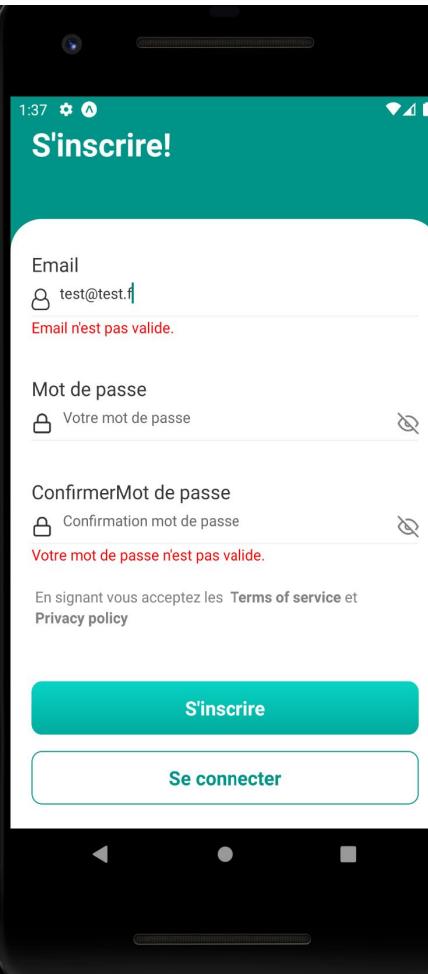
```
secureTextEntry={data.secureTextEntry ? true : false}
```

Puis d'observer le rendu final.

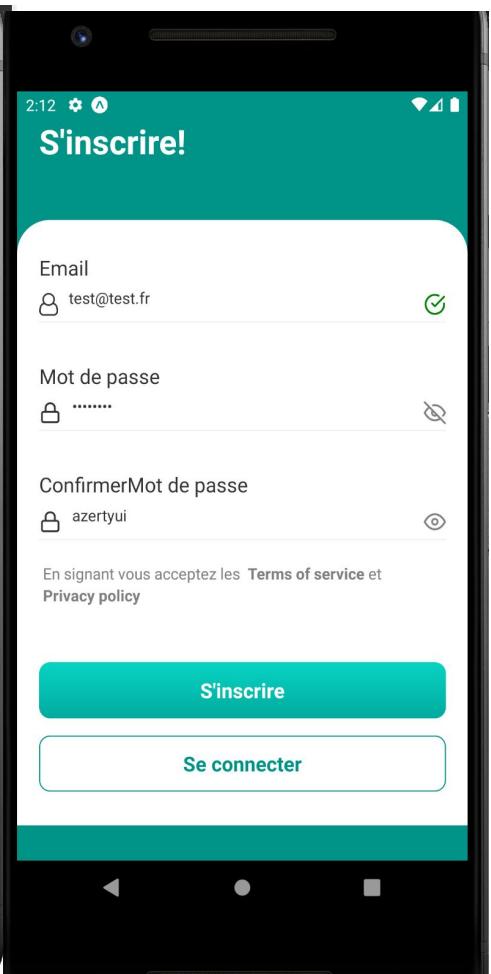
test des alertes



test des messages de validations



test des icônes de validations





III. Conclusion et remerciements

J'arrive donc à la fin de cette présentation de projets où j'ai réalisé:

- Un serveur Backend en nodejs connecter à une base de données MongoDB.
- Un Backoffice en Reactjs contenant une gestion d'utilisateur (API CRUD) connecter au serveur Backend.
- Une application mobile en React Native avec expo connecter au serveur Backend pour se connecter.

Malheureusement nous ne verrons pas (dû aux contraintes liées à la taille du dossier de projet) :

- la réalisation du chat coté backend et dans l'application mobile
- la récupération de contenus via une url et son affichage dans l'application mobile.

Cependant vous pouvez trouver tout le code source et le projet:

le code source accessible sur github :

<https://github.com/mjcc30/backend-cse>
<https://github.com/mjcc30/backoffice-cse>
<https://github.com/mjcc30/app-cse>

Les parties du projet déployés sur heroku et expo :

<https://expo.io/@mjcc/cse-news>
<https://back-office-ce-news.herokuapp.com/>
<https://backend-ce-news.herokuapp.com/>

Je voudrais aussi vous faire part ici, de mon ressenti tout au long de ce projet, mais comme je n'ai assez de place j'ai créé un annexe intitulé "Pourquoi ce projet?", que vous retrouverez dans la page suivante.

Je tiens à remercier d'abord mon formateur Jérôme LESAINT, qui m'a apporté les clés afin de pouvoir travailler en autonomie et de manière autodidacte. Car avant d'apprendre à développer il faut déjà apprendre à apprendre et comprendre.

Je voudrais aussi remercier l'entreprise CONSEILCE qui m'a accueilli avec beaucoup de bienveillance malgré la situation sanitaire et qui m'a permise de réaliser ce projet dont le but premier était de progresser et de m'épanouir dans mon futur métier.

De plus je remercie toute la communauté de développeurs sur internet pour le travail effectué en amont qui m'a permis de répondre à toutes mes questions et de régler les problèmes auxquels j'ai été confronté.

Enfin je vous remercie d'avoir pris le temps de me lire et merci de prendre encore quelques minutes pour lire mon annexe.



Annexe 1 : Pourquoi ce projet ?

Pourquoi ce projet ?

Lorsque Michel Ruiz (Directeur marketing chez Conseil CE) m'a proposé de développer son application, c'est avec plaisir que j'ai accepté.

Quand il m'a demandé sur quelles technologies je voulais travailler, j'ai voulu réaliser ce projet de la conception jusqu'à la production pour me constituer un profil dit « full stack ».

Sur les technologies de la stack MERN, qui sera la base de mon prochain métier grâce à la poursuite de mes études à l'IPSSI, (master 1 "Développeur concepteur d'application") en alternance en tant que développeur fullstack "MERN" chez DOCTEGESTIO.

Au début de ce projet, j'avais énormément de choses à apprendre car je débutais, personnellement j'aimais tout ce que j'apprenais. Je suivais des tutoriels et des cours sur internet, tous intéressants, qui m'ont permis aussi d'améliorer ma compréhension à l'oral en anglais. Puis au fur et à mesure je commençais à voir une progression dans la compréhension du code.

Malgré certains déboires, j'ai pu apprendre à utiliser toutes mes compétences et connaissances pour résoudre mes problèmes. Je pense qu'il n'y a pas de raté ce sont juste des expériences formatrices.

J'ai souvent était cloué devant des écrans à chercher, lire, apprendre des choses, encore et encore jusqu'à y passer la nuit sans m'en rendre compte, je pense que c'est aussi là que se situe mon travail. Certes je produis des lignes et des lignes de codes. Mais je le fais avant tout avec envie et passion...

Le métier de développeur est super, mais les projets sur lesquelles on travaille doivent être encore plus intéressants. Cela va nous donner envie d'apprendre et la qualité de notre travail se fera ressentir.

“Ce que nous devons apprendre à faire, nous l'apprendrons en le faisant.”

Aristote