

Alexandre Leroux

# DOSSIER DE PROJET

---

Réalisation d'une application web

# Sommaire

<b>1.Introduction</b>	<b>4</b>
1.1 Compétences couvertes par le Projet	4
1.2 Résumé du dossier de projet	5
<b>2.Présentations</b>	<b>6</b>
2.1 Présentation Alexandre Leroux	6
2.2 Présentation La Plateforme_	6
2.3 Présentation de l'entreprise : Unistellar	7
2.4 Mon rôle chez Unistellar	8
2.5 Organigramme de l'équipe	8
<b>3.Présentation du projet</b>	<b>9</b>
3.1 Description des besoins	9
3.2 Analyse de l'existant	10
3.3 Environnement de travail	10
3.4 Organisation du travail	11
<b>4.Technologies utilisées</b>	<b>13</b>
4.1 React.js	13
4.2 Redux	13
4.3 Amplify ( AWS )	14

<b>5.Réalisation du projet</b>	<b>14</b>
5.1 Portail de connexion	14
Résumé	14
Architecture	15
Exemple d'un composant	16
5.2 Sécurité de la connexion	19
5.3 Conclusion du portail de connexion	22
5.4 Application utilisateurs ( confidentiel )	23
Résumé	23
Architecture	24
Redux	25
Exemple du code Redux	27
GraphQL	30
Exemple d'un composant	32
5.5 Tests & mise en production	39
Description des tests effectués	39
Test création de compte, connexion et validation du mail	41
Test changement de langage	45
Test changement de password	47
CI/CD AWS	48
Choix des tests à effectuer au déploiement	50
Dashboard Cypress	50
 <b>6.Conclusion</b>	 <b>51</b>

# 1. Introduction

## 1.1 Compétences couvertes par le Projet

Le projet couvre les compétences énoncées ci-dessous.

Activité type 1 « **Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité** » :

- Développer des composants d'accès aux données
- Développer la partie front-end d'une interface utilisateur web

Activité type 2 « **Concevoir et développer la persistance des données en intégrant les recommandations de sécurité** » :

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

Activité type 3 « **Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité** » :

- Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement
- Concevoir une application
- Développer des composants métier
- Construire une application organisée en couches
- Préparer et exécuter les plans de tests d'une application
- Préparer et exécuter le déploiement d'une application

## 1.2 Résumé du dossier de projet

Ce dossier retrace les étapes de la réalisation d'un ensemble d'applications, dédié à un usage interne et externe, réalisées lors de mon alternance dans l'entreprise Unistellar.

Dans un premier temps, une période d'intégration dans l'entreprise a été mise en place. Période durant laquelle ont été effectuées : La prise en main des différents outils, liés au développement ou collaboratifs, la compréhension des besoins de l'entreprise, de leurs produits et des solutions pouvant être apportées, ainsi que la prise en compte des impératifs de la société, des contraintes liées à la logique métier, et des demandes spécifiques de la direction.

Suite à cette période, j'ai pu intégrer pleinement le processus de développement, et des tâches spécifiques liées aux projets m'ont été attribuées ( features, bug fix, spike... ).

Le développement s'est essentiellement fait sur la partie front-end des applications, avec l'utilisation de React.js.

## 2. Présentations

### 2.1 Présentation de Alexandre Leroux

Je m'appelle Alexandre Leroux, j'ai 44 ans, et je vis à Marseille.

J'ai été, durant ces 7 dernières années, photographe professionnel (4 ans en tant que responsable photographie/numérisation aux archives départementales des Hautes-Alpes, et 3 ans en tant que photographe indépendant).

J'ai souhaité faire une reconversion professionnelle fin 2020, et c'est alors que j'ai pu intégrer la coding School, à l'École La Plateforme, pour un cursus de deux ans.

L'année scolaire 2020/2021 a été consacrée à l'apprentissage du code au sein de cette structure ( HTML/CSS, PHP, JS), et cette seconde année s'est déroulée en alternance, autour de différents projets propres à l'école et à l'entreprise.

### 2.2 Présentation de La Plateforme\_

L'école la plateforme est une nouvelle école du numérique, qui vise à former un public venant de tous horizons, tout âge, et sans condition d'accès.

Parmi les différentes formations proposées, celle de la coding school s'oriente sur le développement web et la conception d'applications.

La formation se déroule au rythme des apprenants, la pédagogie est orientée sur une autonomie des étudiants qui doivent rendre des projets permettant d'en débloquent des suivants, demandant plus de compétences. Ainsi, chacun peut évoluer à son rythme, et selon ses capacités/rapidité.

## 2.3 Présentation de l'entreprise : Unistellar

Unistellar est une entreprise récente (2017), née de l'idée d'un de ses fondateurs, Arnaud Malvache, qui, en tant que passionné d'astronomie, se faisait la réflexion que le ciel dit "profond" (nébuleuses, galaxie..Etc) était difficilement accessible avec les instruments optiques disponibles sur le marché.

Fin 2017, leur premier télescope intelligent, l'eVscope; est lancé sur le marché. Il permet de voir en quelques minutes les objets du ciel profond, par un système d'accumulation de lumière, le stacking.

*"La conception du télescope intelligent d'Unistellar permet aux passionnés du ciel du monde entier d'explorer les merveilles de l'Univers depuis chez eux (même dans un ciel avec pollution lumineuse), sans avoir besoin d'être un expert aguerri.*

*Grâce aux technologies exclusives développées par Unistellar (vision amplifiée, reconnaissance de champ, réduction de la pollution lumineuse), l'eVscope 2 est l'instrument spatial grand public le plus puissant et offre une nouvelle expérience conviviale d'exploration du ciel profond. Avec une vitesse et une précision inégalées, le télescope révèle des galaxies, des nébuleuses, des comètes et d'autres objets célestes à des millions d'années-lumière avec des couleurs et des détails sans précédent, même en ville.*

*Les amateurs qui découvrent l'astronomie peuvent profiter d'un nouveau loisir et développer leurs compétences avec un appareil exceptionnel mais convivial, tandis que les astronomes avancés bénéficient d'un appareil facile à transporter et qui peut rapidement passer d'un objet à l'autre, et collecter des données scientifiques précieuses qui aident l'humanité à mieux comprendre l'espace. Les professionnels et les éducateurs peuvent ravir le public avec des vues rapides, nettes et colorées de l'Univers. "*

source : [site internet](#)

## 2.4 Mon rôle chez Unistellar

Il était convenu avant ma prise de poste que je serai affecté à l'équipe front-end.

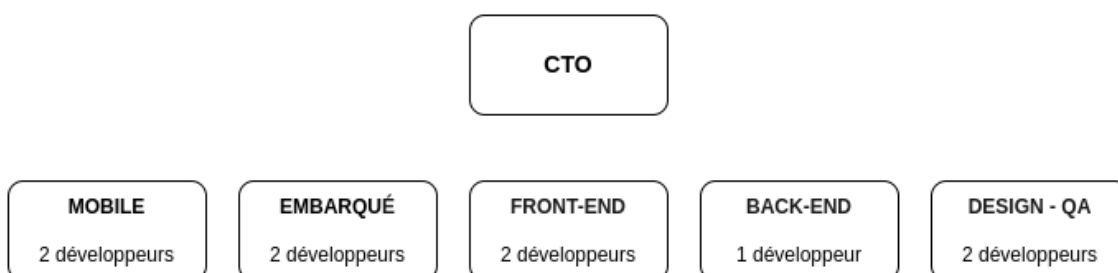
Mon rôle sera alors de participer à l'élaboration de différents projets.

Ces derniers seront à destination des employés d'Unistellar, mais aussi des utilisateurs de l'eVscope.

Quelques notions de back-end seront abordées, mais l'essentiel du développement se fera côté front.

## 2.5 Organigramme de l'équipe

L'équipe software comprend 11 personnes, réparties comme suit :



L'équipe front est donc composée de moi même, ainsi que d'un collègue ayant plusieurs années d'expérience.

Le lead dev de notre équipe est le développeur back-end, avec qui nous travaillons étroitement.



## 3. Présentation du projet

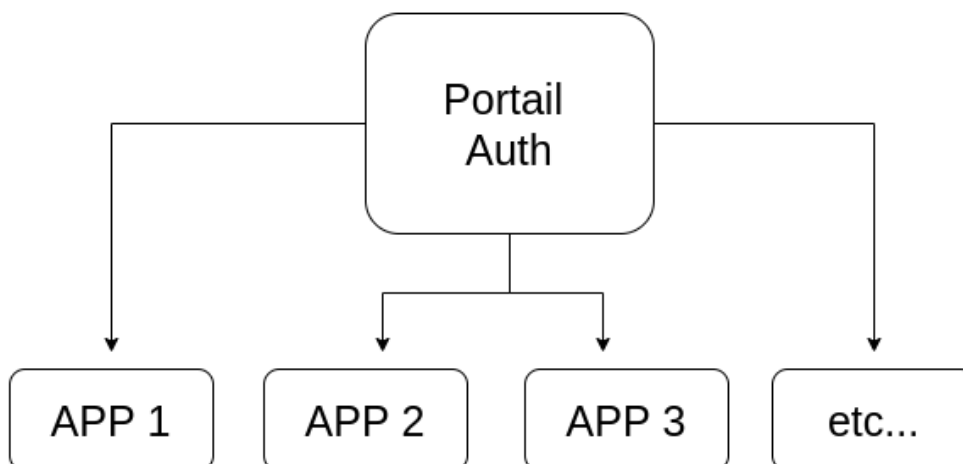
### 3.1 Description des besoins

Unistellar souhaite mettre en place plusieurs applications web, à destination des employés ( application type back-office ) mais aussi des utilisateurs.

Pour des raisons de confidentialité, l'entreprise m'a demandé de ne pas parler explicitement de certains projets.

Je nommerai donc le projet qui ici, me concerne plus particulièrement, comme étant le projet "utilisateurs".

Le concept global de ces réalisations doit s'articuler autour d'un portail de connexion unique, qui permet ensuite de naviguer entre différentes applications, chacune étant hébergée sur son propre sous-domaine, le tout sans avoir besoin de se reconnecter.



## 3.2 Analyse de l'existant

L'ensemble de ce projet démarre de zéro. A mon arrivée, certaines fonctionnalités ont déjà été créées par mon collègue, telles que les bases du portail de connexion.

## 3.3 Environnement de travail

Unistellar fournit à ses employés des pc sous linux (pop-os). L'ensemble des services liés au développement est géré via AWS d'Amazon.

Atlassian est utilisé pour la gestion des projets, et l'aspect collaboratif.

Les différents outils utilisés pour le développement et l'organisation sont les suivants :

- Visual Studio Code
- Git pour le versionning, via CodeCommit d'AWS.
- DynamoDB d'AWS, pour la persistance des données.
- Jira Software pour la gestion des tickets.
- Confluence pour la rédaction de documentation.
- Slack pour la communication interne.

### 3.4 Organisation du travail

L'équipe est donc composée d'un lead dev, et de deux développeurs front-end, dont je fais partie. Le travail est organisé en sprint de deux semaines, et nous faisons une réunion hebdomadaire pour faire le point sur l'avancement des projets.

Des tickets Jira sont attribués pour la répartition des tâches. Jira est également connecté à Slack, ce qui permet d'avoir des alertes via des canaux spécifiques, lorsque les tickets changent d'état.

Il existe 4 états différents pour les tickets :

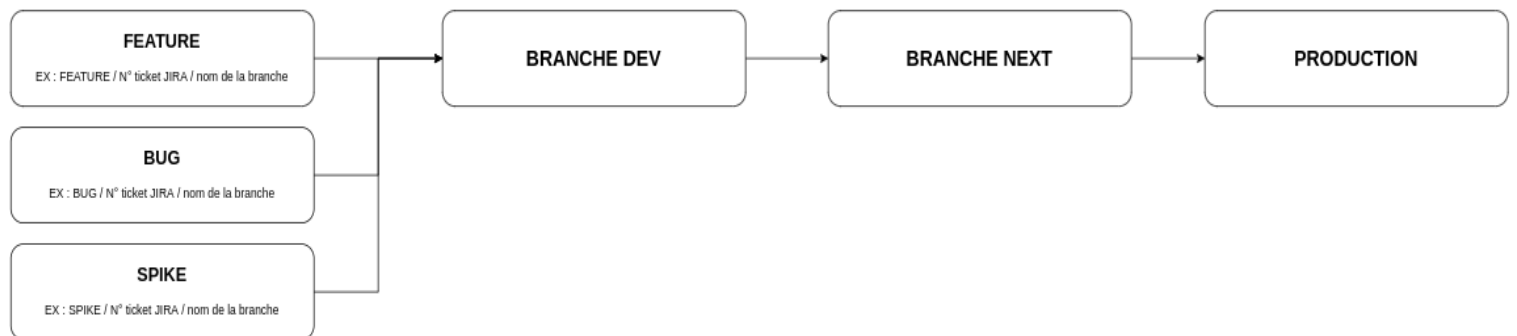
- A faire
- En cours
- In testing
- Fini

Les projets disposent de plusieurs branches de travail :

Les branches sont associées aux tickets Jira, qui décrivent les tâches à effectuer.

Le nommage des branches se fait comme suit :

FEATURE / N° du ticket Jira / nom de la branche



Lorsqu'un ticket est passé en "In testing" par un des développeurs, son collègue fait le code review, pour ensuite faire un merge ou un rebase selon l'état de la branche dev.

## 4. Technologies utilisées

### 4.1 React.js

L'entreprise a fait le choix d'utiliser React.js pour la partie front de ses applications.

React.js est une librairie Javascript, créée par un employé de Facebook, qui permet de créer des sites “single-page”, système qui permet de modifier les données à afficher, sans rechargement de page.

Parmi les atouts de React.js, nous pouvons citer l'utilisation du jsx, une extension syntaxique permettant d'intégrer des éléments semblables à du html, ce qui permet de structurer assez facilement ce que doit être le rendu de notre composant.

Citons également les hooks, présents depuis la version 16.8, qui permettent de gérer un état local plus facilement, le tout sans l'utilisation de classe.

### 4.2 Redux

Redux est une librairie permettant de gérer l'état de notre application, communément appelé le “state”.

Lorsqu'une application devient conséquente, avec beaucoup d'état différents à gérer, l'utilisation de Redux devient intéressante. Elle permet de gérer tout l'état de l'application de façon séparée, dans un “store”, et de le manipuler avec plus de lisibilité.

### 4.3 Amplify ( AWS )

Amplify est une librairie fournie par AWS, permettant de créer rapidement des applications web type fullstack, grâce à l'ensemble des services AWS.

Une seconde librairie, Amplify UI, propose un design système, mettant à disposition tout un ensemble d'interfaces utilisateurs.

## 5. Réalisation du projet

### 5.1 portail de connexion

#### Résumé :

La volonté de l'entreprise est de disposer d'un portail unique de connexion, qui délivrera aux utilisateurs, les autorisations via des tokens jwt.

Ces tokens permettront ensuite de naviguer sur les différentes applications, sans nouvelle connexion.

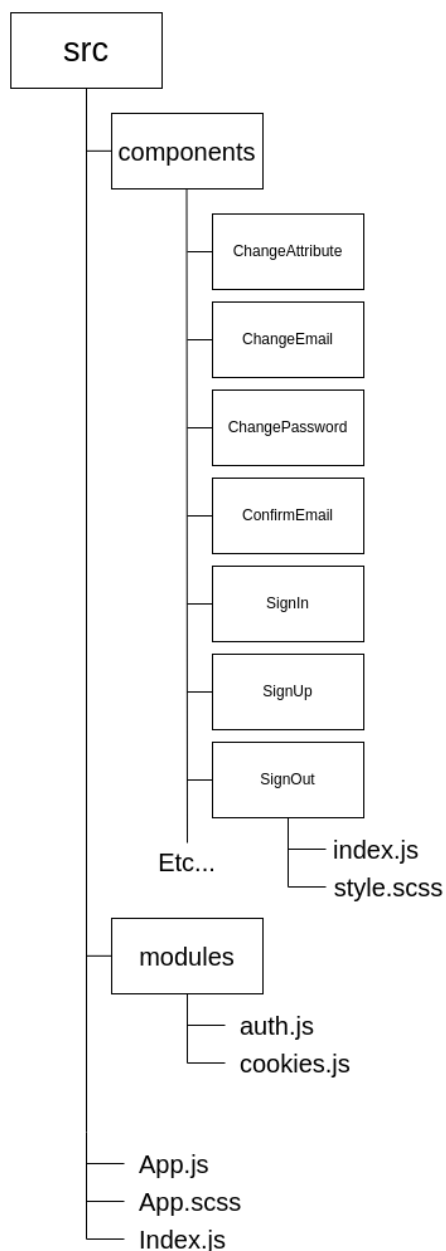
Une des première étape a été de créer le portail de connexion.

Le projet a été réalisé avec la librairie Create-React-App. Celle-ci permet de démarrer rapidement une application single-page (SPA), configurant directement le projet avec notamment Babel, Webpack... etc

## Architecture :

L'architecture est basée sur une séparation des composants dans des dossiers spécifiques. Chaque dossier contient deux fichiers : index.js et style.scss.

Chaque fichier contient un composant unique.



Chaque dossier contenu dans "components" porte le nom du composant contenu. Ces dossiers contiennent deux fichiers, index.js, où se trouve le composant et sa partie render en jsx, et le fichier de style propre au composant.

Le dossier module contient différents fichiers dédiés uniquement à de la logique. Ces fichiers contiennent un ensemble de fonctions liées au nom du fichier, qui seront exportées et utilisées dans les différents composants du dossier "component"

### Exemple d'un composant :

Pour cet exemple, je vais détailler le composant SignUp, qui permet de s'inscrire :

```
import { useState } from 'react';
import { Auth } from 'aws-amplify';
import { Link, useHistory } from 'react-router-dom';
import {
  CheckboxField,
  Loader,
  PasswordField,
  TextField,
  Button,
} from '@aws-amplify/ui-react';

import '@aws-amplify/ui-react/styles.css';
import classNames from 'classnames';
import {
  checkMail,
  checkPasswordRealTime,
  signUp,
  signIn,
} from '../../modules/auth';

import './style.scss';
```

Dans un premier temps, nous retrouvons l'ensemble des imports dans le haut du fichier. Nous avons certains import liées à Amplify, comme la méthode “Auth”, ainsi que les éléments front de la bibliothèque “Amplify UI”. React-Router est également utilisé pour la gestion des urls. Les fonctions de la partie "modules" sont également importées.



```
const SignUp = ({
  onSign,
  redirectFromQuery
}) => {
  const [isLoading, setIsLoading] = useState(false);
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [cgu, setCgu] = useState(false);
  const [cguError, setCguError] = useState('');
  const [error, setError] = useState('');
  const [emailError, setEmailError] = useState(false);
  const [errorMessageesPassword, setErrorMessageesPassword] = useState([]);
```

Nous trouvons ensuite la déclaration du composant, qui est faite via une expression de fonction, qui sera exportée pour le fichier app.js.

Des props sont passés depuis le fichier app.js ( onSign, redirectFromQuery ). Nous avons ensuite les hooks de state, qui nous permettent de gérer les différents états de notre composant :

- Le loader qui s'affiche pour les actions asynchrones.
- email et password qui sont des champs contrôlés ( l'état est modifié sur chaque frappe clavier, et l'événement intercepté modifie la valeur du state de l'input lié ).
- Les cgu, avec la gestion d'erreur liée.
- Les différents types d'erreurs.

```
const handleSignUp = async () => {
  setError('');
  try {
    if (!cgu) throw new Error('CGU acceptance is mandatory');
    setIsLoading(true);
    await signUp(Auth, {
      username: email,
      password,
      attributes: { email, 'custom:tos': '1' },
    });
    const user = await signIn(Auth, email, password);
    onSign(user);
    history.push('/');
    setIsLoading(false);
  } catch (error) {
    console.log('error signing up', error);
    if (!cgu) {
      setCguError(error.message);
    } else {
      setError(error.message);
    }
    setIsLoading(false);
  }
};
```

Des fonctions propres à ce composant sont ensuite déclarées.

La fonction “ handleSignUp “, appelée lors du click sur le bouton “sign-up” du formulaire, est une fonction asynchrone contenant un “try / catch“, qui tente de créer l'utilisateur via la méthode signUp.

Les différents états sont gérés via les “setState” des hooks déclarés au début du composant.

```
const checkMail = (e, setEmailError, setError) => {
  const regex_email = /^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}$/i;
  const isValid = regex_email.test(e.target.value);
  setEmailError(!isValid);
  if (!isValid) {
    setError('Email is not valid');
  } else setError('');
};
```

Le champ “email” du formulaire est un champ contrôlé. Sur chaque frappe clavier, l'événement est récupéré et la valeur de la touche pressée est ajoutée

à l'input. L'email saisi est contrôlé sur chaque événement par une Regex, pour vérifier la bonne syntaxe de celui-ci.

```
return (
  <>
    <form className="sign-form" onSubmit={handleSignUp}>
      <h2 className="sign-form-title">Create your account</h2>
      {error && <h3 className="error">{error}</h3>}
      {errorMessagesPassword.length > 0 && (
        <h3 className="passworderror">
          {errorMessagesPassword.map((error, key) => (
            <div key={key}>{error}</div>
          ))}
        </h3>
      )}
      <div className={classNames('field', { 'field--error': emailError })}>
        <TextField
          id="email"
          type="email"
          value={email}
          onChange={handleChangeEmail}
          onBlur={(e) => checkMail(e, setEmailError, setError)}
          placeholder="Email*"
          hasError={emailError}
          required
        />
      </div>
      <div
        className={classNames('field', {
          'field--error': errorMessagesPassword.length > 0,
        })}
      >
        <PasswordField
          id="password"
          value={password}
          descriptiveText="Must be at least 8 characters, contains"
          onChange={(e) => {
            setPassword(e.target.value);
            handleChangePassword(e);
          }}
        />
      </div>
    </form>
  </>
);
```

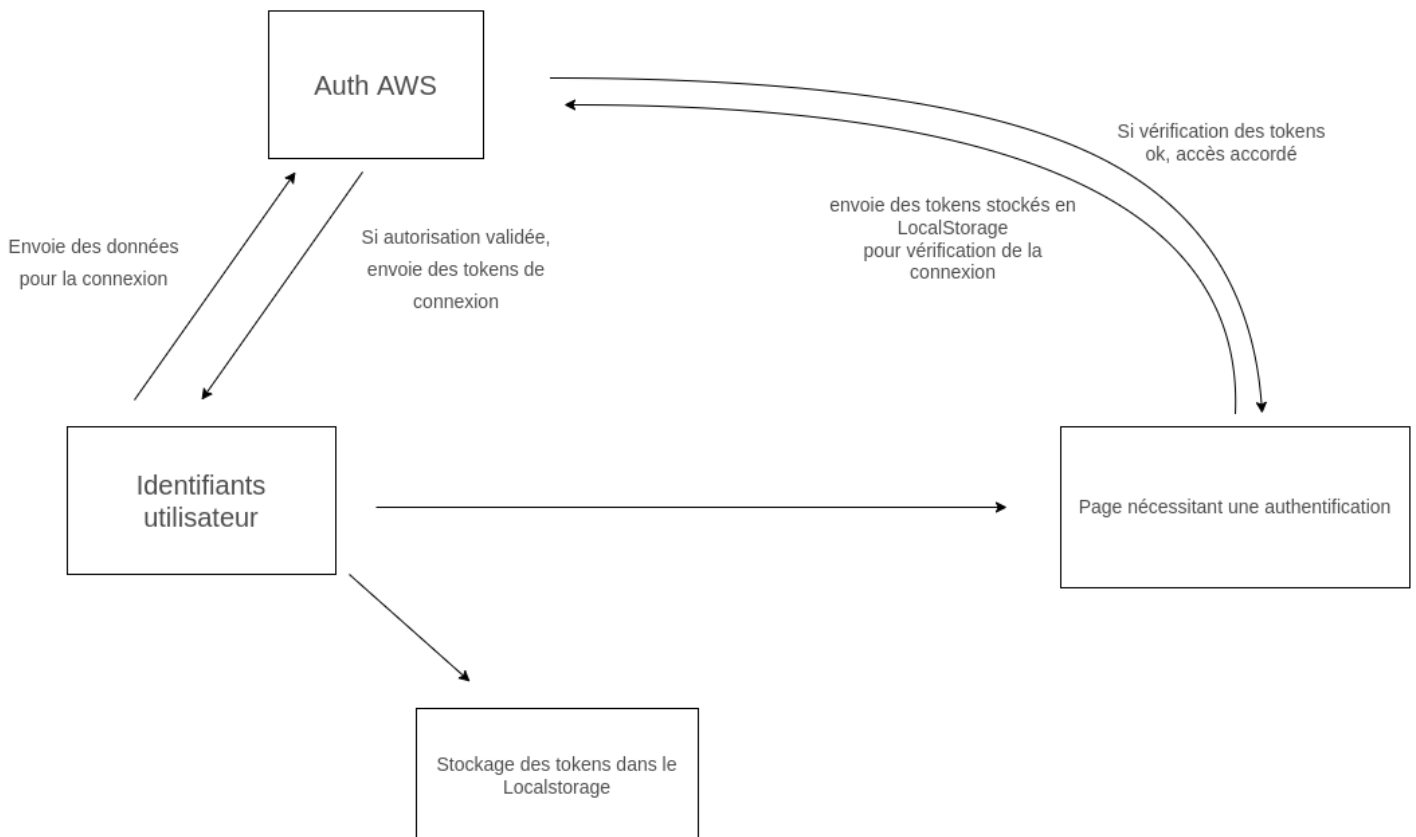
Vient ensuite un extrait du rendu, via le return en jsx.

Nous pouvons voir l'utilisation d'un “fragment”, immédiatement au début du return : Cela permet de wrapper un ensemble de balise html, sans créer de nouveaux nœuds dans le dom.

L'ensemble des composants fonctionne selon cette suite d'exemple.

## 5.2 Sécurité de la connexion

Le système de connexion fournit par AWS fonctionne de la façon suivante :



Lors d'une connexion réussie, le service d'AWS envoie plusieurs tokens de connexion à l'utilisateur, qui sont stockés dans le localstorage.

Lors d'un accès à une page demandant une connexion, les tokens sont envoyés au service Auth de AWS, afin de vérifier l'authentification.

L'entreprise souhaitant un portail de connexion unique, et la possibilité de naviguer d'une application à l'autre (chaque application a son propre sous-domaine) sans reconnexion, nous avons dû repenser l'utilisation de base de ces tokens. En effet, le localstorage est uniquement accessible pour un domaine donné, pas ses sous-domaines.

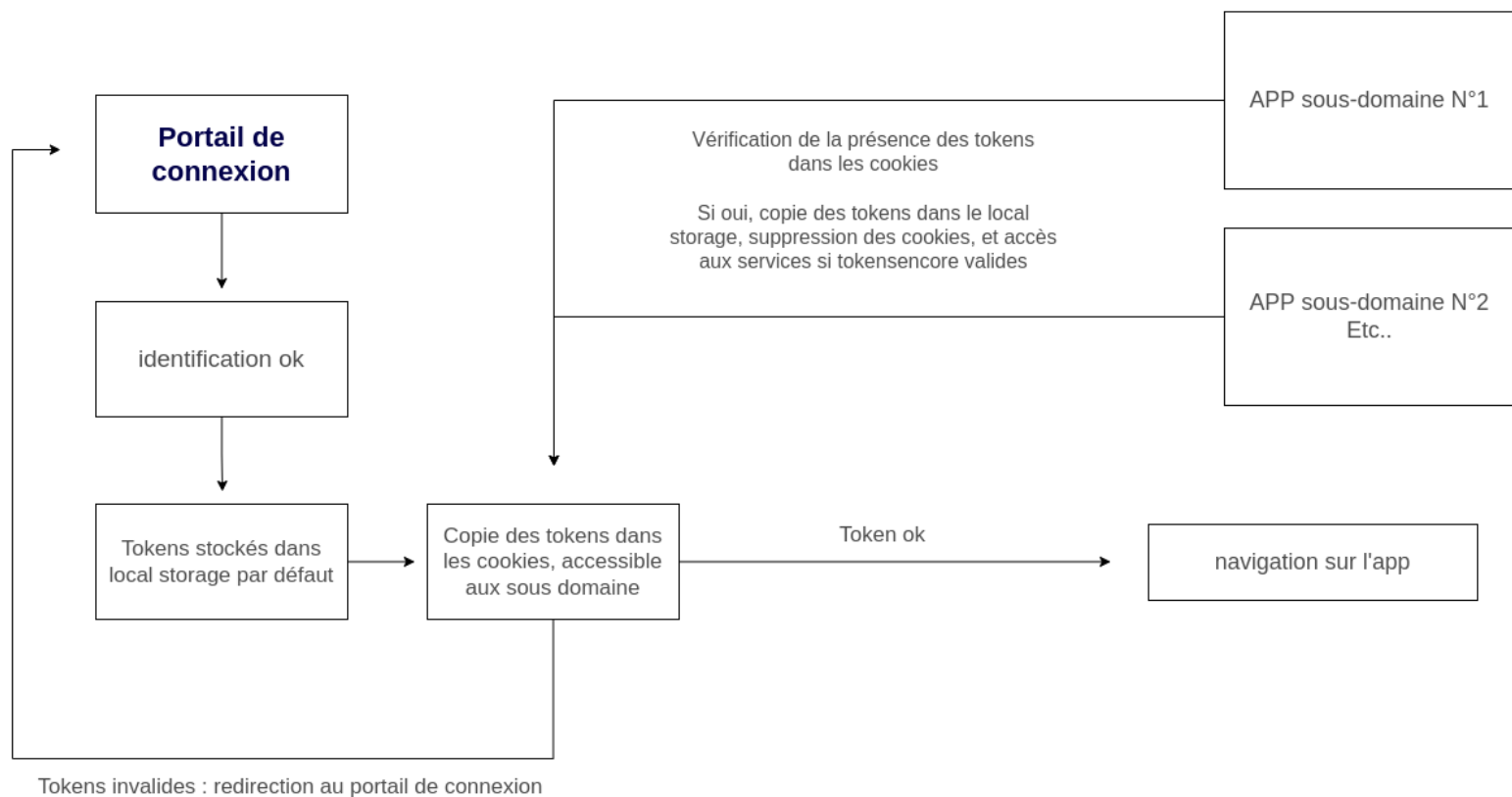
Un transfert des tokens entre le localStorage et les cookies se fait donc entre les différentes applications.

Il m'a été demandé de vérifier s'il était possible de "voler" ces tokens, et si oui, voir l'usage qu'il était possible d'en faire. J'ai donc procédé à différents essais avec Wireshark :

Il m'était parfois possible de forcer manuellement le http ( alors que le https est activé sur l'app en déploiement ) dans l'url, ce qui envoyait les cookies de façon non chiffrée, et je parvenais à les récupérer pour ensuite les réutiliser depuis une autre session, qui simulait un navigateur 'pirate'.

Ce problème ne se produisait pas de façon récurrente, mais j'ai tout de même réussi à le reproduire cette faille à plusieurs reprises, ce qui nécessite sa prise en compte.

Schéma de la connexion :



Les cookies ont donc été paramétrés en "secure" pour empêcher leur envoi depuis une url en http.

Le HttpOnly n'est dans notre cas, pas utilisable, car nous utilisons le javascript pour accéder au cookies, afin de les transférer dans le local storage.

Le risque principal dans notre cas serait les failles XSS et CSRF.

Pour la faille XSS, react échappe par défaut tout le contenu du jsx, ce qui limite les possibilités d'attaque.

Concernant la faille CSRF, j'ai préconisé l'utilisation d'un token CSRF, envoyé lors des requêtes pour authentifier la source.

Pour contrer le vol des tokens d'authentification, l'utilisation d'un RTR ( rotation token refresh ) semble être une bonne solution.

Pour résumer, en l'état actuel de mes connaissances sur ce vaste sujet, l'utilisation en parallèle du localstorage + cookie partiellement sécurisé ne me semble pas être la meilleure solution.

Les token stockés dans les cookies ne peuvent pas être supprimés, sinon la navigation entre les différentes applications demanderait une reconnexion :

En théorie, à chaque ouverture d'une des applications, les cookies contenant les bon tokens sont transférés au local storage. Si ils étaient supprimés immédiatement après le transfert dans le localstorage, la visite d'une autre application sur un des sous domaine nécessiterait alors une nouvelle connexion, ce que l'entreprise aimerait éviter.

Les cookies ne doivent donc pas être supprimés, et comme nous utilisons javascript pour les transferts, nous ne pouvons pas les passer en "httpOnly", ce qui maintient une possible faille de sécurité.

L'ensemble de ces informations a été transmis à mes supérieurs hiérarchiques, lead dev et CTO.

J'ai également rédigé une documentation sur confluence, reprenant l'ensemble de mes recherches, et des tests que j'ai réalisés.

### 5.3 Conclusion du portail de connexion

Le travail sur cette première partie du projet était intéressant à réaliser, notamment sur la gestion des tokens, la sécurité, et l'ensemble des problématiques qui vont avec.

Le travail en équipe avec mon collègue, qui possède des connaissances confirmées dans l'utilisation de React.js, était très formateur. Cela m'a permis d'aborder l'utilisation de cette librairie avec de meilleures pratiques.

## 5.4 Application utilisateurs ( confidentiel )

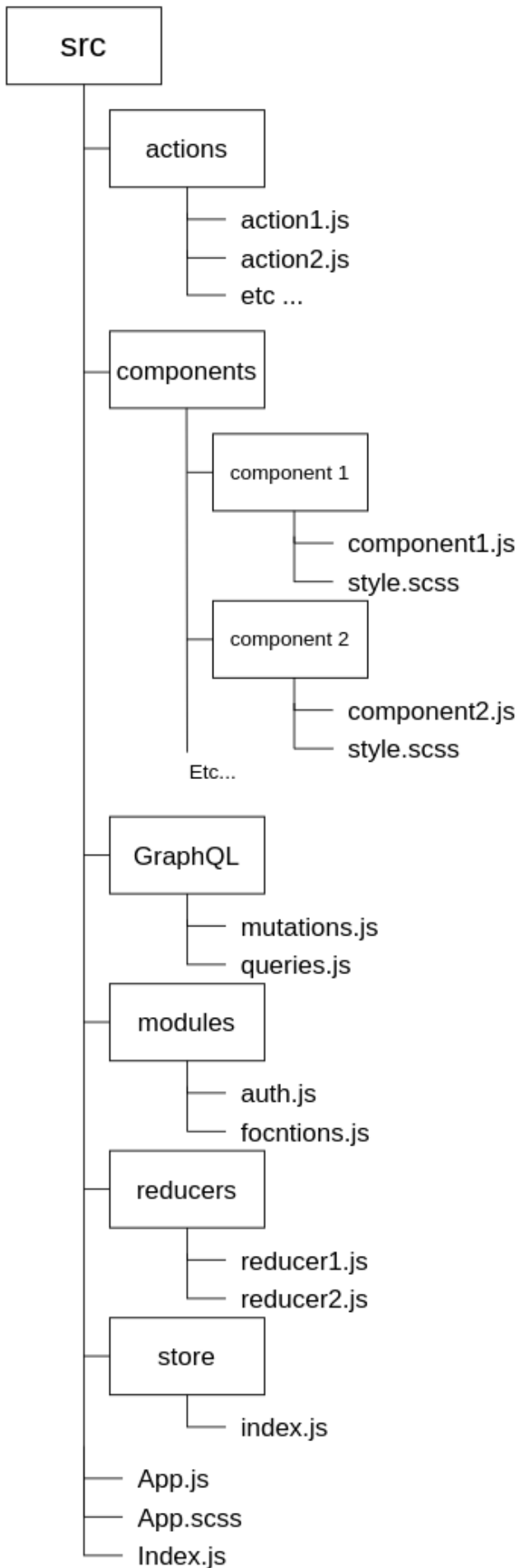
### Résumé :

Suite à la création du portail de connexion, il nous a été demandé de travailler sur une application à destination des utilisateurs possédant des télescopes.

Cette application sera accessible pour les utilisateurs disposant d'un compte, géré par le portail de connexion décrit précédemment.

Ce projet, comme le précédent, a été créé avec create-react-app.

## Architecture :



L'architecture de ce projet est assez similaire à celle du portail de connexion. Quelques différences sont à noter :

Apparition des dossiers “ actions “, “reducers” et “store”, dû à l'utilisation de Redux pour ce projet

Un dossiers graphql est aussi présent, contenant des modèles pour les requêtes en base de données



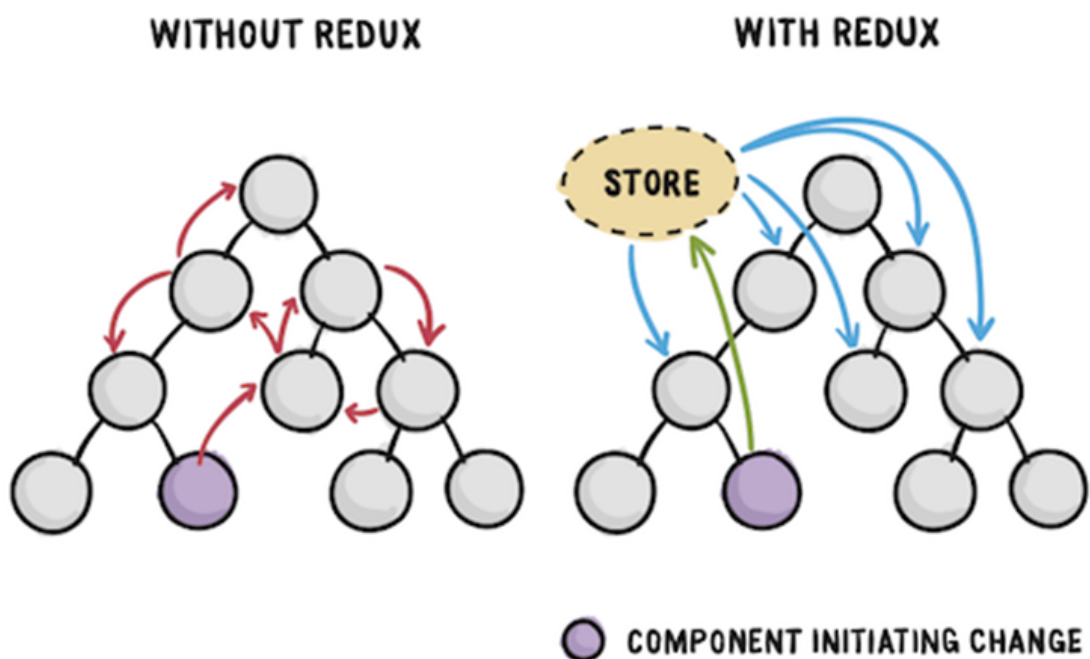
## Redux:

Pour ce projet, qui peut contenir beaucoup d'états différents, interagissant les uns avec les autres, nous avons choisi d'utiliser la librairie Redux, afin de stocker tout le state de l'application hors des composants.

Redux, un peu à l'instar de contexte, présent dans React, est un gestionnaire de state, qui permet de déporter et regrouper l'ensemble du state de l'application, en un seul endroit, appelé le store.

Le fait d'avoir le state regroupé et disponible dans tous les composants, évite ce que l'on appelle le "props drilling" : le fait de devoir passer de parents à enfants, des props qui ne seront pas forcément utiles aux enfants intermédiaires.

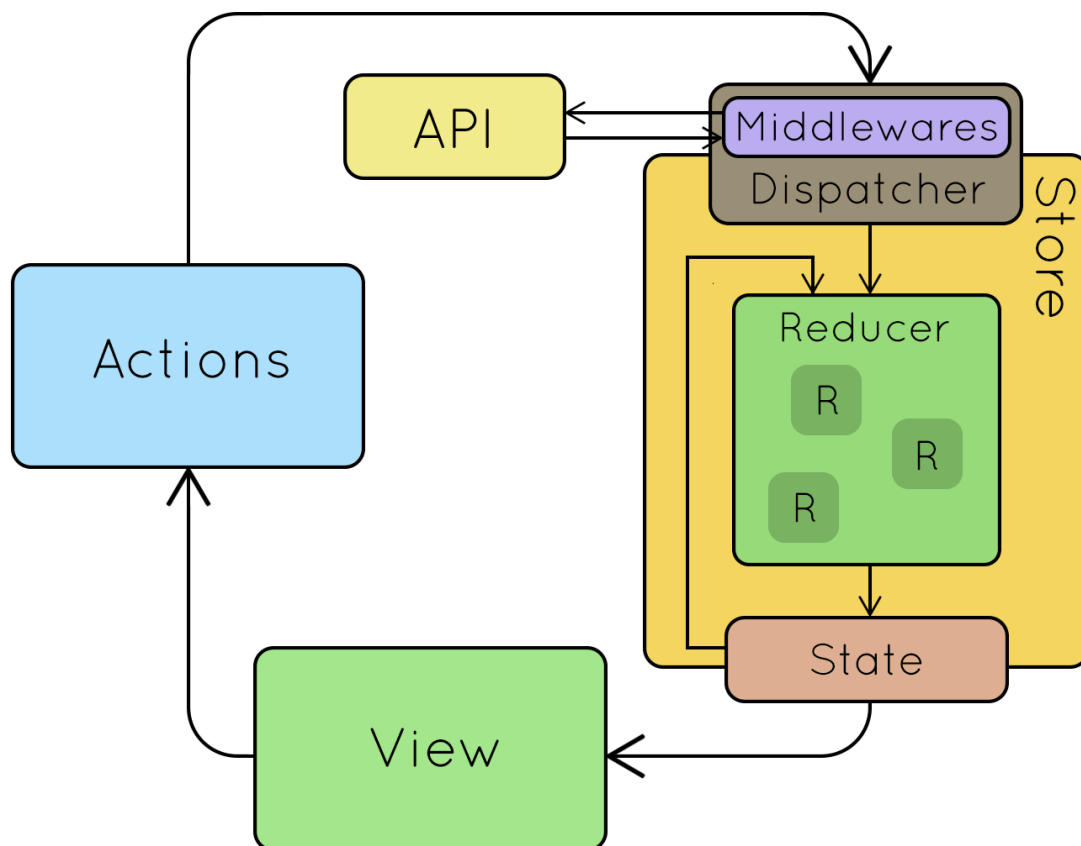
Un "provider" fournit le state aux différents composants, qui ont besoin de le consommer :



source : <https://www.jbvigneron.fr/parlons-dev/mettre-en-place-redux-pour-un-projet-angular/>

L'ensemble de Redux fonctionne ainsi :

- un store est constitué, contenant l'ensemble du state.
- Les reducers vont intercepter les actions pour modifier le state dans le store
- Des actions indiquent au reducer qui les interceptent, les changements d'état qui doivent être appliqués.
- Lorsque le state est modifié, un render des éléments concerné est effectué, avec le nouvel état.



## Exemple du code Redux :

L'ensemble du state et des actions, passées dans des variables, sont initiés dans le reducer :

```
import {
  SET_ACTION_1,
  SET_ACTION_2,
  SET_ACTION_3,
  SET_ACTION_4,
  SET_ACTION_5,
  SET_ACTION_6,
  SET_ACTION_7
} from "../actions/";

const initialState = {
  state1: "",
  state2: false,
  state3: null,
  state4: "",
  state5: true,
  state6: undefined,
  state7: {
    state7_1: "",
    state7_2: false,
    state7_3: null,
    state7_4: ""
  }
};

const reducer = (state = initialState, action = {}) => {
  switch (action.type) {
    case SET_ACTION_1:
      return { ...state, state1: action.state1 };
    case SET_ACTION_2:
      return { ...state, state2: true };
    case SET_ACTION_3:
      return { ...state, state1: false, state4: action.state4 };
    case SET_ACTION_4:
      return { ...state, state1: false, state2: action.state2 };
    case SET_ACTION_5:
      return { ...state, state6: false, state4: action.state4 };
    case SET_ACTION_6:
      return { ...state, state1: action.state2 };
    case SET_ACTION_7:
      return { ...state, state7: { ...state.state7, state7_2: action.state7 } };
    default:
      return state;
  }
};

export default reducer;
```

afin  
de

*préserver la confidentialité du projet, j'ai renommé les actions et le state initial du projet*

Nous trouvons dans un premier temps, les actions passées dans des variables.

Vient ensuite la const “initialState”, qui contient le state d’origine .

Le reducer est enfin construit avec un switch, qui interprétera les actions envoyées par des dispatch, et qui retournera un nouveau state.

La modification peut se faire par un payload, qui peut contenir un certain nombre d’informations, passées en paramètre, qui seront utilisées pour la modification du state.

Lorsque le state est modifié, un render du composant concerné est effectué.

L’application utilisant du code asynchrone, nous avons utilisé la bibliothèque Redux-Thunk, afin de créer des middleware faisant des appels api.

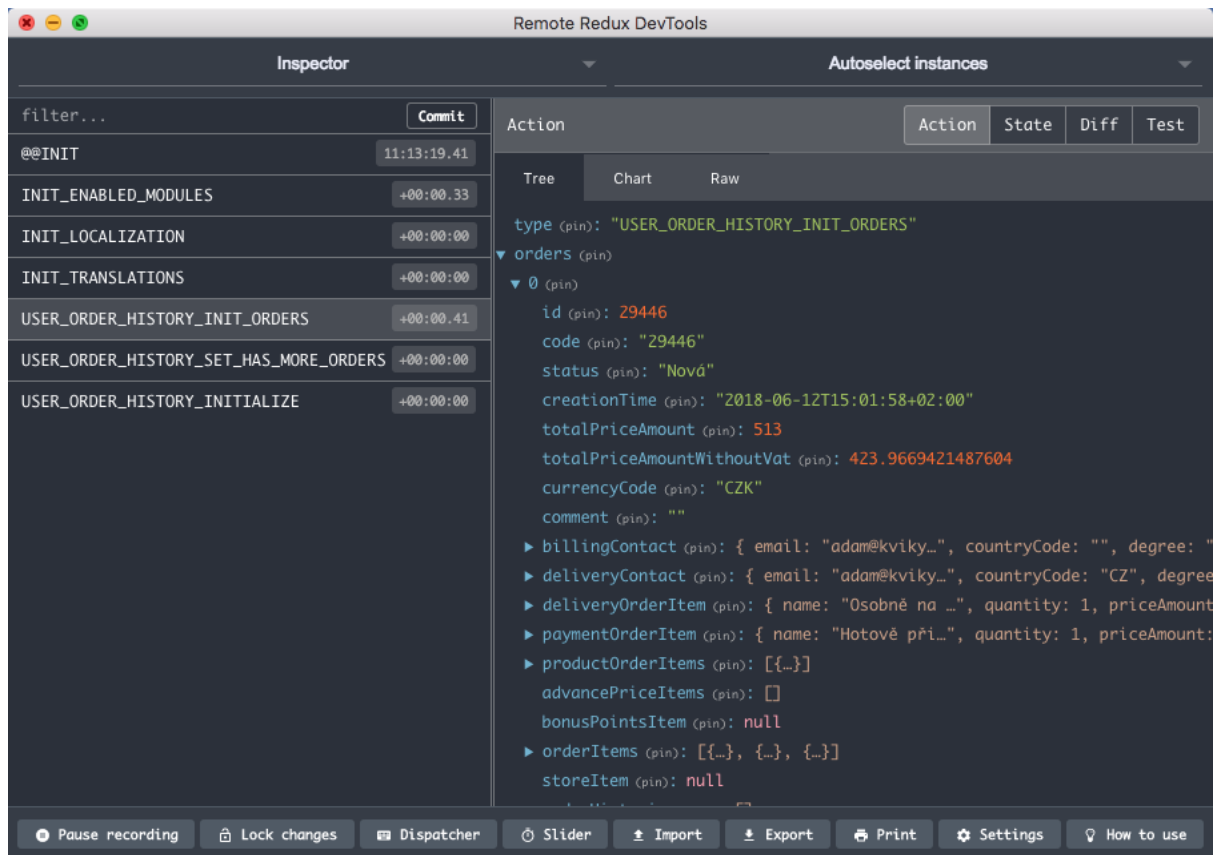
En effet, redux est par défaut, synchrone.

exemple d’un middleware utilisant Thunk :

*afin de préserver la confidentialité du projet, les dispatch et actions ont été renommés*

```
export const action1 = (param) => async (dispatch, getState) => {
  const state = getState();
  try {
    const result = await API.graphql(
      graphqlOperation(req, {
        req1,
        content: state.state1
      })
    );
    dispatch(
      setAction1({
        ...state.state1.state1Details,
        state2: {
          items: [
            ...state.state3.details.items,
            result.data.state5,
          ],
        },
      })
    );
    dispatch(setState5(""));
  } catch (error) {
    console.log(error);
  }
};
```

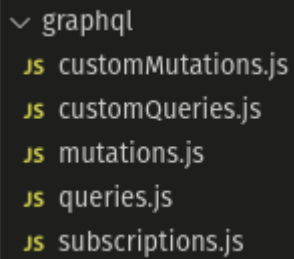
Nous avons également utilisé Redux Devtools pour le développement. Cet outil est très pratique, il permet de visionner en temps réel les modifications et le state actuel.



### GraphQL:

La persistance des données a été faite avec la base de données AWS, DynamoDB.

GraphQL a été utilisé pour les différentes requêtes .



```

  graphql
  ├── customMutations.js
  ├── customQueries.js
  ├── mutations.js
  ├── queries.js
  └── subscriptions.js

```

Le dossier graphql contient plusieurs fichiers :

- mutation.js et queries.js, sont des fichiers créés automatiquement, lors de la mise en place du backend, avec le crud de base en rapport avec les tables présentes en bdd.
- customMutation.js et customQueries.js, contiennent des requêtes spécifiques, que nous avons nous même créées
- subscription.js contient les éléments qui seront mis à jour automatiquement, en cas de modification, via des websocket, avec des événements.

Un des aspect intéressant de graphql est qu'il permet de récupérer les données que l'on souhaite précisément, sur plusieurs tables, en une seule requête, ce qui limite les accès API.

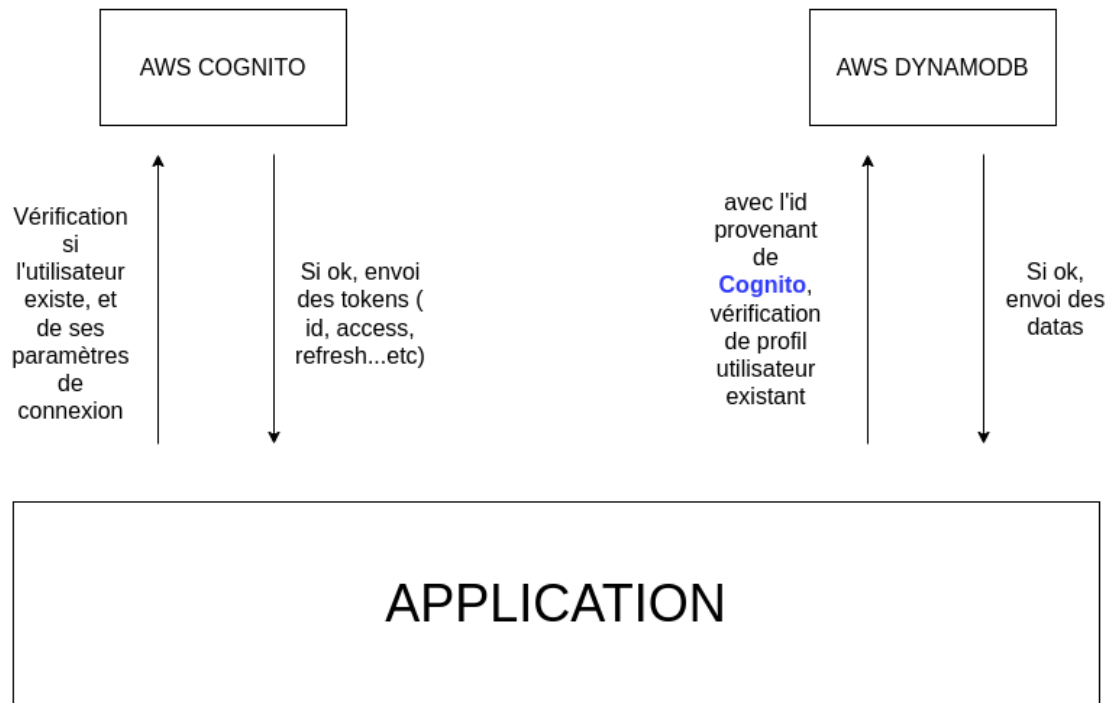
Les requêtes créées dans customMutation.js et customQueries.js sont de ce type. Elles ciblent plusieurs tables en une requête pour récupérer (ou modifier ) les éléments de notre choix :

*afin de préserver la confidentialité du projet, les les noms des tables ont été modifiés*

```
export const customQueries = /* GraphQL */ `
  query customQueries($id: ID!, $sub: ID!) {
    getData(id: $id) {
      data_1
      data_2
      data_3
      data_4
      data_5
      table_2(sortDirection: ASC) {
        items {
          data_1
          data_2
          data_3
          user {
            data_1
            data_2
          }
        }
      }
      data_4 {
        data_1
        data_2
        data_3
      }
      user {
        data_1
        data_2
      }
      table_3(filter: { userID: { eq: $sub } }) {
        items {
          data_1
        }
      }
    }
  }
`;
```

### Exemple d'un composant :

Un utilisateur connecté depuis le portail de connexion peut arriver sur cette application, mais ne pas posséder de compte spécifique à cette application :



Il faut bien séparer AWS Cognito, qui est la gestion des utilisateurs, de leurs droits..etc, et de l'éventuel profil utilisateur, qui lui sera stocké en bdd.

Lors du développement de l'application, il m'a été demandé de mettre en place la modification suivante :

Lorsqu'un utilisateur se rend sur l'application, une vérification doit être effectuée afin de savoir s'il possède un handle. Si ce n'est pas le cas, il doit être redirigé automatiquement sur une page l'obligeant à en saisir un, pour ensuite avoir accès à l'ensemble des fonctionnalités de l'application.



afin de préserver la confidentialité du projet, les les noms des routes et des composants ont été modifiés

```
function App() {
  return (
    <div className="App">
      <Switch>
        <Route path="/error/:code">
          <ErrorPage></ErrorPage>
        </Route>
        <Route path="/">
          <Authenticator
            loginMechanisms={['email']}
            components={components}
            services={services}
          >
            {({ signOut, user }) => {
              return (
                <>
                  <Header signOut={signOut}></Header>
                  <Switch>
                    <Route exact path="/">
                      <Redirect to="/route/" />
                    </Route>
                    <Route exact path="/route_1/">
                      <Component></Component>
                    </Route>
                    <Route path="/route_2/:param">
                      <Component></Component>
                    </Route>
                    <Route path="/route_3/:param">
                      <Component></Component>
                    </Route>
                    <Route exact path="/route_4/:param">
                      <Component></Component>
                    </Route>
                    <Route path="/route_5/:param">
                      <Component></Component>
                    </Route>
                    <Route>
                      <ErrorPage></ErrorPage>
                    </Route>
                  </Switch>
                </>
              );
            }}
          </Authenticator>
        </Route>
      </Switch>
    </div>
  );
}
```

A l'origine, le composant App était construit de cette façon.

Nous utilisons React-Router pour créer les différentes routes. Le composant "Authenticator" présent ici est l'authentification.

Pour gagner en praticité lors du développement, nous avons ajouté la possibilité de créer un compte et de s'authentifier directement dans cette application, sans passer par le portail de connexion décrit précédemment.

Lorsque l'utilisateur arrive sur l'application, il peut donc se connecter, ce qui lui permet d'accéder au return de la callback présent dans le composant Authenticator.

Le render de l'application affiche alors le header, avec la fonction signout, et les composants à afficher selon la route de navigation.

Dans un premier temps, j'ai voulu utiliser un affichage conditionnel, avec une ternaire, dont la conditionnelle se fait sur l'existence ou non du handle :

```
return (
  <UserHandle ?
    <>
    <Header signOut={signOut}></Header>
    <Switch>
      <Route exact path="/">
        <Redirect to="/route/" />
      </Route>
      <Route exact path="/route_1/">
        <Component></Component>
      </Route>
      <Route path="/route_2/:param">
        <Component></Component>
      </Route>
      <Route path="/route_3/:param">
        <Component></Component>
      </Route>
      <Route exact path="/route_4/:param">
        <Component></Component>
      </Route>
      <Route path="/route_5/:param">
        <Component></Component>
      </Route>
      <Route>
        <ErrorPage></ErrorPage>
      </Route>
    </Switch>
  </>
  :
  <UserChoiceHandle></UserChoiceHandle>
```

Mais cette gestion de l'affichage ne convient pas : Une fois l'utilisateur connecté depuis AWS Cognito, un autre appel api se fait depuis graphql vers dynamoDB, afin de voir si l'utilisateur, identifié via cognito, possède un handle enregistré en base de données.

Si cette requête récupère bien un handle existant, un dispatch est effectué dans redux, pour mettre le state à jour concernant l'utilisateur.

Le temps que cet appel api soit géré, notre ternaire affichera toujours par défaut le composant "UserChoiceHandle", car dans un premier temps le UserHandle est false tant que le state n'a pas été mis à jour.

Cette solution ne fonctionne donc pas

J'ai donc modifié le render des composants : J'ai créé un nouveau composant "RouterUser", dans lequel j'ai intégré l'ensemble des routes, avec différentes conditions d'affichage.

```
function App() {
  return (
    <div className="App">
      <Switch>
        <Route path="/error/:code">
          <ErrorPage></ErrorPage>
        </Route>
        <Route path="/">
          <Authenticator
            loginMechanisms={["email"]}
            components={components}
            services={services}
          >
            {( { signOut, user } ) => {
              return (
                <>
                  <Header signOut={signOut}></Header>
                  <RouterUser />
                </>
              );
            }}
          </Authenticator>
        </Route>
      </Switch>
    </div>
  );
}

export default App;
```

Une fois passé l'authentification, Le header sera affiché par défaut, ce qui permettra à l'utilisateur de sign-out si besoin.

Le composant RouterUser, quant à lui, se base sur le state existant pour savoir ce qu'il va devoir afficher.

```
const RouterUser = () => {
  const dispatch = useDispatch();
  const handle = useSelector((state) => state.myProfile?.handle);
  const userHandleExist = useSelector((state) => state.userHandleExist);
  const profileIsLoading = useSelector(
    (state) => state.social.profileIsLoading
  );
  useEffect(() => {
    dispatch(loadMyProfile());
  }, [dispatch]);
}
```

Les constantes déclarées utilisent la fonction “useSelector” de Redux, afin de récupérer certains éléments du state, et effectuer un nouveau render si il y a une mise à jour de ce state.

Nous avons :

- handle, qui sera récupéré si un profil est déjà existant (le “?” est une condition sur l’objet, qui permet de ne pas avoir d’erreur si l’objet en question n’existe pas )
- userHandleExist, qui est un booléen, sur false par défaut.
- profileIsLoading, qui est un booléen également. Cette constante gère l’affichage d’un spinner de chargement, qui s’affiche le temps que les fonctions asynchrone soient terminées.

Le hook “useEffect”, ( qui gère les effets de bords, comme les animations, effets, ou les appels api, et se lance une fois le DOM mis à jour ), lance ici un dispatch “loadMyprofile” .

Il s'agit d'un middleware asynchrone, qui démarre par une requête GraphQL, pour chercher le profil utilisateur, depuis l'id provenant de Cognito.

Selon le résultat, la constante "userHandleExist" est passée à true, et le loader à false, afin d'arrêter l'affiche du spinner.

Si la fonction passe dans le catch, le loader est à nouveau stoppé, en passant à false, et "userHandleExist" est également passé à false (son state initial est "undefined")

```
export const loadMyProfile = () => async (dispatch) => {
  try {
    const result = await API.graphql(
      graphqlOperation(getUser, {
        id: Auth.user.username,
      })
    );
    if (result.data.getUser?.avatar) {
      result.data.getUser.avatarUrl = await Storage.get(
        result.data.getUser.avatar.key
      );
    }
    if (result.data.getUser.handle) {
      dispatch(setUserHandleExist(true));
    }
    dispatch(setMyprofile(result.data.getUser));
    dispatch(setProfileIsLoading(false));
  } catch (error) {
    console.log(error);
    dispatch(setUserHandleExist(false));
    dispatch(setProfileIsLoading(false));
  }
};
```

Enfin, le render de ce composant, sans d'utilisation de ternaire, juste des affichages conditionnels basés sur les éléments du state géré par Redux.

Le flow fonctionne à présent correctement

```

return (
  <>
    {handle && (
      <>
        <Switch>
          <Route exact path="/">
            <Redirect to="/route/" />
          </Route>
          <Route exact path="/route_1/">
            <Composant></Composant>
          </Route>
          <Route path="/route_2/:param">
            <Composant></Composant>
          </Route>
          <Route path="/route_3/:param">
            <Composant></Composant>
          </Route>
          <Route exact path="/route_4/">
            <Composant></Composant>
          </Route>
          <Route path="/route_5/:handle">
            <Composant></Composant>
          </Route>
          <Route>
            <ErrorPage></ErrorPage>
          </Route>
        </Switch>
      </>
    )}
    {userHandleExist === false && (
      <div className="main">
        <UserFirstHandle />
      </div>
    )}
    {profileIsLoading && (
      <div className="main">
        <Loader />
      </div>
    )}
  </>
);
};

export default RouterUser;

```

## 5.5 Tests & mise en production

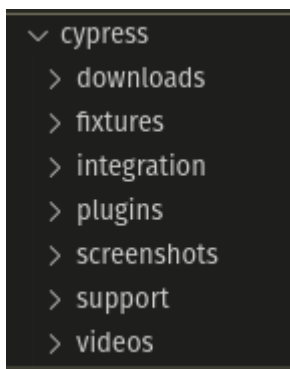
En parallèle des projets décrits ci dessus, il m'a été demandé de me pencher sur les différentes possibilités de test type end-to-end, et de réaliser des essais avec les différentes solutions existantes.

Jest et React Testing Library sont installés par défaut avec Create-React-App, mais ils ne correspondent pas aux besoins définis, à savoir, des tests end-to-end dans un navigateur, pour simuler l'utilisation classique d'un utilisateur.

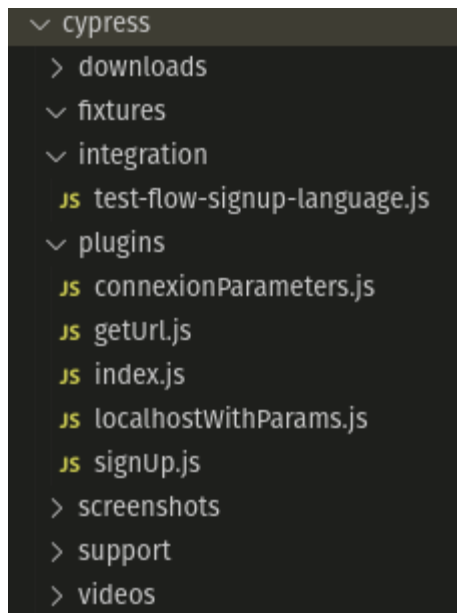
Après quelques recherches et différents essais, le choix s'est porté sur Cypress.io : Cette librairie correspond bien aux attentes qui m'ont été exposées, et il est pris en charge dans les outils CI/CD d'AWS.

### Description des test effectués

Une fois cypress installé dans le projet, nous pouvons accéder aux dossiers suivants, à la racine du projet :



dans le dossier “integration”, nous trouvons le ou les fichiers contenant les tests. Dans le dossier “plugin”, nous trouvons des fichiers contenant des fonctions ou des paramètres.



Les tests qui m’ont été demandés concernent le portail de connexion décrit précédemment.

Les test à effectuer sont :

- création d’un compte utilisateur.
- connexion suite à la création de compte
- Récupération du mail de validation de l’email utilisé pour la création de compte
- récupération du lien présent dans l’email reçu, visite de ce dernier afin de confirmer l’adresse mail du nouveau compte utilisateur.
- Tester les changements de langages sur les différentes pages
- Test du “reset password”, avec récupération du mail contenant le lien permettant de changer de mot de passe
- Création d’un nouveau password, et test de celui-ci



## Test création de compte, connexion et validation du mail :

dans un premier temps, création d'un nouveau compte utilisateur

```
import { connexionParam, dateforMail, localhost } from '../plugins/connexionParameters'
import { localhostWithParams } from '../plugins/localhostWithParams'
import signUp from '../plugins/signUp'
import getUrl from '../plugins/getUrl'
const email = connexionParam.mailTimestamp;
const randomNumberForPwd = Math.round(Math.random()*1000000)

describe('test sign-up, confirm email, change language, reset password', () => {
  //
  // Open Cypress | Set ".only"
  it('Sign-up & email confirm : Sign-up', () => {
    cy.visit(localhost)
    signUp(connexionParam)
    cy.wait(4000)
  })
})
```

La méthode Signup ci-dessus est la suivante :

```
import Amplify from 'aws-amplify';
import { Auth } from 'aws-amplify';
import awsExport from '../../src/aws-exports'

Amplify.configure(awsExport);

async function signUp(connexionParam) {
  try {
    await Auth.signUp({
      username: connexionParam.mailTimestamp,
      password: connexionParam.password,
      attributes: { email: connexionParam.mailTimestamp, 'custom:tos': '1' },
    });
  } catch (error) {
    console.log('error signing up:', error);
  }
}

export default signUp
```

L'ensemble des paramètres de connexion est stocké sur un fichier

“connexionParameter.js”

Afin que chaque test effectué crée un nouveau compte, nous utilisons la méthode suivante :

email = email.de.test+<time stamp>@gmail.com

Nous utilisons donc gmail, avec un “+” après le nom du mail, et un chiffre unique grâce au time stamp.

ex : [email.de.test+1655931746@gmail.com](mailto:email.de.test+1655931746@gmail.com)

La seconde étape est de se connecter avec les identifiants utilisés pour le signUp, et de cliquer sur le bouton permettant d’envoyer un message pour vérifier l’email utilisé lors du signUp.

```
... // go at homepage and first connexion, then click on "resend" button
... cy.get("#email").type(`${email}`);
... cy.get("#password").type(`${connexionParam.password}`);
... cy.get('.amplify-button.amplify-field-group__control.button').click()
... cy.get('.homepage', { timeout: 10000 }).children('button').click()
... cy.get('.homepage').children('p').should('have.class', 'success')
```

Une fois l’email envoyé, j’utilise une méthode de Cypress, le “task”, permettant d’utiliser des scripts en node.js.

Pour cette étape du test, il m’a été demandé de récupérer le mail de vérification en imap, et non par le navigateur directement ( risque de modification du DOM de l’interface mail du fournisseur, qui pourrait empêcher la connexion, et entraînera un échec des tests dans ce cas ).

J’ai donc utilisé la librairie node-IMAP.

Un script a été créé en node.js, qui dans un premier temps se connecte à la boîte mail, recherche ensuite le timestamp passé au mail d’origine dans le corps des messages.

La “task” de Cypress :

```
· // wait for mail, and connect to gmail account for keep mail and link to confirm  
· cy.wait(10000)  
· cy.task('imap', {time:5000, DataForFetch:dateforMail}).then((response)=>{  
·   ···· console.log(response)  
·   ···· let urlFromMail = getUrl(response.text)  
·   ···· let params = localhostWithParams(urlFromMail)  
·   ···· let urlfinal = localhost + "confirmEmail?code=" + params.code  
·   ···· cy.visit(urlfinal)  
· })  
  
· // wait and check if success message is ok  
· cy.wait(10000)  
· cy.get('p[class="success"]')
```

Extrait du script node-imap :

```

var MailParser = require("mailparser").MailParser;
var Imap = require('node-imap');

var imap = new Imap({
  user: 'mail de test',
  password: 'password!',
  host: 'imap.gmail.com',
  port: 993,
  tls: true
});

function openInbox(cb) {
  imap.openBox('INBOX', true, cb);
}

imap.once('ready', function() {
  openInbox(function(err, box) {
    if (err) throw err;
    imap.search(['ALL', ['TEXT', 'DataForFetch']], function(err, results) {
      if (err) throw err;
      var f = imap.fetch(results, {bodies: ''});

      f.on('message', processMessage)
      f.once('error', function(err) {
        console.log('Fetch error: ' + err);
      });
      f.once('end', function() {
        console.log('Done fetching all messages!');
        imap.end();
      });
    });
  });
});

imap.once('error', function(err) {
  console.log(err);
});

imap.once('end', function() {
  console.log('Connection ended');
});

imap.connect();
setTimeout(() => resolve(html), time)
}),

```

Les datas retournées par la réponse du script imap passent par la fonction “getUrl”, afin d’isoler l’url contenue dans le mail, et d’en extraire les paramètres afin de reconstruire l’url en localhost avec le code de reinitialisation temporaire.

Cette procédure est nécessaire car l’url envoyée dans l’email pointe vers un domaine en production, fonctionnant avec un back-end différent. Dans notre cas, l’url doit pointer vers localhost.

```
AWS: Add Debug Configuration | AWS: Edit Debug Configuration
export default function getUrl(data){
  let expression = /(https?:\/\/[^\s]+)/;
  let regex = new RegExp(expression);
  let urlInArray = data.match(regex)
  return urlInArray[0]
}
```

```
AWS: Add Debug Configuration | AWS: Edit Debug Configuration
export function localHostWithParams(urlFromMail){
  let url = new URL(urlFromMail)
  let urlParams = new URLSearchParams(url.search);
  let code = urlParams.get('code')
  let data = urlParams.get('data')
  return {code, data}
}
```

```
let urlFromMail = getUrl(response.text)
let params = localHostWithParams(urlFromMail)
let urlfinal = localhost + "confirmEmail?code=" + params.code
cy.visit(urlfinal)
```

L'url finale est alors visitée afin de valider l'email créé au tout début, avec le timestamp.

### Test Changements de langage

La deuxième partie des tests concerne le changement de langage. les tests sont fait sur les pages avec l'utilisateur deconnecté, et ensuite connecté

Test des changements si l'utilisateur n'est pas connecté.

Vérification si l'utilisateur est connecté. Si oui, signout.

```
// if user is connected, sign-out
cy.get('header a').then(($valueButtonSignOut)=>{
  if($valueButtonSignOut.hasClass("sign-out") === true){
    cy.get('header a button').click()
  }
})
```

Exemple des test de changement de langue :

```
cy.get("select option:selected").then(($valueSelect)=>{
  switch($valueSelect.text()){
    case 'FR':
      cy.get('.sign-form h2').should('have.text', 'Connectez-vous à votre compte')
      cy.get('input#email').should('have.attr', 'placeholder', 'Entrez votre courriel')
      cy.get('select').select('EN')
      cy.get('.sign-form h2').should('have.text', 'Sign in to your account')
      cy.get('input#email').should('have.attr', 'placeholder', 'Enter your email')
      break;

    case 'EN':
      cy.get('.sign-form h2').should('have.text', 'Sign in to your account')
      cy.get('input#email').should('have.attr', 'placeholder', 'Enter your email')
      cy.get('select').select('FR')
      cy.get('.sign-form h2').should('have.text', 'Connectez-vous à votre compte')
      cy.get('input#email').should('have.attr', 'placeholder', 'Entrez votre courriel')
      break;

    case 'ES':
      break;

    default:
      console.log(`no language match`);
  }
})
```

Ensuite, même procédure en re-connectant l'utilisateur.

## Test changement de password

La dernière étape de test est le reset password.

Les étapes sont sensiblement identiques à celles vues précédemment.

Le test effectue une demande de réinitialisation de password, un mail est envoyé avec une url permettant ( avec un code temporaire en paramètre d'url )

La task node-imap va récupérer ce mail. Un nouveau password est généré, avec le password d'origine + un nombre aléatoire :

```
const randomNumberForPwd = Math.round(Math.random()*1000000)
```

```
cy.get('#newpassword').type(connexionParam.password + randomNumberForPwd)
```

Et une connexion est effectuée avec ce nouveau password.

Open Cypress | Set ".only"

```
it('Reset password : Go to home page and click reset password', ()=>{
  cy.visit('localhost')
  cy.get('.forgot-password a').click()
  cy.get('#email').type(`${email}`)
  cy.get('.sign-form-footer button').click()
  cy.wait(10000)
})
```

Open Cypress | Set ".only"

```
it('Reset password : Gmail connection and get mail', ()=>{
  cy.task('imap', {time:5000, DataForFetch:dateforMail}).then((response)=>{
    let urlFromMail = getUrl(response.html)
    let params = localhostWithParams(urlFromMail)
    let urlfinal = localhost + "confirmForgotPassword?code=" + params.code + "&data=" + params.data
    cy.visit(urlfinal)
  })
  cy.get('#newpassword').type(connexionParam.password + randomNumberForPwd)
  cy.get('.sign-form-footer button').click()
  cy.wait(3000)
  cy.get('.sign-form .success')
})
```

Open Cypress | Set ".only"

```
it('Reset password : Test new password', ()=>{
  cy.visit('localhost')
  cy.get("#email").type(`${email}`);
  cy.get("#password").type(`${connexionParam.password + randomNumberForPwd}`);
  cy.get('.amplify-button.amplify-field-group__control.button').click()
  cy.wait(5000)
  cy.get('.storeButtons')
})
```

## CI/CD AWS

Un projet de test en mode “sandbox” a été créé sur AWS, afin de pouvoir essayer certaines fonctionnalités, sans risque d'interférer avec les applications en cours de développement, ou en production.

J'ai créé un projet d'essai, une todo list, contenant un jeu de test end to end, afin de tester le comportement en déploiement.

Le projet est connecté au dépôt CodeCommit d'aws, et lance le déploiement automatiquement lors d'un push sur la branche choisie.



visualisation des différentes étapes :



Last commit

Please visit AWS CodeCommit Co... | d5dbee0 | [AWS CodeCommit - test-nightwatch](#)

Previews

Disabled



Last commit

Please visit AWS CodeCommit Co... | d5dbee0 | [AWS CodeCommit - test-nightwatch](#)

Previews

Disabled



Last commit

Please visit AWS CodeCommit Co... | d5dbee0 | [AWS CodeCommit - test-nightwatch](#)

Previews

Disabled



Last commit

Please visit AWS CodeCommit Co... | d5dbee0 | [AWS CodeCommit - test-nightwatch](#)

Previews

Disabled

## Choix des tests à effectuer lors du déploiement

Il est possible de modifier le fichier `amplify.yml`, dans les builds settings, afin de spécifier quel fichier de test doit être pris en compte lors du déploiement ( c'est une des demande qui m'a été faite, étudier la possibilité de lancer des tests différents selon que l'on soit en environnement de développement sur une machine local, ou bien en déploiement :

```
--spec "cypress/integration/test-todolist.js"
```

## Dashboard Cypress

Une des fonctionnalité qui permet de mieux suivre les différents déploiement, est le dashboard de Cypress.

On connecte l'application au dashboard grâce à une "key", à ajouter aux build settings:

```
test:
  commands:
    - 'npx cypress run --record --key 21124ec8-72f7-45da-8be0-3aea5f6'
```

L'ensemble des test CICD effectué est ensuite répertorié dans le dashboard :

**UniAlex**  
Alexandre Leroux

**Latest runs**

VIEW ALL PROJECTS

Run status  
Run duration  
Test suite size  
Top failures  
Slowest tests  
Most common errors  
Flaky tests

Project settings

cyress

**Latest runs**

FILTER BY: All Time, Status, Not available, Committer, Tag, Flaky Tests

Last updated: 4:05:29pm

Run Name	Status	Run Time	Duration	Not available	Committer	Tag	Flaky Tests
tests ok	✓	alexandre-leroux	Ran 49 mins ago	00:20	Not available	Awscodebuild	# 24
test ok	✗	alexandre-leroux	Ran an hour ago	00:25	Not available	Awscodebuild	# 23
Remise du bon aws-export	✗	alexandre-leroux	Ran 8 days ago	00:25	Not available	Awscodebuild	# 22
Remplacement du aws-export par celui du back-office	✓	alexandre-leroux	Ran 8 days ago	00:21	Not available	Awscodebuild	# 21
suppression variable aws-export. Remplacement du aws-export par celui du social	✓	alexandre-leroux	Ran 8 days ago	00:20	Not available	Awscodebuild	# 20
ajout variable dans aws export, affichée en front	✗	alexandre-leroux	Ran 8 days ago	00:25	Not available	Awscodebuild	# 19
variable enlevée du aws-export, car rejetée (provoque une erreur) au moment du b...	✓	alexandre-leroux	Ran 9 days ago	00:20	Not available	Awscodebuild	# 18

## 6.Conclusion

Cette année d’alternance au sein de l’entreprise Unistellar a été enrichissante, et m’a permis de travailler et monter en compétence sur les différentes parties liées à la création d’un projet.

Le projet “portail de connexion” était enrichissant, car il m’a permis d’aborder, plus en profondeur, les problématiques rencontrées lors de la création de compte utilisateur, et les connexions sécurisées qui vont avec.

Le projet utilisateur était également formateur. Aborder une application contenant beaucoup de composants et de données qui interagissent ensemble, gérer le state de l’application via Redux, et découvrir l’utilisation de GraphQL, était formateur.

Pour finir, le travail en équipe avec un collègue maîtrisant bien l’environnement javascript, et web en général, était un vrai plus. Cela m’a permis d’avoir une meilleure approche du métier de développeur web. Les nombreux échanges que nous avons eu, sur les bonnes pratiques, certains concepts plus poussés, les solutions pouvant être apportées, et l’approche générale du code, m’ont beaucoup apporté. Je tiens à le remercier tout particulièrement pour la disponibilité dont il aura fait preuve.