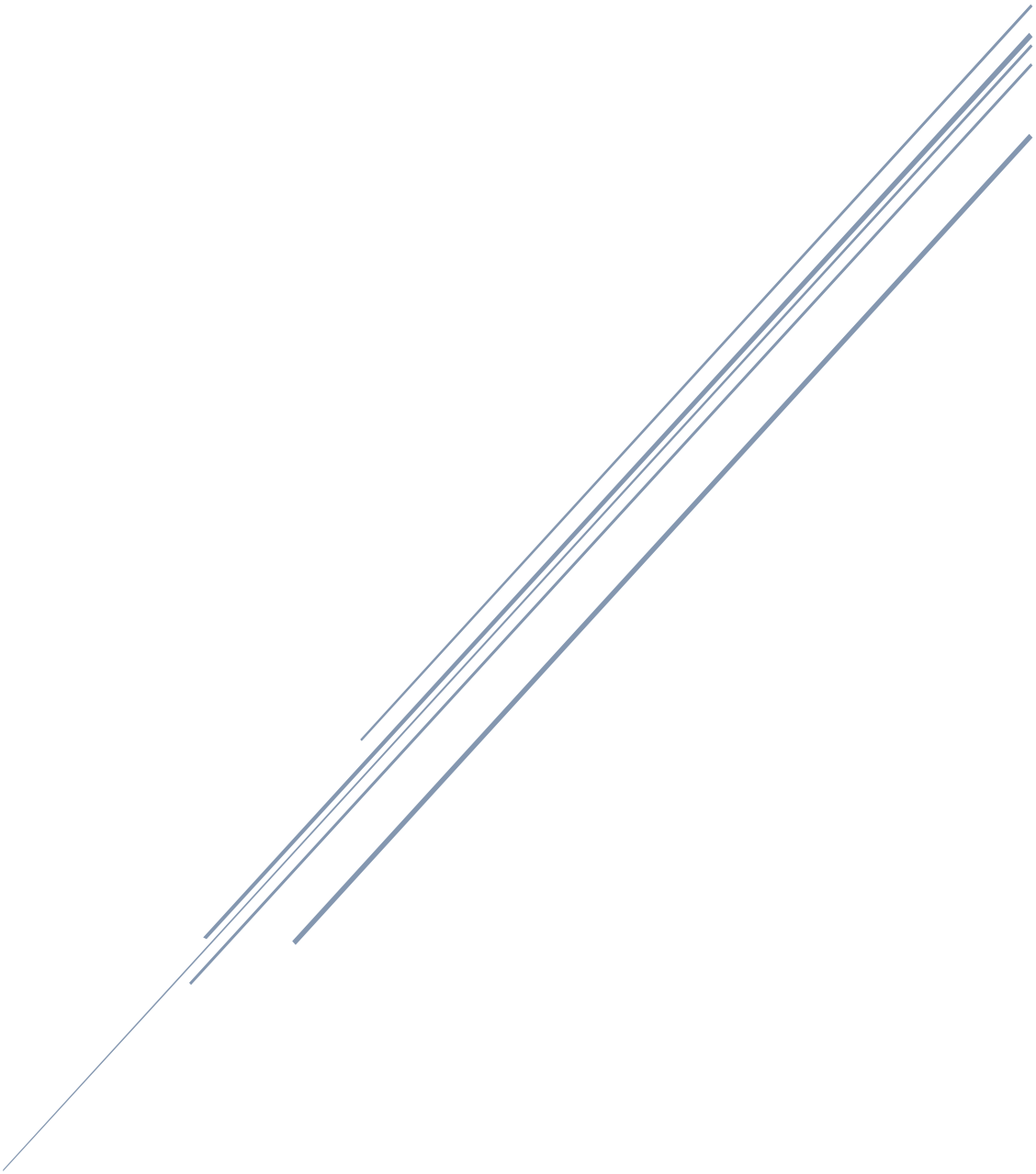
RAPPORT D'ACTIVITE

Nathalie DURASSIER-BONHEUR



EPSI

1ère Année Bachelor Informatique - 2018/2019

REMERCIEMENTS

En premier lieu, je tiens à remercier M. Nicolas DAUDIN, chef de projet chez OBS SA. En tant que maître de stage, il a su m'accompagner tout au long de cette période et a partagé ses connaissances dans le domaine informatique.

Je souhaite remercier aussi M. Yann SCHEPENS et M. Luiguy LEMACON, respectivement expert technique et développeur au sein de l'entreprise OBS SA ainsi que mon maitre de stage pour leur confiance et les connaissances et les compétences qu'ils ont pu partager avec moi. Je les remercie aussi pour leur disponibilité et la qualité de leur encadrement en entreprise.

Je remercie ensuite toutes les équipes d'Orange Business Service, qu'il s'agisse de développeurs, de managers, d'experts technique, de chefs de projets ou autres, pour leur accueil chaleureux et leur disponibilité durant ce stage.

J'adresse enfin mes remerciements au corps professionnel et administratif d'EPSI Nantes pour la qualité de leur enseignement ainsi que pour leur disponibilité.

SOMMAIRE

Orange est une société anonyme à conseil d'administration. En activité depuis 28 ans, Orange est spécialisée dans le secteur d'activité des télécommunications filaires. Orange Business Services SA (OBS SA) est une unité du groupe Orange regroupant ainsi toutes ses activités IT. Auparavant, Orange Application for Business était une filiale d'OBS SA. Désormais rattachée à OBS SA, la société est spécialisée dans les domaines de l'expérience client, du Big Data et des objets connectés et plus précisément dans l'intégration de systèmes et la fourniture de services applicatifs sur mesure ou en mode SaaS. Autrement dit, les domaines de prédilection de la société sont les objets communicants, l'expérience client, le Big Data et l'analytique.

Etroitement lié au secteur de l'informatique et du numérique, le secteur d'activité des télécommunications est en secteur encore en plein développement. Tout comme le secteur d'activité de l'information, le secteur des télécommunications a été touché par la révolution de la téléphonie mobile a fait exploser ces deux secteurs.

Le secteur de l'informatique, numérique et réseau génère énormément d'emplois et affiche une croissance positive. Dans un monde où les nouvelles technologies sont de plus en plus présentes, ce secteur est de plus en plus sollicité. L'informatique et les télécoms deviennent désormais indispensables aux entreprises quel que soit leur secteur d'activité. La majorité d'entre elles gèrent leur système d'informations (contacts, achat* de matières, vente, tenue des stocks, gestion du personnel, etc…) ainsi que les processus de conception et de fabrication de leur production par informatique. Ce secteur fait partie des marchés qui recrutent le plus de jeunes diplômés et peine même à trouver des profils qualifiés.

Le stage s'est étalé sur une durée de cinq semaines, du lundi 6 mai au vendredi 7 juin 2019. La mission confiée devait donc être faisable sur cette courte période. La durée du stage a donc dû être prise en compte lors de la mise en place d'un planning et fait partie des contraintes du projet.

Les locaux sont situés au sein d'un parc d'activité. Les locaux d'OBS SA sont composés de différents bureaux et différents open-space. L'open-space où le stage s'est déroulé est composé entre autres de développeurs, de chef de projets, d'experts techniques, d'architectes. Les deux managers ont leur bureau dans l'open-space. Au total, cet espace rassemble plus d'une vingtaine de personnes. Différentes technos sont utilisées dont le PHP et le javascript*. Chaque membre de l'open-space sait se rendre disponible pour ses collègues, dans la mesure du possible. On distingue différents statuts : salarié, alternant/stagiaire, prestataire.

La mission avait pour but de développer une interface utilisateur d'un chat*. Cette même mission en englobe deux : la conception d'une interface utilisateur puis la réalisation de la partie client du chat. Le langage de programmation a été définit en amont, le javascript en utilisant deux librairies* : Socket.io et ReactJS*. L'interface a été conçue en ReactJS, langage issu du javascript. Ces missions venaient donc complémenter un projet de l'entreprise. Le serveur* du chat a été développé en interne par un développeur prestataire chez OBS SA. Il restait donc à développer une interface pour permettre aux utilisateurs d'utiliser ce chat et avant tout il s'agissait de tester le bon fonctionnement du chat.

Avant d'évoquer les démarches suivies afin de réaliser les missions, nous allons présenter et préciser ces dernières. La réalisation des missions sera ensuite détaillée. Nous conclurons par une évaluation des réalisations et des compétences qui auront été mises en œuvre durant cette période de stage en entreprise.

A – Objectifs des mission données

L'objectif de ce stage était donc de réaliser une interface utilisateur d'un chat et de développer la partie client du chat. La partie serveur du chat a été réalisée en amont par un développeur (prestataire chez OAB). La mission concernait donc la partie client* du serveur*. Le projet du chat fait partie d'un plus grand projet d'OAB, un projet visant la transformation du Groupe Orange.

Il y a eu trois interlocuteurs principaux durant ces cinq semaines de stage : le chef de projet (Nicolas DAUDIN), l'expert technique (Yann SCHEPENS) et le développeur en charge du développement du chat (Luiguy LEMACON). Ces personnes ont joué un rôle déterminant dans le processus de réalisation des missions. L'interaction sur le projet se faisait principalement avec ces trois personnes.

Quant aux enjeux pour l'entreprise, il s'agissait avant tout de tester l'existant. En effet, une base de départ était déjà existante. Le serveur du chat a déjà été développé. Il manquait donc la partie client du chat*, autrement dit la partie front end. Cette partie front end correspond à une interface utilisateur communiquant avec le serveur du chat. L'enjeu est donc important, puisqu'il s'agit de tester le bon fonctionnement du serveur chat.

Le projet a été décomposé en diverses étapes qui ont été déterminées à l'avance avec l'aide du chef de projet et de l'expert technique. La première étape a été de monter en compétence sur ReactJS*. En effet, le javascript* était un nouveau langage, qui n'avait jamais vraiment été pratiqué auparavant. ReactJS est une librairie de Javascript. Cette-dernière permet de créer facilement des interfaces utilisateurs. Ce langage a été choisi en amont par l'équipe du projet. Rappelons que cette mission vient remplir un besoin d'un gros projet d'OBS SA. L'interface devant être réalisée permet de tester le bon fonctionnement du serveur chat réalisé en interne.

Divers logiciels* ont été utilisés, certains étaient connus tandis que d'autres ont été découverts au sein de l'entreprise durant la période de stage. L'environnement de développement choisi est PHP Storm*, déjà utilisé auparavant, aucune phase de découverte ou de prise en main n'a été nécessaire. L'objectif était donc, en très peu de temps, de monter en compétence sur une nouvelle techno et d'être capable de réaliser le produit demandé. L'objectif a donc été atteint.

B – Contraintes, limites et potentiels problèmes à résoudre

Assez vite, on a pu noter quelques contraintes à ce projet. La première étant la découverte du langage ainsi que la contrainte temporaire. Le stage ne comptait seulement 22 jours effectifs dans les bureaux d'OAB. Le projet devait donc tenir en moins de 20 jours, laissant une marge éventuelle pour l'installation du poste de travail.

Des problèmes de proxy ont souvent été relevés. Ce problème est indépendant au projet et vient du proxy de l'entreprise. La configuration proxy de la machine a cependant dû être vérifiée un certain nombre de fois.

Une autre contrainte est la nouveauté des langages utilisés. Le javascript n'avait jamais été utilisé auparavant, ce fut donc une découverte. ReactJS était avant le stage un framework inconnu. Il a donc fallu apprendre à les utiliser et ainsi monter en compétence assez rapidement.

Enfin, la contrainte du projet vient du projet en lui-même. Composé de deux parties, une partie cliente et une partie serveur, la partie serveur avait déjà été utilisée. Il fallait donc s'adapter au code déjà existant.

A – Mise en place d'un suivi de réalisation

Afin de mener à bien ce projet, divers outils de communications ont été utilisés : lignes téléphoniques, mails, Slack*, outils internes à l'entreprise. Slack permet d'échanger avec les différents membres de l'open-space. Cet outil, utilisé par tous les membres de l'open-space, a donc permis de demander de l'aide en cas de besoin. Ces différents outils de communications ont été utilisés dans la gestion du projet notamment.

La période de stage étant assez courte, celle-ci a été organisée dès le début du stage avec un chef de projet*. Dans la construction d'un projet, l'organisation est une chose primordiale. Même si, les chefs de projet se chargent de l'organisation du projet, un développeur doit savoir s'organiser et gérer, optimiser son temps de travail. Savoir gérer un projet est un métier d'où la fonction chef de projet ou autrement dit manager de projet. Le rôle du chef de projet est d'organiser et de conduire le projet de bout en bout. Il est donc responsable des différentes phases du projet. De la traduction des besoins du client en spécifications fonctionnelles, en passant par le suivi de production, jusqu'à la recette utilisateur ou la mise en production, le chef de projet suit le déroulement de la production.

On entend souvent parler des méthodes Agiles en entreprise et pour cause ces-dernières sont utilisées. Il s'agit d'une nouvelle – bien que ces méthodes existent désormais depuis quelques années – manière d'aborder la gestion de projet. C'est une autre approche de la gestion de projet. La méthode employée pour ce projet est la méthode Scrum, signifiant « mêlée » en français. Celle-ci n'a pas été véritablement exploitée par l'apprenant, cependant des travaux ont été attendus comme la création d'un backlog et de user stories ainsi que la réalisation d'un chiffrage.

Ainsi, après deux jours d'installation du poste, de configurations et de découvertes du langage ReactJS, un entretien avec le chef de projet et l'expert technique a eu lieu. La mission du stage a ainsi été détaillée. L'expert technique a expliqué quelles étaient les prochaines étapes du processus de réalisation.

Un premier entretien avec l'expert technique a permis de cibler plus précisément ce en quoi la première mission allait consister et a permis d'établir un cahier des charges*. Aucune véritable contrainte n'a été posée, si ce n'est la durée du stage. La demande de l'expert technique*, qui se place donc en tant que client, est simple : réaliser une interface utilisateur afin d'intégrer un chat. Cette demande constitue donc le cahier des charges. Durant cet entretien, quelques étapes ont tout de même été suggérées : faire une maquette de l'interface, réaliser des user stories afin de créer un backlog et enfin établir un chiffrage afin de chiffrer en heures le besoin de chaque tâche listée. En se basant sur des journées de 7.5 heures effectives par jour en moyenne, on arrive à un total de cent soixante-cinq (165) heures de travail sur vingt-deux (22) jours de stage.

La réalisation de user stories a été demandée. Les users stories consiste à se mettre à la place du client et à déterminer les différentes fonctionnalités nécessaires. Elles permettent notamment de bien respecter le cahier des charges et permettent d'établir les futures tâches à accomplir. Ainsi, une première liste de fonctionnalités a été suggérées. Reprises une par une, certaines fonctionnalités ont été supprimées, faute de temps et étant considérées comme intraitables ou encore non-demandées. Une seconde liste de fonctionnalités a été établie avec une dizaine de fonctionnalité à mettre en place.

Un backlog est un document, mis en place dans le cadre de la méthode agile employée. Destiné à rassembler tous les besoins du client, il contient la liste des fonctionnalités demandées, nécessaires et intervenant dans la production du produit. Grâce le backlog, on obtient une liste de fonctionnalités qui devront être implémentées dans le projet. Ce backlog a été créé en équipe, en appui du chef de projet et de l'expert technique à l'issu de la création des user stories.

Le projet, nous l'évoquions est en deux parties communiquant entre elles et créant donc des échanges, une partie serveur et une partie cliente. Avant de commencer le développement de la partie cliente, ce en quoi consiste la seconde mission du stage, il a fallu réaliser un contrat d'interface. Ce contrat a été fait par le développeur ayant réalisé le serveur du chat. Ce-dernier est nécessaire à la bonne compréhension de la tâche demandée et permet de définir les notions nécessaires afin de comprendre le code. En effet, le document rassemble les événements socket.io* créés en amont dans la partie serveur et précise les données à utiliser pour

créer la partie client. Il établit ainsi le lien entre les deux parties : le serveur et le navigateur (partie client). Le fonctionnement de socket.io sera expliqué par la suite. Ci-dessous, le contrat d'interface :

call event	data	received event	data
listTalk	{user :[user_hash]}	receivedTalks	[{ createAt : "2019-05-27T16 :30 :05,877Z" , hash :[talk_hash], people : [{hash :[user1_hash]}, {hash :[user2_hash]}], updateAt : "2019-05-27T16 :30 :05,877Z" ,), [...]]
openOneToOneTalk	{receiver :[receiver_hash]}	takeRoom	{hash :[talk_hash]} user socket.id join a room init by talk_hash
getTalkMessages	{talk :[talk_hash]}	receivedMessages	[{talk :[Talk_hash], message :[text_message], sender :[sender_hash], createAt : "2019-05-27T16 :54 :35,877Z" }, {talk :[Talk_hash], message :[text_message], sender :[sender_hash], createAt : "2019-05-27T16 :54 :39,154Z" }]
chatMessage	{sender :[sender_hash], msg :[sender_hash], [Talk_hash]}	Received	{talk :[Talk_hash], sender :[sender_hash], message :[text_message]} event broadcast on room
USELESS FOR MANAO			
listUser		receivedUsers	users : [user1, user2]

Comme nous le disions, celui-ci énumère les évènements socket.io à appeler, d'un côté, et les évènements à recevoir de l'autre. Il indique notamment les données qui seront envoyées ou reçues. Ce document qui a été d'une grande aide, a permis de comprendre le fonctionnement du serveur chat ainsi que de comprendre les différents messages créés dans le code. Le fait de préciser les données qui doivent être reçues et/ou envoyées, permet de ne pas utiliser n'importe quelle variable ou donnée et de comprendre ce à quoi chacune d'entre elles correspond. Reprendre un code fait par un tiers est une chose assez délicate. Chaque personne ayant sa façon de voir les choses et de coder, il est important de mettre en place ce genre de document.

Un planning prévisionnel a été mis en place afin de respecter la contrainte de la durée du stage. La seconde mission consistait à concevoir la partie cliente du chat. Les étapes de réalisation du chat ont donc été définies avec le chef de projet et l'expert technique.

Le planning a été créé par le chef de projet avec un outil interne (Jira), il s'agit d'un tableau contenant des tickets. Chaque ticket correspond à une fonctionnalité à développer. Ces fonctionnalités ont été listées en amont et vérifiées par l'expert technique. Ainsi seules les fonctionnalités pouvant être développées dans le temps imparti ont été reportées dans le tableau. Une fois les documents de gestion de projet établis et validés, un chiffrage a été demandé. Le chiffrage consiste en une évaluation du temps que va prendre chacun des tâches listées dans le backlog. Le chiffrage est considéré comme étant une étape très délicate dans un projet. Savoir chiffrer une tâche à accomplir est un exercice complexe et d'autant plus difficile quand on a très peu d'expérience. Vous retrouverez le chiffrage en annexes. Pour ce projet, l'estimation n'est pas très loin de la réalité. Des fonctionnalités ont cependant mis plus de temps à être effectuées et une tâche n'avait pas été chiffrée et s'est donc greffée au chiffrage au cours du projet.

Un premier chiffrage avec uniquement les tâches à effectuer pour la création de l'interface utilisateur* avait été réalisé. La première mission, qui consistait à créer une interface utilisateur pour un chat, a été réalisée en trois jours. Cependant, l'interface a été modifiée lors de la réalisation de la seconde mission afin de respecter le backlog créé. Bien évidemment, les modifications effectuées par la suite font partie du second chiffrage. Le chiffrage élaboré en équipe, avec le chef de projet et l'expert technique avait pour principal objectif d'avoir une interface utilisateur et un chat fonctionnel avant la fin du stage, cet objectif étant respecté.

La mise en place d'un suivi de réalisation a permis de réaliser le projet plus sereinement. Cela a permis notamment d'avoir un produit finit à la fin du stage et de pouvoir le présenter aux différents membres de l'équipe technique.

Les détails sur les différentes étapes du processus de suivis vont vous être détaillées dans la partie suivante.

B – Etapes du processus de réalisation

En arrivant à l'entreprise, la première étape avant de commencer une quelconque mission, a été l'installation du poste de travail. En tant que stagiaire et nouvelle arrivante un livret d'accueil, un poste de travail accompagné de tutoriaux d'installation ont été confiés. Les mots de passe pour accéder au réseau interne et les codes d'accès aux différentes plateformes internes ont été remis.

La machine remise étant sous Windows, il a fallu désinstaller Windows et installer Ubuntu via une clé USB contenant l'ISO du système d'exploitation. Le système d'exploitation* Ubuntu a été une découverte mais la prise en main quant à elle a été facile. La version installée est Ubuntu LTS Bionic Beaver 18.04.

Dans un premier temps, le proxy* a été configuré, d'abord via les paramètres Ubuntu puis en lignes de commandes. Cntlm et vim* ont été installés, vim étant un éditeur de texte et cntlm est un utilitaire permettant de s'authentifier auprès d'un proxy. Le logiciel vim permet de manipuler, modifier, éditer des fichiers texte. Fortement inspiré de vi* (qui est lui aussi un éditeur de texte), il signifie Vi IMproved (« Vi à améliorer », en français). Cntlm a été configuré, via son fichier de configuration* afin qu'il puisse nous connecter au proxy.

Avant la configuration du proxy, reste à savoir ce qu'est un proxy ? Proxy signifie mandataire en français, il s'agit d'un intermédiaire. Lors d'une recherche sur internet sans proxy notre ordinateur nous renvoi la page demandée. Avec un proxy notre ordinateur se connecte au proxy et demande la page requise. On pourrait alors se demander à quoi sert le proxy. Le proxy joue divers rôles. On utilise un proxy d'abord pour une question de sécurité car le proxy gère les requêtes internet et donc peut nous permettre d'autoriser ou d'interdire l'accès aux ordinateurs d'un réseau. Il peut, de plus, masquer les informations concernant l'ordinateur utilisé comme son adresse IP*, son système d'exploitation et le navigateur utilisé. Enfin, le proxy a un côté pratique : il permet d'enregistrer les pages récurrentes (ce qui rappelle le principe du cache, dans ce cas on appelle ce genre de proxy, un proxy-cache.

Après avoir configuré correctement et validé la bonne configuration du proxy, est venue la configuration d'Evolution, un client de messagerie intégré à Ubuntu. Il inclut une messagerie, un mémo, un calendrier, un carnet de contacts et autres outils de communications utiles au sein d'une entreprise.

Une fois ces deux configurations effectuées, leur bon fonctionnement étant validé, Coop Net a été configuré. Application interne pour les employés d'Orange, Coop Net est un outil permettant d'effectuer des conférences en ligne. Cet outil n'a cependant pas été utilisé durant le stage.

Enfin, avant de terminer l'installation complète du poste, un IDE* a été installer, il s'agit de PPHP Storm. Ce choix s'est fait d'un point de vue pratique puisque cet environnement de développement a déjà été utilisé auparavant et donc déjà connu, un gain de temps donc dans la prise en main du logiciel.

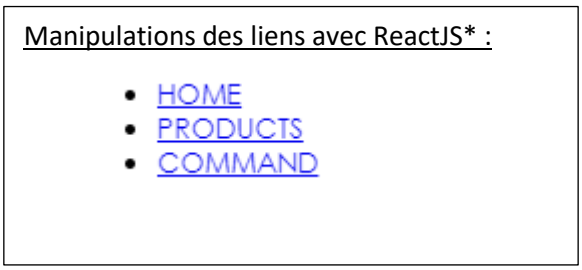
Lors de l'installation du poste de travail, Docker* a dû être installé et configuré en lignes de commandes en suivant un tutoriel créé en interne par l'équipe de l'open-space. Il fallait donc commencer par configurer les paramètres proxy et DNS* pour le logiciel Docker, en modifiant les fichiers de configurations du logiciel, toujours en suivant le tutoriel donné. Docker-compose a été installé par la suite au cours du projet, en se rendant compte qu'il n'avait pas été installé. Docker-compose est utilisé afin de gérer plusieurs containers Docker d'une même application. On peut désormais, en utilisant une seule commande, créer et démarrer tous les services d'une configuration d'un docker.

Afin de découvrir et de comprendre le fonctionnement du langage ReactJS, la documentation du site officiel a été lue et le tutoriel donné a été suivis. Le tutoriel consistait en la création d'un petit jeu, un Tic Tac Toe, autrement dit un morpion. Après avoir suivi le tutoriel, divers objets ont été créés et intégrés au sein d'une même page. Il s'agissait de s'imprégner de la syntaxe du langage ainsi que de comprendre la logique. En effet, la logique de ReactJS est basée sur des « components » (composants), des classes et la notion de « state » (état). Les objets instanciés dans une classe ont un state que l'on peut modifier grâce à la fonction *setState*().

Les composants ReactJS implémentent une méthode *render().* Cette méthode prend les données d'entrée et renvoie ce qu'il faut afficher. Les données peuvent être consultées par la méthode *render()* via *this.props*. En plus de prendre des données d'entrée, accessibles via *this.props,* un composant peut maintenir les données d'état interne, celles-ci sont accessibles via *this.state*. Lorsque les données d'état d'un composant changent, les données de *render()* sont mises à jour en réinvoquant la méthode *render()*. En utilisant les props et le state, nous pouvons mettre en place une petite application de compteur. Cet exemple utilise l'état pour suivre

l'évolution du compteur, il gère aussi l'évènement *onClick* du bouton. On met donc à jour le *state* du compteur grâce à deux buttons, ADD pour ajouter et RETIRE pour retirer.

Après une courte prise en main, les modules de Bootstrap ont été ajoutés au projet. Il existe une librairie Bootstrap dédiée aux applications ReactJS. En effet, les modules comme les tableaux, les buttons sont créées sous forme de components. L'implémentation est donc déjà toute faite. On peut donc utiliser Bootstrap aussi simplement qu'en HTML*. Afin de tester la bonne compréhension du fonctionnement de Bootstrap avec ReactJS, un tableau a été créé. La manipulation des liens a été revu. Tous ces petits éléments de prise en mains et de découverte ont été utiles par la suite car ces divers outils ont été par la suite réutilisés pour la réalisation de l'interface utilisateur, qui rappelons-le est la première mission de ce stage.



Une maquette de l'interface utilisateur a été élaborée à l'aide de l'outil Draw.io. Ce site permet de créer des schémas, des maquettes et autres graphiques. Facile d'utilisation, il laisse libre recourt à notre imagination. Une première maquette avec page d'accueil, page du chat et une page personnelle pour chaque membre du chat a été créée.

Après discussion avec l'expert technique, seule la page du chat a été sauvegardée, la page d'accueil n'étant des plus utiles. Pour ce qui est de la page personnelle seule était plus ambitieuse voire trop ambitieuse et donc infaisable dans le temps imparti. En effet, la page affichait des données de l'utilisateur. Ces données devaient donc être stockées dans une base de données ou sur le serveur du chat. Créer une base de données* aurait été une tâche plus complexe, sans oublier le Règlement Général sur la Protection des Données (la RGPD), qui contraint l'utilisation des données. Ce travail qui aurait donc été trop fastidieux a été mis de côté. Cependant, la question de pseudo d'utilisateur a été soulevée. Nous reviendrons sur cette problématique par la suite. La maquette a ensuite été développée en ReactJS sur une période de deux jours laissant ensuite place à la seconde mission.

A – Réalisation de l'interface utilisateur

Ainsi, la première semaine a été consacrée à la découverte et à la prise en main des différents langages. Cette semaine a ainsi permis de monter en compétences notamment sur le langage principalement utilisé le javascript avec la librairie ReactJS. Le poste de travail a été installé avec les logiciels et les configurations demandés. A l'issu de l'entretien ayant eu lieu avec le chef de projet et l'expert technique, s'en est suivis des recherches sur la mise en forme d'un chat, sur ce qu'était réellement un chat et comment en développer un. Cette phase a ainsi permis de rentrer dans le projet et comprendre ce qui était demandé.

La première mission résidait dans la réalisation d'une interface utilisateur en ReactJS. Avant de débuter le développement, une maquette a été réalisée. Cependant quelques changements ont eu lieu. Une réflexion a d'abord été menée sur les éléments qui allaient composés la page. Node.js* a été installé sur la machine afin de pouvoir développer l'application demandée. Node.js est une plateforme logicielle et évènementielle en Javascript.

Comme nous l'avons vu précédemment ReactJS fonctionne dans une logique de components. Chaque component, ou composant en français, correspond à un élément de la page. Ainsi, divers components ont été créé : une navbar (barre de navigation), une liste de contacts, un composant dédié au fonctionnement du chat – avec les messages, le formulaire d'envois notamment – , un formulaire de connexion, un composant pour la page d'accueil et enfin un composant dédié à la navigabilité* du site.

Ayant eu du mal à installer les paquets manquants afin de créer une application ReactJS, un développeur à utiliser son docker afin d'installer tous les paquets manquants nécessaires au développement de l'application ReactJS.

Docker est une plateforme permettant de partager, déployer et faire tourner des containers d'applications. Cette technologie est un véritable gain de temps en plus d'être facile d'utilisation. Docker utilise le système de conteneurisation. Un docker est composé d'au moins un container et peut en contenir plusieurs. Un container est une unité d'un logiciel qui est contient du code ainsi que toutes les dépendances nécessaires à son déploiement. Son point fort est que les containers peuvent être transférer et utiliser d'un environnement à l'autre. L'application fonctionne rapidement et de manière fiable d'un environnement informatique à l'autre. Une image de conteneur Docker est un ensemble de logiciels léger, autonome et exécutable qui comprend tout ce qui est nécessaire pour exécuter une application : code, exécution, outils système, bibliothèques système et paramètres. Ces images docker deviennent des conteneurs à la seconde où on l'exécute. Que l'on utilise Docker sous Linux ou Windows, les logiciels conteneurisés s'exécuteront toujours de la même façon, quelle que soit l'infrastructure, ce qui permet aussi d'utiliser cette technologie dans les data center entre autres. Les conteneurs isolent les logiciels de son environnement et s'assurent qu'il fonctionne correctement malgré les potentielles différences qu'il peut y avoir. Cette technologie récente est de plus en plus utilisée et est d'une réelle praticité.

Son container docker comportait tous les modules nécessaires à la création d'une application ReactJS. Le problème venait du logiciel npm*. Npm est un gestionnaire de paquets pour Node.js. Ce logiciel gère les dépendances d'une application. La définition qu'on se fait d'une dépendance, en général, est qu'une dépendance représente une entité ou un élément dont sa stabilité, sa conduite, est dictée par une autre entité ou une autre source. Ce que l'on appelle une dépendance, dans le contexte informatique, ne correspond pas tout à fait à cette définition. Il s'agirait plutôt d'un paquet dont l'application en production nécessite. Si la dépendance définie dans le fichier *package.json* de l'application est manquant l'application ne pourra fonctionner correctement.

Les dépendances, *dependencies* en anglais, sont définies dans un fichier JSON* nommé *package.json*. JSON est l'acronyme pour Javascript Object Notation. Les fichiers JSON sont souvent utilisés pour transporter des données d'un server vers une page web, car ce format est léger. De plus, ces fichiers sont simples à comprendre, ils s'auto-suffisent. Ce fichier regroupe ainsi toutes les dépendances installées sur l'application. Vous pouvez retrouver le fichier *package.json* dans les annexes.

Si npm ne tourne pas correctement sur la machine, les paquets nécessaires au développement de l'application ne peuvent pas être installés. Pour contourner ce problème, et afin de pouvoir installer des paquets manquants, Yarn* a été utilisé. Installé dans le container utilisé pour le projet, Yarn est aussi un gestionnaire de

paquets. Bootstrap pour ReactJS a notamment été installer avec le gestionnaire Yarn via la commande *« yarn install [nom du paquet]* ». Pour ce faire, Yarn installé sur le container docker et non sur la machine elle-même, il a fallu rentrer dans le container docker. Pour cela il faut vérifier que l'état du docker, à savoir s'il est actif ou non. La commande à effectuer pour voir les dockers actifs est celle-ci :

```
docker ps
```

On y trouve ainsi l'ID de chacun des containers dockers actifs, leur image, leur date de création, leur statut, le port emprunté et leur nom.

Pour rentrer dans un container docker actif, la commande à effectuer est la suivante :

```
docker-compose exec -it  [container id]  bash
```

En tapant cette commande, on rentre dans le docker et on accède au bash*, on peut ainsi effectuer les commandes souhaiter. On peut ainsi installer les paquets que l'on veut avec le gestionnaire de paquets Yarn, installé sur le container docker.

La mise en page de l'interface a été faite en CSS* (Cascade Style Sheet) ainsi qu'avec un framework* CSS, Bootstrap pour ReactJS.

En ce qui concerne le logo et le nom du chat, le choix s'est fait en milieu de développement.

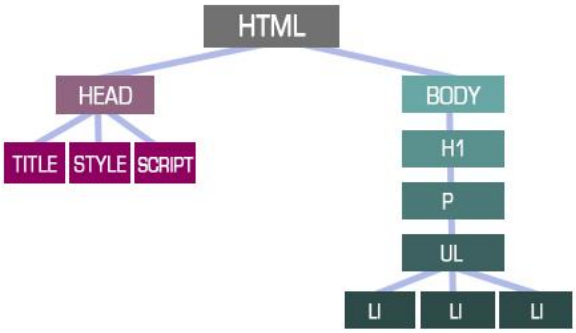
Le chef de projet a demandé de trouver un nom au projet ainsi qu'un logo pour le chat. Après quelques réflexions individuelles puis en équipes, avec les autres développeurs de l'open-space, le nom de Meow est apparu. Meow est un jeu de mot avec les deux définitions du même mot « chat », l'animal et l'application.

Commençons par la création des différents composants, voici la liste de ceux qui ont été créés :

- *tchatApp* : dédié à la partie chat
- *login* : formulaire de connexion
- *listeContact* : gère l'affichage des utilisateurs et des conversations
- *menu* : div de la page d'accueil
- *RoutingApp* : contient une *navbar* et gère la navigabilité du site avec React-Router

- Login Component

Le projet nécessitait un formulaire de connexion. Celui-ci permet à tout utilisateur de rentrer sur le chat avec un pseudo. Le pseudo entré dans le formulaire est stocké dans le stockage de session grâce à la fonction *sessionStorage()*. SessionStorage est un attribut du DOM*.



Le DOM correspond au Document Object Model, il s'agit d'une représentation structurée et orientée objet. Lorsqu'on évoque la notion de DOM, on peut parler d'arbre dans lequel chaque balise implémentée représente un nœud. Par exemple, l'utilisation de la méthode *getElementById()* fait référence à un élément précis de l'arbre, dont l'ID correspond à celui recherché.

Nous l'avions évoqué, le formulaire de connexion demandait de stocker une donnée de l'utilisateur. Or, après réflexion et consultation de l'expert technique, créer une base de données juste pour cela aurait été absurde. Il a donc fallu trouver une solution alternative. Cette solution est l'utilisation de la méthode *sessionStorage()*.

Voici le code de la fonction utilisée pour valider l'inscription de l'utilisateur au chat :

```
handleValidation(){
        const userInput = document.querySelector('[name="userInput"]');
        console.log(userInput);
        sessionStorage.setItem('user', userInput.value);
}
```

Le nom rentré dans le formulaire par l'utilisateur est récupéré avec la fonction *querySelector()*, est ensuite stocké dans une variable. La valeur de cette variable est ensuite stockée dans l'espace de stockage de la session grâce à la fonction *sessionStorage.setItem()*. « *user* » est une clé, c'est via cette clé que l'on peut récupérer la valeur stockée avec la fonction *sessionStorage.getItem()*.

- ListeContact Component

Un autre composant ReactJS a été créé pour contenir les noms des utilisateurs et les conversations. Celui-ci a été placé sur la gauche de la page, et la liste est à la verticale. Les données sont récupérées avec socket.io. Pour ce qui est de l'affichage de chaque utilisateur ou conversation, la méthode *apend()* a été utilisée. Nous reviendrons sur sa réalisation lors de l'explication de la conception de la partie cliente du chat.

- TchatApp Component

Pour ce qui est du composant *tchatApp*, il contient le cœur du chat. Ce composant contient un formulaire qui permet de taper le message que l'on veut envoyer. Ci-dessous, le formulaire d'envois de messages :

```
<form className="formChat*" onSubmit={this.handleSubmit}>
        <input id='messageInput' type="text" name="textMessageName" className="form-control"
        placeholder="Your Message" required/>
        <button className="btn btn-sm" type="submit">Send</button>
</form>
```

On crée une nouvelle variable qui prend la valeur de l'attribut dont le *name* correspond à « *textMessageName* », qui n'est autre que le message tapé dans l'input du formulaire.

```
this.textMessageName = 'textMessageName';
```

Le texte entré dans le formulaire est récupéré avec la méthode querySelector(). Cette méthode renvoie le premier élément du document qui correspond au sélecteur entré (ou *null* si aucun élément n'est trouvé).

```
let chatInput = document.querySelector('[name="' + this.textMessageName + '"]');
```

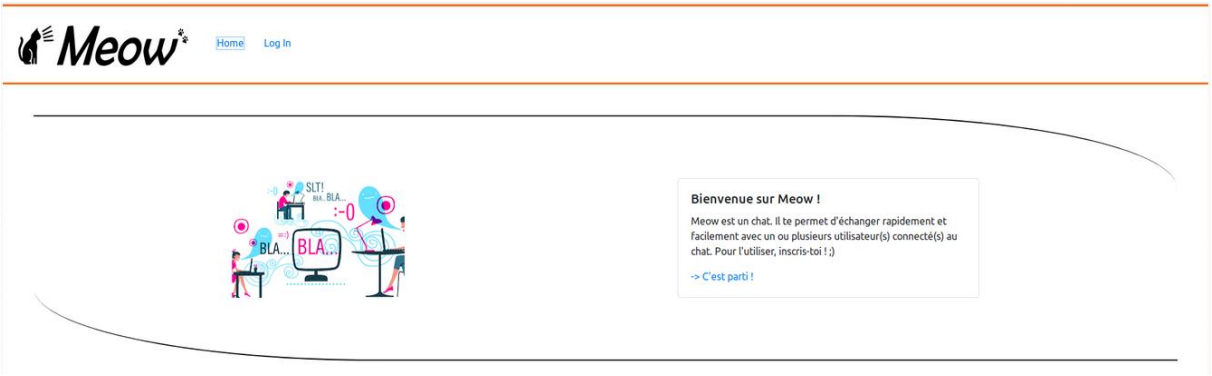
Un state msg a été créé et est mis à jour à chaque message envoyé. La mise à jour est effectué grâce à la fonction *setState()* de ReactJS. Voici un extrait de code illustrant l'utilisation de la méthode *setState()* :

```
this.state = {
    msg: ''
};
handleMessage() {
        let chatInput = document.querySelector('[name="' + this.textMessageName + '"]');
        this.setState({msg: chatInput.value});
}
```

Retrouvez le code complet en annexes. C'est dans ce composant que sont gérés les évènements socket.io. Nous reviendrons sur la création de ce composant *tchatApp*.

- Menu Component

Ce composant n'est pas des plus utiles. Il s'agit simplement d'une balise div, contenant un élément Bootstrap et une image. Ce composant sert de page d'accueil au chat et permet à l'utilisateur d'accéder directement au formulaire de connexion après l'explication du chat via un lien



- RoutingApp Component

Ce component gère la navigabilité de l'interface. ReactJS est généralement fait pour créer du one-page, c'est-à-dire, créer une application qui tient sur une unique page. Cependant, React-Router qui est la librairie standard de routage pour ReactJS permet de créer des routes vers de composants entre autres. Ce module a notamment permis, au sein de l'application, de créer deux boutons dans la barre de navigation qui renvoient, chacun d'entre eux, sur un composant différent. Le bouton « Home » renvoie sur le composant *Menu* et le bouton « Log In » renvoie sur le composant *Login*.

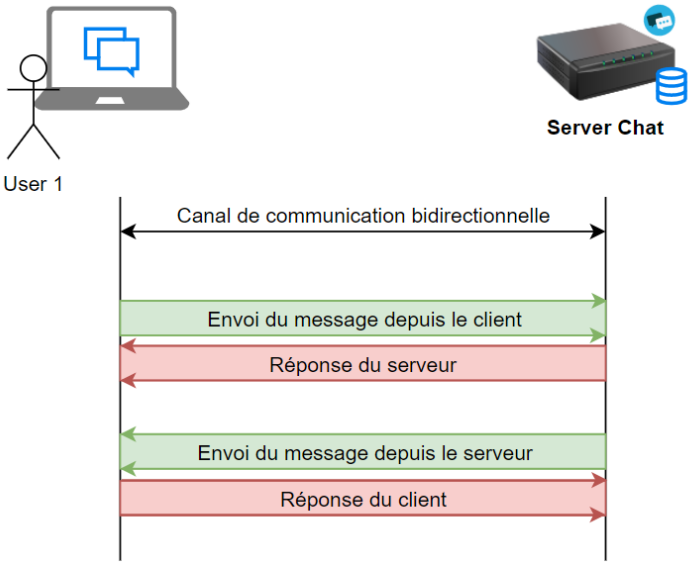
B – Réalisation de la partie client du chat

La première étape de la mission étant donc réalisée et validée, la seconde mission a été décrite plus précisément. Il s'agit de concevoir la partie client du chat en utilisant socket.io. La partie client (navigateur) se fait en javascript.

- Fonctionnement de Socket.io

Afin de réaliser la partie cliente du chat il a fallu utiliser Socket.io. Socket.io est une librairie qui permet de créer des applications en temps réel ainsi que des applications de communication nécessitant un échange entre un navigateur et un serveur (un chat en est un parfait exemple). Il s'agit d'un serveur Node.js et d'une librairie javascript côté client pour le navigateur. Socket.io est un service fiable. Les connexions sont établies malgré la présence de proxy, de pare-feu* ou même d'antivirus. Le mot « socket » vient de l'anglais et peut être traduit par « connecteur réseau » ou encore « interface de connexion ». Socket.io rappelle le système des websockets. Websocket* est une technologie évoluée qui permet d'ouvrir un canal de communication bidirectionnelle entre un navigateur (côté client) et un serveur. WebSocket permet d'envoyer des messages à un serveur* et recevoir ses réponses de manière événementielle sans avoir à aller consulter le serveur pour obtenir une réponse. Socket.io n'est autre qu'une API* WebSocket en Node.js.

Le schéma ci-dessous résume le fonctionnement de Socket.io : une communication directe et bidirectionnelle entre navigateur et serveur, en temps réel.



De façon à monter en compétence et appréhender la logique de socket.io, le tutorial du site a été parcouru. Il s'agissait de concevoir un simple chat. Cette étape a permis de comprendre ce qu'était un évènement.

Avant de commencer à développer réellement la partie client du chat, le développeur en charge de la conception du chat a souhaité voir une fonction qui permette de lever un événement avant de donner accès au répertoire git* ainsi qu'à la base de données.

Une fois ce test passé, les accès au répertoire git du projet chat de l'entreprise ont été fournis. En tant que reporter sur le répertoire du projet, aucun commit n'était permis. Cette étape permettait simplement d'avoir accès à une partie du code du serveur chat : les différents évènements socket.io créés. Avec le contrat d'interface, la liste des divers évènements émis ou reçus étaient déjà connus. L'accès à une partie du code a permis une meilleure compréhension de socket.io.

- La base de données

La base de données quant à elle a été conçue avec MongoDB*. MongoDB permet de créer des bases de données. Ce second logiciel exploité afin de remplir la mission est une base de données orientée documents. Les données de la base de données sont donc stockées en toute flexibilité dans des documents JSON. Les champs peuvent donc être modifier et la structure peut quant à elle évoluer. MongoDB nous permet d'être plus libre et de travailler les données plus aisément.

La base de données du chat a été développée par Luiguy, le développeur en charge du serveur chat. Cette-dernière servait simplement à vérifier le bon fonctionnement du chat.

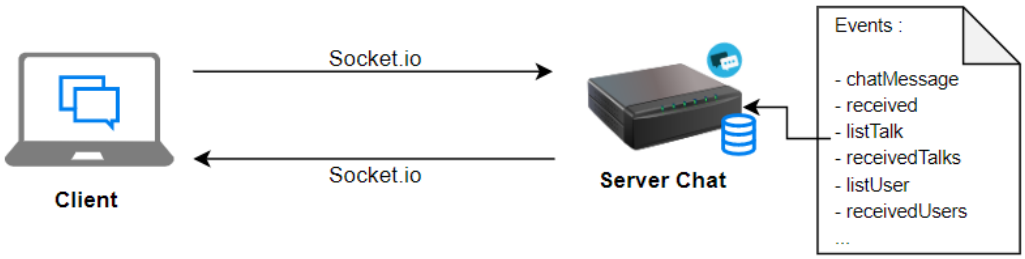
Quelques commandes ont été effectuées sur la base de données. Un nouvel utilisateur a été créé, en lui accordant un mot de passe et des droits suffisant. La recherche des commandes de bases a été nécessaires comme par exemple accéder à la liste des collections.

En effet, les bases de données MongoDB utilisent des collections (appelées « tables » sous MySql*). Voici quelques commandes utilisées dans MongoDB en lignes de commandes :

```
mongodb
use [nom de la bdd]   // la bdd entrée doit existée
show collections //liste toutes les collections existantes

db.[nom de la collection].find() //lister le contenu de la collection
db.[nom de la collection].remove({}) //supprimer tout le contenu de la collection
db.[nom de la collection].remove({[champ] : « … »}) //supprimer un contenu particulier de la collection
```

- Socket.io et la notion d'évènement

En effet, ce que l'on peut appeler un évènement socker.io, est en réalité un message qu'émet ou que reçoit le serveur et/ou le navigateur. Socket.io crée des échanges en temps réel entre le serveur socket.io et le navigateur. Les informations vont dans les deux sens : serveur vers navigateur mais aussi navigateur vers serveur. C'est ce qui permet de recevoir et émettre des messages en temps réel. Depuis le début de ce document, nous employons le terme d'évènements socket.io sans réellement savoir de quoi il s'agit ? On considère un évènement socket.io comme étant un message que l'on émet ou que l'on reçoit. Le message correspond en réalité à une action que l'on précise en créant cet évènement. Lorsqu'on reçoit ou émet le message, l'action choisie lors de la création est effectuée. C'est ce que l'on considère comme étant un évènement. Lorsque le serveur socket.io est lancé et le navigateur connecté au serveur, chacune des deux parties va être à l'écoute de tout évènement, de tout message. Le schéma ci-dessous résume le fonctionnement de Socket.io :



Pour lever un évènement, c'est-à-dire faire appel à ce-dernier, on utilise la fonction *.on()*. En utilisant *.on()*, on fait appel à l'évènement créé. Pour lancer un évènement, on utilise la fonction *.emit()*. Cette fonction permet de faire passer des données. Par exemple, pour lister les utilisateurs et les conversations, la fonction *.on()* a été utilisé, tandis que la fonction *.emit()* a été utilisée afin d'envoyer un message.

Avant de faire quoi que ce soit avec le serveur du chat, il faut avant tout de chose, se connecter au serveur. Pour cela, on doit se connecter au bon port du serveur. Le port* sur lequel est le serveur socket.io du chat est le port 5000, tandis que l'application est sur le port 3000. Ce schéma résume l'étape de connexion au serveur et montre le message que répond le serveur lorsque le navigateur est bien connecté.

Ci-dessous, le code de l'initialisation du chat, autrement la connexion au serveur socket.io :

```
// Initialisation du chat*
function getID() {
    let id = (new Date().getTime() + Math.floor((Math.random()*10000)+1)).toString(16);
    return id;
}
const username = sessionStorage.getItem('user');
let clientID = getID();
console.log('id du client : ', clientID);
const token = 'UmgoIhfJWomN4jLW5wPXoMkIY8l5PPY6Tq8bPjzx6mg';
const socket = io('http://192.168.105.127:5000', {
    query: {
        'clientid': username,
        'Authorization': token
    }
});
```

Le serveur ne laisse pas entrer n'importe qui sur le chat. Il vérifie alors que le *token** soit le bon, que le pseudo de l'utilisateur soit bien défini. Le *token* a été définit en brut dans le code, on vérifie simplement que le *token* entrant soit le même. En se connectant, le *clientid* et le *token* sont envoyés, le serveur fait les vérifications nécessaires et envoie une réponse. Lorsque la connexion est établie, un message identique à celui-ci est envoyé :

```
node        | Authentication of iT7JnY_FwiZIy13oAAAy
node        | token UmgoIhfJWomN4jLW5wPXoMkIY8l5PPY6Tq8bPjzx6mg is Valid for Nathalie
node        | socket iT7JnY_FwiZIy13oAAAy connected
```

Ci-dessous, se trouvent les deux fonctions permettant d'envoyer un message. La fonction *handleMessage()* permet de récupérer la valeur tapée dans l'input du formulaire d'envoi de message et change le state de la variable *msg* via la fonction *setState().* La fonction *handlSubmit()* quant à elle fait appel à l'évènement « chatMessage » du serveur socket.io. L'utilisation de la méthode *.emit()* permet d'envoyer les données que l'on souhaite. On envoie ainsi le pseudo de l'expéditeur, correspondant à la variable *username*, le contenu du message. La *room* permet d'envoyer un message à un utilisateur précis, ce qui n'est pas possible en l'état. La *room* n'a été déclarée. Normalement, lorsqu'on arrive sur le chat, on choisit la personne à qui on souhaite envoyer un message, ce qui crée une *room* (un hash qui correspond au nom de la *room*). Cependant, ce n'est pas le cas actuellement. Il s'agit d'une amélioration à apporter. A ce niveau, le message sera donc envoyé à tous les utilisateurs présents dans la base de données du chat.

```
handleMessage() {
        // Récupération du texte entré dans le formulaire d'envois de message
        let chatInput = document.querySelector('[name="' + this.textMessageName + '"]');
        this.setState({msg: chatInput.value});
}

handleSubmit(event) {
        event.preventDefault();
        let chatInput = document.querySelector('[name="' + this.textMessageName + '"]');
        //envoi d'un message
        console.log(chatInput.value);
        socket.emit("chatMessage", {
            sender: username,
            message: chatInput.value,
            room:localStorage.getItem('room')
        });
}
```

Lorsqu'on envoie un message, voici la réponse du serveur* :

```
node        | { sender: 'Nicolas', msg: 'Très bien et toi ?!' }
node        | chat message
node        | data.talk
node        | undefined
node        | room undefined |sender: Nicolas | message: Très bien et toi ?!
```

Le serveur* indique que le message est bien envoyé, il indique qui en est l'expéditeur et le contenu du message.

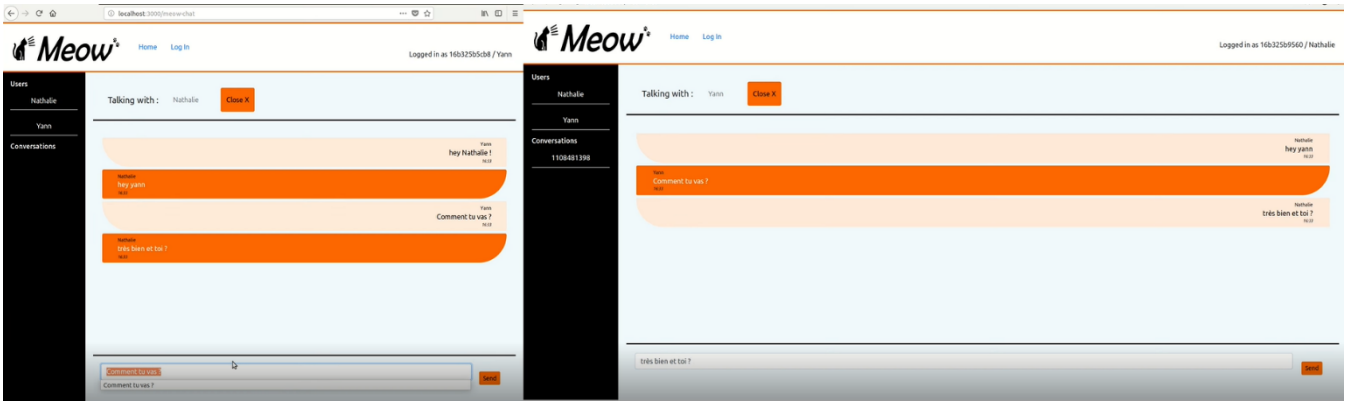
Envoyer un message est un début mais le recevoir est une autre histoire. Recevoir un message fait appel à un autre évènement et une autre fonction se charge de récupérer les données envoyées au serveur et de les afficher. Pour récupérer les messages émis, on utilise la fonction *.on()*. Voici cette fonction :

```
// réception des messages
socket.on("received", data => {
        console.log(data);
        let divMsg = document.createElement('div');
        let divMsgSent = document.createElement('div');
        let divSen = document.createElement('div');
        let divHeure = document.createElement('div');
        const time = new Date().getHours()+":"+ new Date().getMinutes();
        divMsg.className = "messageBox";
        divMsgSent.className = "messageBoxSent";
        divMsg.id = 'msgBox';
        divSen.className = "sender";
        divHeure.className = "details";
        let messages = document.getElementById('messages');

        //différenciation messages reçus/émis
        if(data.sender!==null){
                if(data.sender===username){
                        messages.appendChild(divMsgSent).appendChild(divSen).append(data.sender);
                        messages.appendChild(divMsgSent).append(data.message);
                        messages.appendChild(divMsgSent).appendChild(divHeure).append(time);
                }else{
                        messages.appendChild(divMsg).appendChild(divSen).append(data.sender);
                        messages.appendChild(divMsg).append(data.message);
                        messages.appendChild(divMsg).appendChild(divHeure).append(time);
                }
         }
});
```

Pour permettre d'afficher les messages, on utilise diverses méthodes. La méthode *createElement()* permet de créer un élément HTML spécifié par le type passé en paramètre. Grâce à cette méthode, plusieurs balises *div* sont créées, chacune contiendra les éléments qui lui seront ajoutés par la suite. Les méthodes *appendChild()* et *append()* permettent, en termes généraux, d'ajouter des éléments. La méthode *append()* permet d'insérer le contenu, spécifié par le paramètre, à la fin de chaque élément dans l'ensemble des éléments associés. La méthode *appendChild()* ajoute un nœud à la fin de la liste des enfants d'un nœud parent spécifié.

```
messages.appendChild(divMsg).appendChild(divSen).append(data.sender);
```

Prenons l'exemple du dessus, on ajoute à la balise *div* dont l'ID est « messages » – récupérée par le biais de la méthode *getElementById()* – la div « divMsg » à laquelle on ajoute la div « divSen ». On vient ensuite ajouter à la div « divSen » la donnée « data.sender ». On obtient alors une liste des messages envoyés comme on peut le voir sur l'image ci-dessous qui est une capture de deux interfaces du chat en train de communiquer chacune affichant la liste de messages envoyés.

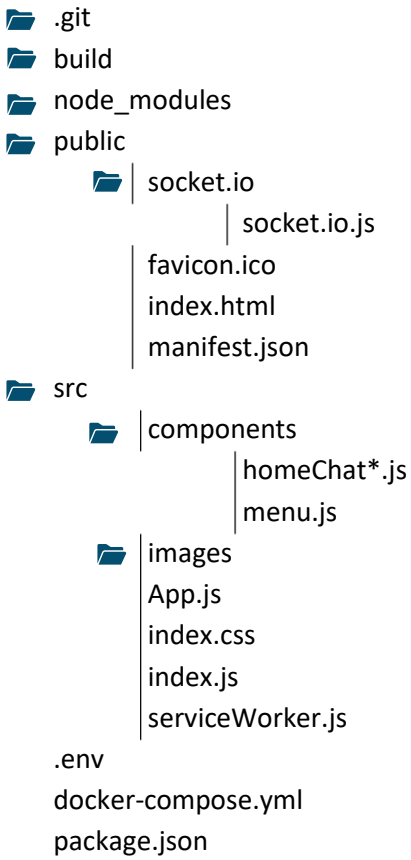


Comme vous pouvez le voir sur les captures des interfaces du chat présentées plus haut, on affiche la liste des utilisateurs ainsi que les conversations définies par un hash. Vous pouvez retrouver d'autres captures en annexes. Pour lister les utilisateurs et les conversations, deux fonctions semblables ont été instanciées. Une première a pour but d'afficher les utilisateurs du chat. Les utilisateurs qui s'affichent sont stockés dans la base de données du chat. Ces fonctions utilisent les mêmes méthode *append()* et *appendChild*(), utilisées pour afficher les messages.

```
// Liste des users
socket.emit("listUser");
socket.on('receivedUsers', data => {
     let listUsers = document.getElementById('listUsers');
     data.forEach(listElements);
     function listElements(element, index, data) {
            let p = document.createElement('p');
            p.className = 'contact';
            p.id = 'contact';
            let receiver = element.hash;
            listUsers.appendChild(p).append(receiver);
            let talkingWith = document.getElementById('talkingWith');
            p.addEventListener('click', clickEvent);
                  function clickEvent(){
                        socket.emit("openOneToOneTalk", {receiver: element.hash});
                        talkingWith.append(element.hash);
                        p.removeEventListener('click', clickEvent);
                  }
            }
});
```

```
// Liste des conversations
socket.emit("listTalk", {user:username});
socket.on('receivedTalks', data => {
       let listTalks = document.getElementById('listTalks');
       data.forEach(listElements);
       function listElements(element, index, data) {
           let p = document.createElement('p');
           p.className = 'contact';
           p.id = 'contact';
           console.log('listUser');
           listTalks.appendChild(p).append(element.hash);
       }
});
```

Pour finir sur la partie réalisation des missions, voici comment est hiérarchisé le projet :



Le répertoire *node_modules* rassemble tous les modules, tous les paquets* installés pour le développement de l'application. Les répertoires *nodes_modules*, public et src ont été créés lors que la création de l'application ReactJS. Le dossier socket.io a été initialisé à la suite de l'installation de socket.io.

Le fichier *docker-compose.yml* est un fichier docker. Il s'agit d'un fichier YAML définissant les services, les réseaux et les volumes du container docker. Par exemple, une définition de service contient la configuration qui est appliquée à chaque container actif pour ce service.

Le fichier *serviceWorker.js* est créé lorsqu'on créer l'application ReactJS. Le fichier serviceWorker.js est un script que votre navigateur exécute en arrière-plan. Il lance des fonctionnalités qui n'ont pas besoin d'une page Web ou d'une interaction avec l'utilisateur et peut inclure diverses fonctionnalités telles que les notifications push ou la synchronisation en arrière-plan.

A – Adéquation du travail effectué

Le planning a été fait dans l'optique d'avoir une interface de chat fonctionnelle avec les fonctionnalités indispensables, avant la fin de la période de stage s. Bien entendu, l'interface aboutie sur une version demandant encore des ajustements ergonomiques notamment et des fonctionnalités à ajouter. La principale fonctionnalité manquante est l'envoi d'un message à une personne précise. En effet, pour l'instant lors de l'envoi du message, celui-ci est envoie à tous utilisateurs de chat et non à une personne précise. Cette fonctionnalité est définie dans la partie serveur du chat. Cependant, la partie cliente ne le permet pas. Il s'agit là de la principale amélioration à faire pour ce chat.

Pour ce qui est de la validation du travail effectué, il y a eu une première démonstration face à l'expert technique et le chef de projet. Cette démonstration a eu lieu le mardi de la dernière semaine de stage, cette date de démonstration a été définie en fonction du planning et du chiffrage et définie avec le chef de projet.

B – Compétences mises en œuvre

Durant ces cinq semaines de stage, diverses compétences ont été mises en œuvre.

La première activité retenue correspond à l'activité du référentiel correspondant à celle-ci : « A 1.4.1 Participation à un projet ». Comme nous l'avons précisé tout au long de ce rapport, la mission confiée rentrait dans le cadre d'un projet plus imposant de la société OBS SA. Il s'agit d'un véritable projet et il a fallu mettre en place divers moyens et outils afin de clôturer la mission. Un prototype sur la base d'un cahier des charges a été réalisé. Les compétences « C1.4.1.1 Etablir son planning personnel en fonction des exigences et du déroulement du projet » et « C1.4.1.2 Rendre compte de son activité » ont été mise en œuvre. Un planning prévisionnel et un planning final ont été mis en place. Un guide en tant que développeur était à disposition, il rassemble les normes, les « bonnes manières », les standards adoptés par les développeurs de l'entreprise. Le travail en équipe et au sein d'un projet a ainsi pu être vécu. Les moyens et outils d'organisation d'un projet nécessaire à sa mise en place et sa réalisation ont ainsi pu être réalisé. Cette compétence est importante dans le processus d'apprentissage pour être un bon développeur.

L'activité « A 4.1.1 Proposition d'une solution applicative » a permis d'identifier les compétences « C4.1.1.1 Identifier les composants logiciels nécessaires à la conception de la solution » et « C4.1.1.2 Estimer […] le délai de la mise en œuvre de la solution ». A partir du cahier des charges, l'ensemble des user stories a été définit, le backlog du produit à développer a été constitué. Chacune des user stories ont été estimées et loties afin de constituer un planning prévisionnel.

On peut ajouter l'activité « A 4.1.2 Conception de l'interface utilisateur d'une solution applicative ». Celle-ci renvoie à trois autres compétences : « C4.1.2.1 Définir les spécifications de l'interface utilisateur de la solution applicative », « C4.1.2.2 Maquetter un élément de la solution et applicative » et « C4.1.2.3 Concevoir et valider la maquette en collaboration avec des utilisateurs ». En effet, toujours à partir du cahier des charges, les maquettes graphiques ont été réalisées. Ces-dernières ont servi à la définition et à l'illustration des user stories.

La compétence « C4.1.4.1 Recenser et caractériser les composants existants ou à développer utiles à la réalisation de la solution applicative dans le respect des budgets et planning prévisionnels », en référence à l'activité « A4.1.4 Définition des caractéristiques d'une solution applicative », a été mise en œuvre. La partie back end* étant déjà existante, les user stories et maquettes validées par l'équipe, il a fallu concevoir techniquement l'application front end pour répondre au besoin fonctionnel.

L'activité « A4.1.9 Rédaction d'une documentation technique » fait référence à la compétence « C4.1.9.1 Produire […] la documentation technique d'une solution applicative et de ses composants logiciels ». Une documentation technique du prototype fourni a été constituer. Ce document permettra à un autre développeur de reprendre la suite du projet. Il s'agit d'un mode d'emploi : il rassemble par exemple les paramètres d'installation de l'application.

La dernière activité réalisée est « A1.3.3 Accompagnement de la mise en place d'un nouveau service ». Celle-ci met en œuvre une compétence acquise durant le stage, « C1.3.3.2 Informer et former les utilisateurs ». Le prototype a été présenté à l'ensemble des équipes techniques, lors du dernier jour du stage. Une trentaine de personnes étaient présentes : développeurs, managers, chef de projet, architectes et experts techniques. Cette présentation avait pour but de présenté l'interface utilisateur, d'en faire une démonstration et de clôturé le stage.

Malgré ces quelques compétences mises en œuvre, d'autres restent à améliorer et beaucoup à développer et acquérir.

CONCLUSION

Ce stage s'est déroulé dans d'excellentes conditions. L'intégration aux équipes s'est faite rapidement et facilement. Les repas du midi ont été pris dans la salle de pause dédiées aux employés, développeurs comme managers, tous les membres des bureaux ont accès à cette salle. Les discussions étaient variées et enrichissantes aussi bien professionnellement que personnellement. Qu'il s'agisse de développeur, de chef de projet ou de manager, les personnes ont été très agréables et ont su se rendre disponibles lorsque cela était nécessaire et possible.

Les missions confiées par l'entreprise ont été des plus enrichissantes. Nécessitant l'apprentissages de nouvelles technos, ces langages de programmation, autrefois inconnus, sont devenus presque courant. En quelques semaines, ce stage a permis de monter en compétences sur de nombreux plans. Les matériels prêtés ont notamment permis de progresser professionnellement car il a fallu les installer. Le côté Dev'Ops s'est notamment fait ressentir et a été quelques fois entendu. Certaines sous-missions comme l'installation et la configuration du poste de travail et de l'environnement de développement ont nécessitées des connaissances en réseau.

En parallèle des missions données, des formations ont été proposées par OBS SA. Dans le cadre de la nouvelle réglementation et en tant que stagiaire chez OBS, une formation en ligne obligatoire a dû être effectuée durant le stage. Cette formation, intitulée Sensibilisation à la protection des données personnelles, avait pour but d'expliquer et de sensibiliser les professionnels de l'entreprise. Elle explique donc ce que sont les données personnelles, les données sensibles et comment celles-ci sont utilisées et gérées sein de l'entreprise. Le groupe OAB peut jouer différents rôles quant à l'utilisation des données personnelles : sous-traitant, responsable ou bien utilisateur, tout dépend du projet et des besoins des clients. Une seconde formation a été proposée, celle-ci n'étant pas obligatoire. Elle concernait les politiques anti-corruption mises en place au sein d'Orange. Ces formations ont été suivies en supplément de la mission à effectuer et ont permis d'enrichir la culture informatique.

D'un point de vue technique, ce stage a été très enrichissant en développement. Un stage en réseau serait donc intéressant afin de découvrir les deux versants. Effectuer un stage dans les deux branches (développement et réseau) serait l'opportunité de mettre en avant le principe Dev'Ops.

- API : interface de programmation d'application. Il s'agit d'un ensemble normalisé de classes, de méthodes ou de fonctions qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels.
- Adresse IP : IP signifie Internet Protocol. Il s'agit du numéro qui identifie chaque ordinateur connecté à Internet, ou plus généralement, l'interface réseau de tout matériel informatique (routeur, imprimante, serveur) connecté à un réseau informatique utilisant l'Internet Protocol.

- Back end : terme désignant un étage de sortie d'un logiciel devant produire un résultat. On l'oppose au front-end qui lui correspond à la partie visible de l'iceberg. Dans un modèle informatique client/serveur, comme dans le cas d'un chat, le « front-end » est généralement un client et le « back-end », un serveur.
- Base de données : permet de stocker et de retrouver l'intégralité de données brutes ou d'informations en rapport avec un thème ou une activité ; celles-ci peuvent être de natures différentes et plus ou moins reliées entre elles.
- Bash : un des nombreux shells existants dans le monde Unix. Un shell est un interpréteur de commandes, et en même temps un langage de programmation.

- Chat : espace de discussion en direct sur Internet. Contrairement à la messagerie instantanée, le chat vous permet de discuter avec tout le monde.
- Chef de projet : spécialiste informatique chargé de piloter une équipe dans l'état d'avancement d'un projet en traduisant en solutions informatiques les demandes des clients ou des services internes.
- Client : logiciel qui envoie des demandes à un serveur. Il peut s'agir d'un logiciel manipulé par une personne, ou d'un bot.
- CSS : signifie *Cascading Style Sheets*, feuilles de style en cascade en français. Il s'agit d'un langage informatique utilisé pour mettre en forme les fichiers HTML entre autres. Ainsi, les feuilles de style, aussi appelé les fichiers CSS, comprennent du code qui permet de gérer le design d'une page en HTML.

- Docker : outil qui peut empaqueter une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel environnement.
- DNS : Domain Name System (système de noms de domaine, en français) est le service informatique distribué utilisé pour traduire les noms de domaine Internet en adresse IP ou autres enregistrements.
- DOM : Document Object Model, interface de programmation d'application (API) pour des documents HTML valides et XML bien-formés. Il définit la structure logique d'un document et la manière d'y accéder et de le manipuler.

- Expert technique : spécialiste d'un domaine particulier, dans ce cadre dans le secteur informatique. L' Expert technique assure principalement le rôle de conseil, d'assistance et de formation. Il peut intervient directement sur la totalité ou sur une partie d'un projet.

- Fichiers de configuration : un fichier de configuration contient des informations de configuration utilisées par un programme informatique pour adapter ou personnaliser son fonctionnement.
- Framework : fournit un ensemble de fonctions facilitant la création de tout ou d'une partie d'un système logiciel. Il donne accès aussi à un guide architectural en partitionnant le domaine visé en modules.

- Git : logiciel de gestion de versions décentralisé. C'est un logiciel libre créé par Linus Torvalds, auteur du noyau Linux. Il s'agit du logiciel de gestion de versions le plus populaire et le plus utilisé.

- HTML : signifie *HyperText Markup Language*. Il s'agit d'un langage informatique utilisé pour créer des pages web. Ce langage permet de réaliser de l'hypertexte à base d'une structure de balisage comme l'indique son nom, langage de balisage d'hypertexte.

- IDE : Environnement de développement, IDE en anglais, pour *Integrated Development Environment*.
- Interface utilisateur : type d'interface informatique qui permet à un usager de manipuler la machine. Elle coordonne les interactions homme-machine et fait partie du design industriel.

- Javascript : langage informatique utilisé sur les pages web. Ce langage à la particularité de s'activer sur le poste client, c'est l'ordinateur qui reçoit le code et qui l'exécute, contrairement à d'autres langages qui sont activés côté serveur. L'exécution du code est faite par le navigateur.

- Librairie : ensemble de fonctions utilitaires, regroupées et mises à disposition afin de pouvoir être utilisées sans avoir à les réécrire. Les bibliothèques logicielles se distinguent des exécutables dans la mesure où elles ne représentent pas une application. Elles ne sont pas complètes, elles ne possèdent pas l'essentiel d'un programme comme une fonction principale et par conséquent ne peuvent pas être exécutées directement. Les bibliothèques peuvent regrouper des fonctions simples comme des fonctions complexes. L'intérêt des bibliothèques réside dans le fait qu'elles contiennent du code utile que l'on ne désire pas avoir à réécrire à chaque fois.
- Logiciel : software en anglais, il s'agit de la partie non tangible de l'ordinateur, c'est-à-dire les programmes disponibles sur une machine. Le logiciel est aussi indispensable au fonctionnement d'un ordinateur que le matériel lui-même.

- MongoDB : système de gestion de base de données orienté documents, ne nécessitant pas de schéma prédéfini des données. Il fait partie de la mouvance NoSQL.
- MySql : base de données relationnelle, très employée sur le Web, souvent en association avec PHP et Apache. MySql fonctionne indifféremment sur tous les systèmes d'exploitation (Windows, Linux, Mac OS notamment).

- Npm : gestionnaire de paquets officiel de Node.js
- Node.js : plateforme logicielle libre et événementielle en JavaScript orientée vers les applications reseau. Il s'agit d'un environnement bas niveau permettant l'exécution de JavaScript côté serveur.

- Paquets : il s'agit d'une archive (fichier compressé) comprenant les fichiers informatiques, les informations et procédures nécessaires à l'installation d'un logiciel sur un système d'exploitation.
- Pare-feu : logiciel et/ou matériel permettant de faire respecter la politique de sécurité du réseau, celle-ci définissant quels sont les types de communications autorisés sur ce réseau informatique. Il surveille et contrôle les applications et les flux de données (paquets).
- Port : un port logiciel est un système permettant aux ordinateurs de recevoir ou d'émettre des informations.
- Php Storm : éditeur pour PHP, HTML, CSS et JavaScript. Cet environnement de développement a été édité par JetBrains.
- Proxy : composant logiciel informatique qui joue le rôle d'intermédiaire entre deux hôtes pour faciliter ou surveiller leurs échanges. Dans un réseau informatique, le proxy est un programme servant d'intermédiaire pour accéder à un autre réseau, généralement internet

- Slack : logiciel de collaboration Cloud qui, en plus d'une fonctionnalité de chat entre utilisateurs ou en groupe, permet de partager des documents. Slack inclut des fonctionnalités de messagerie directe, de notifications et d'alertes, de partage de documents, de discussion de groupe et de recherche de contenu.
- Socket.io : bibliothèque JavaScript pour les applications Web en temps réel. Il permet une communication bidirectionnelle en temps réel entre les clients Web et les serveurs. Il comprend deux parties: une bibliothèque côté client qui s'exécute dans le navigateur et une bibliothèque côté serveur pour Node.js.

- Système d'exploitation : logiciel qui, dans un appareil électronique, pilote les dispositifs matériels et reçoit des instructions de l'utilisateur ou d'autres logiciels (ou applications).

- Token : terme utilisé pour désigner un identificateur.

- Vim : éditeur de texte, logiciel permettant la manipulation de fichiers texte.
- Vi : éditeur de texte en mode texte plein écran écrit par Bill Joy en 1976 sur une des premières versions de la distribution Unix.

WEBOGRAPHIE

https://reactjs.org/

https://www.npmjs.com/

https://www.docker.com/

Create basic login forms using create react app module in reactjs

A Chat Application Using Socket.IO

Making React realtime with websockets

Building a Node.js WebSocket Chat App with Socket.io and React

tutorial socket.io

Créer une interaction serveur/client avec SOCKET.IO - Programmation web

http://dridk.me/MongoDB.html

Le Web temps réel avec Socket.IO

https://www.docker.com/resources/what-container

https://docs.docker.com/install/linux/docker-ce/ubuntu/#set-up-the-repository

Entrer dans un container Docker actif

https://medium.com/dailyjs/combining-react-with-socket-io-for-real-time-goodness-d26168429a34

https://blog.cloudboost.io/creating-a-chat-web-app-using-express-js-react-js-socket-io-1b01100a8ea5

Codes source de l'interface utilisateur du chat :

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import * as serviceWorker from './serviceWorker';
import App from "./App";

ReactDOM.render(<App/>, document.getElementById('root'));

serviceWorker.unregister();

// 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
```

Menu.js

```
import React from 'react';
import {Card} from 'react-bootstrap';
import {Link} from "react-router-dom";
import imgHome from '../images/chat-img2.png';

class Menu extends React.Component{
    render(){
        return(
            <div className="menu">
                <img src={imgHome} alt=""/>
                <Card className="textMenu" style={{ width: '30rem' }}>
                    <Card.Body>
                        <Card.Title>Bienvenue sur Meow !</Card.Title>
                        <Card.Text>
                                    Meow est un chat. Il te permet d'échanger rapidement
                                    et facilement avec un ou plusieurs utilisateur(s)
                                    connecté(s) au chat.
                            Pour l'utiliser, inscris-toi ! ;)
                        </Card.Text>
                            <Card.Link>
                                    <Link className="logLink" to="/login">
                                        -> C'est parti !
                                    </Link>
                                </Card.Link>
                    </Card.Body>
                </Card>
            </div>
        )
    }
}

export default Menu;
```

RoutingApp.js

```
import React from 'react';
import {BrowserRouter as Router, Route, Link} from "react-router-dom";
import Menu from './components/menu.js';
import logo from "./images/logochat.png";
import {Nav} from "react-bootstrap";
import {TchatApp} from "./components/homeChat";

function RoutingApp(){
    return(
        <Router>
            <div className="navBar">
                <Nav className="navbar fixed-top navbar-expand-lg">
                            <a className="navbar-brand" href="/home"><img src={logo}
                            className="logo" alt="logo" /></a>
                     <Nav.Link componentClass='span'><Link to="/home">Home</Link></Nav.Link>
                   <Nav.Link componentClass='span'><Link to="/login">Log In</Link></Nav.Link>
                </Nav>
            </div>
            <Route path="/home" component={Home}/>
            <Route path="/login" component={Login}/>
            <Route path="/meow-chat" component={Tchat*App}/>
        </Router>
    );
}
function Home(){
    return <Menu />
}

//Formulaire de connexion
class Login extends React.Component {
    handleValidation(){
        const userInput = document.querySelector('[name="userInput"]');
        console.log(userInput);
        sessionStorage.setItem('user', userInput.value);
    }
    render() {
        return (
            <form className="form">
                      <p>You must log in to access to Meow Chat*.</p>
                      <input type="text" required placeholder="Choose your pseudo" name="userIn-
                      put"/>
                      <Link to="/meow-chat"><button onClick={this.handleValidation}
                      className="btn btn-sm" type="submit">Log in</button></Link>
            </form>
        );
    }
}
export default RoutingApp;
```

TchatApp.js

```
import React from 'react';
import io from 'socket.io-client';
import {Navbar, Button} from "react-bootstrap";

// Initialisation du chat
function getID() {
    let id = (new Date().getTime() + Math.floor((Math.random()*10000)+1)).toString(16);
    return id;
}
const username = sessionStorage.getItem('user');
let clientID = getID();
console.log('id du client : ', clientID);
const token = 'UmgoIhfJWomN4jLW5wPXoMkIY8l5PPY6Tq8bPjzx6mg';
const socket = io('http://192.168.105.127:5000', {
    query: {
        'clientid': username,
        'Authorization': token
    }
});
console.log('user:', username);

// Chat
export class TchatApp extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            username: sessionStorage.getItem('username'),
            msg: '',
            userSender: '',
            receiverTalkingWith:'You are sending messages to all Meow Members',
        };
        this.handleMessage = this.handleMessage.bind(this);
        this.handleSubmit = this.handleSubmit.bind(this);
        this.textMessageName = 'textMessageName';
    }
    handleMessage() {
        let chatInput = document.querySelector('[name="' + this.textMessageName + '"]');
        this.setState({msg: chatInput.value});
    }
    handleSubmit(event) {
        event.preventDefault();
        let chatInput = document.querySelector('[name="' + this.textMessageName + '"]');
        //envoi d'un message
        console.log(chatInput.value);
        socket.emit("chatMessage", {
            sender: username,
            message: chatInput.value,
            room:localStorage.getItem('room')
        });
        /*console.log('conversation :', localStorage.getItem('room'));
        if(localStorage.getItem('room')!==undefined){
            socket.emit("chatMessage", {
                sender: username,
                message: chatInput.value,
                room:localStorage.getItem('room')
            });
        }*/
    }
```

```
    render() {

        // réception des messages
        socket.on("received", data => {
            console.log(data);
            let divMsg = document.createElement('div');
            let divMsgSent = document.createElement('div');
            let divSen = document.createElement('div');
            let divHeure = document.createElement('div');
            const time = new Date().getHours()+":"+ new Date().getMinutes();
            //+":"+ new    Date().getSeconds();
            divMsg.className = "messageBox";
            divMsgSent.className = "messageBoxSent";
            divMsg.id = 'msgBox';
            divSen.className = "sender";
            divHeure.className = "details";
            let messages = document.getElementById('messages');
            //differenciation messages
            if(data.sender!==null){
                if(data.sender===username){
                    messages.appendChild(divMsgSent).appendChild(divSen).append(data.sender);
                    messages.appendChild(divMsgSent).append(data.message);
                    messages.appendChild(divMsgSent).appendChild(divHeure).append(time);
                }else{
                    messages.appendChild(divMsg).appendChild(divSen).append(data.sender);
                    messages.appendChild(divMsg).append(data.message);
                    messages.appendChild(divMsg).appendChild(divHeure).append(time);
                }
            }
        });

        // Affichage de la conv privée
        function closeTalk(){
            console.log('clear');
            document.location.reload();
        }
        return (
            <div className="containerPage">
                <div className="userName">Logged in as {clientID} / {username}</div>
                <ListeGauche/>
                <div className="chatBox">
                    <div className="headerBox">
                        <Navbar>
                            <Navbar.Brand>Talking with : </Navbar.Brand>
                            <Navbar.Toggle/>
                            <Navbar.Collapse className="justify-content-end">
                                <Navbar.Text>
                                    <div id="talkingWith"></div>
                                </Navbar.Text>
                            </Navbar.Collapse>
                        </Navbar>
                        <Button className="alert-danger" onClick={closeTalk}>Close X</Button>
                    </div>
                    <div className="messagesList" id="messages"></div>
                    <div className="textChat">
                        <form className="formChat" onSubmit={this.handleSubmit}>
                            <input id='messageInput' type="text" name="textMessageName"
                            className="form-control" placeholder="Your Message" required/>
                            <button className="btn btn-sm" type="submit">Send</button>
                        </form>
                    </div>
                </div>
            </div>
        )
    }
}
```

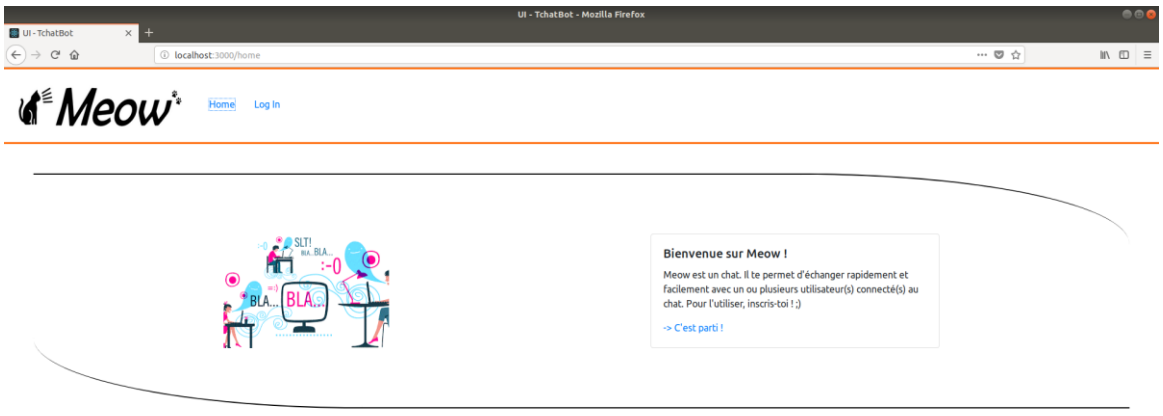
```
// Liste USERS
class ListeGauche extends React.Component {
    state = {
        talkingWith:null
    };
    render(){
        // Liste des users
        socket.emit("listUser");
        socket.on('receivedUsers', data => {
            let listUsers = document.getElementById('listUsers');
            data.forEach(listElements);
                function listElements(element, index, data) {
                    let p = document.createElement('p');
                    p.className = 'contact';
                    p.id = 'contact';
                    let receiver = element.hash;
                    listUsers.appendChild(p).append(receiver);
                    let talkingWith = document.getElementById('talkingWith');
                        p.addEventListener('click', clickEvent);
                        function clickEvent(){
                            socket.emit("openOneToOneTalk", {receiver: element.hash});
                            talkingWith.append(element.hash);
                            p.removeEventListener('click', clickEvent);
                        }
                }
        });

        // Liste des conversations
        socket.emit("listTalk", {user:username});
        socket.on('receivedTalks', data => {
            let listTalks = document.getElementById('listTalks');
            data.forEach(listElements);
            function listElements(element, index, data) {
                let p = document.createElement('p');
                p.className = 'contact';
                p.id = 'contact';
                console.log('listUser');
                listTalks.appendChild(p).append(element.hash);
                /*p.addEventListener('click', clickEvent);
                function clickEvent(){
                    localStorage.setItem('room', element.hash);
                    socket.emit("getTalkMessages", {user: username, talk: element.hash});
                    p.removeEventListener('click', clickEvent);
                }*/
            }
        });
        return(
            <div className="blocGauche">
                <h6>Users</h6>
                <div id="listUsers"></div>
                <h6>Conversations</h6>
                <div id="listTalks"></div>
            </div>
        )
    }
}

export default {TchatApp};
```

Package.json

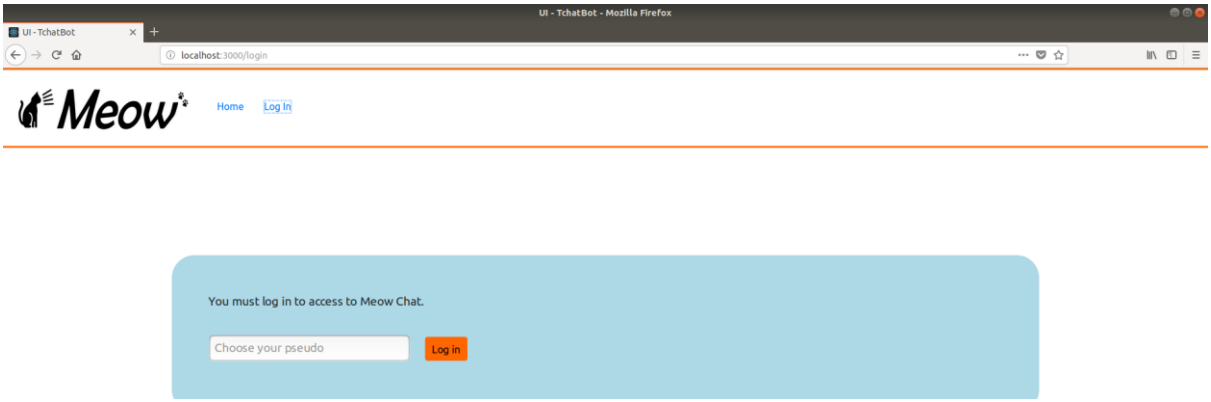
```
{
  "name": "test",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "bootstrap": "^4.3.1",
    "express": "4.16.4",
    "react": "^16.8.6",
    "react-bootstrap": "^1.0.0-beta.8",
    "react-dom": "^16.8.6",
    "react-router": "^5.0.0",
    "react-router-dom": "^5.0.0",
    "react-router-redux": "^4.0.8",
    "react-scripts": "3.0.0",
    "react-spring": "8.0.20",
    "socket.io": "^2.2.0",
    "uws": "10.148.1"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": "react-app"
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  },
  "proxy": "http://172.17.0.1:3128"
}
```

Interface Utilisateur du chat :

- Page d'accueil



- Formulaire de connexion



- Interface du chat

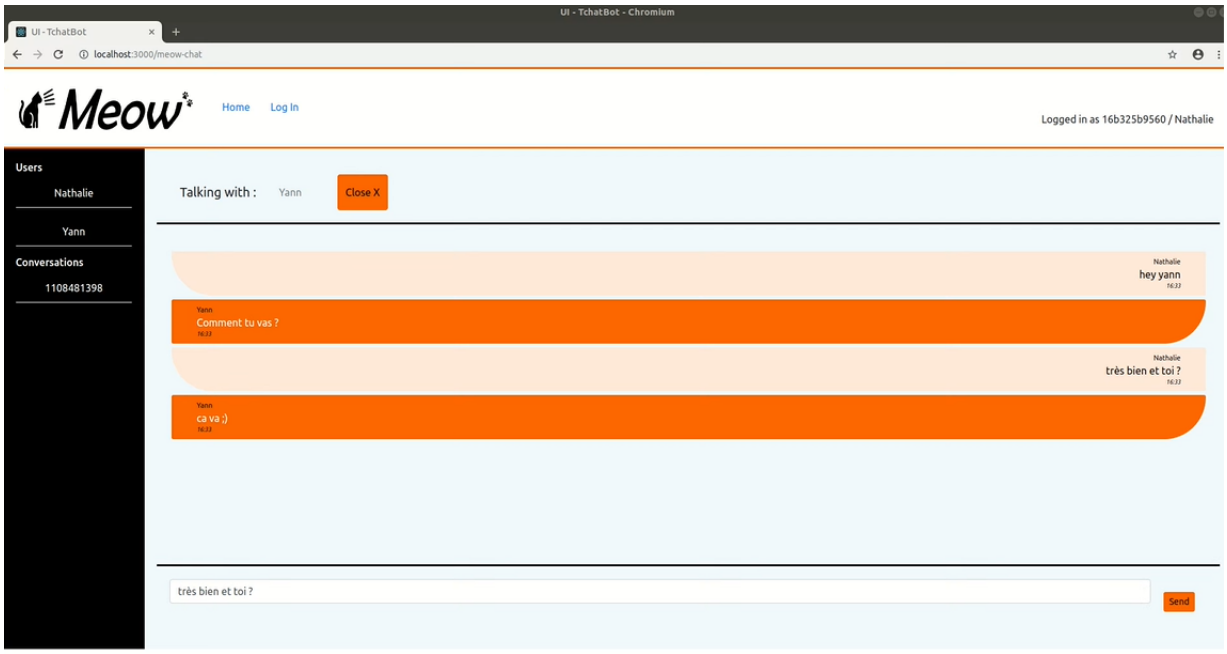


Table des matières