

WorkWithTheBest



Projet chef d'oeuvre pour
La certification du
Titre de Développeur,
Développeuse web.

Formation Développeur,
Développeuse Web fullstack js,
Promotion n°1, Simplon
Saint Quentin en Yvelines,
Du 26/12/2018 au 31/07/2019.

Cabanes Thibault



Sommaire

I. Présentations :

1. Qui suis-je
2. Ma formation à Simplon
3. Mon stage
4. Résumé du projet
5. Compétences du REAC abordées
6. Technologies utilisées

II. Définition du Projet :

1. Les acteurs
2. Les use-cases
3. Les wireframes
4. Les maquettes
5. Schéma relationnel de la base de données

III. Back-End :

1. Mise en place du serveur back avec express et nodeJs
2. Implémentation et échanges avec la base de données
3. Enregistrement des comptes d'utilisateurs
4. Connexion des utilisateurs et Sécurité

IV. Front-End :

1. Mise en place de l'architecture Front-end en React
2. Routes publiques et routes privées
3. Le carousel
4. Component dynamique, crée/met à jour une annonce
5. Le css avec sass

V. Annexes :

I. Présentations

1. Qui suis-je :

Je m'appelles Thibault Cabanes, j'ai 26 ans, deux grands frères de 32 et 34 ans et suis originaire de l'Ile de France.

J'ai d'abord passé un baccalauréat Technologique de Sciences et Technologies de la Gestion, options Comptabilité et Finances d'Entreprises, puis ai entamé un BTS Comptabilité et Gestion des Organisations, que j'ai interrompus à la fin de la première année pour m'orienter vers la menuiserie, choix motivé à cette époque par l'envie de travailler rapidement et aussi par l'envie de créer de mes mains.

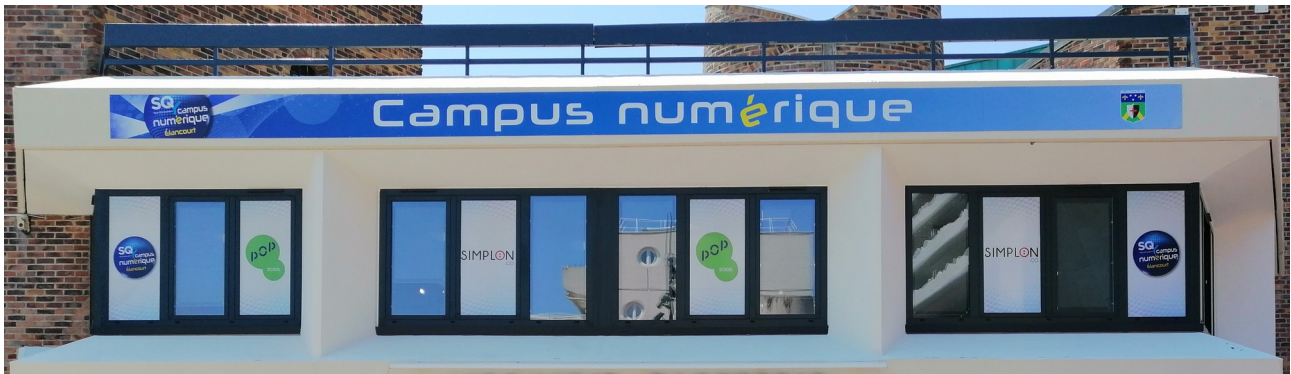
Ayant donc suivis des études de menuiserie et travaillé dans ce domaine pendant 5 ans, j'ai eut l'occasion de découvrir un métier passionnant, que je continuerais de pratiquer à titre personnel, mais qui ne me convenait plus au quotidien.

C'est alors qu'au mois de décembre 2018, enchaînant les contrats d'intérim sur les chantiers de menuiserie à Paris, me demandant comment je pourrais bien me réorienter et re-lancer ma carrière, je reçut un mail de pole emploi, m'annonçant la possibilité, avec l'école SIMPLON, de reprendre mes études afin de devenir développeur web. Aimant les défis, résoudre des problèmes et étant curieux de nature, ma rencontre avec l'univers du code et du développement a donc été une révélation pour moi.

Suivant d'abord des cours d'html et de css sur diverses plateformes en ligne telles que codecademy, sololearn ou le site du zéros (openClassroom), afin de me qualifier pour participer à la formation, j'ai ainsi put me familiariser avec les bases de la structuration et du design d'un site web, avant de me plonger pleinement dans le développement que j'aborderais plus tard.

Par la suite je souhaiterais pouvoir consolider ma montée en compétences, c'est pourquoi je cherche un contrat de professionnalisation, pour poursuivre mes études avec Simplon ou une autre école. Plus tard, j'aimerais pouvoir travailler sur des projets d'envergures, comme par exemple participer au développement d'un framework ou de fonctionnalités complexes.

2. Ma formation à Simplon :



Ma formation à Simplon a débutée le 26 décembre 2018, j'ai eue l'occasion d'y assimiler énormément d'informations.

Ayant commencé avec un ordinateur tournant sous Windows, je suis passé rapidement à un environnement Ubuntu qui m'a parfaitement convenu pour développer, en ce qui concerne l'éditeur de texte, je me suis rapidement orienté vers la version d'essais de Sublime Text avec lequel je me suis trouvé beaucoup d'affinités.

Au cours de l'année nous avons appris à travailler avec Git Hub pour sauvegarder nos projets en ligne ou pour collaborer à plusieurs sur un projet.

Nous avons donc débuté cette formation en travaillant sur le html et le css, pour parfaire ces compétences que nous avons révisé par nous même avant de commencer, ainsi nous avons vu comment structurer une page html, quelques bonnes pratiques pour le SEO, nous avons vu comment rendre le css d'une page responsive à l'aide des media-queries et nous avons vu quelques animations en css.

Nous sommes rapidement entrés dans le vif du sujet, en se mettant au javascript au travers de la conception d'un jeu de dominos sur une page web, l'exercice fut intense mais permit de découvrir toutes les possibilités d'interactions entre le DOM et nos scripts javascript.

J'ai ainsi put découvrir les différents types de données, booléens, nombres, chaînes de caractères, objets, tableaux, fonctions, ainsi que les différents opérateurs tels que les boucles for et while ou les conditionnels if et switch, et ait utilisé à peu près tout pour parvenir à développer mon jeu de dominos (<https://thibtoy.github.io/Mon-Site-Web/domino.html>).

Ensuite nous avons commencé à travailler sur les bases de données (plus particulièrement MySQL) et sur la logique du développement back-end, nous avons vu comment écrire des requêtes en sql, la logique des relations entre les tables, puis nous avons commencé à créer des données, avant de découvrir nodeJs, qui nous permettrait de monter un serveur back-end qui allait nous envoyer les données sur un projet front, nous venions de réaliser notre première connexion entre un projet back et un projet front.

Nous avons continué à travailler sur plusieurs projets mêlant front-end et back-end, comme des petits forums ou de petites galeries commerciales, perfectionnant nos compétences en Sql et en Node, nous faisons nos bases de données, nos jeux de données, nous mettons en place les routes pour accéder aux données via des requêtes préparées et finalement nous injectons les données vers nos projets front pour les exploiter.

Au cours de cette période j'ai eue l'occasion de m'initier à du PHP pour faire du server side rendering et ait donc installé l'environnement lamp sur mon ordinateur, j'ai put m'essayer à du React pour le front-end et j'ai également eue l'occasion de tester des alternatives à MySQL, comme SQLite3 ou mongoDB, ainsi que des méthodes, de communication avec les bases de données, orientées model, les ORM (Object Relational Mapper) comme sequelize ou mongoose.

Nous avons finalement vu les logiques de sécurisation des données et des accès, avec l'encryptage des mots de passes en bases de données, ainsi que l'autorisations d'accès et de modification de certaines données via des tokens.

Tout au long de la formation nous avons également suivis quelques interventions sur les méthodes AGILE et Scrum Master, nous avons fais les deux premiers Moocs de la Réglementation Générale de Protection des Données et nous avons fais quelques ateliers WordPress et Bootstrap.

3. Mon stage :

Dans le cadre de ce stage, effectué lors de ma formation avec Simplon, j'ai été reçu en tant que Développeur Web jr dans l'organisme Smart Economy Solution.

On m'a demandé de travailler sur la solution Manavao, qui est un réseau social de proximité français, ayant pour but de relancer la dynamique territoriale à échelle locale.

On m'a donc demandé plus précisément de travailler sur le développement d'une « maquette commerciale » d'une nouvelle partie du site Web.

Durant ce stage, j'ai pu travailler en télé-travail, malheureusement le Développeur Web avec lequel j'étais sensé communiquer et qui devait m'aider, n'a pas eu beaucoup de temps à m'accorder et j'ai donc dû m'en sortir par moi-même, ce qui fût une expérience constructive.

Je me suis retrouvé dans la situation d'un développeur web devant écouter synthétiser et réaliser la demande de son Client.

J'ai réalisé un diagramme fonctionnel du site, récupéré les quelques plans et chartes graphiques fournis par le client et ait commencé à travailler sur ce projet. Il fallait que je m'intègre à la structure du site Manavao, développée sous Symfony 3.

N'ayant jamais travaillé avec ce framework auparavant, chaque nouvelle tâche à effectuer impliquait une grosse partie de recherches à la fois sur le code existant pour déterminer la techno utilisée (par exemple la connection des users est gérée par FOS User Bundle) et à la fois sur internet afin de comprendre leur fonctionnement (la documentation Symfony est heureusement très complète et quelques youtubeurs comme Lior Chamla pour n'en citer qu'un font d'excellentes vidéos pour expliquer le fonctionnement de Symfony).

J'ai eu à réaliser une plateforme pour Manavao, accessible seulement aux clients, et permettant d'accéder à des fonctionnalités supplémentaires, j'ai donc ajouté de nouvelles routes vers les nouvelles pages, réalisé ces pages, restreint l'accès de ces pages aux clients, créé la table client en base de donnée puis l'ai liée à la table des users.

J'ai créé les tables nécessaires à l'attribution des contenus, les ai liées à la table user, puis j'ai commencé à remplir la page principale avec les données récupérées de Manavao.

Lors de ce stage j'ai donc eût l'occasion de découvrir et d'utiliser Symfony, j'ai donc pris connaissance de la structure d'un projet sous ce framework, vu les controllers, le système de rendus des vues en Twig, j'ai également vu les Entités et le système ORM de doctrine que j'ai trouvé extrêmement pratiques.

Je me suis servit des controllers pour injecter des données de la bdd aux différentes vues, j'ai pris connaissance du système de Rôles et des firewalls, je m'en suis servit pour restreindre l'accès à la nouvelle plateforme, j'ai créé une toute petite partie Admin permettant d'enregistrer un nouveau compte client, accessible seulement pour l'Admin et j'ai finalement commencé à implémenter des fonctionnalités en javascript, en faisant un peu d'AJAX et en utilisant un controller pour utiliser une requête DQL écrite dans un Repository d'Entité.

Tout le long du stage j'ai communiqué quotidiennement avec mon patron via Slack ou directement dans un espace de co-working, ai travaillé sur une branche que j'ai créé à partir de la branche Master, ai fait une pull request pour appliquer sur le site en production, une modification de css que le patron m'a demandé d'effectuer le premier jour, puis j'ai push régulièrement les avancées de mon travail sur la branche de développement que j'avais créé.

Le stage c'est mal déroulé et terminé car l'employeur espérais qu'à la fin, on puisse mettre en production une maquette fonctionnelle que j'aurais développé tout seul en un mois, ce qui était pour moi impossible, à trois mois de formation et sans encadrement technique, l'employeur ne m'a donc à ce jour pas signé les papiers de fin de stage.

Malgrés tout ça, j'ai put découvrir l'architecture symfony et m'en suis reservit par la suite pour faire des API ou des back-end avec une plateforme admin et j'ai put consolider mes connaissances en bases de données, donc ce stage a quand même été bénéfique pour moi.



4. Résumé du projet :

Le projet WorkWithTheBest a pour but d'être une plateforme de recrutement, permettant de poster des candidatures ou offres d'emplois, en les référençant à des villes ou des types d'activités, sur des périodes plus ou moins longues (un jour, un mois, un an...), ce qui permet de cibler tous types de contrats, de l'intérim au CDI.

On pourra ainsi rechercher des offres ou des candidatures d'emplois, par département, par région, par type d'activité, par durée ou par temporalité. On pourra modifier ou supprimer ces offres/candidatures que l'on fait, ainsi que les rendre actives ou inactives pour qu'elles puissent apparaître ou non sur le réseau.

Lorsque deux acteurs se mettront d'accord, ils signeront un contrat virtuel pour la durée de l'engagement, à la fin du contrat (et pas avant), chaque acteur pourra noter l'autre (une seule note par contrat) sur divers critères, pour un candidat, on notera la ponctualité, l'implication et le savoir-être, pour un employeur, on notera le salaire versé, la prise en compte des salariés et l'ambiance de travail dans la structure.

Ce système qui a pour risque majeur de faire fuir les acteurs mal notés (qui ne pourrions que constater leur défauts personnels et choisir de s'améliorer ou de rester fermés à la critique) a pour attrait la mise en avant des profils motivés ou des entreprises impliqués dans l'amélioration du tissu social (salaire équivalent au travail fournit, travailleur faisant montre d'initiative et qui ne passe pas la journée les mains dans les poches).

Deux acteurs ayant travaillé ensemble (via la plateforme WorkWithTheBest ou non) pourront créer un lien pour suivre plus facilement les offres/candidatures de l'autre partie, un lien créé permet une notation mutuelle comme après un contrat.

Un candidat à un poste pourra consulter les notes et profils des entreprises qui le contacte, afin de se faire une idée sur chacune des propositions qui lui seront faites.

De même une entreprise qui propose un emploi pourra consulter les notes des différents candidats, afin de choisir le profil qui lui semblera le mieux.

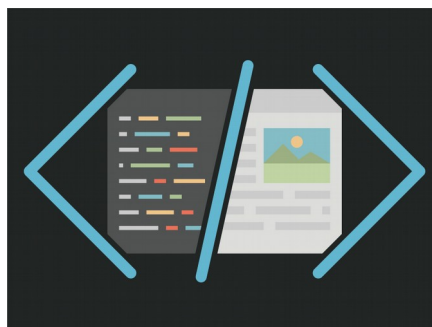
5. Compétences du REAC abordées :

-Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité :

- 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 ou e-commerce.

-Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité :

- Créer une base de données,
- Développer les composants d'accès aux données,
- Développer la partie back-end d'une application web ou web mobile
- Elaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce



6. Technologies utilisées :

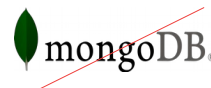
Back-end :

Serveur :



Ayant initialement choisis le framework symfony pour développer le back-end de mon application web, j'ai finalement décidé d'utiliser nodeJS et express, que je maîtrisais un petit peu moins bien, à fin de m'améliorer et de pouvoir concevoir toutes les logiques d'authentification, de routing et de sécurité « from scratch ».

Base de données :



Ce projet nécessitant énormément de relations entre les tables, j'ai décidé de m'orienter vers MySQL, que je maîtrisais bien, afin de ne pas perdre trop de temps à concevoir ma base de données, j'ai cependant fais des tests pour l'enregistrement et l'authentification des utilisateurs avec mongoDB et l'O.D.M mongoose.

Autres :



J'utilises le module jsonwebtoken pour générer mes tokens de sécurité, le module nodeMailer et le serveur de mail de google pour envoyer des mails, le module passwordHash pour hasher les mots de passe, il faut que je changes pour bcrypt et finalement j'utilises le module mysql pour me connecter à ma base de données et lui envoyer des requêtes, j'ai fais un essais avec sequelize mais ai préféré créer moi même mes requêtes, sans utiliser d'ORM.

Front-end :

Interface en React :



J'ai choisis le framework React, pour développer l'interface client de mon application web, j'aime le concept d'import de composants et de cycle de vie de ceux-ci. J'ai dû bricoler un peu pour parvenir à certains résultats, n'ayant pas sût trouver le temps d'apprendre à utiliser redux, mais je ne manquerais pas de me pencher dessus, pour pouvoir concevoir efficacement des applications monolithique.

Gestion des requêtes avec Axios :



J'utilises Axios pour effectuer les requêtes entre mon front et mon back, il permet d'utiliser les méthodes get, post, put et delete.

II. Définition du projet

1. Les acteurs :

-Jean Tasseau : Homme, 32 ans, Menuisier confirmé qui aime voyager, il a besoin de pouvoir trouver du travail aux différents endroits où il se rend, pour des durées variables, avec WorkWithTheBest, Jean renseigne les différentes localités où il sera prêt à aller travailler,

-Comptables Sans Frontières : Cette organisation proposant des services comptables aux entreprises de la région, cherche un comptable à placer dans une entreprise pour deux mois, elle veut quelqu'un de compétent, elle peut chercher parmi les profils comptables de WorkWithTheBest disponibles dans sa région, les comptables les mieux notés par les autres employeurs.

-Jeanne Jeannette : Femme, 26 ans, Jeune diplômée de master en économie sociale et solidaire, cherche un poste à la hauteur de ses talents, elle poste donc une offre en renseignant les lieux où elle voudrait travailler, elle pourra ensuite consulter les notes des entreprises qui la contacteront, afin de faire le meilleur choix possible.

-Gerald Geranium : Homme, 23ans, Jeune électricien, travaille pour une entreprise extraordinaire avec un salaire et des avantages plus que convenables, il souhaite faire valloir à quel point son entreprise est exceptionnelle et peut donc lui attribuer une bonne note dans les diverses catégories (salaire, ambiance de travail, qualité du management et prise en compte du travailleur), cela implique bien sûr que son entreprise ait un compte sur WorkWithTheBest et reconnaisse le lien salarié => employeur qui les réunis.

-Les Maçonneries DeLaBrouette : Maçonnerie, se sert souvent de WorkWithTheBest pour combler ses besoins en main d'oeuvre, elle est très bien notée par les différentes personnes qui y ont travaillé et a pour habitude de bien les noter car satisfait de leur travaux, Cette fois-ci ça c'est mal passé, La Maçonnerie peut donc laisser une mauvaise note sur le profil du travailleur concerné (ponctualité, comportement, qualité du travail et prise de conscience des attentes de l'entreprise).

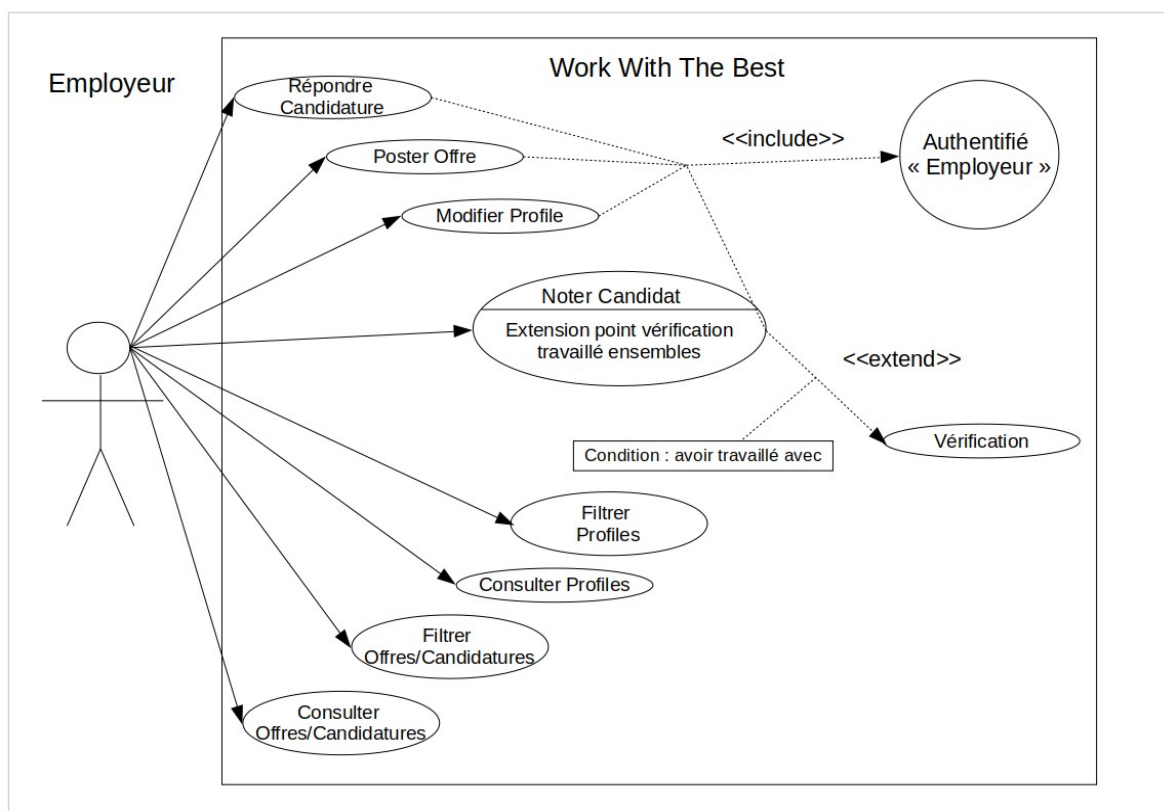
-Pappeterie LesPetitsPapiers : Entreprise de taille moyenne, a embauché Jade Piston il y a deux ans, à l'occasion d'un événement récurrent. Ces deux acteurs s'étant bien notés et ayant acceptés de se linker, l'entreprise peut voir que cette année, Jade sera disponible à la même période et peut donc lui proposer de revenir travailler pour eux.

2. Les use-cases :

L'application WorkWithTheBest pourra interagir avec trois types d'utilisateurs: les candidats, les employeurs et les visiteurs .

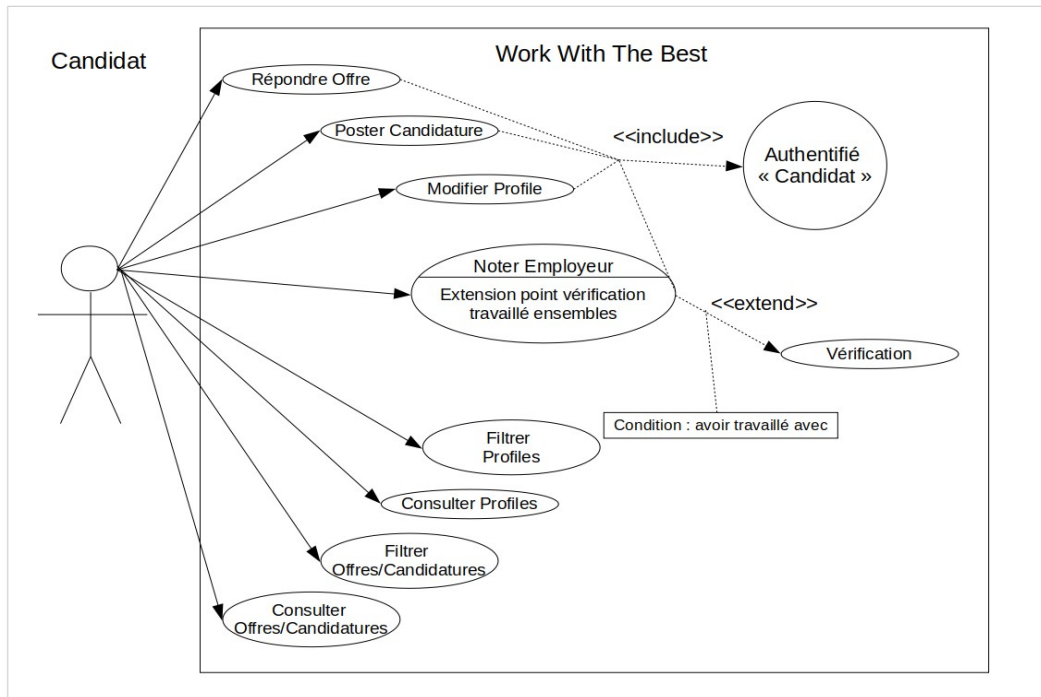
De base, les use-cases candidats et employeurs héritent des fonctionnalités du use-case visiteur: la consultation et filtration des candidatures et des offres, ainsi que la consultation des profiles candidats/employeurs.

Le use-case employeur :



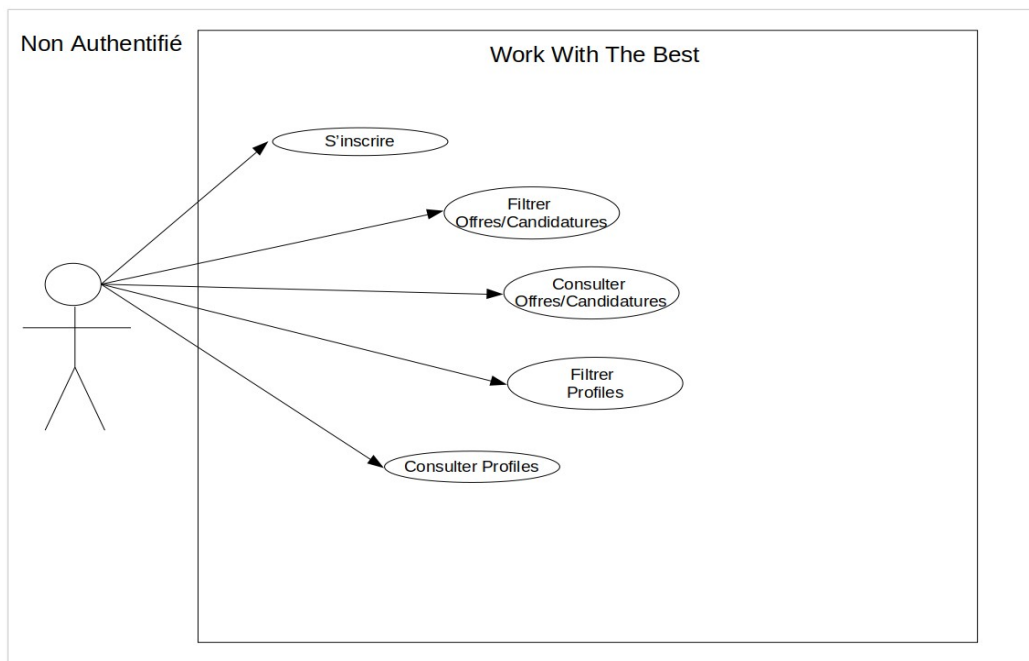
Un employeur pourra répondre à une candidature, poster une offre, modifier son profile ou gérer ses offres si il est connecté avec un compte employeur, de plus, si un contrat avec un candidat se termine il pourra si il le souhaite, noter ce candidat.

Le use-case Candidat :



Un candidat pourra répondre à une offre, poster une candidature, modifier son profile ou gérer ses candidatures si il est connecté avec un compte candidat, de plus, si un contrat avec un employeur se termine il pourra si il le souhaite, noter cet employeur.

Le use-case visiteur :

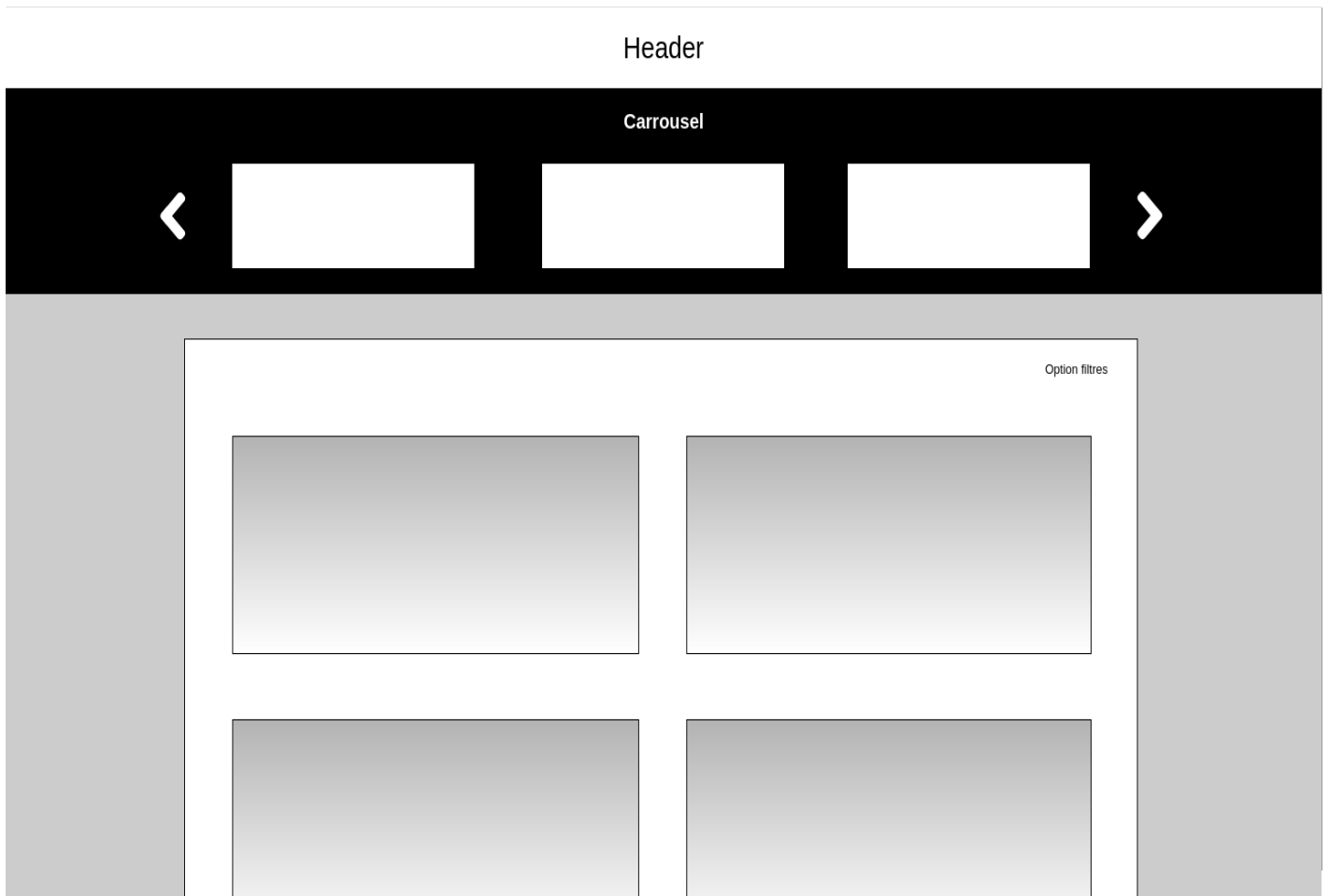


Un visiteur non authentifié pourra simplement s'inscrire ou voir et filtrer les offres/candidatures ainsi que les candidats/employeurs.

3. Les Wireframes :

Les wireframes ont été réalisés avec le logiciel pencil.

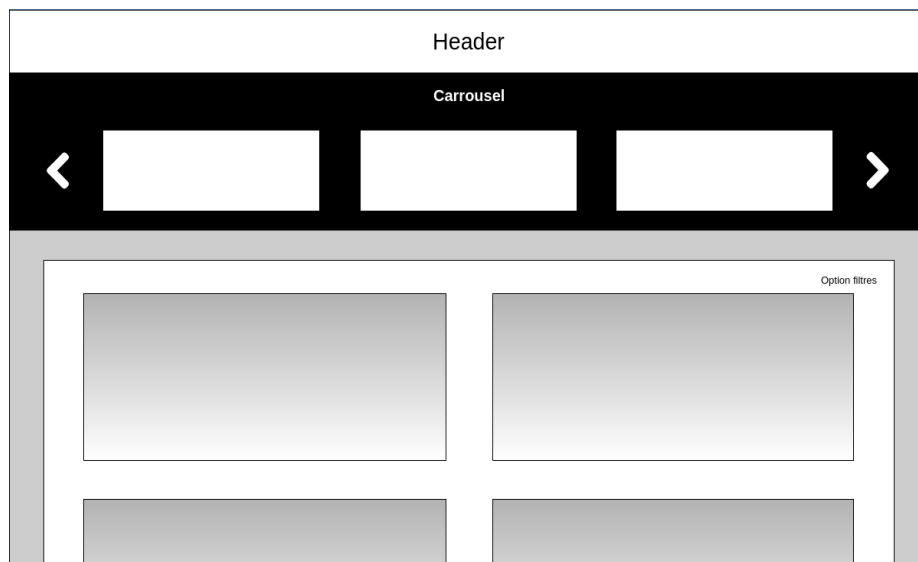
Wireframe pc :



Le wireframe pc du dashboard possède un header (qui contiendras la nav bar), un carrousel qui présenteras les offres ou candidatures, et une page qui afficheras les offres ou candidatures et qui permettras de filtrer celles-ci.

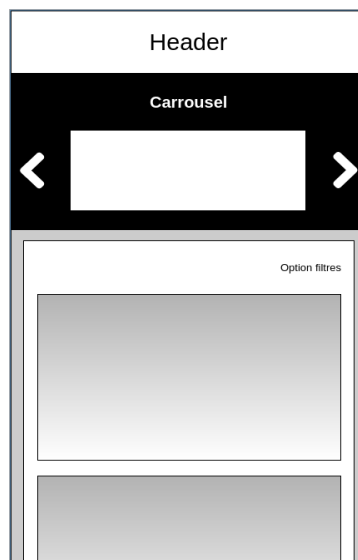
J'ai choisis de faire un affichage épuré, ainsi la nav bar se situe dans un menu déroulant pour ne pas encombrer la fenêtre du navigateur.

Wireframe tablette :



Le wireframe tablette reprends le design du wireframe pc, avec un focus sur le corps de la page, on accorde également plus de place au carrousel, proportionnellement à la page.

Wireframe smartphone :

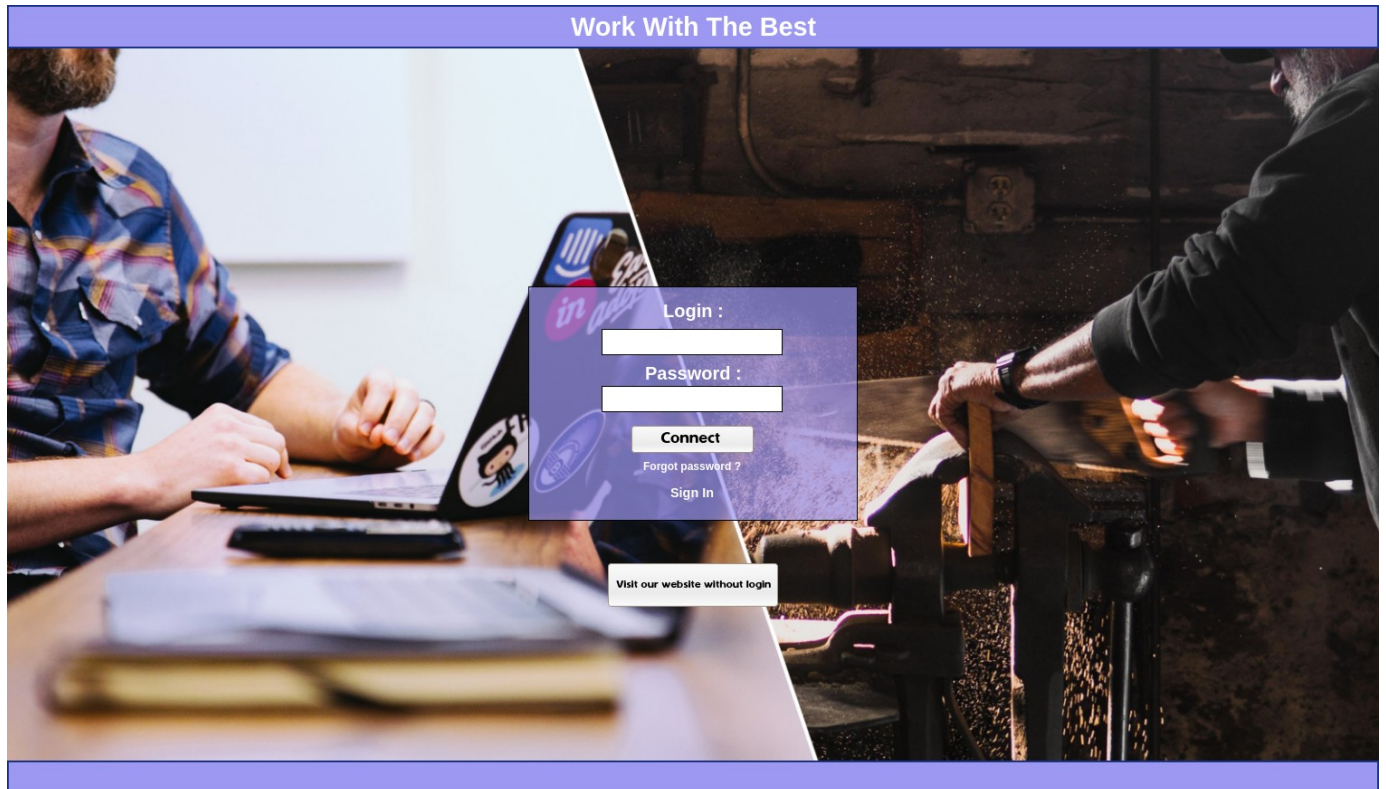


Le wireframe smartphone apporte quelques modifications, nous n'affichons plus qu'un seul élément dans le carrousel, et l'affichage des offres ou candidatures passe de deux à une colonne, il est également prévu de transformer la nav bar en menu hamburger.

4. Les maquettes :

Les maquettes ont été réalisées avec le logiciel pencil.

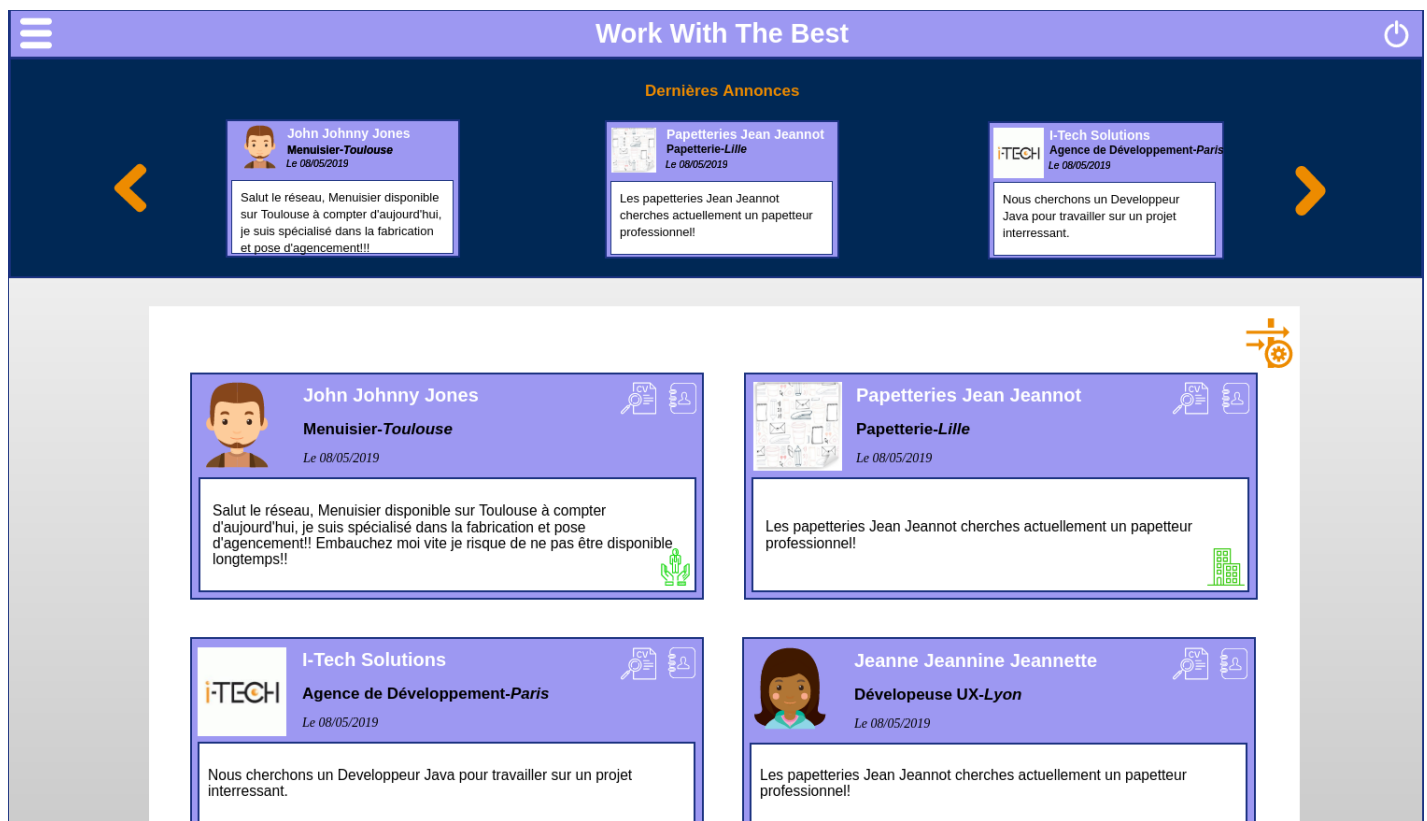
Maquette pc :



La landing page : Il s'agit de la première page du site sur laquelle les utilisateurs arriveront, le menu ne s'affiche pas, on peut choisir de se connecter, de créer un compte, ou bien de visiter le site avec un profile anonyme.

Suivre la route pour la création de compte nous mèneras à une page demandant de choisir entre la création d'un compte candidat ou d'un compte employeur.

La route pour visiter le site anonymement mèneras quand à elle à une page dashboard permettant seulement de consulter ou filtrer les offres et candidatures.

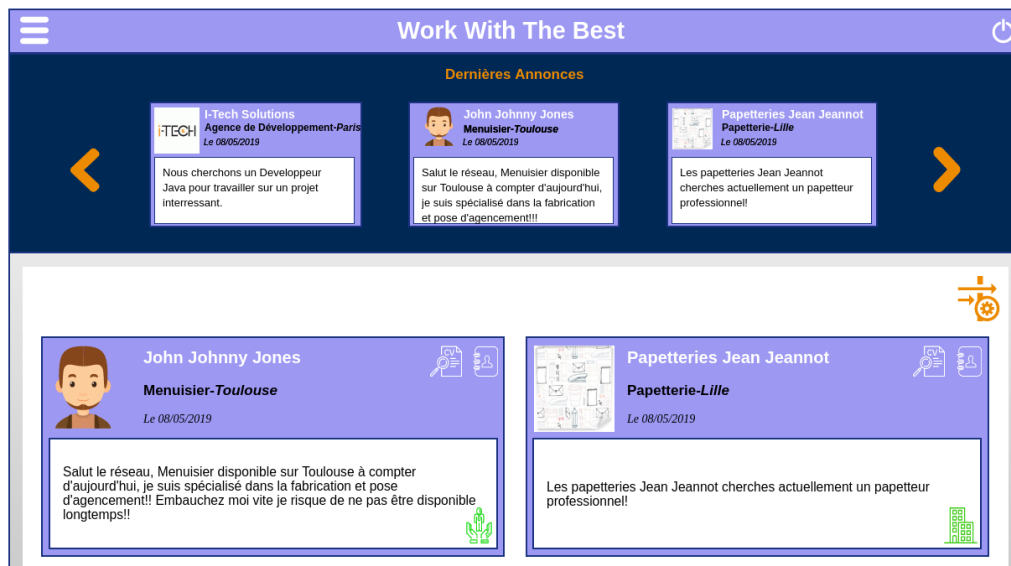


Le dashboard : Le dashboard sera une page possédant un carrousel qui affichera les dernières annonces postées, ainsi qu'un corps de page permettant d'afficher les annonces en les filtrant.

Les annonces sur le dashboard possèdent des liens pour consulter le profil de celui qui poste l'annonce ou lui signaler notre intérêt.

Le menu affiche un bouton pour faire descendre la nav bar et un bouton pour se déconnecter, la nav bar contiendra une route vers un formulaire pour créer une nouvelle offre, ainsi qu'une route pour consulter et gérer les offres que l'on a déjà posté.

Maquette Tablette :



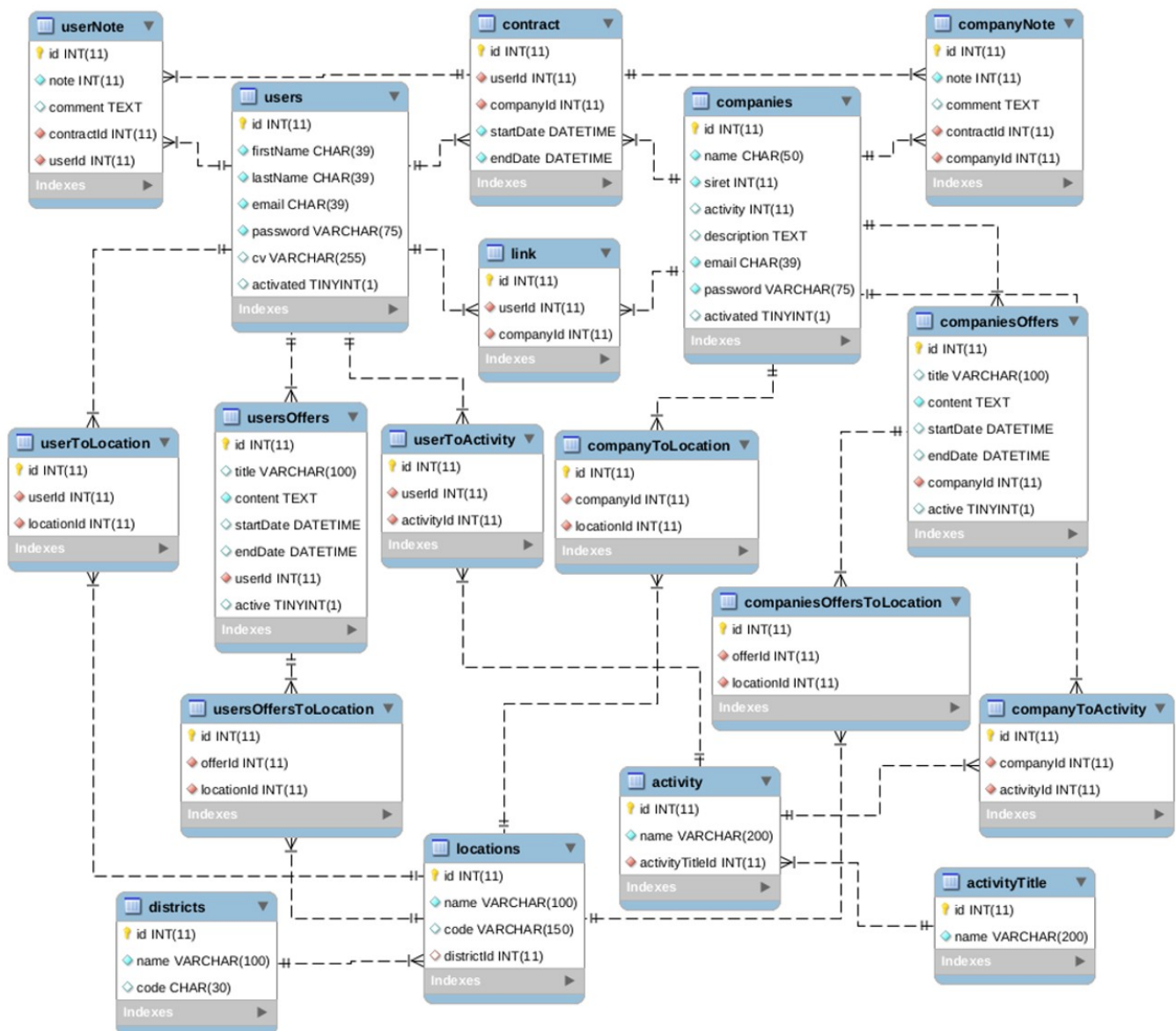
Le dashboard : Similaire à la maquette pc, on focalise sur le corps de la page et on accorde un peu plus d'importance au carousel.

Maquette smartphone :



Le dashboard : On redispone les éléments, on affiche plus qu'une seule offre à la fois dans le carousel, et l'affichage des offres passe de deux à une colonne.

5. Schéma relationnel de la base de données :



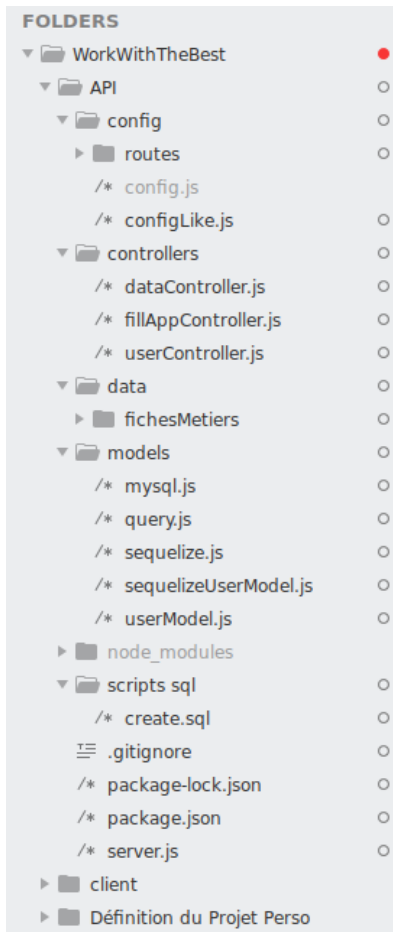
Le schéma de la base de donnée de Work With The Best est assez complexe, puisque chaque table est liée à une autre, et que certaines relations nécessitent la création de tables de jointures.

J'ai choisis de séparer toutes les tables candidats et employeurs, pour éviter de faire des tables avec des champs qui seront systématiquement laissés vide en fonction du rôle de l'utilisateur, cette structure complexifie la création de ma base de données, mais simplifie mes requêtes par la suite.

III. Back-end

1. Mise en place du serveur back-end avec express et nodeJs :

Architecture de l'API :



Voici l'arborescence des fichiers de mon back-end, à la racine, le fichier `server.js`, qui permet d'initialiser serveur et routes, `package.json` qui permet de déclarer les modules dont dépend le fonctionnement de notre API, pour qu'ils s'installent automatiquement avec la commande `npm install` et finalement le `.gitignore` qui permet d'ignorer certains dossiers ou fichiers (grisés sur l'arborescence à gauche), pour ne pas les push sur le repository distant.

Cela permet notamment d'ignorer le node module qui est volumineux et qui s'installera automatiquement quand le contributeur lancera la commande `npm install`.

On trouve ensuite le dossier `config`, qui contient le routeur, ainsi que les fichiers de config, l'un contenant la config utilisée et ignorée par le `gitignore`, l'autre représentant un fichier de config à remplir.

Viens le dossier des contrôleurs, qui contient un contrôleur pour gérer les logiques d'inscription, de connexion et d'autorisations d'accès, un contrôleur pour gérer le contenu de l'application et des utilisateurs et finalement un contrôleur que j'utilise pour enregistrer mes jeux de données et de fausses données en base.

Le dossier `data` contient des fiches métiers xml que j'exploite avec mon contrôleur `data`.

Le dossier `models` contient mon fichier de connexion à la base de données et mon fichier de requêtes, il contient également les modèles `mongoose` et `sequelize` que j'avais testé.

Le dernier dossier, `scripts sql`, contient le script de création de la base de données.

Server.js :

J'ai donc choisis d'utiliser nodeJs et le module Express pour réaliser l'architecture de mon back-end. Pour les mettre en place, il faut les initialiser dans un fichier server.js :

```
//Modules utilisés
const express = require('express');
//const mongoose = require('mongoose');
const bodyParser = require('body-parser');

//mongoose.connect('mongodb://localhost/WWTBDdb', {useCreateIndex: true, useNewUrlParser: true});

//On définit express dans notre constante "app"
const app = express();

//On prépare le body parser
const urlencoded = bodyParser.urlencoded({extended:true});

app.use(urlencoded);
app.use(bodyParser.json());

app.use(function(req, res, next){
  res.setHeader('Access-Control-Allow-Headers', 'X-Requested-With, content-type');
  res.setHeader('Access-Control-Allow-Origin', 'http://localhost:3000');
  res.setHeader('Access-Control-Allow-Methods', 'GET, POST, Options, PUT, DELETE');
  res.setHeader('Access-Control-Allow-Credentials', true);
  next();
});

const userRoute = require('./config/routes/user');
userRoute(app);

//à commenter une fois les datas insérées
const dataRoute = require('./config/routes/data');
dataRoute(app);

const fillAppRoute = require('./config/routes/fillApp');
fillAppRoute(app);
//Mise en place du port d'écoute
app.listen(8000, () => console.log('Listening on port 8000'));
```

Ici on requiert express et body-parser avec la commande 'require()', puis on initialise express en l'utilisant dans notre constante app, on prépare le body-parser et on l'utilise avec express.

Puis on déclare les headers pour définir les règles d'accès à notre API, ici on autorise l'accès seulement au localhost:3000, qui sera le serveur de notre front, et on autorise les méthodes get, post, options, put et delete.

On déclare nos routes, en requérant nos routeurs et en leur donnant app en paramètre.

Finalement on demande à notre API de se lancer sur le localhost:8000 avec la commande 'app.listen()'.

Les routeurs :

Il y a trois routeurs qui gèrent les requêtes faites à l'API, leur structure est similaire.

```
module.exports = function(app) {  
  const fillAppController = require('../controllers/fillAppController.js');  
  
  app.route('/carrousel')  
    .post(fillAppController.carrousel);  
  
  app.route('/wordResearch')  
    .post(fillAppController.wordResearch);  
  
  app.route('/addOffer')  
    .post(fillAppController.addOffer);  
  
  app.route('/offers')  
    .post(fillAppController.offers);  
  
  app.route('/getOffer')  
    .post(fillAppController.getOffer);  
  
  app.route('/updateOffer')  
    .put(fillAppController.updateOffer);  
}
```

Ici on récupère notre constante app passée en paramètre dans le fichier server.js, elle représente une instance d'express.

On requiert le controleur qui gèreras ce qu'il se passeras sur nos routes.

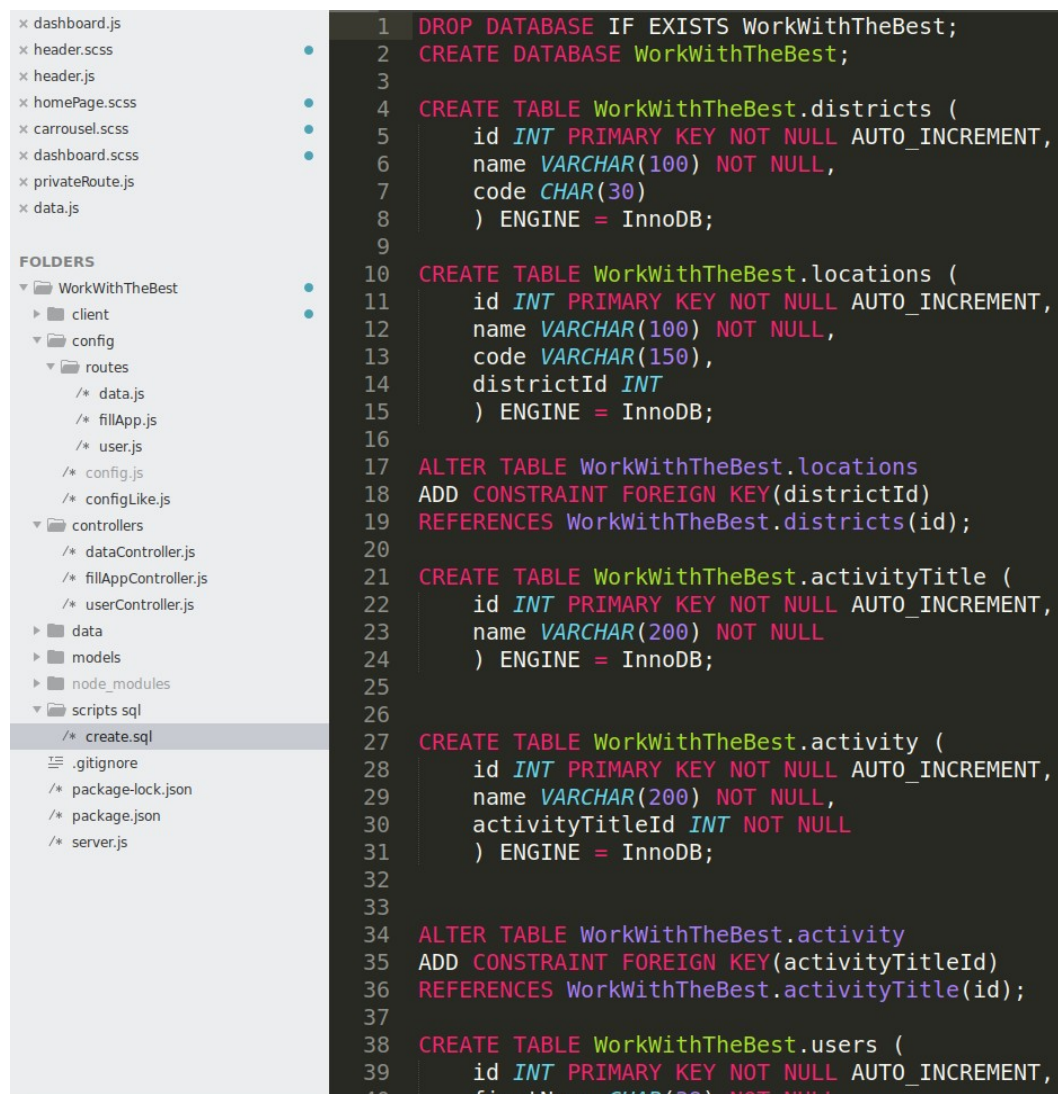
On utilise la méthode route de express pour lui dire de surveiller une url spécifique, puis on déclare la méthode d'accès que l'on s'attend à recevoir (post, put, get etc...) ainsi que la méthode du controleur à utiliser sur cette url si la méthode d'accès correspond.

Pour désactiver les routes qui permettent de remplir la base de données au lancement du projet, il suffit de commenter les lignes qui requierent et utilisent notre data routeur dans le fichier server.js.

2. Implémentation et échanges avec la base de données :

Création de la base de données :

J'ai réalisé un script de création de données nommé 'create.sql' situé dans le dossier script sql, pour l'exécuter, il suffit, dans son terminal, de se placer dans le dossier script sql, et de lancer la commande 'mysql -u 'nom d'utilisateur' -p < create.sql', ceci créera toutes les tables en base de données ainsi que les relations et tables de jointure.



```
1 DROP DATABASE IF EXISTS WorkWithTheBest;
2 CREATE DATABASE WorkWithTheBest;
3
4 CREATE TABLE WorkWithTheBest.districts (
5     id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
6     name VARCHAR(100) NOT NULL,
7     code CHAR(30)
8 ) ENGINE = InnoDB;
9
10 CREATE TABLE WorkWithTheBest.locations (
11     id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
12     name VARCHAR(100) NOT NULL,
13     code VARCHAR(150),
14     districtId INT
15 ) ENGINE = InnoDB;
16
17 ALTER TABLE WorkWithTheBest.locations
18 ADD CONSTRAINT FOREIGN KEY(districtId)
19 REFERENCES WorkWithTheBest.districts(id);
20
21 CREATE TABLE WorkWithTheBest.activityTitle (
22     id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
23     name VARCHAR(200) NOT NULL
24 ) ENGINE = InnoDB;
25
26
27 CREATE TABLE WorkWithTheBest.activity (
28     id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
29     name VARCHAR(200) NOT NULL,
30     activityTitleId INT NOT NULL
31 ) ENGINE = InnoDB;
32
33
34 ALTER TABLE WorkWithTheBest.activity
35 ADD CONSTRAINT FOREIGN KEY(activityTitleId)
36 REFERENCES WorkWithTheBest.activityTitle(id);
37
38 CREATE TABLE WorkWithTheBest.users (
39     id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
40     firstName CHAR(30) NOT NULL
```

The screenshot shows a code editor with a file explorer on the left. The file explorer lists files like dashboard.js, header.scss, and folders like WorkWithTheBest, client, config, routes, controllers, data, models, node_modules, and scripts sql. The 'scripts sql' folder is expanded, showing 'create.sql'. The main editor area displays the SQL script for creating the database and tables, including foreign key constraints.

Ce script supprime la base de donnée work with the best si elle existe, pour la créer ensuite. On utilise le Sql 'CREATE TABLE' pour déclarer les tables et leurs champs, puis on utilise les commandes 'ALTER TABLE' et 'ADD CONSTRAINT' pour déclarer les clés étrangères permettant de lier les tables.

Connexion avec le serveur Mysql :

Pour être capable d'interagir avec notre base de données, il faut d'abord s'y connecter, pour se faire on écrit le fichier 'mysql.js' dans le dossier 'models' !

```
const mysql = require('mysql');

const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'root',
  database: 'WorkWithTheBest'
});

connection.connect(function(err){
  if (err) throw err;
  console.log('db connected')
})

module.exports = connection;
```

Pour se connecter à notre serveur mysql, on commence par récupérer le module mysql, puis on déclare une constante connection, dans laquelle on précise les paramètres d'accès à la base de données à l'aide de la méthode 'createConnection()' de mysql.

On utilise ensuite la méthode 'connect()' sur notre constante afin d'initialiser la connection entre notre base de données et notre serveur back-end.

Finalement on exporte la constante connection que l'on utilisera plus tard pour faire nos requêtes

Pour sécuriser la base de données, il est conseillé d'utiliser une connexion mysql avec un profile à droits restreints pour ajouter ou consulter des données, et une connexion avec un profile à droits supérieurs pour modifier ou supprimer des données

Des requêtes dynamiques :

Ayant choisis de ne pas passer par un ORM mais plutôt de composer moi-même mes requêtes, j'ai décidé de créer des requêtes dynamiques. M'inspirant du nom des méthodes des ORM, j'ai nommé mes requêtes dynamiques 'find()', 'create()', 'update()' et 'delete()'.

Chaque information reçue depuis le front est traitée par une fonction 'mysqlEscape()' afin d'empêcher les injections sql.

La fonction 'find()' :

```
exports.find = function(options, results) {
  let query = 'SELECT '
  if (options.distinct) query += 'DISTINCT ';
  query += options.fields+ ' FROM '+options.table;
  if (options.innerJoin) query += innerJoin(options.innerJoin);
  if (options.where) query += where(options.where);
  if (options.orderBy) query += ' ORDER BY '+options.orderBy.field+ ' '+options.orderBy.order;
  if (options.limit) query += ' LIMIT '+options.limit;
  console.log(query);
  return db.query(query, function(err, data){
    if (err) return results(err, null);
    else if(data.length === 0) return results(null, false);
    else if(data.length === 1) return results(null, data[0]);
    else {
      let total = [];
      data.forEach(function(item){
        let result = {};
        for (let prop in item){
          result[prop] = item[prop];
        }
        total.push(result);
      });
      return results(null, total);
    }
  });
}
```

Cette fonction, qui est composée de beaucoup de conditions qui la rendent modulable, peut composer une requête simple, comme une très complexe.

Concrètement on déclare notre variable qui sera la requête, puis par rapport aux options passées au moment où on appelle la fonction, la requête se constitue au fur et à mesure.

Une fois que la requête est constituée, nous utilisons notre connection mysql vu précédemment (attribuée à la constante db), pour effectuer une requête au serveur mysql, on lui passe notre requête par la variable query, puis on utilise une fonction callback pour traiter les résultats.

Si il se produit une erreur, on la renvoie avec un resultat null.

Si il n'y a pas de resultat on retourne une erreur nulle et un resultat faux.

Si il n'y a qu'un resultat, on retourne une erreur et ce seul résultat sous forme de row data packet.

Si on obtiens plusieurs résultats, on les traite pour les extraire du row data packet en copiant les données sur un nouvel objet, puis on enregistre chaque objet dans un tableau avant de renvoyer une erreur null et le tableau de résultats.

Tandis que la requête se constitue, elle passe par des conditions qui renvoient à une fonction montant une partie de la requête, comme la fonction 'innerJoin()' :

```
if (options.innerJoin) query += innerJoin(options.innerJoin);
```

innerJoin est une propriété de l'objet 'options' passé en paramètre à la fonction 'find()'.

```
function innerJoin(innerJ) {  
  let query = '';  
  for (let prop in innerJ) {  
    query += ' INNER JOIN '+innerJ[prop].table+' ON '+innerJ[prop].on;  
  }  
  return query;  
}
```

Ici on récupère les paramètres passés dans notre fonction innerJoin avec la variable innerJ.

On déclare une variable query qui recevra le morceau de requête à retourner, puis pour chaque inner join à faire, on ajoute à la requête une ligne 'INNER JOIN' avec les conditions du 'ON' que l'on a paramétré .

InnerJ étant un objet d'objets (chaque propriété de innerJ est un objet permettant le paramétrage d'un innerJoin) on itère dedans en utilisant la boucle 'for in'.

L'appel de la fonction 'find()' :

Maintenant que la fonction find est déclarée, il suffit de l'appeler en lui passant plus ou moins de paramètres :

Ici un appel simple, on déclare dans un objet nommé 'params' la propriété 'table' sur laquelle on veut requêter (ici offer.type + 'Offers' renverras 'usersOffers' ou 'companiesOffers'), la propriété 'fields' pour les champs que l'on veut récupérer et la propriété 'where' pour les conditions.

```
let promise1 = new Promise(function(resolve, reject){
  let params = {
    table: offer.type+'Offers',
    fields: 'title, content, startDate, endDate, ownerId, active',
    where: {[offer.type+'Offers.id']: offer.id},
  }

  query.find(params, function(err, data){
    if (err) reject(err);
    else resolve(data);
  });
});
```

Cette configuration renvoie la requête suivante :

```
SELECT title, content, startDate, endDate, ownerId, active
FROM usersOffers WHERE usersOffers.id = '6'
```

Et donne le résultat suivant :

```
RowDataPacket {
  title: 'Développeur Web',
  content: 'Dev web junior cherches une alternance près de Paris',
  startDate: 2019-09-17T22:00:00.000Z,
  endDate: 2020-08-29T22:00:00.000Z,
  ownerId: 3,
  active: 1
}
```

Puisqu'il n'y a qu'un seul résultat, on reçoit un row data packet.

Nous allons maintenant voir une requête plus complexe, la base reste la même, on déclare les propriétés 'table' et 'fields' de notre objet 'params', puis on lui déclare des propriétés supplémentaires, ici, une propriété innerJoin constituée de trois propriétés représentant chacune un innerJoin à faire, une propriété 'orderBy' pour ordonner les résultats par ordre décroissant d'id, puis une propriété 'limit' valant 5 pour limiter le nombre de résultats à 5.

```
let params = {
  distinct: true,
  fields: req.body.type+'Offers.ownerId, '+req.body.type+'Offers.id, title, content, '+name+', locations.name AS location, startDate, endDate',
  table: req.body.type+'Offers',
  innerJoin: {
    first:{table: req.body.type, on: req.body.type+'.id = '+req.body.type+'Offers.ownerId'},
    second:{table: req.body.type+'OffersToLocation AS ToLoc', on: req.body.type+'Offers.id = ToLoc.offerId'},
    third:{table: 'locations', on: 'ToLoc.locationId = locations.id'}
  },
  where:{active: 1},
  orderBy: {field: req.body.type+'Offers.Id', order: 'DESC'},
  limit: 5,
};

query.find(params, function(err, data){
```

Cette configuration renvoie la requête suivante :

```
SELECT DISTINCT companiesOffers.ownerId, companiesOffers.id, title, content, companies.name, locations.name AS location, startDate, endDate FROM companiesOffers
INNER JOIN companies ON companies.id = companiesOffers.ownerId INNER JOIN companiesOffersToLocation AS ToLoc ON companiesOffers.id = ToLoc.offerId INNER JOIN locations ON ToLoc.locationId = locations.id WHERE active = '1' ORDER BY companiesOffers.Id DESC LIMIT 5
```

Et donne les résultats suivants :

Ici puisqu'il y a plusieurs résultats, on ne reçoit plus un row data packet mais un tableau d'objets.

```
{
  ownerId: 6,
  id: 4,
  title: 'Lorem Ipsum',
  content: 'recherchons Ipsum Lorem pour IpsumLoremopsum',
  name: 'La compagnie Gengis Khan',
  location: 'Doméliers',
  startDate: 2019-07-29T23:58:44.000Z,
  endDate: 2019-08-28T23:58:44.000Z
},
{
  ownerId: 4,
  id: 3,
  title: 'Animateur(trice) en aqua gymnastique',
  content: 'recherchons animateur(trice) pour un ' +
    "poste dans notre superbe spa, le Mamy\\'s " +
    'SPA',
  name: "Mamy\\'s SPA",
  location: 'Salans',
  startDate: 2019-07-29T23:58:44.000Z,
  endDate: 2019-08-28T23:58:44.000Z
},
{
  ownerId: 3,
```

3. Enregistrement des comptes d'utilisateurs :

L'enregistrement des comptes des utilisateurs s'effectue depuis le 'userController', sur la route localhost:8000/signUp.

Concrètement il faut utiliser la méthode 'POST' pour envoyer le profile user sur la route adéquate, et le controleur se chargeras du reste :

```
exports.signUp = function(req, res) {
  req.body.password = pH.generate(req.body.password);
  let type = req.body.type;
  delete req.body.type;
  let params = {fields: 'id', table: type, where:{email: req.body.email}};
  query.find(params, function(err, user){
    if (err) res.status(400).json(err);
    else if(!user) {
      let params = {table: type, fields: req.body};
      query.create(params, function(err, data){
        if (err) res.status(400).json(err);
        else {
          let token = jwt.sign({id: data.insertId, table: type}, config.SECRET);
          sendValidationMail(req.body.email, token);
          res.status(201).json({created: true, message:data});
        }
      });
    }
  });
  else res.status(200).json({created: false, message: "This Account already exists"});
};
```

Cette fonction signUp est dynamique, elle gère à la fois l'enregistrement d'un compte candidat comme l'enregistrement d'un compte employeur, puisqu'il s'agit d'une méthode d'un controleur, on récupères en paramètre les objets 'request' et 'response' qui nous sont nécessaires.

Les informations étant vérifiées via le front, le controleur ne re vérifies pas l'intégrité des données.

On commence par hasher le mot de passe avec le module passwordHash, on attribue le type de table (users ou companies) à une variable 'type', pour supprimer cette propriété de notre request.body.

Puis on recherche avec l'email si un utilisateur existe dans la table concernée, si il existe déjà, on renvoie l'objet response avec un statut 200 et un message d'erreur 'this account already exists', si il n'existe pas, on enregistre un nouveau compte dans la table.

Le compte est créé non-activé par défaut, on génère un token d'accès contenant l'id de l'utilisateur créé et la table concernée ('users' ou 'companies'), puis on l'envoie dans un lien, que l'utilisateur pourra cliquer pour activer son compte.

Une fois que l'utilisateur aura cliqué sur le lien reçu par email, une page s'ouvrira dans son navigateur vers la route 'localhost:8000/activateAccount', prise en charge également par le 'UserController' qui s'occupera de la validation du compte, puis qui redirigera vers la page d'accueil de Work With The Best si la validation de son compte fonctionne, ou sur une page affichant un message d'erreur 'Authentication Faillure, maybe the link that you followed is expired', si le token est invalide.

```
exports.activateAccount = function(req, res) {
  let token = req.query.token;
  jwt.verify(token, config.SECRET, function(err, decoded){
    if (err) res.status(400).json('Authentication Faillure, maybe the link that you followed is expired :3');
    else {
      query.update({table: decoded.table, fields: {activated: 1}, where:{id: decoded.id}})
        .then(() => {
          res.redirect('http://localhost:3000/');
        })
        .catch((err) => {
          console.log(err);
          res.status(400).json('Activation faillure, something went wrong');
        })
    }
  })
}
```

La fonction 'activateAccount()':

On récupère dans 'request.query' le paramètre 'token' qu'on a passé en 'GET', puis on vérifie la validité du token avec le même secret qui l'a encodé.

Si le token est invalide, on renvoie un status 400 avec un message d'erreur, si le token est valide, on met à jour le profil de l'utilisateur en base de données.

On passe le champ 'activated' à true, en utilisant les informations récupérées dans le token (la table à mettre à jour ainsi que l'id de l'utilisateur).

Si la mise à jour en base de données réussit, on est redirigé vers l'accueil du site Work With The Best, si la query échoue, on renvoie un status 400 avec un message d'erreur.

Le lien est envoyé par la fonction suivante ‘sendValidationMail()’ :

```
function sendValidationMail(mail, token) {
  let transporter = nodeMailer.createTransport({
    service: 'gmail',
    auth: {
      user: config.MAIL,
      pass: config.GMAILKEY
    }
  })
  let mailConfig = {
    from: '<noreply>',
    to: mail,
    subject: 'Account validation WorkWithTheBest',
    html: '<!DOCTYPE html>'+
      '<html lang="en">'+
      '<head>'+
      '<meta charset="utf-8" />'+
      '</head>'+
      '<body>'+
      '<div><p>Congratulations, your registration on WorkWithTheBest is almost done.<br />'+
      'Click on the link below to validate your account!</p><br />'+
      '<a href="http://localhost:8000/activateAccount?token='+token+'">Click Here</a>'+
      '</div>'+
      '</body>'+
      '</html>'
  }
  transporter.sendMail(mailConfig, function(error, info){
    if(error){
      return console.log(error);
    }
  });
  transporter.close();
}
```

La fonction sendValidationMail() fonctionne grâce au module nodeMailer, on crée une instance en utilisant la méthode ‘createTransport()’, on précise le serveur de mail utilisé (ici j’utilise le serveur gmail avec mon compte personnel), on précise les identifiants de connexion, puis on stock le tout dans une variable ‘transporter’.

On déclare ensuite les paramètres d’envoi du mail ainsi que le template du mail (qu’il est également possible d’importer pour réduire la taille de la fonction et augmenter sa lisibilité).

Puis on utilise notre variable transporter et sa méthode ‘sendMail()’ pour envoyer le mail en lui passant les paramètres déclarés au dessus, avant de clore notre connexion avec le serveur de mail grâce à la méthode ‘close()’ de notre instance transporter.

Pour utiliser le serveur d’envoi de mail de google, il faut posséder un compte google, et le configurer pour obtenir une clé API pour votre application.

4. Connexion des utilisateurs et Sécurité :

La sécurisation d'une application implique de trouver un moyen de protéger certains accès et d'authentifier les utilisateurs qui veulent y accéder.

J'ai choisis d'utiliser le système des json web tokens, qui permet de ne pas avoir de système de sessions sur le serveur back, concrètement, on stocke un token, avec une validité limitée dans le temps, sur le navigateur de l'utilisateur (soit dans un cookie, soit dans le local storage mais ce dernier est déconseillé).

Quand un utilisateur voudra accéder à certaines fonctionnalités du site (une page protégée, un formulaire à envoyer), son token sera envoyé au back pour être vérifié, si le temps du token a expiré ou si le token est un faux, l'utilisateur ne pourra pas accéder à la fonctionnalité souhaitée, ainsi les marges de manœuvres d'un utilisateur mal intentionné sont réduites.

J'ai choisis d'attribuer une validité de 15 minutes aux tokens, pour que l'utilisateur n'ait pas à se reconnecter tout le temps, à chaque vérification de token, si il reste moins de 5 minutes de validité au token, on génère un nouveau token valide pour 15 minutes, que l'on renverra à la place de l'ancien.

La connexion des comptes utilisateurs se fait depuis le 'userController', sur la route localhost:8000/login.

```
exports.login = function(req, res) {
  let params = {fields: '*', table: req.body.type, where:{email: req.body.email}};
  query.find(params, function(err, user){
    console.log(user);
    if (err) res.status(400).json(err);
    else if (!user || user.activated === 0) res.status(200).json({message: 'This user does not exists'});
    else if (pH.verify(req.body.password, user.password)) {
      let name = (user.firstName)?user.firstName+' '+user.lastName: user.name;
      let token = jwt.sign({logged: true, id:user.id, name: name, role: req.body.type}, config.SECRET);
      res.status(200).json({authenticate: true, token: token, message: 'Successfully connected'});
    }
    else res.status(200).json({authenticate: false, message: 'Incorrect password'});
  });
}
```

Ici on paramètre notre recherche avec la table de l'utilisateur recherché et son email, on lance la recherche, si il se passe une erreur, on renvoie un status 400 avec l'erreur.

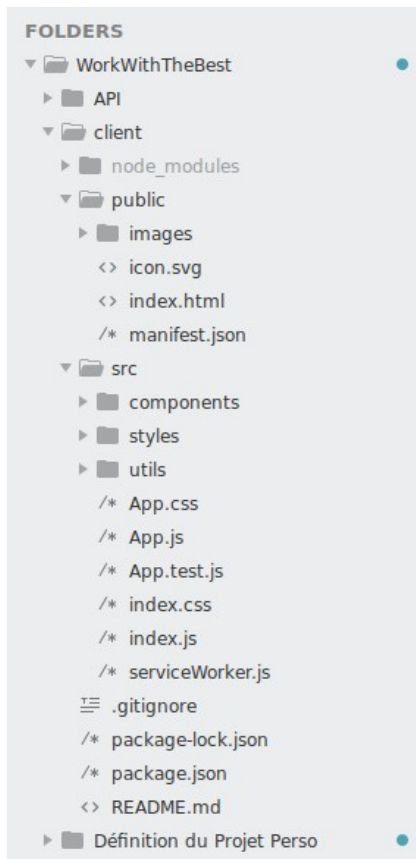
Si on ne trouve pas d'utilisateur ou si l'utilisateur n'a pas été activé, on renvoie un statut 200 avec un message 'This user does not exists'.

Si on trouve un utilisateur actif, on vérifie son mot de passe avec la méthode 'verify' de passwordHash, si le mot de passe est invalide, on renvoie un status 200 avec le message 'incorrect password', si le mot de passe est valide, on génère un token avec les infos de l'utilisateur, que l'on renvoie dans un status 200.

IV. Front-end

1. Mise en place de l'architecture Front-end en React :

Architecture de l'application :



React est une bibliothèque javascript libre, développée par l'équipe de facebook, dont le but est de générer une application monopage, grâce à la création de composants ou 'components' dépendants d'un état, qui génèrent une page html à chaque changement d'état.

En procédant ainsi, on limite les ressources utilisées par le navigateur, qui ne recharge la page que quand cela est nécessaire.

Voici l'arborescence des fichiers de mon front-end en React, à la racine, le fichier index.js qui initialise notre application et le fichier app.js qui est un component nous permettant de définir les différentes routes de l'application.

Toujours à la racine, le package.json et le .gitignore, qui fonctionnent comme précédemment présenté dans la partie back-end.

Viens ensuite le dossier 'public', contenant nos images, ou autres documents utilisés dans l'application, on y trouve notamment le fichier 'index.html', à partir duquel React a accès au navigateur.

Notre application accède aux fichiers contenus dans le dossier 'public', grâce à la variable 'process.env.PUBLIC_URL'.

Finalement le dossier où tout se passe, 'src', il contient les fichiers de nos composants React, dans un sous dossier nommé 'components', il contient également les fichiers scss de nos composants React, dans un sous dossier nommé 'styles' et il contient finalement le fichier 'API.js', qui nous permettra d'avoir des échanges avec notre serveur back-end, dans un sous dossier nommé 'utils'.

Mise en place :

Pour démarrer ce projet, je me suis placé à la racine de mon projet avec mon terminal et ai utilisé la commande 'npx create-react-app client', qui a créé un dossier client contenant l'architecture de base d'un projet React.

Le fichier index.js est le fichier qui assure le bon fonctionnement de React, ce fichier importe React et ReactDOM ainsi que le composant App et utilise la méthode 'render()' de ReactDOM pour afficher le contenu de notre composant App, dans l'élément html du DOM possédant l'id 'root'.

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';
import {BrowserRouter as Router} from 'react-router-dom';
ReactDOM.render(<Router><App /></Router>, document.getElementById('root'));

// If you want your app to work offline and load faster, you can change
// unregister() to register() below. Note this comes with some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

Le composant 'App' est ici enrobé par le composant 'BrowserRouter' de react-router-dom, cela me permet plus tard d'afficher différents composants en fonction de l'url et ainsi de simuler une plateforme multipage.

Utils :

Le dossier 'utils' contient le fichier API.js, ce fichier contient les méthodes de connexion à notre API.

Pour effectuer les requêtes vers le back-end de l'application, j'utilise le middleware axios, j'écris les différentes connexions dont j'ai besoin et je les appelle ensuite dans les composants quand c'est nécessaire.

En retournant une méthode de axios, on retourne une promesse, après l'appel de la fonction updateOffer, il faudra traiter les résultats dans un '.then()' et les erreurs dans un '.catch()'.

```
updateOffer: function(body) {
  return axios.put(path+'/updateOffer', body, {headers});
},
```

Cette fonction prend 'body' en paramètre, il s'agit du 'request.body' que l'on récupéreras dans le back-end.

2. Routes publiques et routes privées :

Les routes publiques:

C'est le composant 'react-router-dom' mentionné précédemment qui gère le système de routes de l'application, il n'est pas installé dans la base d'un projet react, il faut donc le récupérer avec la commande 'npm install --save react-router-dom'.

On enrobe ensuite le composant App du BrowserRouter comme vu précédemment. Enfin, dans notre fichier app.js, on déclare les différentes routes dans une balise 'switch'.

```
import React, {Component} from 'react';
import {Route, Switch} from 'react-router-dom';
import Dashboard from './components/dashboard.js';
import HomePage from './components/homePage.js';

import './styles/styles.scss';

export default class App extends Component {

  render() {
    return (
      <div className="App">
        <Header />
        <div id="App-content">
          <Switch>
            <Route exact path="/" component={HomePage} />
            <PrivateRoute exact path="/dashboard" component={Dashboard} />
          </Switch>
        </div>
      </div>
    );
  }
}
```

Avec cette disposition, le header apparaît sur toutes les pages puisqu'il est en dehors du switch, le fait de changer l'url ne fera qu'intervenir le contenu de la balise « App-content ».

Dans les balises <Route /> on place le terme 'exact' avant la propriété 'path' pour signifier que le composant ne peut être rendu strictement que pour l'url mentionnée, si on ne l'utilise pas, pour la racine du site '/', la route '/dashboard' commençant par la même adresse renverra toujours le composant de la route '/', je le précise également sur toutes mes routes pour qu'on ne puisse pas accéder au composant si l'url ne correspond pas parfaitement.

On utilise la balise <Link to='path' /> de react router dom pour créer des liens vers nos routes.

```
<Link to="/dashboard">Dashboard</Link>
```

Les routes privées :

La route privée fonctionne comme la route publique, à ceci près qu'elle passe par une phase d'authentification avant de retourner une balise <Route /> de react router dom.

Elle est un composant React que l'on déclare dans un fichier 'privateRoute.js'. Puisqu'elle hérite du 'life cycle' de React, on peut utiliser la méthode 'componentWillMount()' pour faire un appel vers notre API et vérifier si l'utilisateur possède un token valide.

```
componentWillMount() {  
  let that = this;  
  API.isAuth()  
    .then(data => {that.setState({logged: true,  
      loaded: true, user: data.data.user});})  
    .catch(err => {that.setState({loaded: true});});  
}
```

Cette fonction s'effectue au chargement du nouveau composant, avant de l'afficher avec la méthode 'render()'.

La méthode 'isAuth()' de API.js retourne une promesse de requête axios, on traitera un résultat valide dans le .then() et une erreur ou refus d'authentification dans le .catch().

```
isAuth: function(){  
  let token = localStorage.getItem('token');  
  return axios.post(path+'/authenticated', {token}, {headers: headers})  
},
```

API.isAuth() 'post' le token récupéré dans le local storage vers la route d'authentification de notre API.

```
exports.authenticated = function(req, res) {  
  jwt.verify(req.body.token, config.SECRET, function(err, decoded){  
    if (err) res.status(400).json(err);  
    else {  
      res.status(200).json({auth: true, user: decoded});  
    };  
  });  
}
```

La route '/authenticated' mène sur une méthode du userController qui vérifie le token.

Sur cette route de notre API, si le token reçu est invalide, on retournera une réponse avec un statut 400 ainsi qu'une erreur qui déclenchera le .catch() de notre componentWillMount(), si le token est valide, on retournera une réponse avec un statut 200 ainsi qu'un objet avec les données de l'utilisateur stocké dans le token, ce qui aura pour effet de déclencher le .then() .

Déclencher le .then() aura pour effet de faire passer deux states à true, le loaded et le logged, tandis que déclencher le .catch() ne fera passer qu'un seul state à true, le loaded.

Grâce à ces states, nous allons pouvoir décider d'orienter l'utilisateur vers la page qu'il demande, si il peut y accéder, sinon il sera réorienter vers la page de connexion.

```
render() {  
  const {logged, loaded, user} = this.state;  
  if (loaded) {  
    if(logged) return (<Route path={this.props.path}  
      render={props => <this.props.component {...props} user={user} /> />)  
    else return window.location = '/login';  
  }  
  else return null;  
}
```

Le state 'loaded' n'existe que pour régler un problème d'asynchronisme, en effet, puisque la méthode `API.isAuthenticated()` déclenche une requête vers notre back-end, qui prend un certain temps, si l'on ne vérifie pas que la réponse est arrivée avec la variable 'loaded', ce code nous redirigera pour toujours vers la page '/login' puisqu'il n'attendra jamais la réponse de notre API.

Si la réponse est revenue et que l'on est connecté, on renvoie un composant `<Route />`, avec le path passé en propriété à notre route privée et le component passé de la même manière.

On pourrait utiliser l'attribut 'component={this.props.component}' de notre route pour rendre le composant voulu, mais l'on ne pourrait alors pas lui passer de propriétés de cette manière, si l'on veut passer des propriétés à nos routes, il faut utiliser l'attribut 'render' et déclarer une fonction qui renverra le composant voulu, avec les propriétés voulues, ici je passe dans la propriété 'user', le profile de l'utilisateur récupéré après avoir décodé le token.

Le problème d'asynchronisme rencontré avec ma route privée m'a bloqué un certain temps, j'ai finalement trouvé la solution en cherchant sur stackoverflow et en trouvant le sujet de quelqu'un qui avait rencontré le même problème que moi.

<https://stackoverflow.com/questions/49309071/react-private-router-with-async-fetch-request>

3. Le carousel :

J'ai décidé de coder le carousel moi même, à la fois pour m'amuser et pour m'entraîner.

J'ai donc commencé à réfléchir à comment faire fonctionner un carousel et ai finalement opté pour une solution avec une nodeList, concrètement, si le carousel défile vers la gauche, on utilise la méthode 'appendChild' sur le carousel en passant le premier enfant de la nodeList, afin que celui-ci se retrouve à la fin de la nodeList, inversement, si le carousel défile vers la droite, on utilise la méthode 'insertBefore' en passant le dernier enfant de la nodeList, afin de l'insérer avant le tout premier.

```
carousel.appendChild(carousel.childNodes[0]);
carousel.insertBefore(carousel.childNodes[length-1], carousel.childNodes[0]);
```

Cette logique fonctionnant très bien mais étant dépourvue d'animations, il me fallait embellir le tout avec des mouvements et des transitions.

J'ai donc décidé d'utiliser des classes css pour gérer les animations, je les attribues et les retires pour donner certains effets.

```
carousel.childNodes[0].classList.add('CarouselLeftMove');
carousel.childNodes[half].classList.remove('CarouselMajorScale');
carousel.childNodes[half+1].classList.add('CarouselMajorScale');
```

Je me suis vite rendu compte qu'il me faudrait régler un problème d'asynchronisme et de logique pour lesquels j'ai décidé d'implémenter une fonction asynchrone, pour laisser le temps à l'animation de se faire, avant d'échanger les positions de nos éléments dans les tableaux.

```
let system = async function(that, move) {
  let carousel = document.getElementById('CarouselShow');
  let length = carousel.childNodes.length;
  let half = (Math.round(length/2))-1;
  if (move !== 'Left') {
    carousel.childNodes[0].classList.add('CarouselLeftMove');
    carousel.childNodes[half].classList.remove('CarouselMajorScale');
    carousel.childNodes[half+1].classList.add('CarouselMajorScale');
    await that.waitForCarousel(carousel);
  }
}
```

Ici on récupère la « galerie » du carousel qui diffuse nos différents éléments, on récupère le nombre d'éléments contenu dedans et l'on fait une petite opération pour avoir un repère sur la moitié de la nodeList, enfin, en fonction du mouvement, on attribue nos classes et l'on attend que les animations se terminent pour déclencher le mouvement dans la nodeList.

```

waitForCarousel = (carousel, length) => {
  return new Promise(resolve => {
    setTimeout( () => {
      if (length) {
        carousel.childNodes[length-1].classList.remove('CarouselRightMove');
        carousel.insertBefore(carousel.childNodes[length-1], carousel.childNodes[0]);
      }
      else {
        carousel.childNodes[0].classList.remove('CarouselLeftMove');
        carousel.appendChild(carousel.childNodes[0]);
      }
      this.setState({workin:true});
      resolve(true);
    }, 300);
  })
}

```

Le mouvement dans la nodeList est déclenché par la fonction 'waitForCarousel', on utilise un setTimeout, que l'on fait durer autant que nos transitions, puis quand elles sont finies, on lance la fonction déclarée dans le setTimeout, qui retire la classe de mouvement puis déplace les éléments dans la nodeList.

Pour éviter qu'un utilisateur un petit peu trop nerveux fasse buger le carousel en cliquant trop vite sur les flèches, j'utilise un state de react qui conditionne le mouvement, et qui passe false au début de celui-ci et true à la fin, ainsi, tant que la fonction de mouvement en cours avec le carousel n'est pas terminée, elle ne peut pas être re déclenchée.

Pour vulgariser la logique finale, les éléments se déplacent de manière animée, une fois l'animation terminée, on leur retire leur classe, donc ils reprennent tous leur place sans aucune transition (déclarée dans la classe de mouvement) et dans le même laps de temps, les changements en début ou fin de la nodeList sont effectués, ce qui reste invisible à l'oeil.

Le contenu du carousel est rempli dynamiquement en récupérant les infos de la base de données, demandées à notre API.

Les classes CSS :

```

.CarrouselRightMove {
  transition: margin 0.3s ease-out;
  margin-right: -65%;
}

.CarrouselMajorScale {
  box-shadow: 0 0 0.8em white;
  transform: scale(1.05);
  filter: brightness(100%);
  opacity: 1;
}

```

J'utilise donc une marge négative pour déplacer les éléments sur la page, avec une transition dite 'ease-out'.

J'ai donné un effet de mise en avant avec la classe majorScale, qui applique une aura blanche autour de l'élément, et qui passe sa luminosité à 100% (contre 70% sans cette classe).

4. Composant dynamique

J'utilises un composant dynamique pour la création ou l'update d'offres, ainsi j'évites des duplications de code inutiles.

La version create :

Il me fallait créer un composant pour permettre aux utilisateurs de créer de nouvelles offres, j'ai donc utilisé la technique de React, qui consiste à lier nos inputs aux states, afin de stocker leurs valeurs dynamiquement dans l'objet 'state', ceci permet de les récupérer plus facilement au moment de l'envoi.

```
export default class NewOffer extends React.Component {
  constructor() {
    super();
    this.state = {
      role: React.createRef(),
      title: React.createRef(),
      content: React.createRef(),
      startDate: React.createRef(),
      endDate: React.createRef(),
      ownerId: React.createRef(),
      active: React.createRef(),
    };
  }
}
```

On attribut aux propriétés du states des méthodes `React.createRef()`, pour permettre plus tard d'injecter des données existantes pour un update.

```
<input className="FormInput" name="startDate" type="date"
  value={this.state.startDate} onChange={this.handleChange}/>
```

On attribut ensuite les states aux valeurs des inputs et on ajoutes un écouteur d'évènements 'onChange' qui déclencheras la fonction `handleChange`.

```
handleChange = event => {
  this.setState({
    [event.target.name]: event.target.value
  });
}
```

Cette fonction attribut à la propriété du state, possédant le même nom que l'input déclenchant l'évènement, la valeur de cette input, ainsi, à chaque fois que le contenu d'un input est modifié, le state est mis à jour.

```
else API.addOffer(this.state)
  .then(data => {console.log(data)})
  .catch(err => {console.log(err)});
})
```

Finalement, au moment du submit, après avoir effectué la vérification des champs, on envoie

le state sur la route '/addOffer' de notre API pour créer une nouvelle offre.

La version update :

Après avoir finis de développer le composant permettant aux utilisateurs de créer une offre, je me suis penché sur l'update, voyant que j'allais avoir à dupliquer du code entre mes deux composants, j'ai voulu trouver un moyen d'adapter mon composant en fonction du cas.

Dans le cas où l'utilisateur voudrait créer une offre, on affiche un formulaire vide, dans le cas où l'utilisateur voudrait mettre à jour une offre existante, on affiche un formulaire rempli avec les données de cette offre.

Pour parvenir au résultat que je désirais, j'ai utilisé un paramètre d'url pour définir l'identifiant de l'offre recherchée.

```
<PrivateRoute exact path='/offers/:id' component={NewOffer}/>
```

Ainsi, je peux maintenant surveiller l'url pour savoir si l'utilisateur veut utiliser le composant pour créer une offre ou pour en mettre une à jour :

```
componentWillMount() {  
  let that = this;  
  let body = {id: this.props.match.params.id, type: this.props.user.role}  
  if (body.id) {  
    API.getOffer(body)  
  }  
}
```

Si body.id est défini (donc si on est sur une url comprenant un('/:id)'), on récupère l'offre recherchée, en requêtant notre API, ce qui nous permet ensuite de traiter ces données pour les attribuer au state, pour remplir automatiquement le formulaire avec les informations contenues en base de données.

Le formulaire de mise à jour fonctionne comme le formulaire de création, au changement des valeurs des inputs, le state sera actualisé.

```
if (this.props.match.params.id){  
  API.updateOffer(this.state).then(data => {console.log(data)});  
}
```

Au moment de soumettre le formulaire, on vérifie si l'url possède un paramètre 'id', et si c'est le cas on envoie le nouveau state sur la route 'updateOffer' de notre API.

```
componentDidUpdate() {  
  if (this.state.path !== window.location.pathname) document.location.reload();  
}
```

Puisque React est basé sur une logique monopage, lorsque je passais directement d'une page de mise à jour à une page de création, le composant ne changeait pas, il n'actualisait pas son contenu, j'ai donc pensé à cette méthode ci-dessus pour forcer l'update d'un composant au changement d'url.

5. Le css avec sass :

Pour utiliser sass, il faut installer le module node-sass qui permet de compiler les fichiers scss en css.

J'ai choisit sass car il permet de déclarer des fonctions (nommées mixin), d'utiliser des variables et de faire des 'sous-déclaration' css visant les nœuds enfants de l'élément ciblé par la classe définie.

```
#NavBar {
  text-align: center;
  margin-top: -2.8vh;
  width: 100%;
  height: 3%;
  transition: all 0.4s ease-out;
  box-shadow: 0 0.1em 0.4em black;
  ul {
    @include flxRowCtr();
    justify-content: space-evenly;
  }
}
```

J'ai créé un fichier scss par component afin d'accroître la lisibilité du design, ce fichier est le scss du component header qui inclus la navbar de l'application. Ici on cible tous les éléments ul de l'élément ayant l'id 'NavBar'.

Je définit certaines données comme des couleurs, une police ou une classe css dans des variables, contenues dans un fichier 'config.scss'.

```
$color7: #9361ff;
$color8: #b3b1e0;

$width: 45%;

$buttonSize: 80px;

@mixin flxRowCtr() {
  display: flex;
  justify-content: center;
  align-items: center;
}
```

Il suffit maintenant d'importer le fichier 'config.scss' pour utiliser les déclarations stockées dans des variables, ceci accroît la maintenabilité car maintenant il n'y a plus qu'à modifier la couleur dans le fichier config, pour la modifier partout ailleurs sur le site.

Finalement, on importe le fichier 'config.scss' dans chacun de nos fichiers scss, et on utilise nos variables comme on le souhaite.

```
@import './config.scss';

.OfferBox{
  width: 40%;
  height: 17vw;
  margin-bottom: 5%;
  @include flxColCtr();
  background-color: $color4;
  justify-content: space-evenly;
  flex: 0 0 40%;
}
```

Exemple d'import et d'utilisation des variables stockées dans le fichier config.scss.

