



# brigad

Let's work

Rapport de stage  
Concepteur développeur informatique  
Robin Saffioti

# SOMMAIRE

<b>CHAPITRE 1 - PRESENTATION GENERALE</b>	<b>4</b>
1.1 Avant-Propos	5
1.1.1 Présentation de l'entreprise	5
1.1.2 Présentation du stagiaire	6
1.2 Abstract	7
1.3 Remerciements	8
1.4 Démarche générale	9
<b>Chapitre 2 - Cahier des charges</b>	<b>10</b>
2.1 Le projet	11
2.2 Gestion du projet	12
2.3 Définition des besoins	13
2.4 Technologies imposées	14
2.4.1 React.JS	15
2.4.2 Redux	16
2.4.3 Babel	17
2.4.4 Webpack	18
2.4.5 Scss Sass	19
2.4.6 Node.JS	20
2.4.7 Hapi	21
2.4.8 Bookshelf	22
2.4.9 Knex	22
2.4.10 PostgreSQL	23
<b>Chapitre 3 - Analyse</b>	<b>24</b>
3.1 Le Diagramme de cas d'utilisation	25
3.1.1 Définition	25
3.1.2 Représentation graphique	26
3.1.3 Fiche de description textuelle d'un cas d'utilisation	28
3.2 Les maquettes	30
3.3 Le diagramme de navigation	32
3.3.1 Définition	32
3.3.2 Représentation graphique	33
3.4 Le diagramme de séquence système	35
3.4.1 Définition	35
3.4.2 Représentation graphique	36
3.5 Le diagramme d'activité	38
3.5.1 Définition	38

3.5.2 Représentation graphique	39
<b>Chapitre 4 - Conception de la base de données</b>	<b>41</b>
4.1 La démarche utilisée	42
4.2 Le diagramme de classes	43
4.3 Extrait du schéma de la base de données	44
4.4 SQL: Le LDD	45
4.4.1 Extrait du code de création de la base de données	46
4.4.2 Extrait du code de création d'une table	47
<b>Chapitre 5 - Conception de l'application</b>	<b>49</b>
5.1 Architecture du projet	50
5.2 Diagramme de séquence détaillé	52
<b>Chapitre 6 - Développement</b>	<b>53</b>
6.1 Les interfaces	54
6.2 Partie Back-End	57
6.2.1 Introduction	57
6.2.2 Les routes	58
6.2.3 Les fichier de config des routes	60
6.2.3.1 POST	60
6.2.3.1 GET	65
6.2.4 Les modèles Bookshelf	67
6.3 Partie Front-End	71
6.3.1 Introduction	71
6.3.2 Code de l'interface de présentation du TypeForm	72
6.3.3 Code de la modale d'affichage des réponses du TypeForm	76
6.3.4 Reducer Redux	79
<b>Chapitre 7 - Déploiement</b>	<b>81</b>
7.1 Diagramme de déploiement	82
7.2 Le déploiement	84
<b>Chapitre 8 - Conclusion</b>	<b>85</b>
8.1 Retour d'expériences	86
<b>Chapitre 9 - Annexes</b>	<b>87</b>
9.1 Extrait du CSS	88

# CHAPITRE 1 - PRESENTATION GENERALE

---

---

## 1.1 Avant-Propos

---

### 1.1.1 Présentation de l'entreprise

Fondé en novembre 2015, Brigad est le premier service spécialisé dans le recrutement par SMS dans l'univers de l'hôtellerie et de la restauration. La Start-up met en relation des indépendants avec des entreprises ayant besoin de personnel. Son algorithme permet aux professionnels de trouver immédiatement la personne disponible, avec un profil adapté, pour un extra ou pour un CDI. Le service est accessible par SMS, application, serveur vocal et sur le site Internet brigad.co.

Dès son lancement en janvier 2016, Brigad s'est donné pour mission de permettre aux restaurateurs de ne plus jamais manquer de personnel, y compris à la dernière minute, avec un service de placement instantané par SMS (85 % des demandes sont satisfaites en moins de 5 minutes). Brigad connaît un succès immédiat et compte aujourd'hui des milliers d'heures de travail distribuées (15 000 heures en juin 2017). La Start-up devient rapidement le nouvel allié RH de plus d'une centaine d'établissements emblématiques de la capitale et de milliers de « brigadiers ».

Devant le succès de cette première solution, Brigad vient d'intégrer de nombreux services complémentaires tels que la gestion des contraintes contractuelles, de paiement et d'assurance.

Brigad s'est implanté à Lyon depuis Novembre 2017 et aussi à Londres depuis Avril 2018.

---

### 1.1.2 Présentation du stagiaire

Passionné d'informatique depuis toujours, j'ai réellement découvert le développement informatique il y a à peu près six ans avec les Arduinos et Raspberry pi. Et depuis, je pratique cette discipline en autodidacte.

J'ai commencé en fabriquant un video wall avec des Raspberry pi pour des amis musiciens qui voulaient renforcer leur présence scénique à l'aide de vidéos.

J'ai ensuite découvert la domotique DIY, et je fabrique plein d'outils qui me permettent de contrôler ma maison en combinaison d'assistant à commande vocale tel que Google Home.

Après avoir travaillé plus de 6 ans dans la production audiovisuelle en tant que technicien du son, je me suis rendu compte que ce milieu n'était pas fait pour moi.

En plus du caractère précaire du statut d'intermittent du spectacle, s'ajoutait une envie de stabilité dans la vie de tous les jours.

Après avoir fait un bilan de compétences qui s'est rapidement conclu par une décision de faire une formation liée au développement informatique, j'ai donc entrepris une reconversion professionnelle afin d'exercer un métier qui me correspondait plus.

En mars de cette année, un ami m'a parlé d'une de ses connaissances qui était devenue Head of engineering d'une Start-up qui était phase de recrutement et que cela pourrait être une bonne occasion pour trouver un stage.

En contactant le head of engineering de cette Start Up, il m'a fait savoir que deux mois et demi de stage semblaient un peu courts pour le CTO, mais que si je le désirais je pouvais passer un entretien d'embauche classique qui déboucherait sur un stage de pré embauche dans le cas où je réussirais.

Je me suis donc auto formé sur la bibliothèque React Js pendant 1 mois et demi avant de passer l'entretien.

Suite à la réussite de l'entretien technique j'ai passé un second entretien avec le CTO, CEO qui a conduit à l'obtention du stage.

---

## 1.2 Abstract

During my traineeship at Brigad I used to talk and write in English everyday.

Some of the employee are not french, one of the Product Manager is from Singapour therefore, do not speak french, as for all the staffs in the London Office.

So the policy at Brigad is to have all our conversation in slack (Professional communication tool) meetings and presentations in English so everyone can understand and participate

---

## 1.3 Remerciements

Je tiens à remercier:

- Messieurs Brieuc Maitre, Jean Lebrument et Florent Malbranche pour m'avoir donné la chance de faire mon stage au sein de Brigad.
- Messieurs Adrien Harnay et Thibault Malbranche, mes tuteurs pour leur patience, leur confiance et leur aide précieuse.
- Messieurs Pascal Buguet et Sébastien Maloron pour leur disponibilité, leur patience et leur enseignement tout au long de ma formation.
- Mes camarades de formation Jessica, Nicolas, Miguel, Joseph, Michel, Frédéric et Kevin, pour leur aide et soutien.
- Soulafe ma compagne, pour m'avoir incité et poussé à entreprendre une reconversion professionnelle et pour son soutien sans faille.
- Nahel, mon fils, pour avoir fait ses nuits à l'âge de deux mois et m'avoir laissé de l'énergie pour travailler.



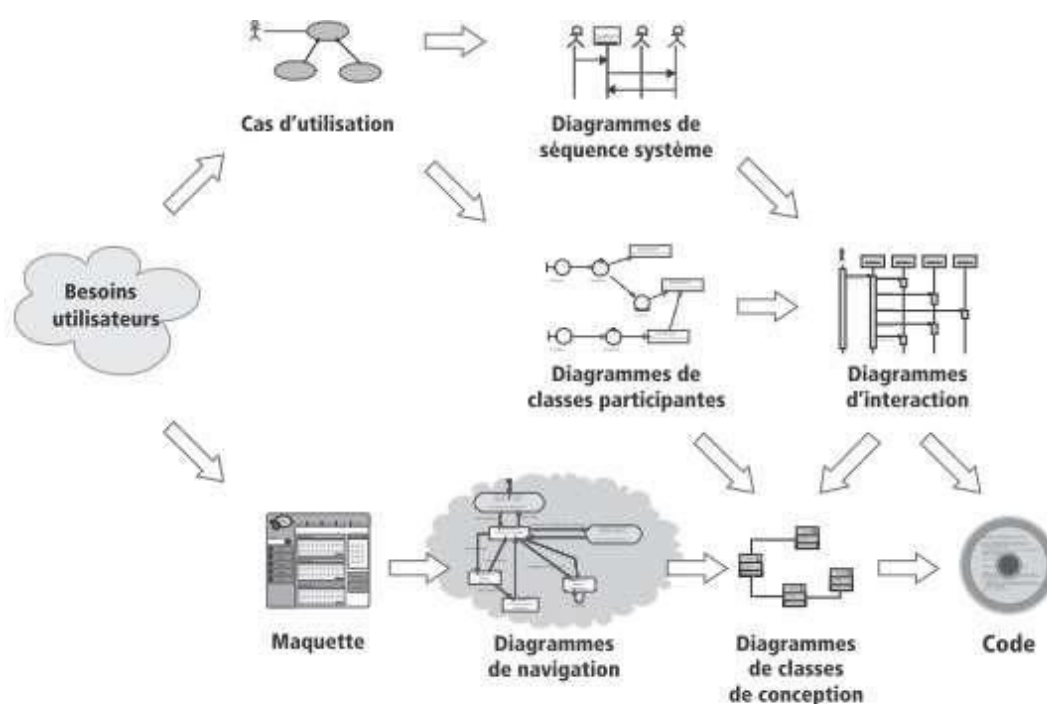
## 1.4 Démarche générale

En tant que développeur, concepteur nous devons être à même d'anticiper tous les besoins utilisateurs.

Dans cet objectif nous devons passer par plusieurs étapes :

- phase d'analyse
- phase de conception

Pour ce projet j'ai choisi d'utiliser une démarche 2TUP (2 Tracks Unified Process) avec le langage UML, préconisé par Pascal Roques dans son ouvrages "UML2 - Modéliser une application web", qui peut se représenter selon le schéma suivant.



**Figure 1-20** Schéma complet du processus de modélisation d'une application web

## Chapitre 2 - Cahier des charges

---

---

## 2.1 Le projet

La société Brigad possède deux plateformes Web importantes.

**Le job board** : Qui est l'interface publique de la société. Elle est séparée en deux parties :

- La partie travailleurs indépendants de la restauration (Job Seeker ou Brigaders) sur laquelle les utilisateurs se connectent et gèrent leurs offres de missions, peuvent voir le nombre d'heures travaillées, leurs rémunérations etc ...
- La partie société (Business) sur laquelle les restaurants, hôtels etc... qui travaillent avec Brigad se connectent pour faire une demande d'extra pour une mission.

**L'admin panel** : Qui est l'interface privée sur laquelle les employés de Brigad se connectent pour avoir des informations sur les Brigaders et les business, faire la mise à jour des profils, valider les missions envoyées par les business ...

Pour faciliter l'inscription des Brigaders, Brigad a décidé de refaire en partie de leur système d'inscription (registration funnel).

Le registration funnel est composé de 3 phases :

- Renseignement du nom, prénom, adresse mail et mot de passe dans une modale sur l'écran d'accueil. (cf. photo 1)
- Renseignement du genre, date de naissance, adresse, et numéro de téléphone. (cf photo 2)
- Renseignement des compétences/expériences, CV et recommandations (cf photo 3)

Brigad souhaitait remplacer la phase 3 par un Typeform.

Typeform est un outil en ligne de création de formulaire modulable et "APIsable".

Il devait également être possible de voir les réponses du Typeform sur la page du profil d'un Brigader sur l'admin panel.

---

## 2.2 Gestion du projet

La gestion du projet s'est faite avec une méthode agile, avec un tableau kanban sur la plateforme Jira. Chacune de mes tâches était découpée en tickets.

Les tickets m'étaient assignés soit par le Lead développeur web ou Back-end, ou par le Product manager.

Une fois les travaux du ticket terminés je les passais en phase de "code review" qui était effectué par les Leads développeurs.

Si le code est approuvé il passe en phase "Q&A ready". Phase pendant laquelle le product manager teste les nouvelles fonctionnalités mises en place. Si il approuve, le ticket passe en "release ready" et est prêt à être envoyé en production. Sinon on crée un nouveaux ticket avec une nouvelle itération décrivant les nouvelles fonctionnalités.

En ce qui concerne la gestion du code, on utilise git comme outil de versionning.

Pour chaque ticket/fonctionnalité, on crée une nouvelle branche .

## 2.3 Définition des besoins

Plusieurs réunions ont eu lieu entre le Head of Product, le CTO et Head of Engineering pour définir les besoins.

Job-Board :

- Les Brigaders devront utiliser un TypeForm dans la phase 3 du “registration funnel” afin de renseigner leurs compétences/expériences
- En fonction de la langue choisie dans la page d'accueil (Anglais ou Français) le TypeForm sera affiché dans la langue préalablement choisie.
- En fonction de sa localisation (Angleterre ou France) le futur Brigader aura un TypeForm en fonction du pays
- Si il n'est ni en France ni en Angleterre, il ne pourra pas remplir le TypeForm. Une page lui indiquant qu'il n'est pas dans la bonne zone géographique sera affiché.

Admin Panel :

- Les employés s'occupant de la gestion des utilisateurs doivent voir si un Brigader a rempli le TypeForm ou non.
- Si le Brigader a rempli le TypeForm l'employé doit pouvoir visualiser les réponses.
- L'employé doit pouvoir visualiser le nombre de fois où un Brigader a rempli le TypeForm (dans le cas ou il l'aurait rempli une première fois avec des valeurs erronées).

---

## 2.4 Technologies imposées

Les plateformes sur lesquelles j'ai eu à travailler étant déjà existantes, les technologies utilisées furent choisies lors de la création de la société en 2016.

Voici une liste exhaustive des technologies utilisées au sein de Brigad.

Front End :

- React.Js
- Redux
- Babel
- Webpack
- Javascript ES6
- Scss Sass

Back End :

- Node.js
- Hapi
- Kenex
- Bookshelf
- PostgreSQL

Etant en stage de préembauche en tant que développeur Front end, j'ai du prendre du temps pour me familiariser avec Node.js et les différents frameworks utilisés, qui ne feront quasiment plus partie de mon quotidien une fois en CDI.

L'intégralité du code Javascript est en train de subir une migration vers typescript.

---

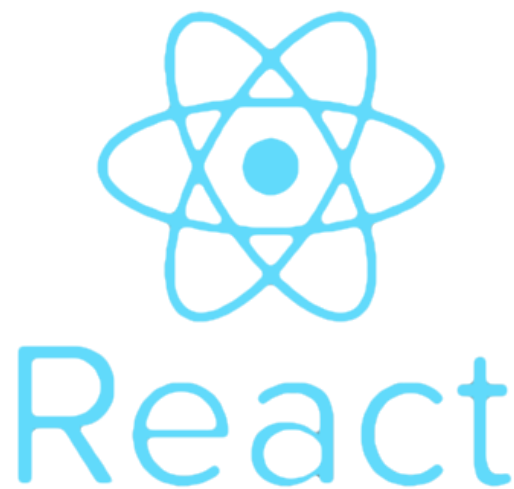
### 2.4.1 React.JS

React.js est une librairie javascript de création d'interface utilisateur créé en 2013 par Facebook.

React est basé sur un virtual-dom: un composant React ne crée pas de HTML mais une représentation sous forme d'objets et de noeuds de ce à quoi le HTML final doit ressembler.

Le virtual-dom va prendre en compte cette représentation, la comparer au DOM réel et en déduire les opérations minimales à exécuter pour que le DOM réel soit conforme au virtuel.

C'est grâce à cet outil que React peut partir du principe qu'il est plus simple de "remplacer" toute l'interface quand elle doit être modifiée plutôt que de modifier au fur et à mesure le DOM comme jQuery ou AngularJS pouvaient le faire.



---

## 2.4.2 Redux

Redux est un dérivé de l'architecture Flux.

Il permet de créer un Store (Conteneur) qui contient des états, qui réagit à des actions dispatchées et auquel on peut "s'abonner" pour être notifié de tout changement.

On peut également y ajouter des middlewares qui permettent d'effectuer des tâches avant les actions.

L'avantage est que l'on peut partager un état au sein de notre application web, entre deux bouts de code qui n'ont rien à voir entre eux ou qui n'ont aucun parent commun.

Redux s'utilise avec plusieurs librairies/framework Javascript, dont React.js Angular.js ou encore Vue.js.





---

### 2.4.3 Babel

Babel est une suite d'outils qui permet de convertir du code Javascript ECMAScript 2015+ (ES6) en code Javascript compréhensible par l'ensemble des navigateurs actuels (Javascript ES5).

Il permet également de transformer le format JSX utilisé par react en code Javascript classique.

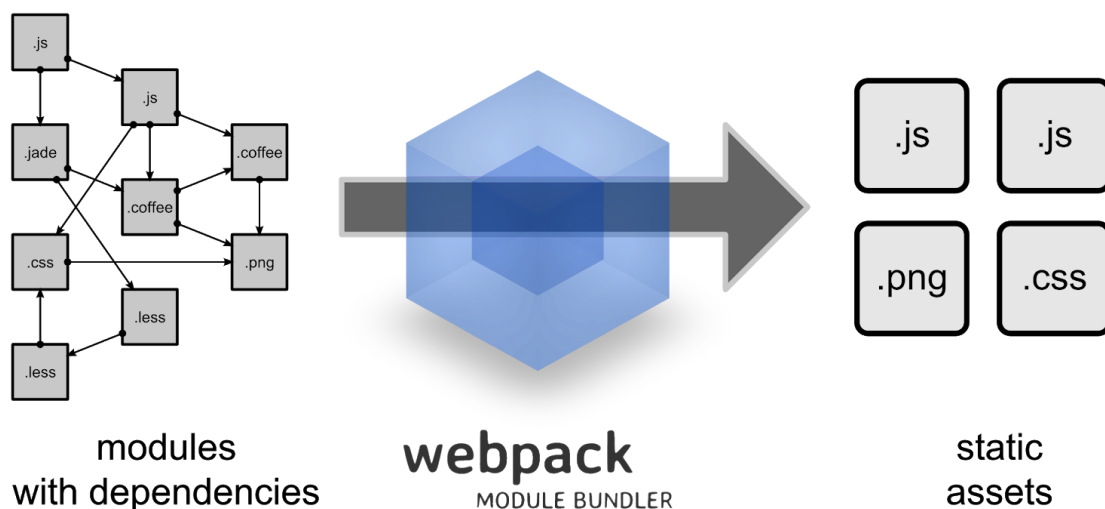
The word "BABEL" is written in a large, bold, yellow font with a thick, hand-drawn brushstroke texture. The letters are slightly slanted and have irregular, feathered edges, giving it a dynamic and artistic appearance.

## 2.4.4 Webpack

Webpack est un module bundler . C'est un outil qui prend tous les fichiers javascript, html, css et les regroupe dans un ou plusieurs modules.

Il permet de faciliter l'accès aux ressources statiques telles que les images, les polices de caractères, etc...

Comme un dessin vaut mieux que mille mots, voici le principe de webpack en illustration:



---

### 2.4.5 Scss Sass

Sass (Syntactically Awesome Style Sheets) est un préprocesseur pour le langage CSS.

Il permet de créer du code CSS en offrant une syntaxe plus simple et un code facilement réutilisable et maintenable.

Il est donc possible de créer des variables, des fonctions, faire des calculs et plein d'autres fonctionnalités que l'on peut retrouver dans un langage de programmation classique.

Ce code est ensuite converti en CSS classique, interprétable par n'importe quel navigateur.



---

## 2.4.6 Node.JS

Node.js est une plateforme de développement en Javascript. Elle permet grâce à des frameworks de créer des serveurs web ou applications avec le langage Javascript

Elle se distingue des autres plateformes grâce à son système non bloquant permettant d'effectuer des entrées/sorties asynchrones.

Node.JS est construit sur Chrome's V8 JavaScript Engine qui équipe le navigateur Chrome.

Node est construit sur un modèle event driven, c'est à dire que le code que l'on va écrire est basé sur un système d'évènement. Les éléments que l'on va créer vont émettre des événements auxquels on pourra souscrire afin de lancer des tâches spécifiques à leurs émissions.



---

### 2.4.7 Hapi

HAPI est un framework Javascript pour Node.js développé par 'Walmart Labs'. Il permet de créer un serveur web node qui sera capable d'écouter et de répondre à des requêtes.

Grâce à Hapi on peut facilement créer des routes (URL) dans le but de faire une API Rest.

HAPI dispose de plusieurs bibliothèques, pour la validation des données, la gestion des erreurs, etc ...



---

### 2.4.8 Bookshelf

Bookshelf est un ORM (object-relational mapping) pour Node.JS.

Il permet de lier/mapper un objet javascript avec une table de base de données. Bookshelf crée des modèles qui correspondent au contenu d'une table.

Mais il peut également rajouter des éléments dans ses modèles tel que des fonctions.

Bookshelf est souvent combiné avec le query builder Knex



---

### 2.4.9 Knex

Knex est un query builder pour Node.Js. Il est compatible avec plusieurs type de base de données relationnelle : Postgres, MSSQL, MySQL, MariaDB, SQLite3, et Oracle.



---

### 2.4.10 PostgreSQL

PostgreSQL est un puissant système open-source de gestion de base de données relationnelle.

PostgreSQL respecte le plus possible les normes SQL (Structured Query Language).

Ce SGBDR supporte une grande partie du standard SQL tout en offrant de nombreuses

fonctionnalités modernes comme par exemple:

- des requêtes complexes
- les clés étrangères
- l'utilisation de triggers (déclencheurs)
- les vues modifiables
- l'intégrité transactionnelle



## Chapitre 3 - Analyse

---



---

## 3.1 Le Diagramme de cas d'utilisation

---

### 3.1.1 Définition

**Les diagrammes de cas d'utilisation permettent de recueillir, analyser et organiser les besoins utilisateurs.**

Un DCU permet de représenter graphiquement les besoins d'un utilisateur, en illustrant les fonctionnalités de l'application à l'aide de schéma facilement compréhensibles.

Il sert à donner une vision du comportement d'une application. Un DCU est un ensemble de cas d'utilisation.

Il faut donc déterminer et définir les acteurs du cas d'utilisation, ainsi que les fonctionnalités auxquelles ils ont accès.

Comme mon travail porté sur deux plateformes différentes, j'ai fait deux DCU, un pour chaque plateforme.

Dans le DCU correspondant à la plateforme Job board l'acteur est un Brigadier/JobSeeker.

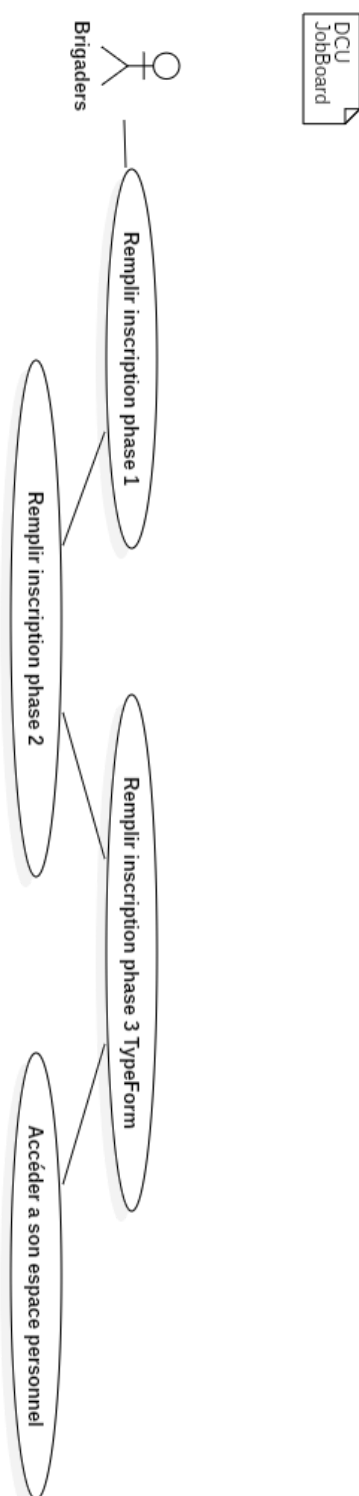
Dans celui correspondant à l'admin panel l'acteur est un employé de Brigad travaillant dans le pôle Onboarding ou Care.

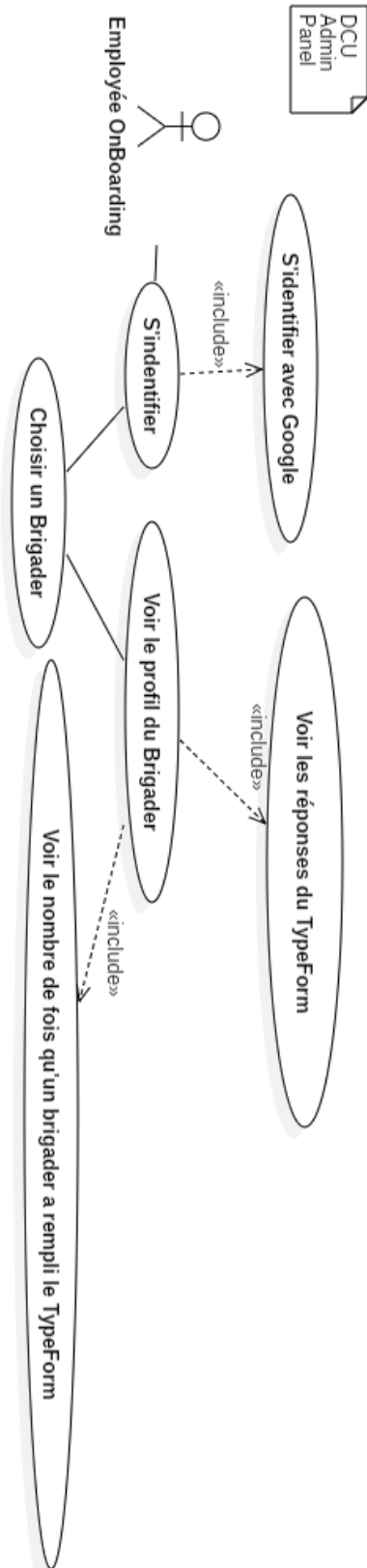
Pôles:

- Onboarding : Ils ont pour missions de gérer et valider les profils des nouveaux Brigadiers et de leur assigner des tags en fonction de leurs compétences.
- Care : Ils ont pour missions d'assurer le suivi des Brigadiers une fois que ceux-ci ont effectué leurs premières missions.

Dans mon cas comme les parties qui m'étaient assignées étant bien spécifiques les diagrammes de cas d'utilisation sont succincts.

### 3.1.2 Représentation graphique





### 3.1.3 Fiche de description textuelle d'un cas d'utilisation

La fiche de description textuelle d'un cas d'utilisation permet de décrire les actions qui seront réalisés par une fonctionnalité.

Le tableau ci dessous décrit le scénario nominal de "S'inscrire sur la plateforme Brigad" pour un Brigader et "Voir les réponses du TypeForm" pour un employé de Brigad.

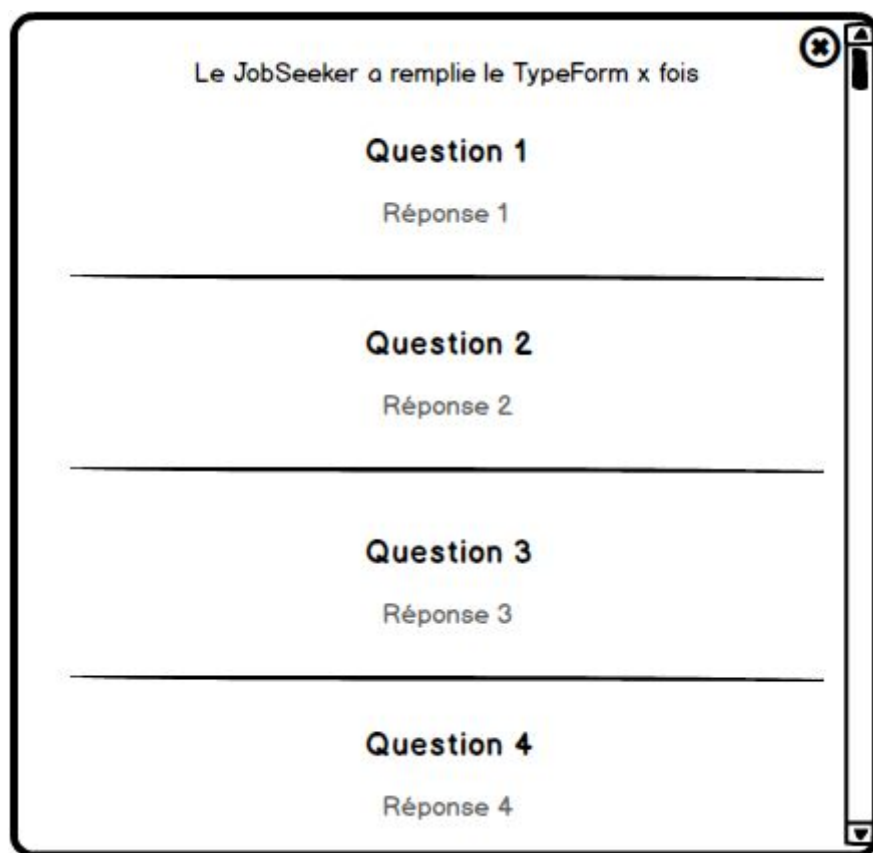
ETAPES	DESCRIPTION
<b>Identification du CU</b>	Titre: S'inscrire sur la plateforme Brigad Résumé: Enregistrer les réponses du TypeForm d'un nouvel utilisateur dans la base de donnée Acteurs: Brigader
<b>Précondition</b>	Serveur back end disponible et fonctionnel. Base de données disponible et fonctionnelle. Plateforme TypeForm disponible et fonctionnelle.
<b>Scénario nominal</b>	1 - cliquer sur le bouton "recevoir du travail". 2- Remplir la phase 1 d'inscription. 3 - Remplir la phase 2 d'inscription. 4 - Remplir le TypeForm. 5 - Accéder à son espace personnel.
<b>Scenarri d'erreurs du cas nominal</b>	1 - Un des champs obligatoire de la phase 1 ou 2 n'est pas remplis et l'utilisateur valide le formulaire. 2 - Message d'erreur "Veuillez renseigner ce champs". 3 - L'adresse du Brigader ne se situe si en France ni en Angleterre. 4- Message lui indiquant qu'il n'est pas dans la bonne zone géographique
<b>Post-conditions</b>	Les informations sont persistées en base de données. Le brigader accède à son espace personnel
<b>IHM (interface homme-machine)</b>	Iframe qui affiche le TypeForm

ETAPES	DESCRIPTION
<b>Identification du CU</b>	Titre: Voir les réponses du TypeForm Résumé: Récupérer les réponses du TypeForm depuis la base de données et les afficher Acteurs: Employée de Brigad
<b>Précondition</b>	Serveur back end disponible et fonctionnel. Base de données disponible et fonctionnelle.
<b>Scénario nominal</b>	1- L'utilisateur s'identifie 2 - L'utilisateur choisit un Brigader. 3 - L'utilisateur accède au profil d'un Brigader 4- Il clique sur le bouton "réponse TypeForm 5 - Il visualise les réponses et le nombre de fois que le TypeForm a été remplis.
<b>Scenarri d'erreurs du cas nominal</b>	1 -Le serveur back end ne marche pas
<b>Post-conditions</b>	Les informations sont persistées en base de données. Le Brigader accède à son espace personnel
<b>IHM (interface homme-machine)</b>	Iframe qui affiche le TypeForm

## 3.2 Les maquettes

Pour les maquettes de l'IHM qui ne m'ont pas été fournis par le Product Designer, j'ai utilisé Balsamiq Mockups, un outil gratuit de maquettage d'interfaces.

Maquette de la modale qui affiche les résultats du TypeForm.



Le JobSeeker a rempli le TypeForm x fois

**Question 1**  
Réponse 1

---

**Question 2**  
Réponse 2

---

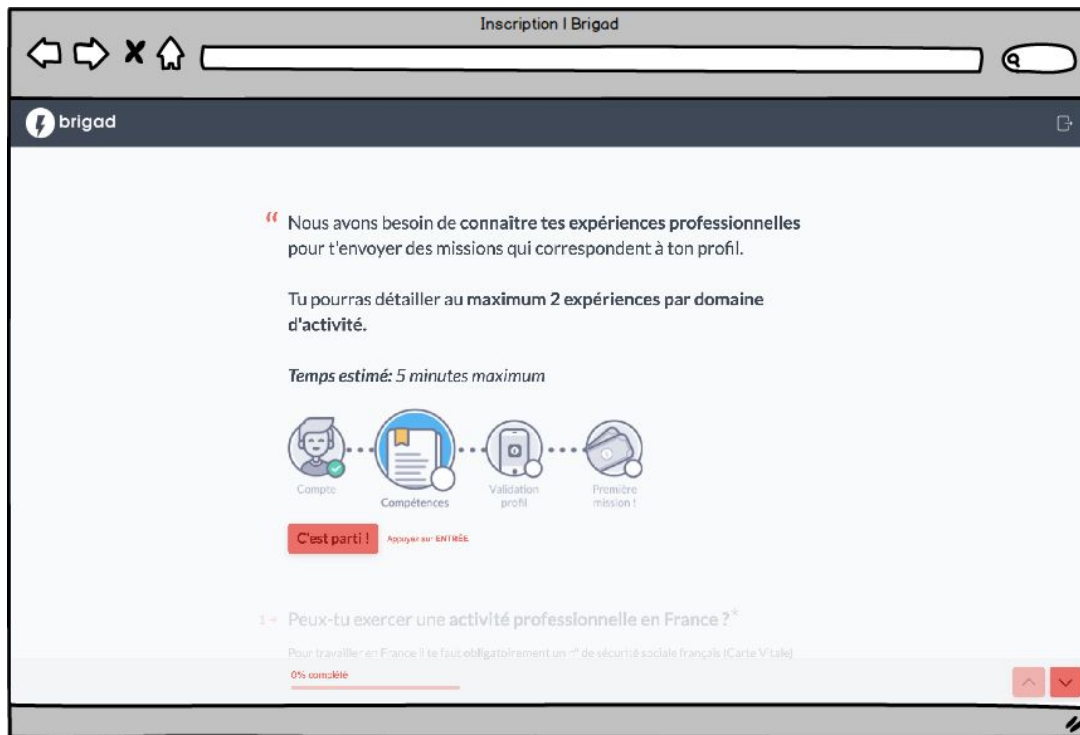
**Question 3**  
Réponse 3

---

**Question 4**  
Réponse 4

The image shows a wireframe of a modal window. At the top, it says 'Le JobSeeker a rempli le TypeForm x fois'. Below this, there are four sections, each with a bold question label and a corresponding response label, separated by horizontal lines. The modal has a standard window frame with a close button in the top right corner and a scroll bar on the right side.

Maquette de la page de la phase 3 d'inscription, qui affiche le TypeForm.



Maquette de la page à afficher quand le Brigader à une adresse ni en France ni en Angleterre.



---

## 3.3 Le diagramme de navigation

---

### 3.3.1 Définition

Le diagramme de navigation (DNAV) est utilisé pour représenter les cheminement de l'application entre les différents écrans.

Le début de la navigation est représenté par un rond plein, noir.  
Les transitions entre les écrans sont représenté par des flèches qui pointe vers un état.

Chaque état, est un écran, une page qui est représenté par un rectangle avec des coins arrondis.

#### **JobBoard :**

L'utilisateur arrive sur la page d'accueil du site, clique sur le bouton "recevoir du travail", une modale s'ouvre où il renseigne, son nom, prénom, mail et mot de passe, il clique sur le bouton valider, qui le dirige vers la phase 2 d'inscription, ou il doit renseigner des information supplémentaire. Il clique sur le bouton valider et est dirigé sur la page de la phase 3 d'inscription : le TypeForm si son adresse si situe en France ou en Angleterre. Sinon une page lui indiquant qu'il n'est pas dans la bonne zone géographique s'affiche à la place.

#### **Admin Panel :**

L'utilisateur s'identifie avec google, il clique sur l'onglet "lister les Brigaders".

Il est dirigé vers une page avec la liste des Brigaders. Il clique sur un Brigader.

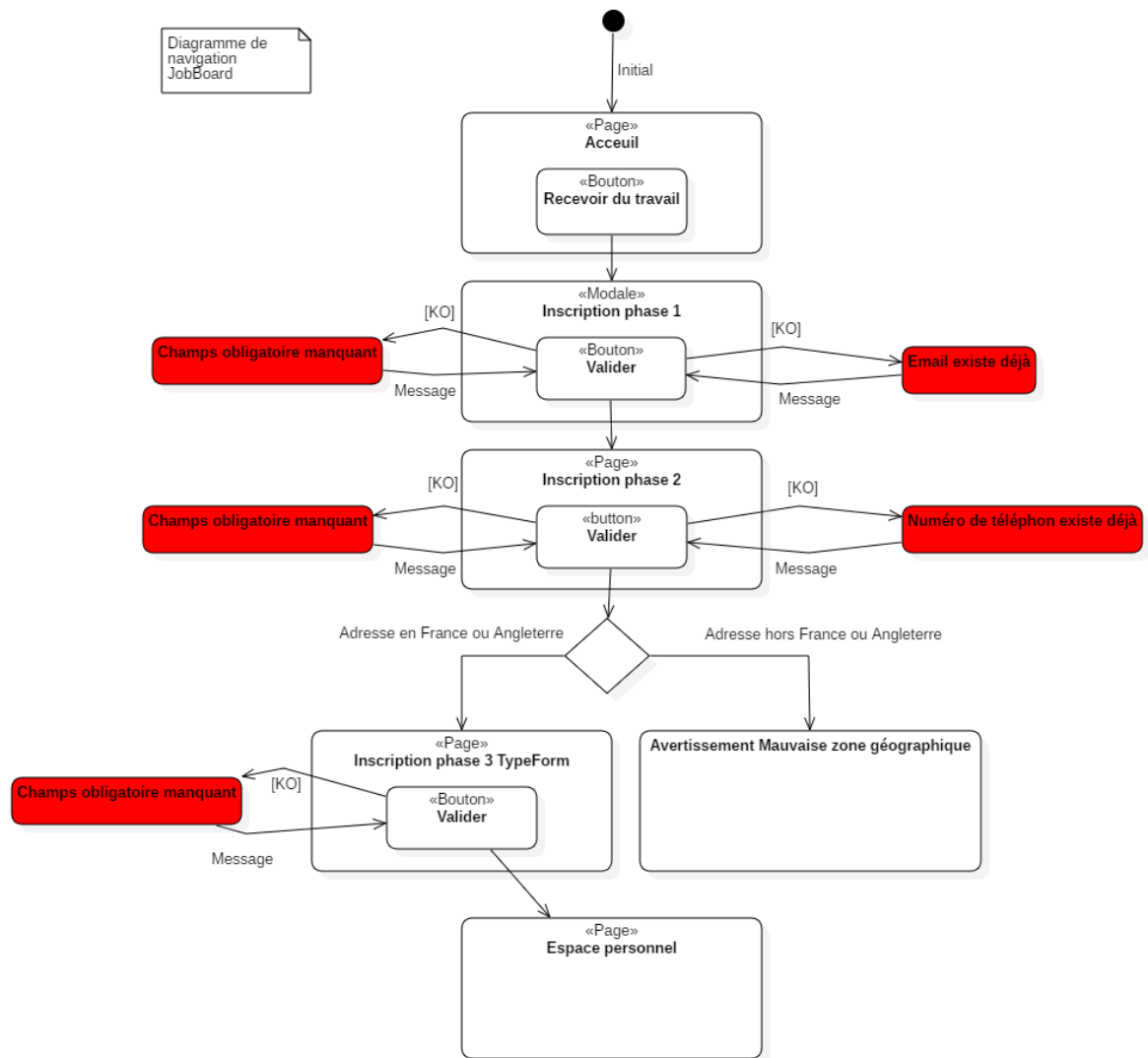
Il est amené sur "la page profil d'un Brigader".

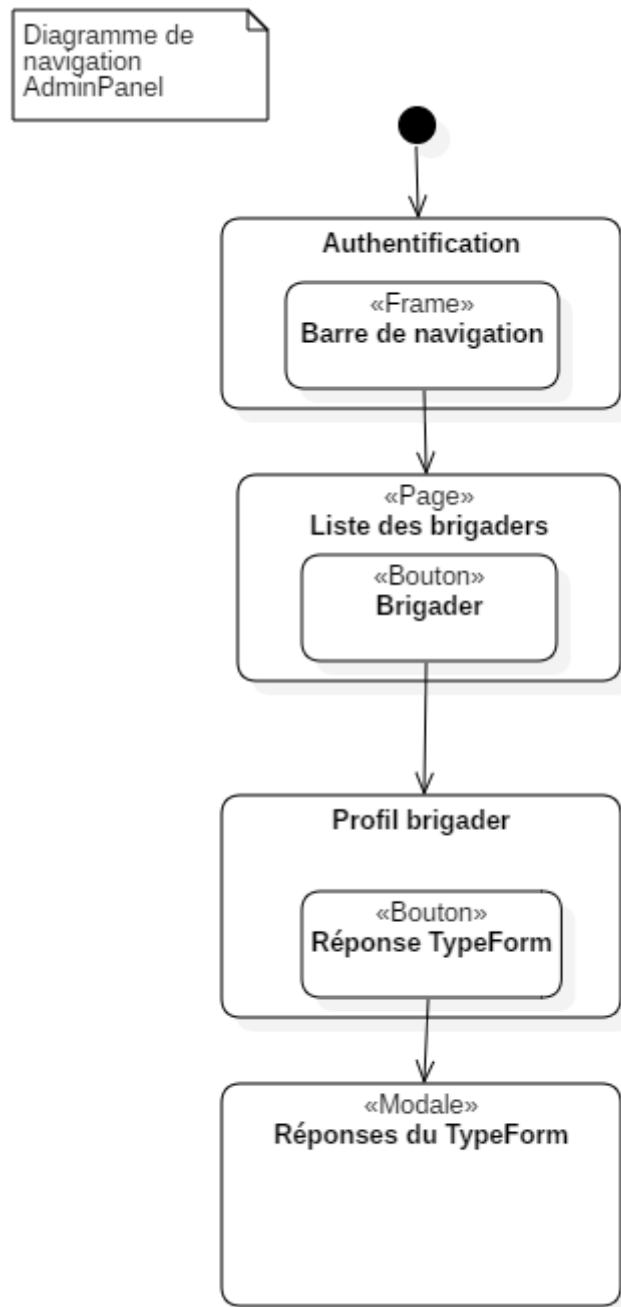
Si le Brigader a rempli le TypeForm, le bouton "Réponse TypeForm" est cliquable, et quand il clique dessus, s'affiche une modale avec les réponse du TypeForm.

Si le Brigader n'a pas rempli le TypeForm, le bouton "Réponse TYpeForm" est grisé et non cliquable.



### 3.3.2 Représentation graphique





---

## 3.4 Le diagramme de séquence système

---

### 3.4.1 Définition

Le diagramme de séquence système sert à décrire le comportement du système en adéquation avec un cas d'utilisation définis précédemment.

Il permet de formaliser les interactions entre acteurs, systèmes, à l'aide de message dans un point de vue chronologique et spatial.

Le temps est représenté verticalement, tandis que l'espace horizontalement.

Dans le DSS on compte 4 concepts principaux : l'acteur, la classe ou l'objet, le message et l'activation.

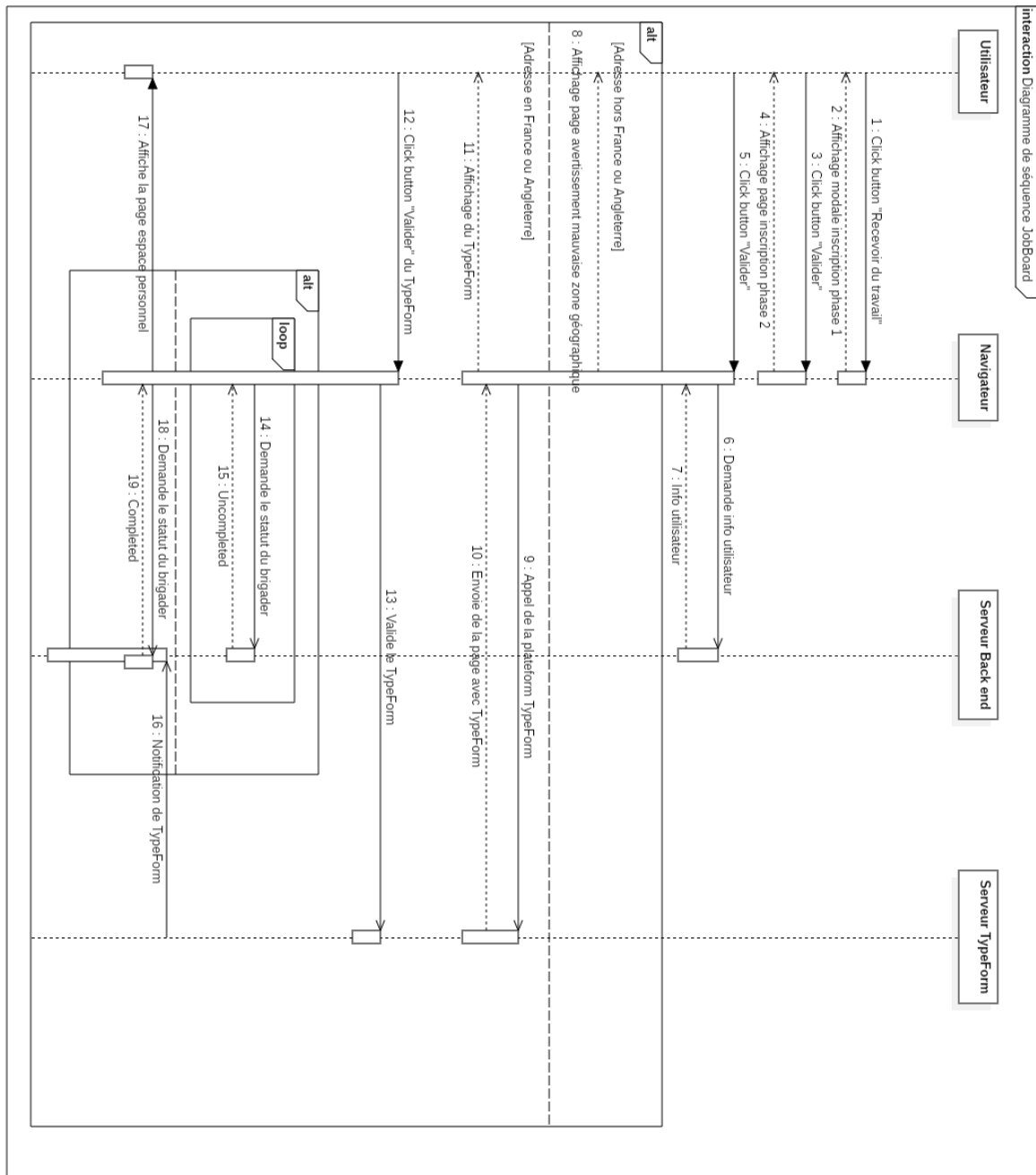
Les acteurs et classes sont représentés par des rectangles ou pas des icônes et une ligne pointillée appelée ligne de vie.

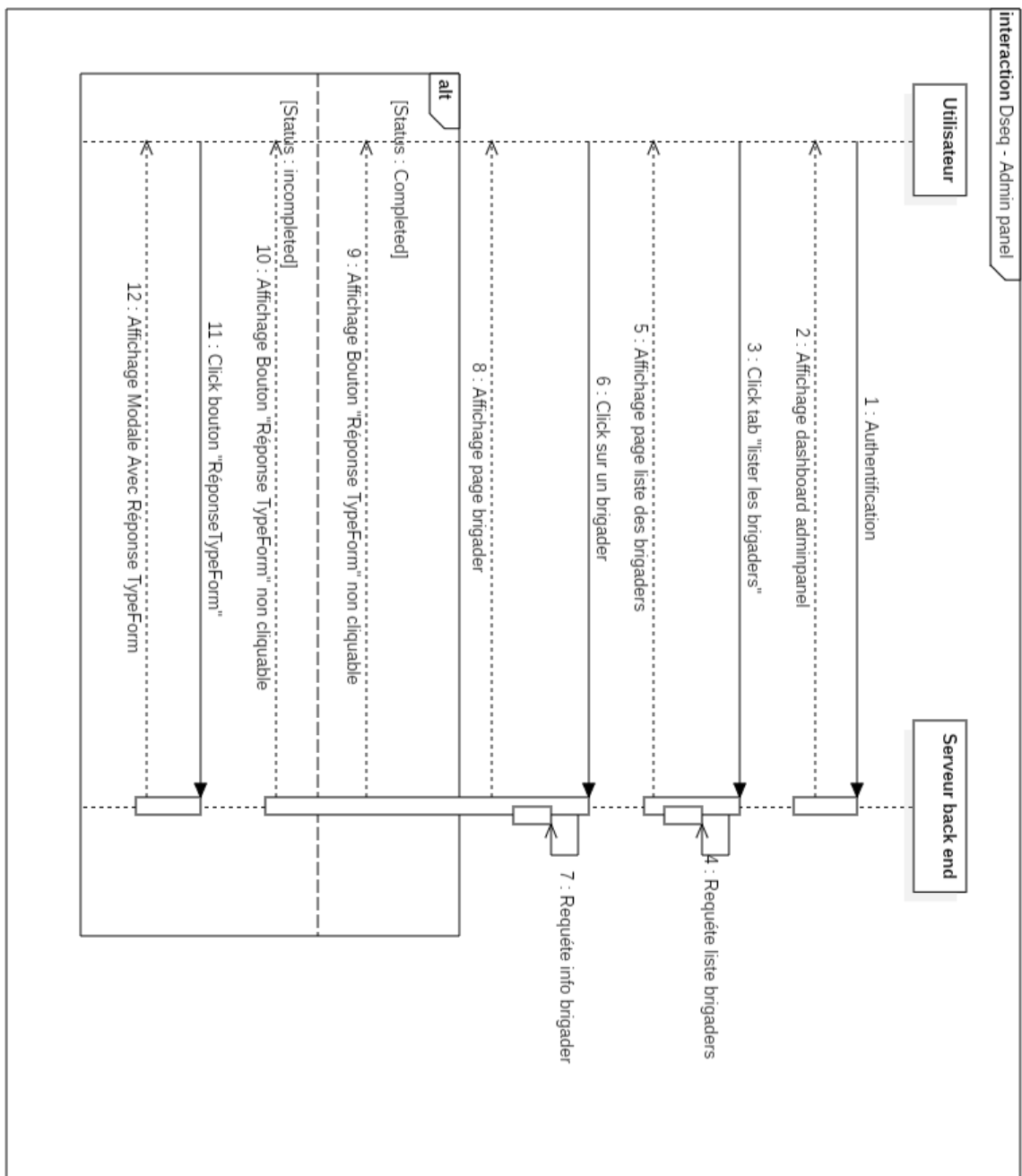
Le message est représenté par une flèche horizontale, qui va de l'émetteur vers le destinataire:

Il existe 5 types de message:

- Message de type CALL (synchrone) représenté par une flèche fermée et pleine.
- Message de type RETURN (synchrone) représenté par une flèche ouverte avec un trait en pointillé.
- Message de type SEND (asynchrone) représentée par une flèche ouverte avec un trait plein
- Message de type CREATE représenté par une flèche fermée et pleine avec un trait plein. Le trait a l'indication <<create>> au dessus de lui.
- Message de type DESTROY représenté par un trait plein avec une flèche pleine et fermée. Le trait a l'indication <<destroy>> au dessus de lui.

## 3.4.2 Représentation graphique





---

## 3.5 Le diagramme d'activité

---

### 3.5.1 Définition

Un diagramme d'activité (DAC) sert à représenter le déroulement d'un cas d'utilisation du point de vue de l'utilisateur.

Pour faire un DAC il faut montrer ces composants essentiels qui sont, les actions de l'utilisateur et les transitions.

On peut également y montrer les cas alternatifs (à l'aide d'un losange) et donc des transitions avec des gardes (OK/KO) en fonction des résultats.

### 3.5.2 Représentation graphique

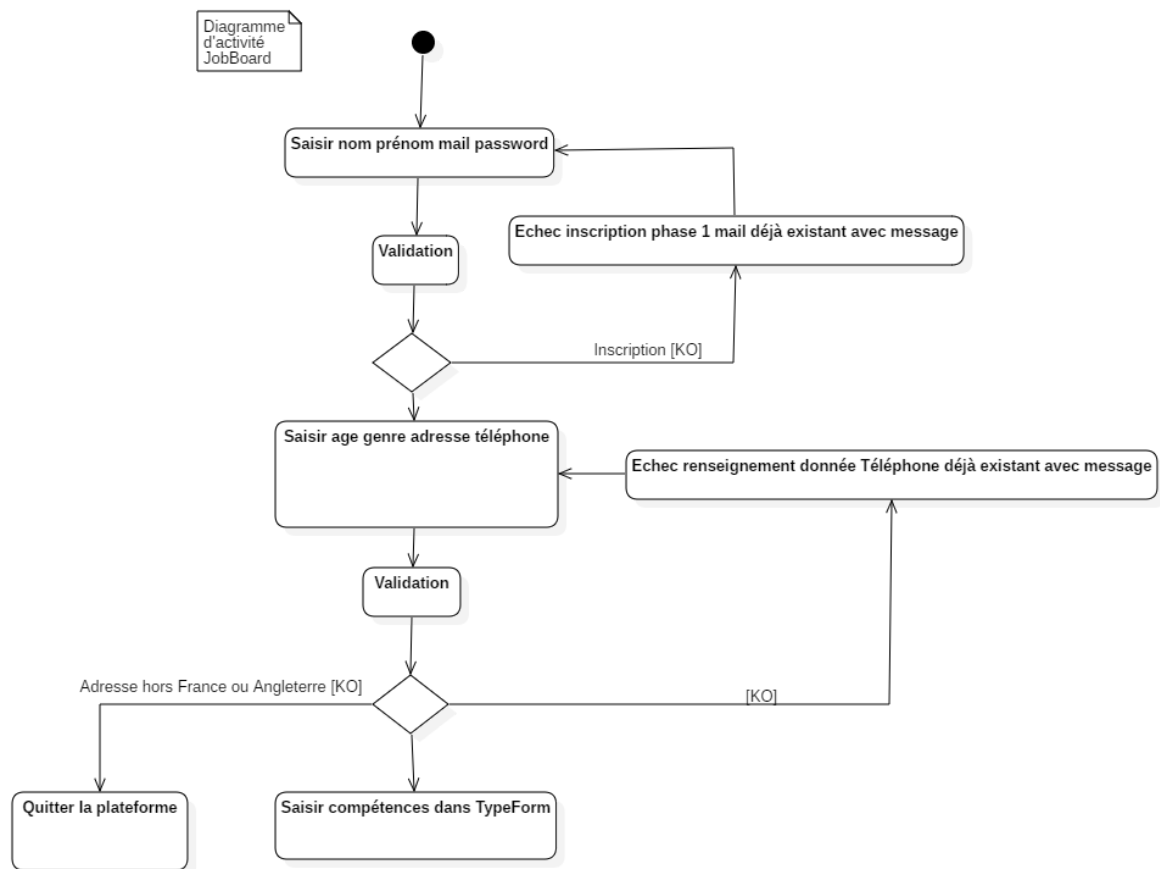
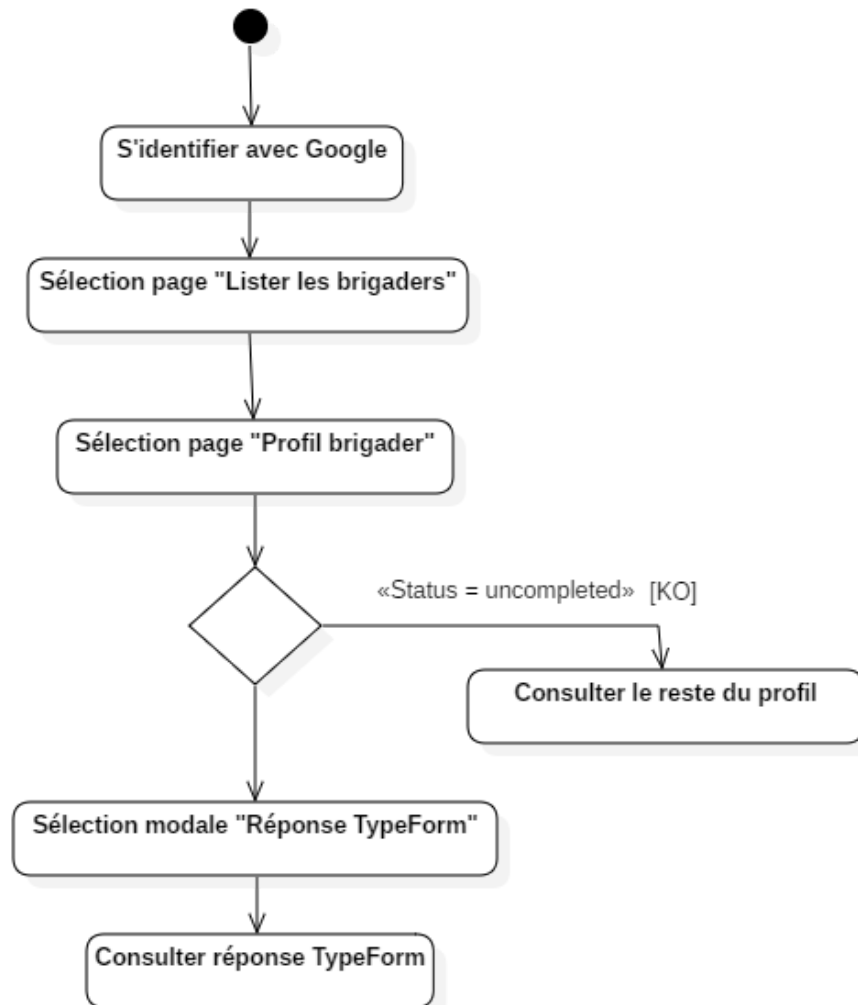


Diagramme  
d'activité  
Admin Panel



## Chapitre 4 - Conception de la base de données

---

---

## 4.1 La démarche utilisée

Afin de concevoir le diagramme de classes et donc la base de donnée, il est important d'utiliser une démarche qui nous permet d'obtenir un modèle de données qui soit normalisé et au moins de 3<sup>ème</sup> forme normale selon Codd.

Cette 3<sup>ème</sup> forme normale possède plusieurs avantages :

- Limiter les redondances de données
- Limiter l'espace disques nécessaires
- Limiter les incohérences de données
- Éviter les anomalies de mise à jour

Pour obtenir au modèle conceptuel de données plusieurs méthodes s'offre à nous:

La démarche par l'analyse du discours qui consiste à repérer les classes et les associations à partir du discours. Un nom devient classe et un verbe une association.

La démarche ascendante qui consiste à analyser les documents disponibles et permet de définir les éléments suivants :

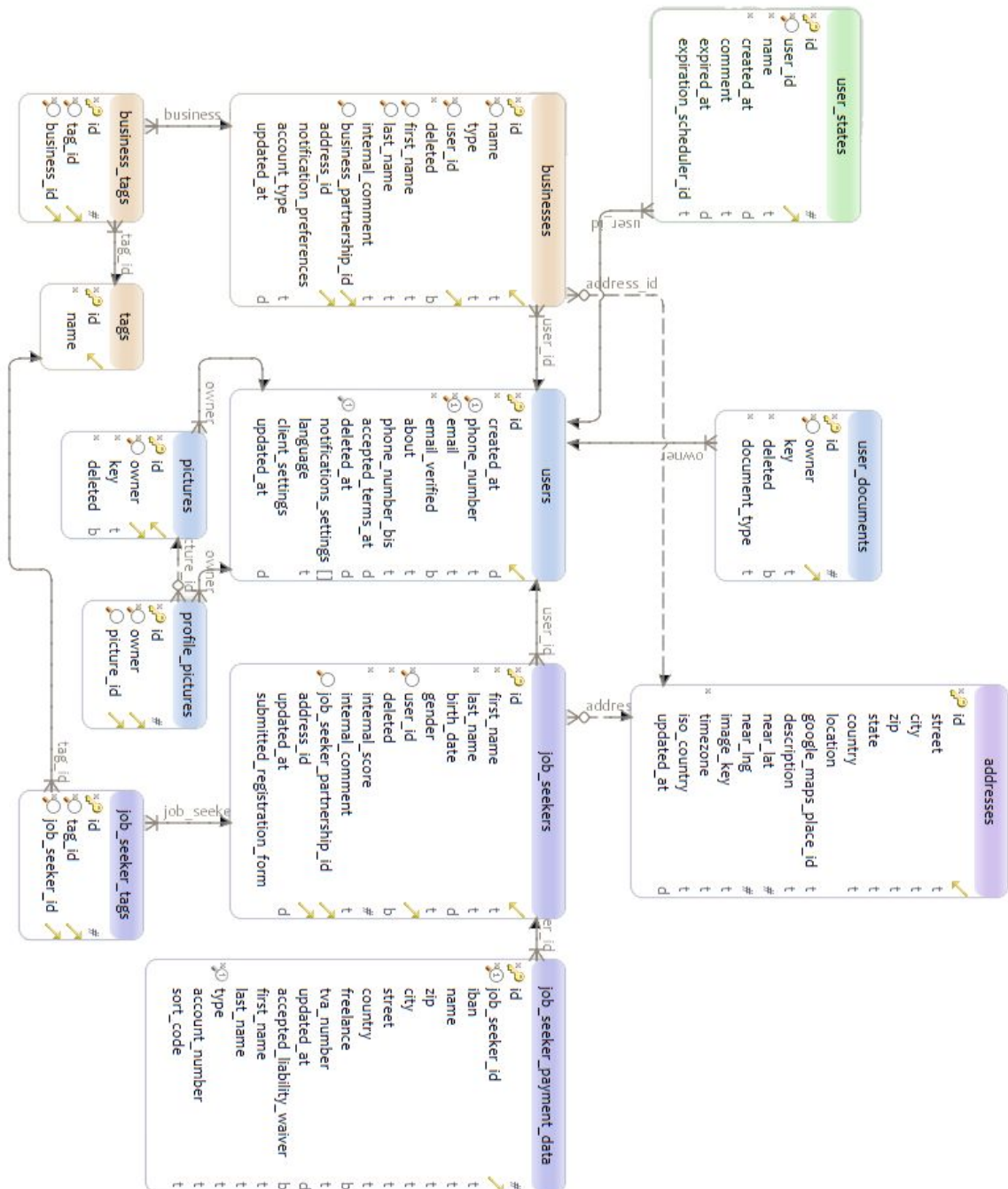
- Le dictionnaire des données
- La matrice des dépendances fonctionnelles
- Le graphe des dépendances fonctionnelles

Dans mon cas la base de données étant déjà en place je réaliserais la modélisation en me basant sur le schéma de la base de données.

---

## 4.2 Le diagramme de classes

## 4.3 Extrait du schéma de la base de données



---

## 4.4 SQL: Le LDD

Le SQL (Structured Query Language) est un langage informatique normalisé utilisé pour exploiter des bases de données relationnelles.

Le SQL est composé de 4 sous langages:

- LDD : Langage de Définition des Données (CREATE, ALTER, DROP)
- LCD : Langage de Contrôle des Données (GRANT, REVOKE)
- LMD : Langage de Manipulation des Données (INSERT, UPDATE, DELETE, SELECT)
- LCY Langage de Contrôle de Transactions (COMMIT, ROLLBACK)

Le LDD est donc utilisé pour manipuler les structures de données d'une base de données.

Les trois types principaux de commandes SQL de définition de données sont:

- CREATE: création d'une structure de données (base de données, table)
- ALTER : modification d'une structure de données
- DROP : suppression d'une structure de données

En annexes vous trouverez une partie du code de création de la base de données. Brigad dispose de plus d'une dizaine de base de données avec un total de plus de 200 tables, c'est donc pourquoi, j'ai dû en mettre qu'une partie.

---

#### 4.4.1 Extrait du code de création de la base de données

Chez Brigad les bases de données sont des bases de données PostgreSQL. Toute les base de données étant déjà en place, j'ai passé beaucoup de temps à l'analyse des ces bases, et en particulier celle sur laquelle j'ai travaillé.

La base de donnée sur laquelle j'ai travaillé s'appelle <<brigad-product>> et compte 61 tables.

```
CREATE DATABASE brigad-product  
  WITH OWNER = postgres  
        ENCODING =  
        TABLESPACE =  
        LC_COLLATE =  
        LC_TYPE =  
        CONNECTION LIMIT =
```

#### 4.4.2 Extrait du code de création d'une table

Une table de base données est un ensemble de données organisés sous forme de tableaux. Une table est donc composée de colonnes et de lignes.

Une table de base de données possède généralement (ce n'est pas obligatoire, mais très fortement conseillé) une clé Primaire: Elle est une contrainte d'unicité et est composé d'une ou plusieurs colonnes. Elle permet d'identifier chaque ligne de la table de manière unique. Une clé primaire ne peut donc être nulle.

La clé primaire a la particularité d'être indexé

Une table peut également comprendre zéro, une ou plusieurs clés étrangères (FOREIGN KEY).

Les clés étrangères sont des colonnes qui correspondent à une clé primaire dans une autre table.

Elles font le lien entre deux tables. Elles sont également des contraintes (CONSTRAINT) vue qu'elles font références (REFERENCE) à la clé primaire d'une autre table, c'est une contrainte référentielles.

```
CREATE TABLE job_seekers (  
    id SERIAL PRIMARY KEY,  
    first_name character varying(255) NOT NULL,  
    last_name character varying(255) NOT NULL,  
    birth_date date,  
    gender text CHECK (gender = ANY (ARRAY['male'::text,  
'female'::text])),  
    user_id uuid NOT NULL REFERENCES users(id),  
    deleted boolean NOT NULL DEFAULT false,  
    internal_score integer NOT NULL DEFAULT 0,  
    internal_comment text,  
    job_seeker_partnership_id integer REFERENCES  
job_seeker_partnerships(id),  
    address_id uuid REFERENCES addresses(id),  
    updated_at timestamp with time zone DEFAULT '2018-07-16  
14:12:49.03017+00'::timestamp with time zone,  
    submitted_registration_form jsonb  
);
```

```
-- Indices -----  
  
CREATE UNIQUE INDEX job_seekers_pkey1 ON job_seekers(id int4_ops);  
CREATE INDEX job_seekers_job_seeker_partnership_id_idx ON  
job_seekers(job_seeker_partnership_id int4_ops);  
CREATE INDEX job_seekers_user_id_idx ON job_seekers(user_id  
uuid_ops);
```



## Chapitre 5 - Conception de l'application

---

## 5.1 Architecture du projet

Le projet étant déjà en place, ce n'est pas moi qui en est crée l'architecture, mais je vais ici vous expliquer comment il est architecturé.

Tout d'abord côté Back end nous avons une API Rest développé en Javascript avec le framework Hapi.

Grâce à Hapi on peut facilement créer les routes (URL) de l'API Rest. Chaque routes fait des appels à la base de donnée PostgreSQL à l'aide de l'ORM Bookshelf et du Query builder Knex.

Bookshelf crée des modèles avec lesquels on peut exploiter les données venant de la base de donnée.

Côtés Front-end nous avons un serveur Node.Js avec le framework express qui permet générer l'interface graphique développé avec le framework React.Js.

Usuellement React.Js est exécuté coté client, directement dans le navigateur. C'est à dire que webpack et babel s'occupent de transformer tout le code React (JSX et ES6) en javascript classique, qui lui va créer les éléments HTML.

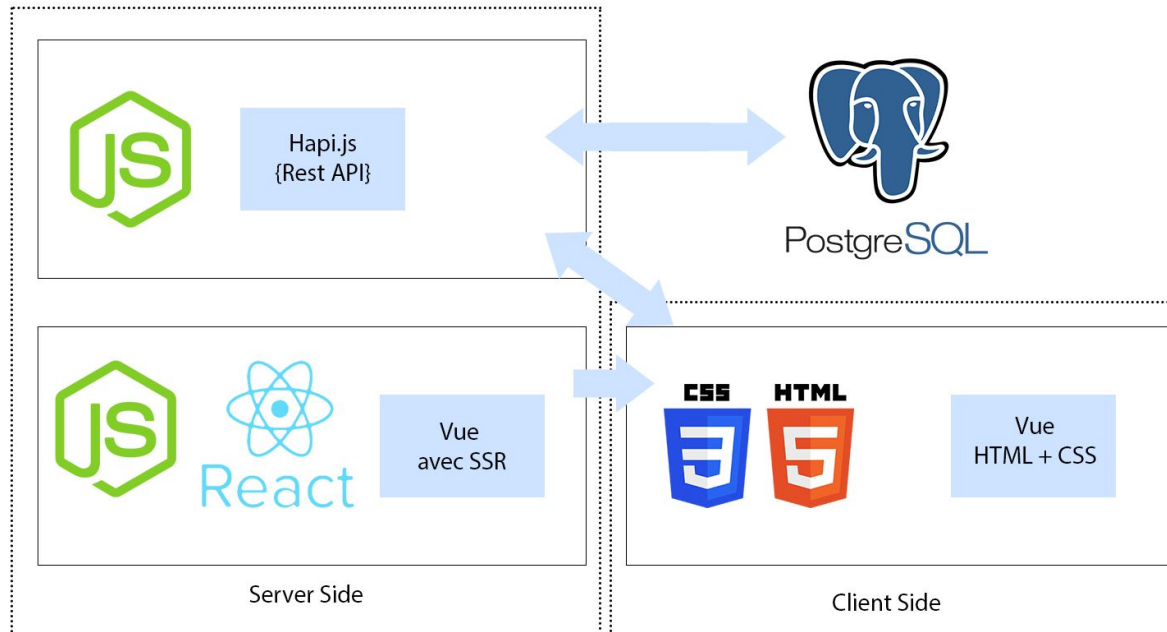
En soit il n'y a pas vraiment de HTML. Le peu de vrai HTML qu'il y a, est le index.html dans lequel on lie le fichier javascript créé par React dans une balise <script>.

Exemple du index.html du projet React classique :

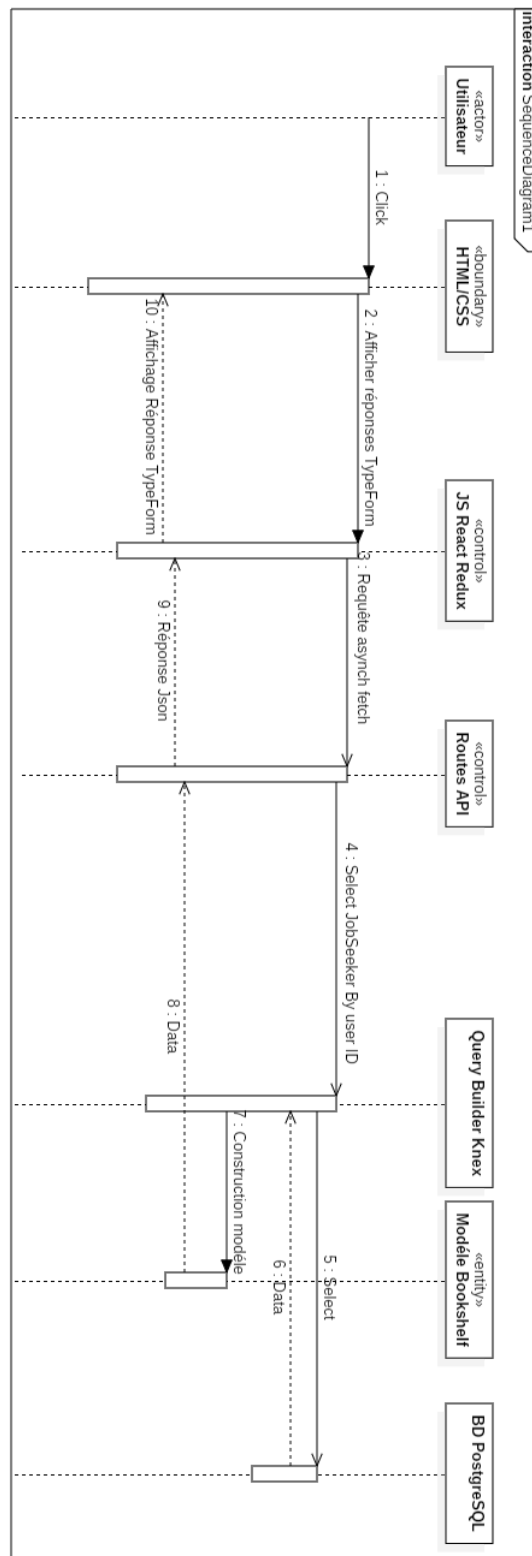
```
<!DOCTYPE html>
<html>
  <head>
    <title>React App</title>
  </head>
  <body>
    <div id="root"></div>
    <script src="/bundle.js"></script>
  </body>
</html>
```

Une fois qu'il a construit ces éléments, Javascript va les Append (Ajouter) à la div dont l'id est root.

Ici nous exécutons React.js côté serveur, pour faire ce qu'on appelle du Server Side Rendering (SSR). C'est à dire que toute la partie Javascript qui crée le HTML est fait du côté du serveur et envoie donc directement le HTML au client. Ce qui permet de faire du SEO (Search Engine Optimization) dans l'optique d'être référencé par le moteur de recherche.



## 5.2 Diagramme de séquence détaillé



## Chapitre 6 - Développement

---

## 6.1 Les interfaces

Interfaces d'un profil Brigader avec le bouton "Réponse TypeForm" sur l'admin panel.

The screenshot displays a web interface for managing Brigader profiles. The top navigation bar is blue and contains several menu items: CONFIGURATION, PAIEMENTS, LISTER LES BUSINESS, LISTER LES BRIGADERS, LISTER LES OFFRES, CRÉER UNE OFFRE, and WORKTIME. A French flag icon is also present.

The main content area shows the profile of a Brigader named **Robin SAFF**. The profile is marked as "Complète". The profile details are organized into two columns:

- Left Column:**
  - Adresse email:** robin1@brigad.com
  - Numéro de mobile:** +33 6 20 20 20 76
  - Genre:** Homme
  - Age:** 32 ans
  - IBAN:** N/A
  - Numéro SIREN:** N/A
  - Taux d'acceptation:** 0%
  - Taux de réponse:** 0%
  - Langue préférée:** Français
  - Créé le:** 20/07/2018
  - Description:** N/A
- Right Column:**
  - Rue:** 34 Rue de Rivoli
  - Ville:** Paris
  - Code postal:** 75004
  - Région:** Ile-de-France
  - Pays:** France
  - Partenariat:** N/A
  - Curriculum Vitae:** N/A
  - Pièce d'identité:** N/A

Below the profile details, there is a section for "Compétences" (Skills) with a dropdown menu and a "Réponse du TypeForm" button. At the bottom, there is a "Commentaire interne" (Internal Comment) field with a "Tags" section and an "Editer le profil" (Edit profile) button. The footer of the interface includes "HISTORIQUE DES STATUTS" and "Mettre à jour le statut" (Update status).

## Interfaces avec la modale d'affichage des réponses du TypeForm

**Complete**

**Robin SAFF**

0 recommandations

Compétences

Le Brigader a rempli le TypeForm 2 fois

Peux-tu exercer une activité professionnelle en France ?

Oui

As-tu de l'expérience en Salle ?

oui

Sélectionne le nombre de mois/années d'expérience que tu as en Salle

1 - 3 ans

Quel est le dernier poste que tu as occupé ?

Serveur

Nom de l'établissement ?

Hôtel du roi

Quand as-tu commencé à travailler en tant que {{answer\_90388853}} chez "{{answer\_90388906}}"? ?

34 Rue de Rivoli  
Paris  
75004  
Ile-de-France  
France

N/A  
N/A  
N/A

CONFIGURATION

PAIEMENTS

LISTER LES BUSINESS

LISTER LES BRIGADERS

LISTER LES OFFRES

CRÉER UNE OFFRE

WORKTIME

HISTORIQUE DES STATUTS

METTRE À JOUR LE STATUT

## Interface avec intégration du TypeForm

“ Nous avons besoin de connaître tes expériences professionnelles pour t'envoyer des missions qui correspondent à ton profil. Tu pourras détailler au maximum 2 expériences par domaine d'activité.

**Temps estimé: 5 minutes maximum**

Compte ··· Compétences ··· Validation profil ··· Première mission !

**C'est parti !** Appuyez sur ENTRÉE

1 → **Peux-tu exercer une activité professionnelle en France ?\***

Pour travailler en France il te faut obligatoirement un n° de sécurité sociale français (Carte Vitale)

0% complété

> <



---

## 6.2 Partie Back-End

---

### 6.2.1 Introduction

J'omet volontairement la présentation de toute la partie de création de serveur car elle n'a pas été codée par mes soins.

J'ai également volontairement coupé des parties du code pour des raisons de clarté et de concision, car certains fichiers font plusieurs centaines de lignes de codes.

Premièrement j'ai dû créer de nouvelles routes, afin de pouvoir accéder aux données voulues via l'API.

J'ai donc créé une route avec une requête POST pour pouvoir soumettre les informations du TypeForm et une avec une requête GET pour accéder à ces données.

Ensuite j'ai défini les actions à effectuer quand on appelle ces routes.

## 6.2.2 Les routes

/service-core/admin-panel/index.js

```
import * as getJobSeekerRegistrationForm from
'./actions/get-job-seeker-registration-form';
...
export default function() {
  async function register(server, _, next) {
    try {
      server.route([
...
{
  method: 'GET',
  path: '/v1/job-seekers/{user_id}/registration-form',
  config: getJobSeekerRegistrationForm.v1,
},
...
]);
    next();
  } catch (error) {
    next(error);
  }
}
```

/service-core/hooks/index.js

```
import * as postRegistrationForm from './actions/post-registration-form';
...
export default function() {
  async function register(server, _, next) {
    try {
      server.route([
...

```

```
{
  method: 'POST',
  path: '/v1/registration-form',
  config: { ...postRegistrationForm.v1, auth: false },
},
...
]);
  next();
} catch (error) {
  next(error);
}
}
```

**method** : définit la méthode de la requête (GET, POST, PUT, PATCH, DELETE)

**path** : correspond à l'URL de la route, les accolades ({} ) correspondent au paramètre passer.

**config** : le fichier dans lequel sont définis les actions à effectuer une fois que la route est appelé.

---

## 6.2.3 Les fichier de config des routes

---

### 6.2.3.1 POST

#### POST

Définitions des actions à effectuer quand la route POST est appelé.

Récupération du modèle JobSeeker et du modèle User.

Requête sur la table user avec jointure sur la table job-seeker.

Récupération des informations envoyé par le hook de TypeForm.

Formatage des réponses du TypeForm.

Calcul du nombre de fois que le TypeForm à été soumis.

Insertion des données dans la base de données.

Actualisation du statut du job-seeker.

post-registration-form.js

```
import Joi from '@brigad/util-joi';
import Boom from 'boom';
import { get } from 'lodash';

export const v1 = {
  tags: ['api', 'public'],
  handler: async ({ payload, plugins }, reply) => {
    const { job_seeker: jobSeekerModel, user: userModel } = plugins.v1.models;
    const { service_emitter: serviceEmitterExtension } = plugins.v1.extensions;

    const getUser = async userId =>
      userModel
        .query(qb =>
          qb
            .join('job_seekers', 'job_seekers.user_id', 'users.id')
            .where('users.id', '=', userId),
        )
        .fetch({
          columns: userModel.columns,
        });

    try {
```

```
if (
  !get(payload, 'form_response.hidden.user_id')
  || Joi.validate(payload.form_response.hidden.user_id, Joi.string().uuid())
  .error
) {
  return reply().code(202);
}

const user = await getUser(payload.form_response.hidden.user_id);
if (!user) {
  return reply().code(202);
}

const form = payload.form_response.definition.fields.map((question) => {
  const answerObj = payload.form_response.answers.find(
    elem => elem.field.id === question.id,
  );
  let answers = [];
  if (answerObj) {
    if (answerObj.type === 'choice') {
      answers = [answerObj.choice.label || answerObj.choice.other].filter(
        item => item,
      );
    } else if (answerObj.type === 'choices') {
      answers = answerObj.choices.labels;
    } else {
      answers = [answerObj[answerObj.type]];
    }
  }
}

return {
  id: question.id,
  answers_type: answerObj.type,
  question_type: question.type,
  question: question.title.replace(/<V?[^>]+(>|$)/g, ''), // Remove HTML markup
  answers,
};
});
```

```
const submittedRegistrationForm = user
  .related('job_seeker')
  .get('submitted_registration_form');
const submittedCount
  = (submittedRegistrationForm
    ? submittedRegistrationForm.submitted_count
    : 0) + 1;

const jobSeeker = await user.related('job_seeker').save(
  {
    submitted_registration_form: {
      registration_form: form,
      submitted_count: submittedCount,
    },
  },
  { patch: true },
);

user.set('job_seeker', jobSeeker);
const stateNameBeforeVerification = get(user.get('state'), 'name');
if (['uncompleted', 'completed'].includes(stateNameBeforeVerification)) {
  await jobSeekerModel.verify_registration_completed(user);
  await user.refresh({ columns: userModel.columns });

  if (stateNameBeforeVerification !== get(user.get('state'), 'name')) {
    await serviceEmitterExtension.emit('onJobSeekerStateUpdated', {
      user: {
        ...user.toJSON(),
        pushEnabled: await userModel.hasPushNotificationEnabled(
          user.get('id'),
        ),
      },
    });
  }
}

return reply({
  submitted_registration_form: {
    registration_form: form,
```

```
    submitted_count: submittedCount,
  },
});
} catch (error) {
  return reply(Boom.wrap(error));
}
},
response: {
  status: {
    202: Joi.object().allow(null),
    200: Joi.object().keys({
      submitted_registration_form: Joi.object().keys({
        registration_form: Joi.array().items(
          Joi.object().keys({
            id: Joi.string().required(),
            answers_type: Joi.string().required(),
            question_type: Joi.string().required(),
            question: Joi.string().required(),
            answers: Joi.array().required(),
          }),
        ),
      submitted_count: Joi.number()
        .integer()
        .min(1)
        .required(),
    }),
  }),
},
},
};
```

Avec Bookshelf et Knex

```
const getUser = async userId =>
  userModel
    .query(qb =>
      qb
        .join('job_seekers', 'job_seekers.user_id', 'users.id')
        .where('users.id', '=', userId),
    )
```

```
.fetch({  
  columns: userModel.columns,  
});
```

En SQL:

```
SELECT * FROM users  
INNER JOIN job_seekers ON job_seekers.user_id = users.id  
WHERE user_id = ?
```

Avec Bookshelf et Knex

```
const jobSeeker = await user.related('job_seeker').save(  
  {  
    submitted_registration_form: {  
      registration_form: form,  
      submitted_count: submittedCount,  
    },  
  },  
  { patch: true },  
);
```

En SQL:

```
UPDATE job_seekers  
SET submitted_registration_form = ?  
WHERE user_id = ?
```



### 6.2.3.1 GET

Définitions des actions à effectuer quand la route GET est appelé.

Récupération du modèle JobSeeker.

Requête sur la table job-seeker.

Renvoie des données

get-job-seeker-registration-form.js

```
import Boom from 'boom';
import Joi from 'joi';

export const v1 = {
  validate: {
    params: Joi.object().keys({
      user_id: Joi.string().uuid(),
    }),
  },
  handler: async ({ params, plugins }, reply) => {
    const { job_seeker: jobSeekerModel } = plugins.v1.models;

    try {
      const jobSeeker = await jobSeekerModel
        .query(qb => qb.where('user_id', '=', params.user_id))
        .fetch({
          columns: jobSeekerModel.columns,
        });

      if (!jobSeeker) {
        return reply(Boom.notFound('The job_seeker requested does not exist'));
      }

      return reply({
        submitted_registration_form: jobSeeker.get(
          'submitted_registration_form',
        ),
      });
    } catch (error) {
```

```
    return reply(Boom.wrap(error));
  }
},
response: {
  status: {
    200: Joi.object().keys({
      submitted_registration_form: Joi.object()
        .keys({
          registration_form: Joi.array().items(
            Joi.object().keys({
              id: Joi.string().required(),
              answers_type: Joi.string().required(),
              question_type: Joi.string().required(),
              question: Joi.string().required(),
              answers: Joi.array().required(),
            }),
          ),
      submitted_count: Joi.number()
        .integer()
        .min(1),
    })
    .allow(null),
  }),
},
},
};
```

Avec Bookshelf et Knex

```
const jobSeeker = await jobSeekerModel
  .query(qb => qb.where('user_id', '=', params.user_id))
  .fetch({
    columns: jobSeekerModel.columns,
```

En SQL:

```
SELECT * FROM job_seekers
WHERE user_id = ?
```

## 6.2.4 Les modèles Bookshelf

Une modèle Bookshelf contient les colonnes de la table mais également des données supplémentaires, tel que des fonction ou des attributs qui sont des jointure sur d'autre tables

Les modèles de l'orm Bookshelf sont créé de la sorte.

(Je n'ai gardé que les parties cohérente avec le code précédemment montré).

Modèle de Job-seeker

```
/* eslint-disable no-param-reassign,camelcase */
import { get, round } from 'lodash';

import User from './user';
import JobSeekerPayment from './job-seeker-payment';
import JobSeekerZones from './job-seeker-zones';
import JobSeekerPayedOffersList from './job-seeker-pay-ed-offers-list';

import Tag from './tag';
import Address from './address';
import JobSeekerTag from './job-seeker-tag';
import UserState from './user-state';

import JobSeekerPartnership from './job-seeker-partnership';
import JobSeekerPaymentData from './job-seeker-payment-data';

class JobSeeker {
  constructor(params) {
    const { database, service_files } = params;

    if (database.initialised(JobSeeker.name)) {
      return database.get(JobSeeker.name);
    }
    const extendedModel = database.bookshelfClient.Model.extend(
      {
        tableName: 'job_seekers',
        tag_list() {
          return this.belongsToMany(new Tag(params)).through(
```

```
        new JobSeekerTag(params),
    );
},
partnership() {
    return this.belongsTo(
        new JobSeekerPartnership(params),
        'job_seeker_partnership_id',
    );
},
address() {
    return this.belongsTo(new Address(params), 'address_id');
},
payment_data() {
    return this.hasOne(new JobSeekerPaymentData(params));
},
virtuals: {
    acceptance_rate() {
        return !this.get('total_propositions')
            ? 0
            : round(
                this.get('total_accepted_propositions')
                / this.get('total_propositions')
                * 100,
                2,
            );
    },
},
{
    columns: [
        'job_seekers.id',
        'job_seekers.first_name',
        'job_seekers.last_name',
        'job_seekers.birth_date',
        database.bookshelfClient.knex.raw(
            "date_part('year',age(birth_date)) as age",
        ),
        'job_seekers.gender',
        'job_seekers.internal_score',
        'job_seekers.internal_comment',
        'job_seekers.job_seeker_partnership_id',
    ],
}
```

```
    'job_seekers.submitted_registration_form',
      abstract: abstractColumns,
required: [
  'first_name',
  'last_name',
  'birth_date',
  'gender',
  'address_id',
  'submitted_registration_form',
],
async verify_registration_completed(userReq) {
  const userJson = userReq.toJSON();
  const obj = {
    ...userJson,
    ...userJson.job_seeker,
    ...userJson.address,
  };
  const tab = [...this.required, ...new User(params).required];
  const isCompleted = tab.reduce(
    (last, key) => last && obj[key] && obj[key] !== '',
    true,
  );
  if (
    !isCompleted
    && ['completed', 'validated'].includes(get(userJson.state,
'name'))
  ) {
    await new UserState(params).add(userJson.id, 'uncompleted');
  } else if (
    isCompleted
    && get(userJson.state, 'name') === 'uncompleted'
  ) {
    await new UserState(params).add(userJson.id, 'completed');
  }

  return userReq.refresh({
    columns: database.columns.user,
  });
},
```

```
    },  
  );  
  database.set(JobSeeker.name, extendedModel);  
  return extendedModel;  
}  
}  
  
export default JobSeeker;
```

---

## 6.3 Partie Front-End

---

### 6.3.1 Introduction

La partie Front end est principalement articulé autour de deux bibliothèques Javascript :

**React:** qui sert à créer les vues et gérer le raisonnement logique de l'application

**Redux:** qui nous sert à faire les requêtes HTTP vers l'api et contenir ses informations dans un state général accessible depuis n'importe quel point de l'application.

React utilise une syntaxe particulière, le JSX. JSX permet de créer des composants comme si ils étaient des balises HTML. Chaque composant peut recevoir des propriétés/paramètres appelées props.

Comme le code est extrêmement vaste je ne vais en mettre qu'une partie.

### 6.3.2 Code de l'interface de présentation du TypeForm

Code de l'interface de la présentation du TypeForm

```
import React from 'react';
import PropTypes from 'prop-types';
import { injectIntl, intlShape } from 'react-intl';

import { connect } from '@brigad/redux-rest-easy';
import { retrieveOwnProfile, getOwnProfile } from 'src/redux/profiles';
import { invalidateJobSeekerId } from
'src/redux/profiles/jobSeekersPrivate';
import { stringifyQueryParameters } from '@brigad/shared/utils/StringUtils';
import { UserJobSeekerPrivateProps } from
'src/proptypes/api-models/UserPrivateProps';
import { getToken } from '@brigad/shared/utils/TokenUtils';
import { getUserIdFromToken } from '@brigad/shared/utils/UserTokenUtils';
import LoadingContent from
 '@brigad/shared/components/atomic/loading-content/js/LoadingContent';
import Flex from '@brigad/shared/components/atomic/flexbox/js/Flex';
import { getJobSeekerRegistrationTypeform } from 'src/utils/TypeFormUtils';
import { isUserCompleted } from
 '@brigad/shared/utils/business-logic/UserLogicUtils';
import WrongCountry from './WrongCountry';

import styles from
 '../css/components/JobSeekerRegistrationTypeForm.scss';

const MAX_TRIES = 10;
const getUserId = () => getUserIdFromToken(getToken());

class JobSeekerRegistrationTypeForm extends React.Component {
  tries = 0;

  state = {
    showLoader: true,
    shouldFetchUser: true,
  };
};
```



```
componentDidMount() {
  this.listener = window.addEventListener('message', (event) => {
    if (event.data === 'form-submit') {
      this.setState({
        showLoader: true,
      });
      this.waitForUserCompleted();
    }
  });
}

componentWillUnmount() {
  window.removeEventListener('message', this.listener);
}

hideLoader = () => {
  this.setState({ showLoader: false });
};

checkIfUserCompleted = (normalizedPayload) => {
  const keyName = Object.keys(normalizedPayload)[0];
  const users = normalizedPayload[keyName];
  const userId = Object.keys(users)[0];
  const user = users[userId];

  if (isUserCompleted(user)) {
    this.setState({
      showLoader: false,
      shouldFetchUser: false,
    });
    this.props.onSuccess();
  }
};

waitForUserCompleted = async () => {
  await this.props.invalidateUser();
  await this.props.retrieveUser(this.checkIfUserCompleted);
  this.tries += 1;
};
```

```
    if (this.tries < MAX_TRIES && this.state.shouldFetchUser) {
      setTimeout(this.waitForUserCompleted, this.tries * 1000);
    }
  };

  render() {
    const {
      intl: { locale },
      user,
    } = this.props;
    const content = !['FR', 'GB'].includes(user.address.iso_code) ? (
      <WrongCountry />
    ) : (
      <>
        <iframe
          title="regularise-job-offer"
          src={`${getJobSeekerRegistrationTypeform(
            user.address.iso_code,
            locale,
          )}${stringifyQueryParameters({
            email: user.email,
            user_id: user.id,
            first_name: user.job_seeker.first_name,
            last_name: user.job_seeker.last_name,
          })}` }
          onLoad={this.hideLoader}
          className={styles.iframe}
        />
        {this.state.showLoader && (
          <Flex justifyContent="center" className={styles.loader}>
            <LoadingContent bigSize />
          </Flex>
        )}
      </>
    );
    return content;
  }
}
```

```
JobSeekerRegistrationTypeForm.propTypes = {
  retrieveUser: PropTypes.func.isRequired,
  invalidateUser: PropTypes.func.isRequired,
  user: UserJobSeekerPrivateProps.isRequired,
  onSuccess: PropTypes.func.isRequired,
  intl: intlShape.isRequired,
};

const mapStateToProps = state => ({
  user: getOwnProfile(state),
});

const mapDispatchToProps = dispatch => ({
  retrieveUser: onSuccess => dispatch(retrieveOwnProfile({ onSuccess })),
  invalidateUser: () => dispatch(invalidateJobSeekerId(getUserId())),
});

export default connect(
  mapStateToProps,
  mapDispatchToProps,
)(injectIntl(JobSeekerRegistrationTypeForm));
```

### 6.3.3 Code de la modale d'affichage des réponses du TypeForm

Code de la modale de présentation des réponses du TypeForm

```
import PropTypes from 'prop-types';
import React from 'react';
import { FormattedDate } from 'react-intl';

import Flex from '@brigad/shared/components/atomic/flexbox/js/Flex';
import Item from '@brigad/shared/components/atomic/flexbox/js/Item';
import LoadingContent from
'@brigad/shared/components/atomic/loading-content/js/LoadingContent';
import Modal from '@brigad/shared/components/atomic/modal/js/Modal';
import FormattedMessage from
'@brigad/shared/components/atomic/intl/js/FormattedMessage';

import ExternalLink from
'@brigad/shared/components/atomic/links/js/ExternalLink';

import styles from '../css/JobSeekerTypeFormModal.scss';

const parseAnswers = (type, answer) => {
  switch (type) {
    case 'text':
    case 'choice':
    case 'email':
    case 'choices':
    case 'number':
      return answer;

    case 'url':
    case 'file_url':
      return (
        <ExternalLink aspect="blue-link" href={answer}>
          <FormattedMessage id="Grammar.link" />
        </ExternalLink>
      );
    case 'boolean':
      return answer ? (
```

```
      <FormattedMessage id="Grammar.yes" />
    ) : (
      <FormattedMessage id="Grammar.no" />
    );

    case 'date':
      return <FormattedDate value={answer} />;
    default:
      return null;
  }
};

const JobSeekerTypeFormModal = (props) => {
  const content = !props.isLoading ? (
    props.data.map((form, index) => {
      const Separator = !!index && <hr />;

      const answerRes = form.answers.map((answer, indexAnswers) => (
        // eslint-disable-next-line react/no-array-index-key
        <Item key={indexAnswers} className={styles.answers}>
          {parseAnswers(form.answers_type, answer)}
        </Item>
      ));

      return (
        <div key={form.id}>
          {Separator}
          <Flex className={styles.questions}>
            <Item>{form.question}</Item>
          </Flex>
          <Flex column>{answerRes}</Flex>
        </div>
      );
    })
  ) : (
    <LoadingContent bigSize />
  );

  return (
```

```
<Modal show={props.show} onHide={props.onHide} className={styles.modal}>
  <div className={styles.submissions}>
    <FormattedMessage

id="JobSeekerDetailsPage.JobSeekerDisplayInfos.modal.numberOfSubmission"
    values={{ number: props.metadata.submittedCount }}
  />
  </div>
  {content}
</Modal>
);
};

JobSeekerTypeFormModal.propTypes = {
  data: PropTypes.arrayOf(PropTypes.object).isRequired,
  metadata: PropTypes.shape({ submittedCount: PropTypes.number }).isRequired,
  isLoading: PropTypes.bool.isRequired,
  show: PropTypes.bool.isRequired,
  onHide: PropTypes.func.isRequired,
};

export default JobSeekerTypeFormModal;
```

### 6.3.4 Reducer Redux

Code du Reducer Redux pour `jobSeekerTypeFormAnswers`

```
import { createResource } from '@brigad/redux-rest-easy';
import { normalize } from 'normalizr';

import { getURL } from '@brigad/shared/utils/APIUtils';

import { jobSeekerTypeFormAnswersSchema } from
'src/schemas/jobSeekerTypeFormAnswers';

const jobSeekerTypeFormAnswers =
createResource('jobSeekerTypeFormAnswers')({
  retrieveJobSeekerTypeFormAnswers: {
    method: 'GET',
    url: `${getURL('core', 1)}/job-seekers/:userId/registration-form`,
    normalizer: ({
      submitted_registration_form: { registration_form: registrationForm },
    }) => normalize(registrationForm, jobSeekerTypeFormAnswersSchema),
    metadataNormalizer: ({
      submitted_registration_form: { submitted_count: submittedCount },
    }) => ({ submittedCount }),
  },
});

const {
  actions: {
    retrieveJobSeekerTypeFormAnswers: {
      perform: retrieveJobSeekerTypeFormAnswers,
    },
  },
  selectors: {
    retrieveJobSeekerTypeFormAnswers: {
      request: {
        getResource: getJobSeekerTypeFormAnswers,
        getMetadata: getJobSeekerTypeFormAnswersMetadata,
        hasSucceeded: hasRetrievedJobSeekerTypeFormAnswers,
      },
    },
  },
}
```

```
    },  
    },  
    },  
  } = jobSeekerTypeFormAnswers;  
  export {  
    retrieveJobSeekerTypeFormAnswers,  
    getJobSeekerTypeFormAnswers,  
    hasRetrievedJobSeekerTypeFormAnswers,  
    getJobSeekerTypeFormAnswersMetadata,  
  };
```

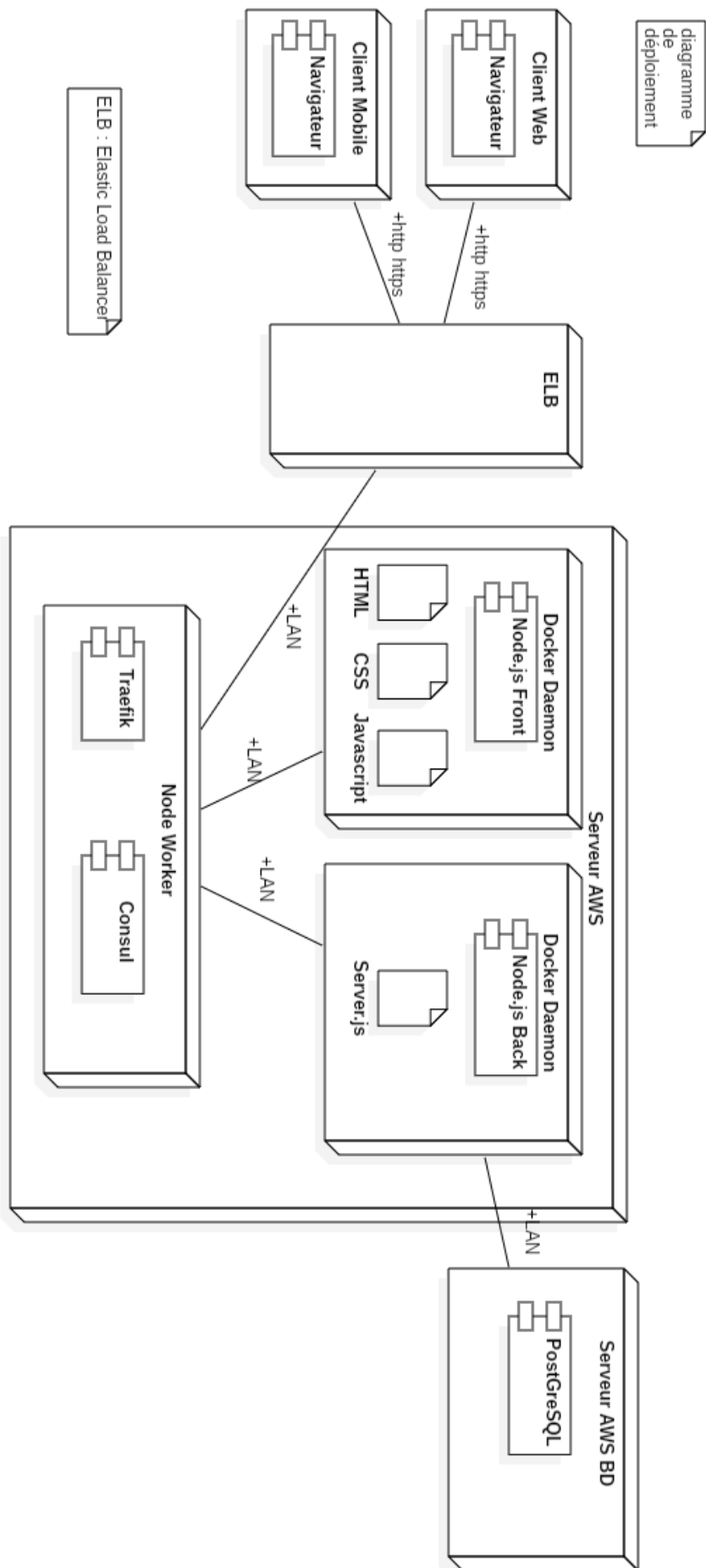


## Chapitre 7 - Déploiement

---

---

## 7.1 Diagramme de déploiement



---

## 7.2 Le déploiement

Chez Brigad la stratégie de développement et de déploiement est celle qu'on appelle déploiement continu.

Les développeur travaille en local sur une branche correspondant à un tâche (généralement un ticket Jira).

Une fois sont travail terminé il envoie (push) sa branche sur Github et fait une demande de Pull Request (PR). Chaque Pull Request déclenche un build dans CircleCI.

Circle CI est un outils intégré à Github qui permet de lancer la construction d'un projet et de lancer ses test, récupérer le rapport de couverture de code et le déploiement.

Si tout les test passent, un autre développeur (en général le lead) va relire le code et s'assurer qu'il est propre et ne crée aucun conflit. Si tout est bon la PR sera fusionné (merge) avec la branche master de développement.

Ensuite on crée une PR de la branche master, pour la fusionner avec la branche Production. Quand la Pull request est accepté la branche Master dev et merge dans la branche Production et CircleCI va construire la nouvelle image Docker et l'envoyer sur le serveur de production.

## Chapitre 8 - Conclusion

---

---

## 8.1 Retour d'expériences

Ce stage de trois mois au seins d'une start-up grandissante, m'a conforté dans l'idée d'avoir choisi le domaine qui me correspond et dans lequel j'ai désormais envie d'évoluer.

En plus d'avoir consolider mes connaissances acquises en autodidacte et durant la formation : UML, HTML/CSS, Javascript, React.JS, Redux.JS, Node.JS, SQL ... j'y ai appris à travailler en équipe et à voir en pratique concrète comment répartir des tâches grâce aux méthodes agiles .

En plus de l'aspect bénéfiques de la pratique, j'ai pu avoir un retour direct sur mon travail, car les différentes options que j'ai implémenté on était déployer en production et on déjà eu un retour Utilisateur. Ce qui m'as permit de comprendre et d'apprendre de certaine de mes erreurs et me gratifier de mes réussites.

Pour conclure je peux dire que l'objectif de ma reconversion professionnelle est atteint, puisqu'à l'issue du stage Brigad m'a proposé de continuer l'aventure avec eux en me proposant un CDI.

## Chapitre 9 - Annexes

---

## 9.1 Extrait du CSS

### JobSeekerTypeFormModal.scss

```
@import '~@brigad/shared/styles/texts.scss';
@import '~@brigad/shared/styles/colors.v3.scss';

.header {
  max-width: 300px;
}

.questions {
  @include subTitleBold;

  margin-bottom: 30px;
}

.answers {
  font-weight: $bold-weight;
  color: $medium-grey;
}

.submissions {
  margin-bottom: 30px;
}
```

### JobSeekerRegistrationTypeForm.scss

```
.iframe {
  position: fixed;
  top: 60px;
  left: 0;
  right: 0;
  bottom: 0;
  height: calc(100vh - 60px);
  width: 100vw;
  border: 0;
```



```
}  
  
.loader {  
  position: fixed;  
  top: 60px;  
  left: 0;  
  right: 0;  
  bottom: 0;  
  height: calc(100vh - 60px);  
  width: 100vw;  
  border: 0;  
  z-index: 10;  
  background-color: rgba(255, 255, 255, 0.9);  
}
```