

# **DOSSIER DE PROJET**

En vue de l'obtention du Titre Professionnel  
**Concepteur Développeur d'Applications**

## **LorenzO'Ticket**

Réalisé par  
**HERBET LE FAUCHEUR Tony**

Centre de formation  
**O'clock**

Promotion  
**Phénix**

Référent de formation  
**RIVEY Fanny**

# SOMMAIRE

<b>1 - Introduction:</b>	<b>5</b>
<b>1a - Apothéose :</b>	<b>5</b>
<b>1b - LorenzO'Ticket :</b>	<b>5</b>
Pourquoi ce nom ? :	5
Contextualisation du projet :	5
Description initiale du projet :	5
Why this website and what will be the use ? :	6
En quoi la partie conception a influencé la partie développement ? :	6
<b>2 - Compétences du référentiel couvertes par le projet :</b>	<b>7</b>
<b>2a - Activité type 1, Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité :</b>	<b>7</b>
CP1 - Maquetter une application :	7
CP2 - Développer une interface utilisateur de type desktop :	7
CP3 - Développer des composants d'accès aux données :	7
CP4 - Développer la partie front-end d'une interface utilisateur web :	8
CP5 - Développer la partie back-end d'une interface utilisateur web :	8
<b>2b - Activité type 2, Concevoir et développer la persistance des données en intégrant les recommandations de sécurité :</b>	<b>9</b>
CP6 - Concevoir une base de données :	9
CP7 - Mettre en place une base de données :	9
CP8 - Développer des composants dans le langage d'une base de données :	9
<b>2b - Activité type 3, Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité :</b>	<b>9</b>
CP9 - Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement :	9
CP10 - Concevoir une application :	9
CP11 - Développer des composants métier :	10
CP12 - Construire une application organisée en couches :	10
CP13 - Développer une application mobile :	10
CP14 - Préparer et exécuter les plans de tests d'une application :	10
CP15 - Préparer et exécuter le déploiement d'une application :	11
<b>3 - Cahier des charges :</b>	<b>11</b>
<b>3a - Les rôles :</b>	<b>11</b>
<b>3b - User Stories :</b>	<b>11</b>
Les User Stories du MVP :	11
Les User Stories des fonctionnalités secondaires :	12
<b>3c - Schémas :</b>	<b>13</b>
Architecture :	13
Diagramme de Navigation (envoi de message) :	14
MCD, Modèle Conceptuel des Données :	15

Modèle Entité-Association, ERD (Entity-Relationship Diagram) :	17
MLD, Modèle Logique des Données (LDM, Logical Data Model) :	18
MPD-R, Modèle Physique des Données Relationnelles :	19
Workflow :	20
Routes :	20
Diagramme de Séquence pour la création d'un ticket :	21
Cas d'utilisation (Use Case) pour le MVP :	22
<b>3d - Charte graphique :</b>	<b>23</b>
Typographies :	23
Couleurs :	23
<b>3e - Wireframes :</b>	<b>23</b>
<b>4 - Spécifications techniques :</b>	<b>27</b>
<b>4a - Versionning :</b>	<b>27</b>
<b>4b - Front :</b>	<b>27</b>
Organisation des dossiers et fichiers :	28
<b>4c - Back :</b>	<b>28</b>
Organisation des dossiers et fichiers :	29
<b>4d - Déploiement :</b>	<b>30</b>
<b>5 - Organisation :</b>	<b>34</b>
<b>5a - L'équipe :</b>	<b>34</b>
<b>5b - Outils :</b>	<b>34</b>
<b>5c - Méthodologie :</b>	<b>37</b>
<b>5d - Conventions et règles :</b>	<b>39</b>
Git :	39
Code :	39
<b>5e - Github :</b>	<b>41</b>
Que faire pour commit ? :	41
Rebase :	41
Reset last commit :	41
Squash :	41
Delete branch :	42
Amend :	42
Stash:	42
Github :	43
<b>6 - Réalisations :</b>	<b>43</b>
<b>6a - Journal de bord :</b>	<b>43</b>
Journée 1 - Sprint 0	43
Journée 2 - Sprint 0	43
Journée 3 - Sprint 1	44
Journée 4 - Sprint 1	44
Journée 5 - Sprint 2	44
Journée 6 - Sprint 2	44
Journée 7 - Sprint 3	44

Journée 8 - Sprint 3	45
Journée 9 - Sprint 4	45
Journée 10 - Sprint 4	45
Journée 11 - Sprint 5	45
Journée 11 - Sprint 5	45
Journée 12 - Sprint 6	46
Journée 13 - Sprint 6	46
Journée 14 - Sprint 7	46
Journée 15 - Sprint 7	46
Journée 16 - Sprint 7	46
Journée 17 - Sprint 7	47
Journée 18 - Sprint 8	47
Journée 19 - Sprint bonus	47
<b>6b - Tests :</b>	<b>48</b>
<b>6c - Validation des données :</b>	<b>51</b>
<b>6d - Explication d'une fonctionnalité :</b>	<b>53</b>
<b>7 - Recherches :</b>	<b>62</b>
<b>7a - Veilles sur la sécurité :</b>	<b>62</b>
<b>7b - Recherches :</b>	<b>65</b>
<b>8 - Conclusion :</b>	<b>67</b>
<b>8a - LorenzO'Ticket :</b>	<b>67</b>
<b>8b - Améliorations :</b>	<b>68</b>
<b>8c - Remerciements :</b>	<b>68</b>

# 1 - Introduction:

## 1a - Apothéose :

Dans le cadre de ma formation chez O'clock, nous devons travailler pendant un mois sur un projet de fin de formation, ce que nous appelons l'Apothéose

Dans ce but, tous les élèves pouvaient proposer des idées de projet et nous devions ensuite faire une liste de ceux sur lesquels nous aimerais travailler.

Nous avons ensuite été répartis en groupe sur les différents projets qui ont été proposés.

Le projet qui m'a été assigné est LorenzO'Ticket.

## 1b - LorenzO'Ticket :

Pourquoi ce nom ? :

Il existe une tradition non officielle et tacite lors d'une formation chez O'clock qui veut que les élèves nomment leurs projets en commençant ou en utilisant un "O".

Ce projet a été initialement proposé par notre référent de promotion et avait comme nom officieux "gestion de ticket". Suite à une blague au sein du groupe, en début de projet, à propos de Lorenzo Lamas et ayant en tête la tradition O'clockienne, nous avons commencé à appeler le projet LorenzO'Ticket pour plaisanter mais cela a fini par rester.

Je vais donc vous présenter le projet auquel j'ai participé et qui s'appelle "LorenzO'Ticket".

Contextualisation du projet :

Description initiale du projet :

*Votre super agence web, BetterCode Corporation, aurait besoin d'une solution pour gérer les erreurs et demandes d'assistance remontées par ses clients. Et quoi de mieux, pour une agence web, que de créer en interne un outil sur mesure ? Après tout, on est jamais mieux servi que par soi-même !*

*On vous a donc choisis pour réaliser cette solution maison. Le brief est très court, c'est à vous de proposer ce qui vous semble le mieux correspondre au besoin. Dans l'idée, le patron voudrait que ce soit hyper simple pour les clients, qui ne sont pas toujours très familiers avec l'outil informatique. Idéalement, pas besoin de compte (la demande pouvant émaner de n'importe quelle personne travaillant au sein de l'entreprise cliente), juste un formulaire assez court mais permettant de qualifier au mieux son besoin. Et une fois le ticket publié, on lui enverrait un lien de suivi de sa demande.*

*En interne, par contre, ce n'est pas la même tisane. Il va falloir imaginer une interface pour centraliser toutes les demandes et simplifier leur gestion par vos collègues de l'agence. Les*

*tickets seront traités par les différents services et chaque réponse sera accompagnée d'un moyen d'identifier qui a répondu.*

Why this website and what will be the use ? :

The fake web agency, BetterCode Corporation, wanted a solution to handle their clients website errors and assistance needs.

For the clients this is a means to report issues they may encounter on their applications made by the agency.

They will be able to create tickets to report any issues they may have.

They can follow the advancement of said tickets and post messages to add information.

For the intervenors, meaning the various workers of the agency, this is a means to centralize all the problems that need to be fixed.

Employees will be able to attribute tickets for themselves and multiple employees could work on the same ticket.

They too could post messages to ask for more information or to give information regarding the advancement of the ticket.

Three roles will be available :

- Intervenor, this is an employee that will work on tickets
- Lead, in a future update, team lead will be able to assign tickets to intervenors. This is both to help the agency manage their workforce, but also because we can't expect clients to know exactly what service is concerned for their ticket.
- Admin, they will be able to do everything but managing the back office will be their primary role. They will be in charge of creating accounts for the clients with the emails previously given by them.

En quoi la partie conception a influencé la partie développement ? :

Ce projet ayant été proposé par Fanny, notre référent de promotion, nous n'avions pas dans l'équipe de Product Owner. Nous avons donc, en équipe, partagé ce rôle et avons choisi de faire des modifications par rapport à la proposition initiale.

L'objectif principal étant de simplifier la procédure, au maximum pour le **Client**, nous ne lui demandons pas de savoir à quelle catégorie appartient son problème, et par conséquent à quel service sera attribué son ticket. Par exemple, un problème de css ou de marketing, ne sera pas réglé par les mêmes personnes.

Nous avons donc pensé à créer un rôle **Lead**, qui filtre les tickets en les attribuant aux **Intervenors**, qui eux, interviendront sur les tickets.

Initialement, la proposition de projet spécifiait que le **Client** devrait avoir accès au formulaire de création de ticket sans connexion. Nous avons décidé d'en mettre une, pour des raisons de sécurité, pour s'assurer que les tickets soient bien créés par des clients de l'agence.

Toujours dans un souci de simplicité d'utilisation pour le client, sa connexion se fait uniquement avec l'adresse mail de son entreprise, préalablement fournie et ajoutée en base de données par un **Admin** de l'agence.

## 2 - Compétences du référentiel couvertes par le projet :

**2a - Activité type 1, Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité :**

### CP1 - Maquetter une application :

Nous avons rédigé un cahier des charges lors des six premiers jours, il contient des wireframes ou maquettes fonctionnelles, pour les versions mobile, tablette et desktop de l'application.

Nous avons également réalisé un schéma du workflow afin de prévoir les cheminements possibles lors de la navigation ainsi qu'une liste préliminaire des routes, ou url, dont nous aurions besoin.

Un choix pour les typographies et les couleurs a aussi été fait à ce moment-là.

### CP2 - Développer une interface utilisateur de type desktop :

Nous n'avons pas réalisé d'interface exécutable uniquement sur desktop mais un site que nous avons essayé, autant que faire se peut, de rendre adapté à toutes les tailles d'écran. L'interface pour taille desktop est fonctionnelle et a été prioritaire par rapport à celle pour téléphones, plus d'informations à ce sujet sur la CP 13.

Pour la réalisation de l'interface utilisateur, nous avons suivi les wireframes réalisés lors de la phase de conception et quelques tests unitaires liés à l'interface ont été écrits.

### CP3 - Développer des composants d'accès aux données :

Pour les besoins du projet, nous avons dû créer une API qui permet à un utilisateur authentifié, via notre front-end, de voir et/ou manipuler les informations de notre base de données lorsque cette manipulation est prévue pour cette utilisateur.

## CP4 - Développer la partie front-end d'une interface utilisateur web :

Comme énoncé précédemment, nous avons développé la partie front-end en utilisant la bibliothèque React et il est dans sa quasi totalité dynamique et adapté à la taille de l'écran.

- React :
  - React est une bibliothèque front avec une grande communauté, donc une documentation complète et à jour
  - Il s'agit de la bibliothèque que nous maîtrisons le mieux
  - Il permet une meilleure scalabilité (de par son fonctionnement par composants) dans une optique d'évolution de l'application
- Apollo :
  - Aide à la consommation de l'API côté front, faire les requêtes, réutilisation des types (apollo-client)
- TypeScript & Jest :
  - Amélioration de la qualité et sécurité du code:
    - Types
    - Tests
- React router dom:
  - Gestion des routes de notre application côté front

## CP5 - Développer la partie back-end d'une interface utilisateur web :

Lors de la phase de conception, nous avons préparé la création de la base de données en réalisant divers schémas qui sont disponibles dans la partie Cahier des charges.

Nous avons utilisé :

- PostgreSQL :
  - Système de gestion de base de données relationnelle et objet
- GraphQL :
  - Customisation des requêtes à la BDD pour n'avoir que ce qui est voulu selon la situation (légèreté des requêtes)
  - Fortement typé
- Apollo :
  - Permet la création et la mise en place d'une API qui implémente GraphQL (apollo-server)
- NodeJS :
  - Afin d'avoir un environnement homogène côté front et back
- Joi :
  - Validation des données

## **2b - Activité type 2, Concevoir et développer la persistance des données en intégrant les recommandations de sécurité :**

### **CP6 - Concevoir une base de données :**

Une base de données relationnelle a été réalisée pour ce projet.

Plus d'informations sur la conception dans la CP10 et la partie Cahier des charges.

### **CP7 - Mettre en place une base de données :**

En utilisant les documents faits pendant la phase de conception, nous avons mis en place une base de données PostgreSQL afin d'avoir une persistance de nos données et pouvoir y accéder via notre API.

### **CP8 - Développer des composants dans le langage d'une base de données :**

Nous avons choisi d'utiliser GraphQL pour communiquer avec notre base de données.

Quelques fichiers SQL ont été écrits afin de créer et d'initialiser puis peupler notre base de données. Les requêtes de l'API ont été construites via la bibliothèque Knex.

## **2b - Activité type 3, Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité :**

### **CP9 - Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement :**

Nous avons travaillé en utilisant divers outils de collaboration et avons mis en place des conventions et règles et ce en travaillant de façon agile.

Plus d'informations dans la partie Organisation.

### **CP10 - Concevoir une application :**

En plus de ce qui a déjà été énoncé plus tôt, nous avons prévu, lors de la rédaction du cahier des charges, les différentes fonctionnalités nécessaires à notre application.

Pour ce faire, nous avons rédigé des User stories et les avons organisées pour définir un MVP, ou produit minimum viable.

Plusieurs autres schémas ont également été réalisés :

- Diagramme de navigation
- Diagramme de cas d'utilisation pour le MVP
- Diagramme de séquence pour la création d'un ticket
- Diagramme entité relation
- Modèle conceptuel des données
- Modèle physique de données relationnelles
- Schéma de l'architecture

Le cahier des charges est disponible dans la suite de ce document.

#### CP11 - Développer des composants métier :

Différents composants métier ont été développés pour les besoins de l'application. Nous avons suivi ce qui avait été prévu lors de la phase de conception.

#### CP12 - Construire une application organisée en couches :

Après de nombreuses discussions lors de la phase de conception, nous avons choisis de repartir et développer l'application avec une architecture 4 couches :

- Presentation Layer, les vues de notre application.
- Business Layer, pour la manipulation des données.
- Data Access Layer, la partie API.
- Persistence Layer, pour notre base de données.

#### CP13 - Développer une application mobile :

Nous sommes partis du postulat que l'agence n'avait pas une clientèle ayant un besoin fort d'utiliser l'application sur téléphone, nous n'avons donc pas réalisé d'application mobile et nous nous sommes contentés de rendre le site responsive comme énoncé sur la CP2.

L'hypothétique augmentation de ce besoin a été prise en compte lors de la conception. En effet, en choisissant de faire notre interface utilisateur en React, il serait alors plus simple de réaliser une application à part entière en React Native pour téléphone par exemple.

Il serait même envisageable de tout basculer sur un monorepo en React Native pour toutes les plateformes via l'utilisation de la bibliothèque react-native-web pour avoir une application utilisable entre autres sur Android, iOS, Windows, macOS et web.

#### CP14 - Préparer et exécuter les plans de tests d'une application :

Des tests manuels ont été réalisés au fur et à mesure du développement pour le front et le back.

Des tests unitaires ont été écrits pour la partie front mais seulement pour la page de connexion car nous avons manqué de temps et voulions finir un maximum de fonctionnalités du MVP en priorité.

Une intégration continue a été mise en place via les Github actions et une Pull Request ne pouvait pas être merge sans la réussite des tests.

## CP15 - Préparer et exécuter le déploiement d'une application :

Nous avons déployé l'application, plus d'informations sur le sujet dans la suite de ce dossier.

## 3 - Cahier des charges :

### 3a - Les rôles :

**Client:** Personne qui crée des tickets, un client de l'agence.

**Intervenors:** Personne qui s'occupe de traiter les tickets, un employé de l'agence.

**Lead:** Chef d'une équipe précise d'intervenors et qui peut attribuer des tickets à ses intervenors (v2), un employé de l'agence.

**Admin:** Gestion du back office (tous les droits), un employé de l'agence.

### 3b - User Stories :

#### Les User Stories du MVP :

Le MVP (Minimum viable product) est une version simple et épurée d'un projet qui contient seulement les fonctionnalités jugées nécessaires au fonctionnement et à l'utilisation concrète du projet afin d'avoir un produit viable dès que possible.

Les autres fonctionnalités seront ajoutées par la suite lors du développement.

En tant que **Client**, je voudrais pouvoir :

- me connecter pour utiliser le site
- me déconnecter lorsque je n'ai plus besoin d'utiliser le site
- avoir un formulaire, afin de remonter un problème
- avoir une page, afin de lister tous mes tickets
- afficher le détail d'un ticket, afin de voir l'état d'avancement
- sur la page d'un ticket, ajouter un message afin de donner plus d'informations
- fermer mon ticket lorsque celui ci a été traité

En tant qu'**Intervenor**, je voudrais :

- avoir un espace personnel afin de changer mon mot de passe
- avoir un formulaire, afin de me connecter au back office
- pouvoir me déconnecter lorsque je n'ai plus besoin d'utiliser le site
- consulter tous les tickets
- consulter tous les tickets qui me sont attribués afin de connaître ma charge de travail
- afficher le détail d'un ticket, afin de voir toutes les informations le concernant
- changer le "statut" d'un ticket en "ouvert/en cours/fermé" afin de définir son état de prise en charge
- pouvoir m'attribuer un ticket afin de le prendre en charge
- sur la page d'un ticket, ajouter un message pour échanger avec le client

En tant que **Lead** je voudrais pouvoir :

- faire tout ce que peut faire un **Intervenor**

En tant qu'**Admin**, je voudrais :

- faire tout ce que peut faire un **Lead**
- CRUD Client
- CRUD Ticket
- CRUD Employee
- CRUD Message

Les User Stories des fonctionnalités secondaires :

En tant que **Client**, je voudrais pouvoir:

- avoir un champ "type de problème"
- avoir un champ "urgence"
- recevoir un email lorsqu'il y a un changement sur mon ticket
- ajouter une raison lorsque je ferme un ticket
- recevoir un nouveau mdp si je l'ai oublié afin de pouvoir me connecter

En tant qu'**Intervenor**, je voudrais:

- changer le "type de problème" d'un ticket
- avoir un bouton, afin de filtrer les tickets par niveau d'urgence
- avoir un bouton, afin de filtrer les tickets que je me suis assigné
- ajouter une raison lorsque je ferme un ticket
- recevoir un nouveau mot de passe si je l'ai oublié afin de pouvoir me connecter

En tant que **Lead** je voudrais pouvoir:

- attribuer un ticket à un de mes *intervenor*
- avoir une page pour voir tous mes *intervenor*

En tant qu'**Admin**, je voudrais:

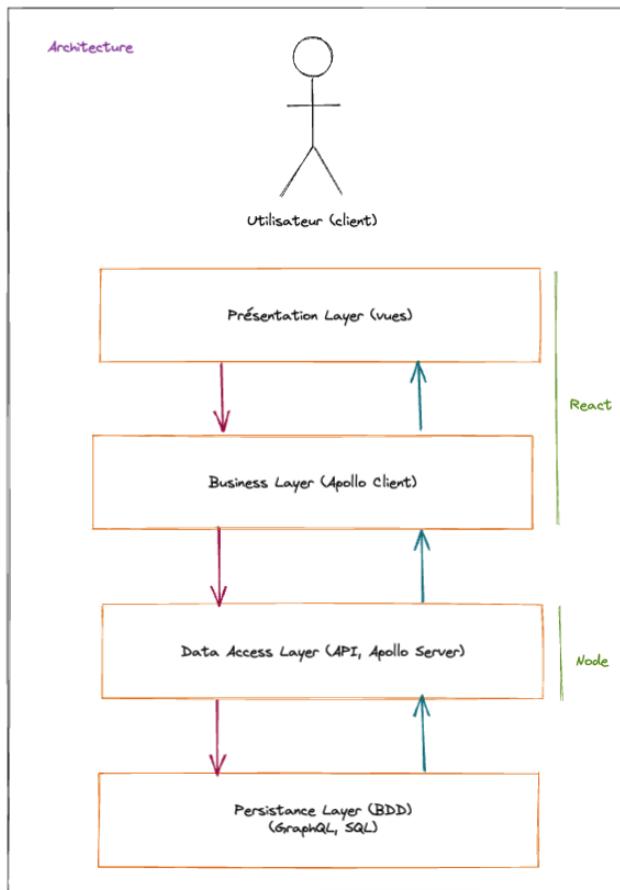
- attribuer un ticket à quelqu'un, afin de le gérer

### 3c - Schémas :

Nous avons réalisé plusieurs schémas.

Lorsque cela était approprié, nous avons utilisé le langage de modélisation unifié, ou UML (Unified Modeling Language).

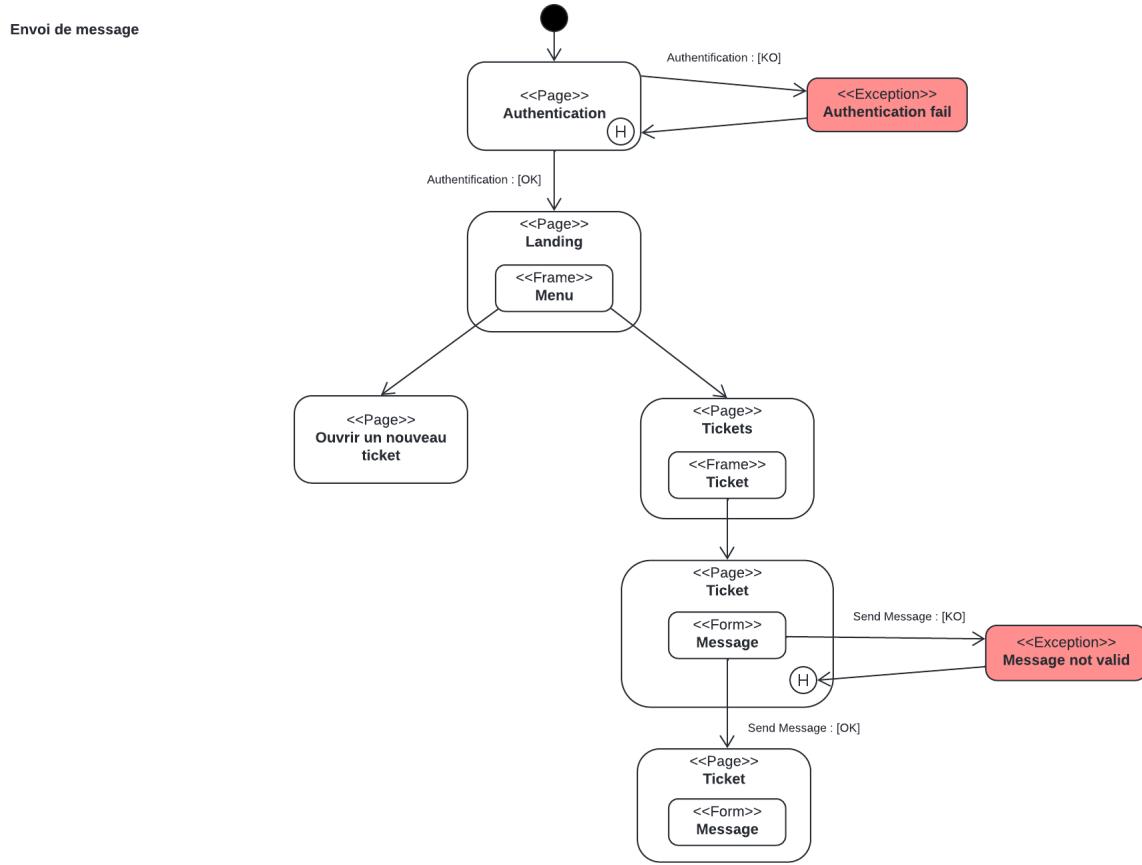
### Architecture :



Nous avons décidé d'utiliser une architecture en quatre couches.

- La première couche, Presentation Layer, fait référence à l'interface utilisateur. Cette couche est ce que l'utilisateur voit. Elle est réalisée à l'aide de React.
- La deuxième couche, Business Layer, est la partie logique derrière notre interface utilisateur. C'est elle qui sera chargée d'enregistrer les informations côté front-end et de faire les appels à l'API. Elle est réalisée avec la partie state de React et Apollo Client.
- La troisième couche, Data Access Layer, est entièrement dédiée à notre API, elle est principalement réalisée via NodeJS et Apollo Server.
- La quatrième et dernière couche, Persistance Layer, est tout simplement notre base de données SQL.

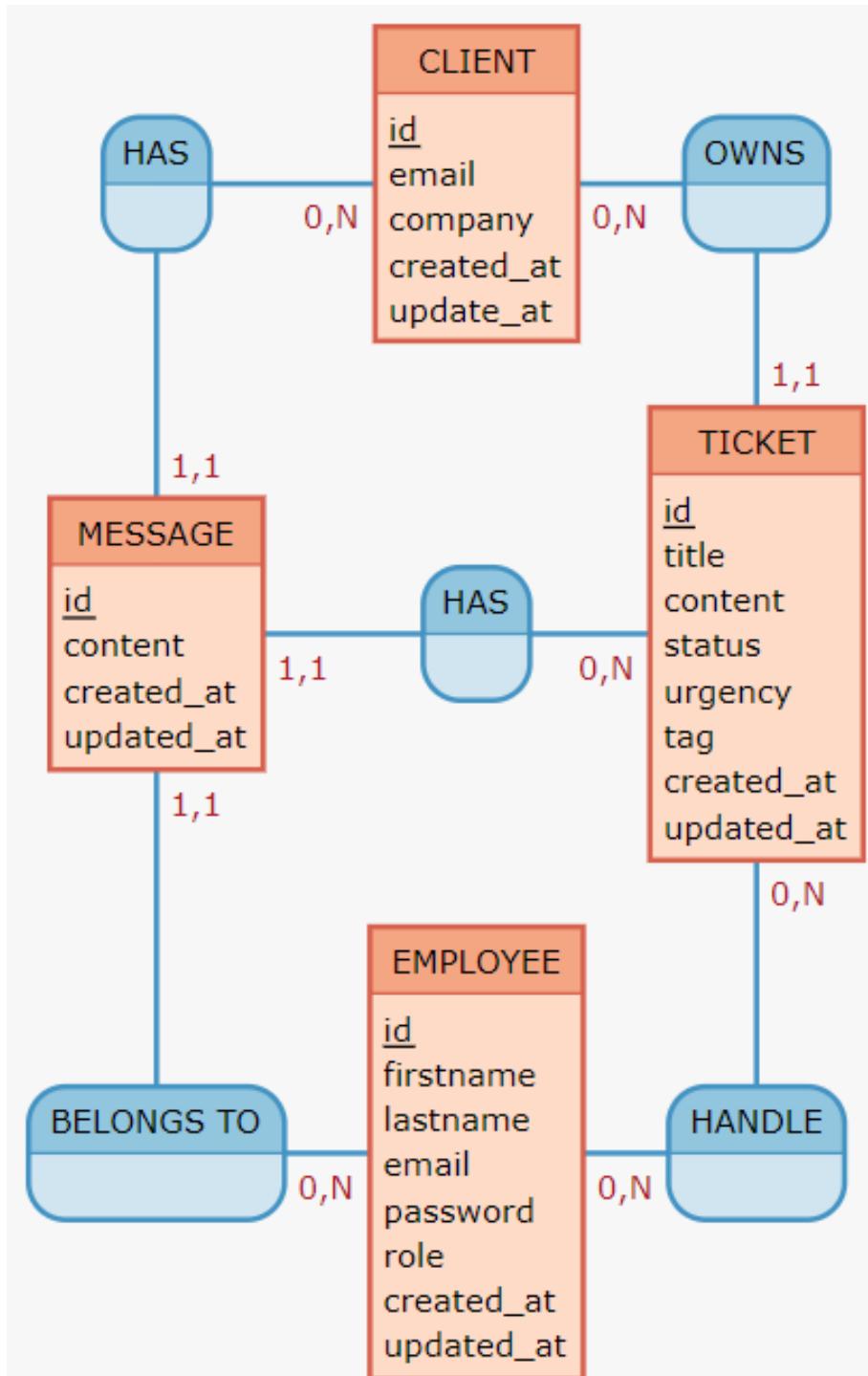
## Diagramme de Navigation (envoi de message) :



Nous avons ici, essayé de faire un diagramme d'état (state) d'une fonctionnalité. Ce type de diagramme a pour but de définir tout le cheminement nécessaire pour une fonctionnalité type. Ici nous avons choisi la fonctionnalité d'envoi d'un message. Lorsqu'un utilisateur arrive sur l'application, il doit se connecter. Si la connexion échoue nous arrivons dans state différent, représenté par un H, un autre diagramme pour ce cheminement et ces spécificités seraient nécessaires. Si la connexion réussit nous arrivons sur la page d'accueil (il faudrait renommer Landing sur le schéma). Depuis cette page d'accueil nous avons deux choix possibles, soit ouvrir un nouveau ticket, soit aller sur un ticket déjà existant. Nous avons détaillé le choix d'aller sur un ticket existant, car la suite est quasiment identique. Pour ce choix, il faut donc aller sur la page des tickets puis choisir un ticket. Nous sommes redirigés vers la page du ticket choisi et nous pouvons donc écrire un message. Ici aussi un state différent est possible, si l'envoi du message réussi nous restons sur la page du ticket et le message s'affiche.

## MCD, Modèle Conceptuel des Données :

Le MCD ou modèle conceptuel de données est la représentation des données ainsi que leurs relations.



Nous pouvons voir ici que notre base de données contiendra quatre entités ou tables représentées en rouge avec leurs propriétés ou champs inscrits en dessous.

Les relations entre nos entités sont représentées en bleu et un mot indique la nature de cette relation.

Les annotations entre nos entités et relations servent à déterminer le type de relation entre nos entités, elles s'appellent des cardinalités et se lisent de cette façon :

Un Client à 0 ou Plusieurs Ticket.

Un Ticket à 1 et 1 seul Client.

Un Client à 0 ou Plusieurs Message.

Un Message à 0 ou Plusieurs Client.

Un Message à 1 et 1 seul Ticket.

Un Ticket à 0 ou Plusieurs Message.

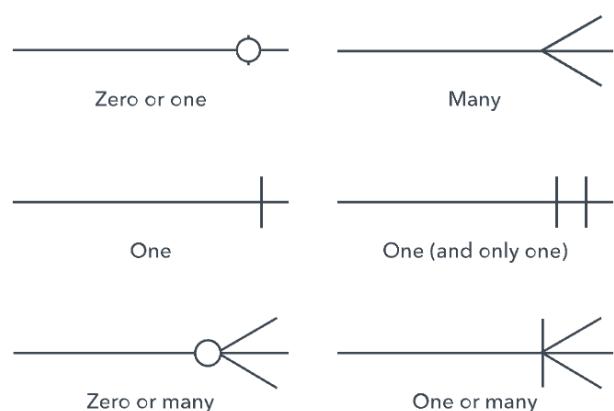
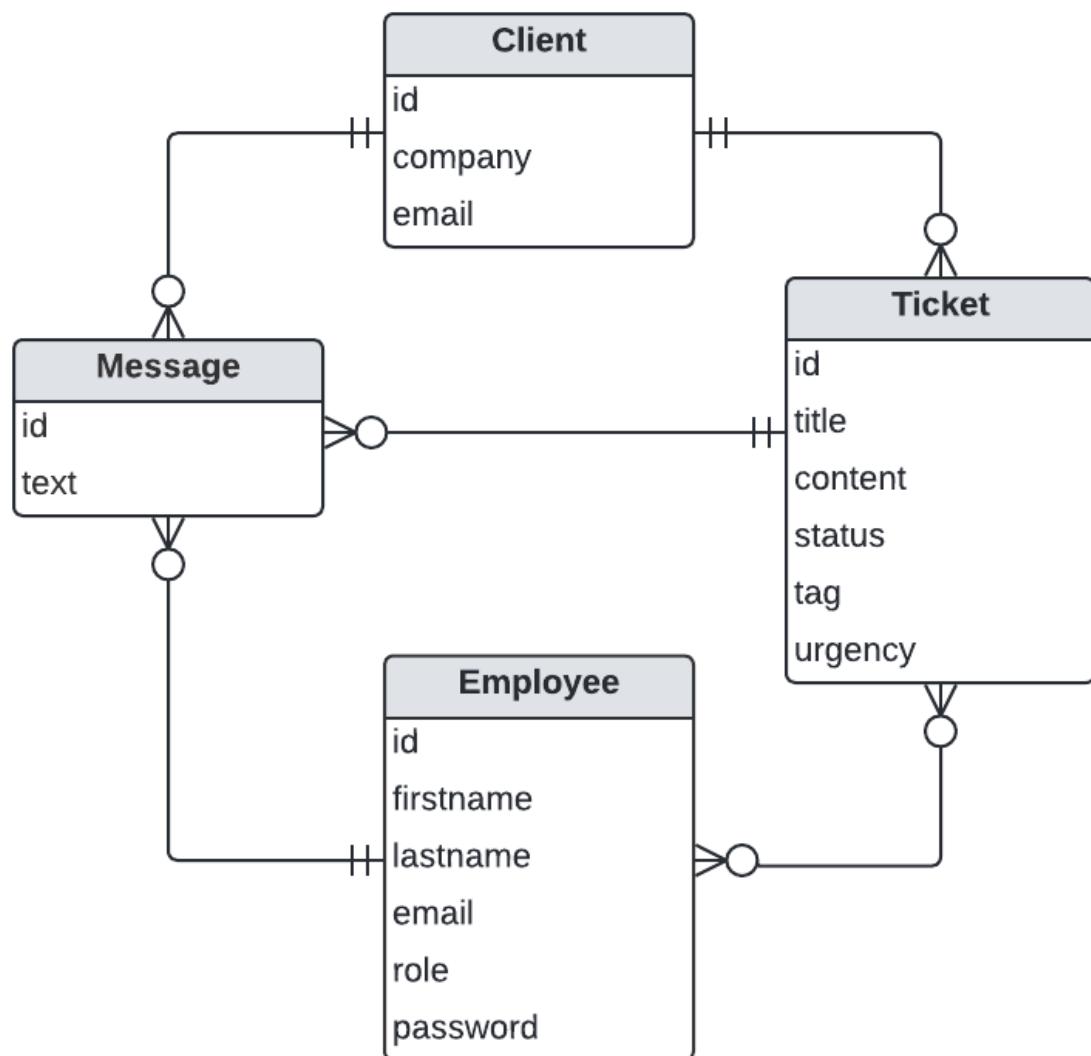
Un Employee à 0 ou Plusieurs Ticket.

Un Ticket à 0 ou Plusieurs Ticket.

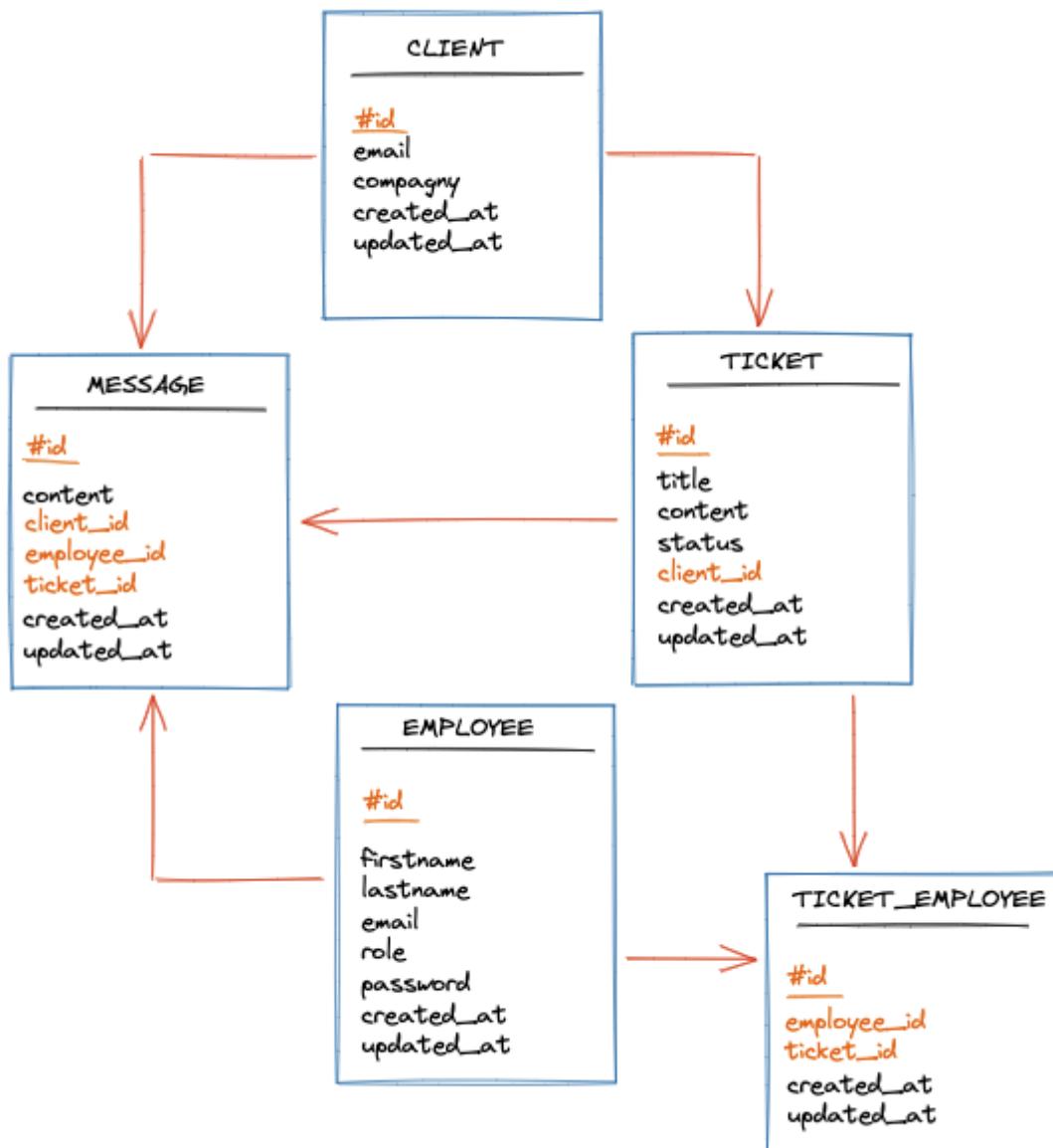
Un Employee à 0 ou Plusieurs Message.

Un Message à 1 et 1 seul Employee.

Modèle Entité-Association, ERD (Entity-Relationship Diagram) :

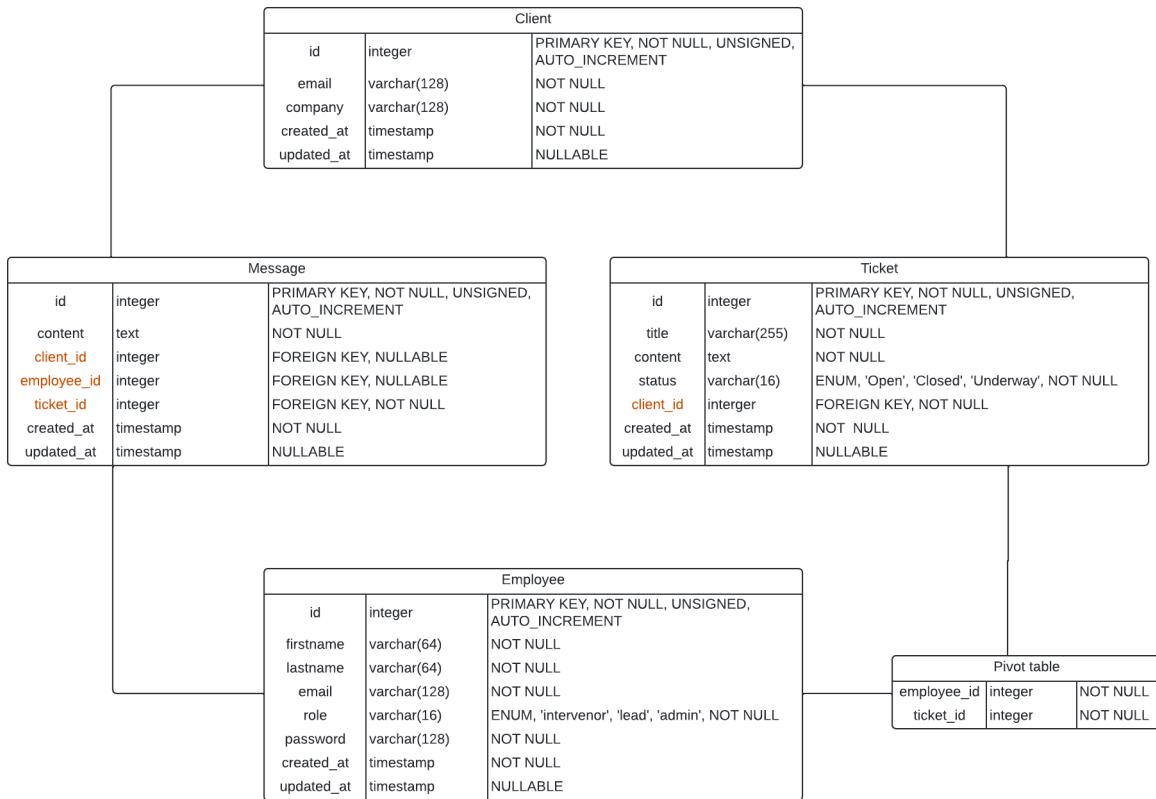


MLD, Modèle Logique des Données (LDM, Logical Data Model) :



Les entités deviennent des tables, les cardinalités disparaissent et les clefs étrangères apparaissent ainsi que la table pivot.

## MPD-R, Modèle Physique des Données Relationnelles :



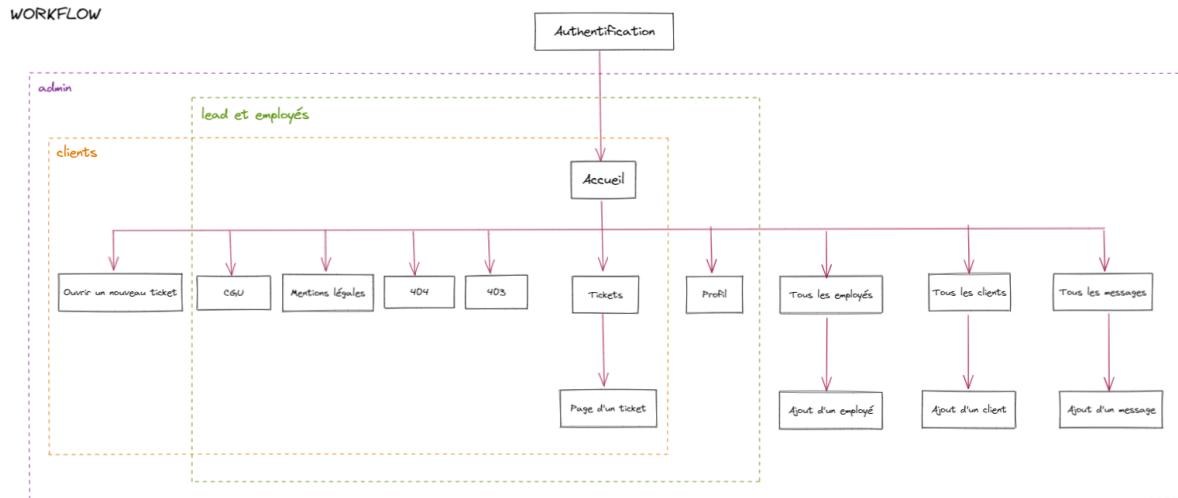
Ici aussi, par rapport à un MCD, les entités deviennent des tables, les relations et les cardinalités disparaissent et les clefs étrangères apparaissent.

Les propriétés se transforment en champs (ou attributs) et on note le type (integer, varchar etc) de ces champs ainsi que leurs paramètres (not null, PK, etc).

Nous n'avons pas ressenti le besoin d'écrire un dictionnaire de données, car les informations sont déjà présentes sur le MPD-R.

## Workflow :

Le workflow ou flux de travaux, dans le cadre de la réalisation d'une application web est la représentation de la navigation du site en suivant son arborescence.



Chaque page est représentée par une forme rectangulaire qui contient son nom.

Les flèches représentent le cheminement basique. On peut donc voir qu'il faut forcément être authentifié pour pouvoir accéder à l'accueil et que depuis cet accueil nous pouvons aller sur toutes les pages.

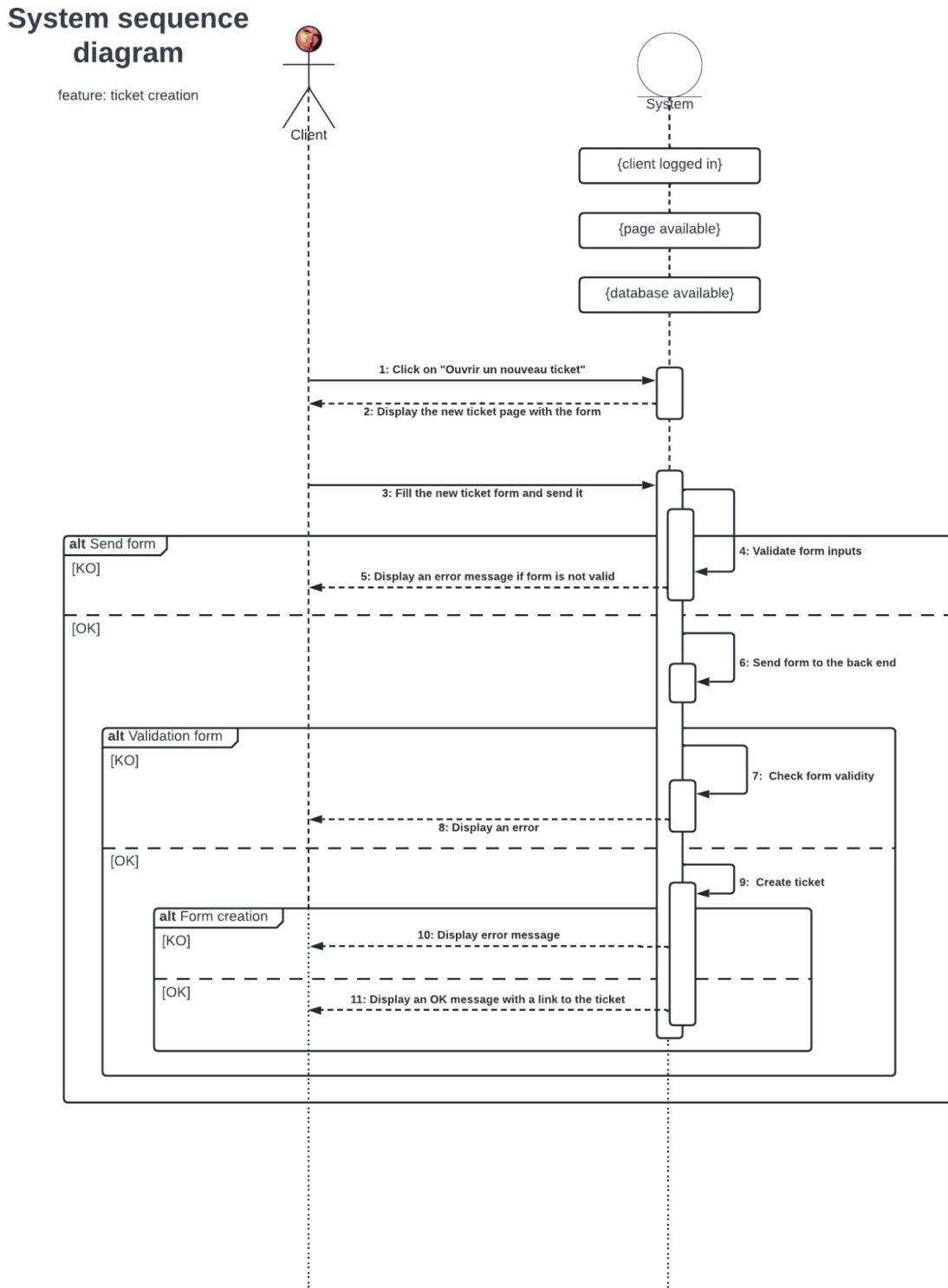
Les rectangles de couleurs représentent les rôles et englobent les pages auxquels ils auront accès. Les clients ne possèdent donc pas les droits pour la page de "tous les employés" par exemple.

## Routes :

URL	Authentification	méthode	vue	commentaire
/	non	mutation	formulaire de connexion	deux types de connexion pour les clients et employés
/accueil	oui	mutation	page de présentation de l'application et liens disponibles	pages disponibles selon les rôles (droits d'accès)
/creer-un-ticket	oui	mutation	formulaire d'ajout de ticket	uniquement pour les clients
/tickets	oui	query	liste de tous les tickets	liste des tickets personnels pour les clients, et de tous les tickets pour les employés
/ticket/:id	oui	query	page détail d'un ticket	pour tout le monde, selon les rôles (un client n'aura accès qu'à ses tickets)
/employes	oui	query	liste de tous les employés	uniquement pour l'administrateur de l'application
/ajout-employe	oui	mutation	formulaire d'ajout d'un employé	uniquement pour l'administrateur de l'application
/clients	oui	query	liste de tous les clients	uniquement pour l'administrateur de l'application
/ajout-client	oui	mutation	formulaire d'ajout d'un client	uniquement pour l'administrateur de l'application
/messages	oui	query	liste de tous les messages	uniquement pour l'administrateur de l'application
/ajout-message	oui	mutation	formulaire d'ajout d'un message	pour tout le monde
/profil	oui	mutation	modification information profil d'un employé	modification de ses informations (mot de passe)
/cgua	oui	-	page des conditions générales d'utilisation de l'application	pour tout le monde
/mentions-legales	oui	-	page des mentions légales de l'application	pour tout le monde
*	oui	-	page d'erreur 404	pour tous ceux qui se sont perdus
/403	oui	-	page d'erreur 403	pour tous ceux qui n'ont pas les droits

Nous avons simplement prévu à l'avance les urls que nous utiliserons. Les autres informations servent simplement à bien définir à quoi serviraient ces urls et la façon de les utiliser.

## Diagramme de Séquence pour la création d'un ticket :



Un diagramme de séquence représente les interactions entre plusieurs entités ou groupes, ce sont les acteurs.

Ici nous avons choisi de représenter la création d'un ticket et les entités sont l'utilisateur (le Client) et l'application (notre System). Une explication détaillée des symboles et composants est disponible sur [lucidchart](#).

Voici une rapide explication :

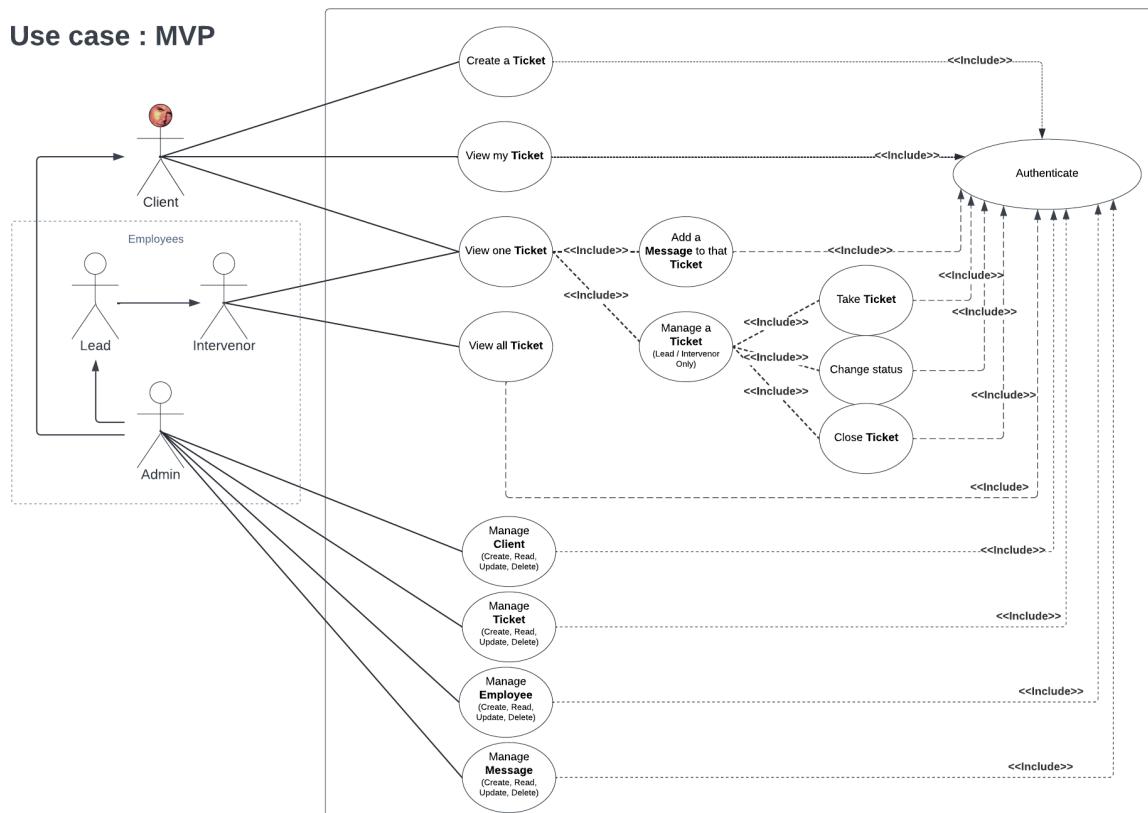
Les lignes qui partent des acteurs représentent le temps.

Les rectangles sur ces lignes de temps, représentent le temps nécessaire à une action.

Les flèches représentent des interactions synchrones ou asynchrones, selon si elles sont respectivement pleines ou en pointillés.

Les rectangles qui contiennent un “alt” en haut à gauche sont des scénarios alternatifs qui peuvent avoir plusieurs conditions. Ici nous avons mis “KO” pour représenter qu'il y a eu un problème et “OK” pour quand tout se passe comme prévu et que la suite des événements est possible.

## Cas d'utilisation (Use Case) pour le MVP :



Ce schéma est la représentation des différentes interactions possibles entre les utilisateurs et notre application. Ici nous n'avons mis que les fonctionnalités prévues dans notre MVP. Sur la gauche nous avons les utilisateurs possibles. Les flèches représentent l'héritage, il faut comprendre qu'un Admin peut également faire ce qu'un Lead peut faire. Un lead peut faire ce qu'un Intervenor peut faire, donc un Admin peut faire ce qu'un Intervenor peut faire. Les cercles représentent les objectifs, aller sur tel ou tel page. Tous ces objectifs sont reliés par une flèche à “Authenticate” car pour tous nos objectifs il est nécessaire de se connecter. On peut également voir qu'aller sur la page d'un ticket par exemple, donne la possibilité d'écrire un message sur ce ticket.

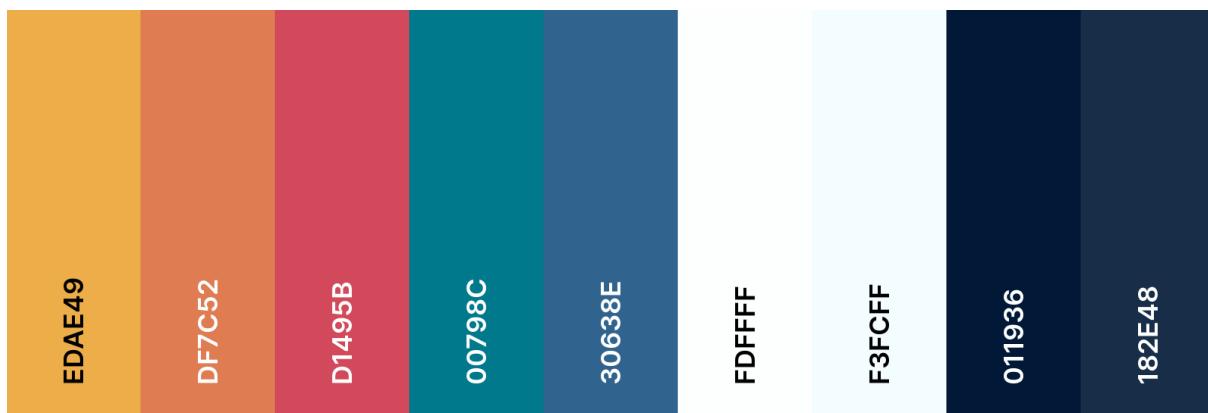
### 3d - Charte graphique :

#### Typographies :

Nous avons fait le choix d'utiliser deux polices sans serif pour une meilleure lisibilité :

- Roboto
- Montserrat

#### Couleurs :



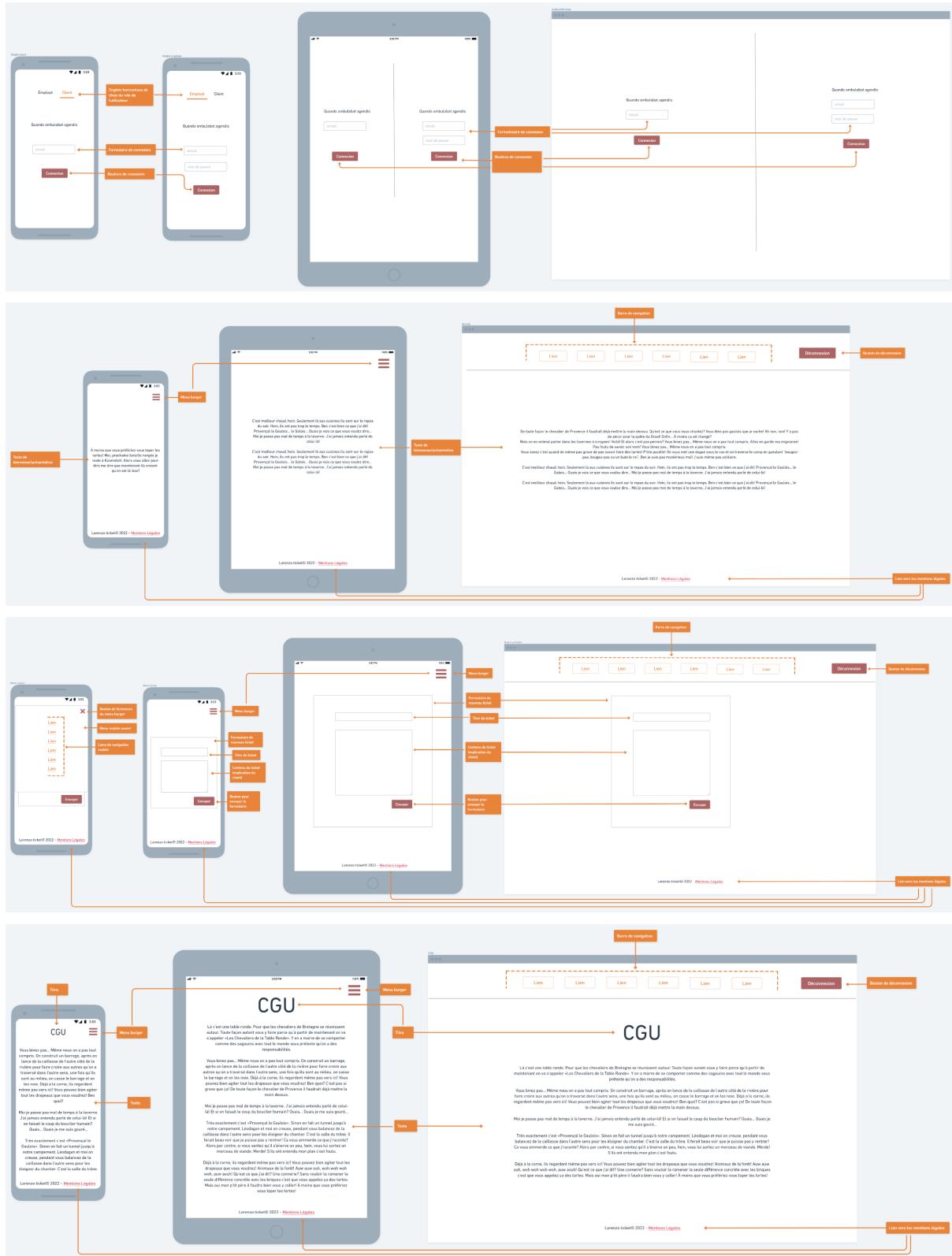
### 3e - Wireframes :

Des wireframes pour mobile, tablette et desktop ont été réalisées pour toutes les pages prévues initialement pour l'application en version MVP. En effet, nous n'avons pas inclus les fonctionnalités de recherche, filtres et pagination par exemple qui ont été jugées secondaires.

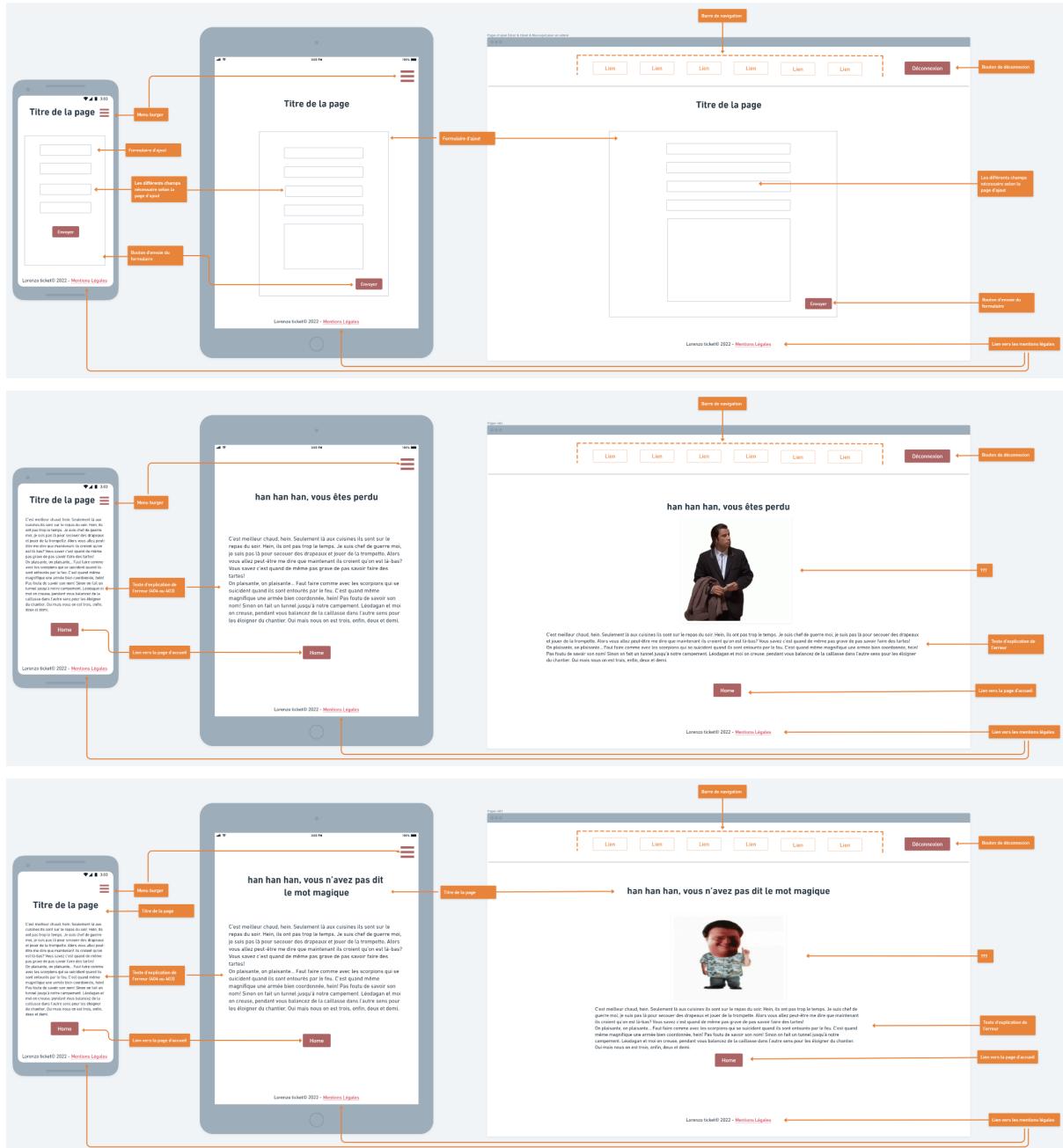
Les wireframes servent lors de la conception à définir l'interface utilisateur et les éléments qu'elle contient, leurs emplacements approximatifs ainsi que toutes informations nécessaires à la compréhension sous forme de légendes.

Le but étant de mettre "sur papier" ce qui a été fait précédemment afin que toute l'équipe ait la même vision du projet sans ambiguïté et ainsi faciliter le développement.

Au cours du développement, nous avons débattu sur les pages "listes" qui sont les pages de tous les messages, tous les clients, tous les employés et tous les tickets. Nous étions en train d'envisager une nouvelle approche pour la modification des informations en utilisant une modale ce qui demanderait de changer ces wireframes. Des informations complémentaires sur le sujet sont visibles sur un screenshot du ticket trello dans la section "Outils" de ce dossier.







## 4 - Spécifications techniques :

### 4a - Versionning :

Nous avons choisi d'utiliser la bibliothèque commitizen pour nommer nos commit. Cette bibliothèque, via ligne de commande, permet de rédiger et formater les messages de commit.

En effet, en utilisant la ligne de commande, une suite de choix nous sera proposée pour écrire un message avec un préfix en fonction du type du commit puis une description courte et enfin une description longue optionnelle. Bien sûr, tout est configurable, nous avons par exemple choisi une option qui ajoute un emoji en fonction du préfix et cela afin de comprendre d'un simple coup d'œil l'objectif du commit sans avoir à lire le message.

### 4b - Front :

Comme nous étions tous familier avec React, c'est donc naturellement que nous avons choisi de faire le front en l'utilisant.

Cependant, nous avons tout de même réfléchi aux conséquences de ce choix comme j'ai pu en parler plus tôt. Ainsi, nous avons estimé que cela offrait également des possibilités intéressantes pour l'utilisation de React Native si besoin.

React est également une bibliothèque avec une grande communauté ce qui offre une pseudo assurance sur sa sécurité et sa maintenabilité ainsi qu'une facilité à trouver des informations et de la documentation.

React est une bibliothèque JavaScript qui fonctionne sur le principe de Single Page Application qui permet de créer des interfaces utilisateurs dynamiques, en découplant les éléments de chaque page en "composants" écrit en JSX (JavaScript Syntaxe eXtension). Le JSX pourrait être décrit comme un mélange entre JavaScript et HTML.

Les données nécessaires pour l'affichage dynamique sont stockées dans le "state" et les composants se basent sur ce state et ses changements pour l'affichage en comparant le DOM (Document Object Model) avec un DOM virtuel créé par React qui permet de ne rafraîchir que les composants où les données ont été changées.

React Router Dom a été utilisé pour la gestion des routes.

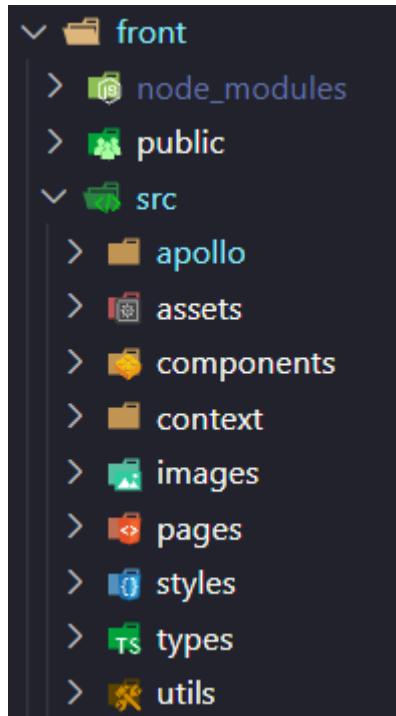
TypeScript a été utilisé afin de nous aider sur la détection préventive des erreurs liés aux types.

L'écosystème Apollo permet la création et la consommation d'API en GraphQL. Pour les requêtes à notre API, nous avons donc utilisé Apollo Client qui est fortement typé.

Jest a été utilisé comme framework de test avec l'aide de React Testing Library pour les tests spécifiques à l'interface utilisateur.

Sass a été utilisé pour la gestion et l'écriture du css.

## Organisation des dossiers et fichiers :



Pour le front, nous avons un dossier public qui contient l'index.html ainsi que le favicon.

Le dossier src contient lui :

- **apollo** : tous les fichiers liés aux requêtes côté front, nos mutations et queries ainsi que leurs types.
  - **assets** : les gif pour les pages 404 et 403, le dossier images aurait dû être déplacé dans ce dossier.
  - **components** : nos composants React qui ne sont pas les pages et les composants réutilisables. On y trouve notamment les composants Header, Footer et Message.
  - **context** : la mise en place du Context de React afin d'avoir accès aux informations de l'utilisateur partout.
  - **images** : les différentes images utilisées côté front.
- Aurait du être déplacé dans les assets.
- **pages** : les composants React qui servent de pages à notre site.
  - **styles** : les fichiers scss.
  - **types** : les interface et enum que nous avons créer pour l'utilisation de Typescript
  - **utils** : les méthodes qui ne sont pas liées

directement à un composant et peuvent être utilisées à plusieurs endroits. On y trouve par exemple la gestion du localStorage ou des fonctions pour la gestion des classes css.

## 4c - Back :

Nous avons décidé d'utiliser NodeJS afin d'avoir un environnement back en JS et donc être homogène sur tout le projet. Cela était également nécessaire, même si des alternatives existent, pour l'utilisation de GraphQL.

GraphQL est un langage de requête qui permet de customiser les requêtes et n'avoir que le nécessaire et ce sous forme d'objet. Nous voulions avoir cette souplesse possible qu'offre GraphQL par rapport à une API RESTful qui demande bien souvent de devoir faire plusieurs requêtes et nous fournit plus d'informations que nécessaire. Nous avions également à cœur de revoir cette technologie vue en cours.

Knex est une bibliothèque de construction de requêtes qui nous a permis de faire des requêtes SQL préparées, renforçant ainsi la sécurité et ce de façon simple.

JWT, JSON Web Token, nous a permis de renforcer la sécurité des requêtes à notre API.

Apollo Server a été utilisé pour l'implémentation de notre API en GraphQL.

La base de données a été faite en PostgreSQL qui est un prérequis pour une API en GraphQL.

PG nous a permis de connecter notre base de données PostgreSQL avec notre environnement NodeJS.

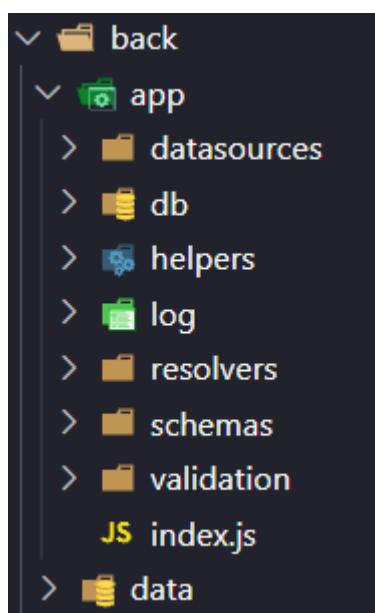
Bunyan est une bibliothèque de journalisation, elle nous a permis pendant le développement d'afficher des informations dans le terminal. Une utilisation en production est possible mais nécessiterait des modifications de la configuration.

Dataloader permet d'ajouter un système de cache à nos requêtes. Si la même requête a déjà été effectuée dans le laps de temps prédéfini, la réponse renvoyée sera celle qui a été mise en cache au lieu de refaire la requête. Cette fonctionnalité peut être désactivée au cas par cas si besoin.

## Organisation des dossiers et fichiers :

Pour le back, nous avons un dossier data qui contient les fichiers SQL pour la création, la structure et les données de développement de notre base de données.

Le dossier app contient lui :



- **datasources** : ce sont les fichiers contenant toutes nos requêtes à la base de données.
  - **db** : fichier de connexion à la base de données.
  - **helpers** : les fichiers pour le JWT et le logger.
  - **log** : les fichiers de log.
  - **resolvers** : les fichiers faisant la liaison entre les schémas et les datasources. Si un schéma contient des informations qui ne sont pas directement dans la table associée, il faut faire la liaison avec les datasources nécessaires. Les traitements supplémentaires à faire avant une requête sont également faits ici (exemple: validation des données).
    - **schémas** : les fichiers graphQL pour définir la "forme" de nos objets notamment nos entités de la base de données.
    - **validation** : les fichiers comportant les règles pour la validation des données.

## 4d - Déploiement :

Pour la mise en ligne de l'application, nous avons opté pour l'hébergeur o2switch, comme Fidia possédait déjà un hébergement chez eux.

Nous avons créé un fichier .htaccess à la racine du répertoire front, pour assurer la réécriture des url, une fois en environnement de production :

```
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteBase /subcategory
    RewriteRule ^index\.html$ - [L]
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteCond %{REQUEST_FILENAME} !-l
    RewriteRule . /index.html [L]
</IfModule>
```

Ensuite, nous avons choisi un domaine déjà existant et créé un sous-domaine qui serait alloué au projet.

The screenshot shows the o2switch web interface for managing domains. On the left, there's a sidebar with links: 'Espace Technique', 'Espace Client', and 'Documentation'. The main area has a header 'Sous-domaines' with a search bar and some icons. Below the header, there's a section titled 'Créer un sous-domaine' with fields for 'Sous-domaine' (containing 'lorenzo-tickets'), 'Domaine' (containing 'monevm.com'), and 'Racine du document' (containing '/'). A blue 'Créer' button is visible. Below this, there's a section titled 'Modifier un sous-domaine' with a table showing one row: 'api.lorenzo-tickets.monevm.com' under 'Sous-domaines', 'Atteindre /api.lorenzo-tickets.monevm.com/back' under 'Racine du document', 'not redirected' under 'Redirection', and 'Supprimer' and 'Gérer la redirection' under 'Actions'.

Une fois créé, nous lui avons installé un certificat SSL pour qu'il ait le protocole https et ainsi être sécurisé.

O2switch permettant de le faire facilement, nous avons donc procédé comme suit:

Gestion des Flux

Performance

Sécurité

SSH Accès SSH IP Bloqueur d'adresses IP SSL/TLS

Manage API Tokens ModSecurity SSL/TLS Status

Let's Encrypt™ SSL ImunifyAV

Logiciel

SitePad Website Builder Setup Node.js App Sélectionner une version de PHP

Setup Python App Setup Ruby App

Avancé

Terminal Tâches Cron Index

Pages d'erreur Gestionnaires Apache Types MIME

Showing 1 to 10 of 13 entries

Le renouvellement de vos certificats a été vérifié pour la dernière fois le 26 Jan 2023

Previous 1 2 Next

Générer un nouveau certificat

Choisissez parmi l'un de vos noms de domaine ci-dessous. Une nouvelle clé et un nouveau certificat seront rajoutés au gestionnaire SSL/TLS.

Show 10 entries

Nom de domaine Hôtes alternatifs Actions

lorenzo-tickets.monevm.com	lorenzo-tickets.monevm.com, www.lorenzo-tickets.monevm.com,	<a href="#">+ Générer</a>
----------------------------	---	---------------------------

Showing 11 to 17 of 17 entries

Previous 1 2 Next

Cliquez ici pour visiter le gestionnaire SSL/TLS de cPanel. Merci de noter qu'en cas de suppression d'une clé privée ou d'un certificat en utilisation, vous devrez procéder à une réinstallation par la fonction Let's Encrypt SSL.

Let's Encrypt™ is a trademark of the Internet Security Research Group. All rights reserved.

cPanel 106.0.14 Annuaire Mises à jour Documentation

## Let's Encrypt™ SSL

Le service Let's Encrypt a des limites sur le nombre de certificat SSL générable par domaine. Cliquez ici pour avoir plus d'information à ce sujet.

Merci d'indiquer tout hôte complémentaire à inclure au certificat

Installation du certificat pour : lorenzo-tickets.monevm.com

Nom de domaine	Type	Inclure ?	Inclure le Wildcard* ?	Ajouter les sous-domaines liés à cPanel
lorenzo-tickets.monevm.com	Sub	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/> (cpanel,webmail,webdisk,pccontacts,pccalendars)
www.lorenzo-tickets.monevm.com	Sub Alias	<input checked="" type="checkbox"/>	<input type="checkbox"/>	

\* Cette option désactive les emails d'alertes pour les renouvellements de certificat  
\*\* Les sous-domaines de cPanel sont les sous-domaines de la forme cpanel., webmail., qui permettent d'accéder à cPanel (ou au webmail) à partir de sous-domaines. Attention aux limites de Let's Encrypt

Choisissez une méthode de validation (toutes sont automatiques)

Par défaut, la méthode HTTP est recommandée. Si votre domaine utilise les serveurs DNS d'o2switch, vous pouvez aussi utiliser la méthode DNS.

Si vous souhaitez un certificat SSL wildcard, vous devez choisir la méthode de validation DNS.

http-01  dns-01

Veuillez choisir un type de clé de certificat (la valeur par défaut est recommandée)

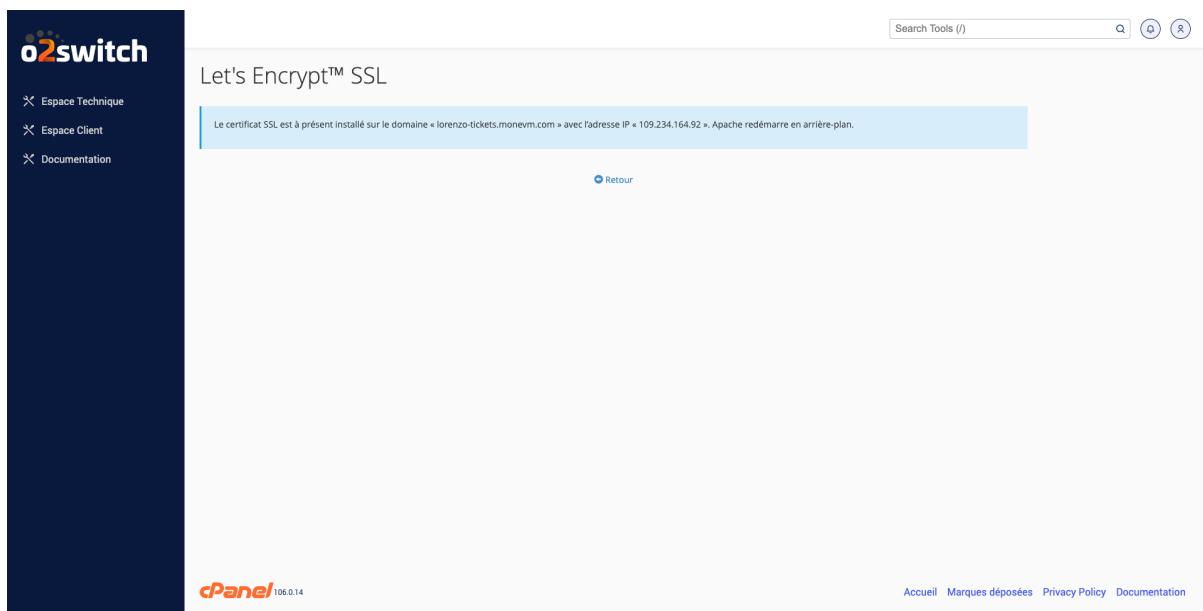
RSA 2048-bit

**Avertissement :** décochez les domaines se terminant par .o2switch.fr, .o2switch.net ou .universe.wf dans la liste des hôtes alternatifs, sinon cela va très probablement provoquer une erreur à cause des limites de Let's Encrypt.

[Générer](#) Si vous souhaitez annuler la génération

Si vous souhaitez réutiliser le certificat SSL d'un autre hôte virtuel, Cliquez ici.

Retour



Nous pouvions dorénavant, voir que notre domaine avait bien le protocole https:

Nom de domaine	Hôtes alternatifs	Etat	Validation	Expiration	Actions
lorenzo-tickets.monevm.com	www.lorenzo-tickets.monevm.com	Installé	http-01	04 May 2023	<a href="#">Supprimer</a> <a href="#">Réinstaller</a> <a href="#">Voir</a>
monevm.com					<a href="#">Voir</a>

Pour mettre le site en ligne, nous avions deux solutions. Soit en connexion FTP (File Transfer Protocol), soit directement en clonant le projet sur le serveur distant en ligne de commande.

Le build de l'application étant impossible à faire sur le serveur, faute de mémoire, nous avons opté pour le faire en FTP.

Nous avons fait pointer le server sur le dossier `build` du projet :

The screenshot shows the o2switch web interface for managing domains. On the left, there's a sidebar with links for 'Espace Technique', 'Espace Client', and 'Documentation'. The main area is titled 'Modifier un sous-domaine' (Edit sub-domain). It has tabs for 'Sous-domaines', 'Racine du document', 'Redirection', and 'Actions'. Below these tabs is a search bar and a 'Rechercher' button. The main content area is mostly empty, with a large gray placeholder. At the bottom, there's a status bar with the URL 'lorenzo-tickets.monevm.com', the status 'not redirected', and buttons for 'Supprimer' (Delete) and 'Gérer la redirection' (Manage redirection). A red circle highlights the URL and the redirection management button.

Puis, une fois connectés sur le serveur, via FileZilla, il nous a suffit de transférer, en glisser-déposer, le répertoire du projet directement sur le domaine concerné.

À gauche le serveur local et à droite le serveur distant.

The screenshot shows the FileZilla interface with two panes. The left pane, labeled 'Site local', shows the local file structure at '/Users/fidia/Sites/CDA/projet-08-gestion-de-tickets-client/front/'. The right pane, labeled 'Site distant', shows the remote file structure at '/lorenzo-tickets.monevm.com/front'. A red border surrounds both panes. The bottom status bar indicates '11 fichiers et 4 dossiers. Taille totale : 1606599 octets' on the left and '11 fichiers et 4 dossiers. Taille totale : 1606599 octets' on the right, along with other connection details.

Par la suite, à chaque modification de code, nous devions refaire un build de l'application et refaisions le transfert en FTP.

Nous avons procédé de la même manière pour le déploiement de l'api, sur un autre sous-domaine.

Nb : Cette partie a été écrite par toute l'équipe lors du déploiement en préparation des dossiers de projet.

## 5 - Organisation :

### 5a - L'équipe :

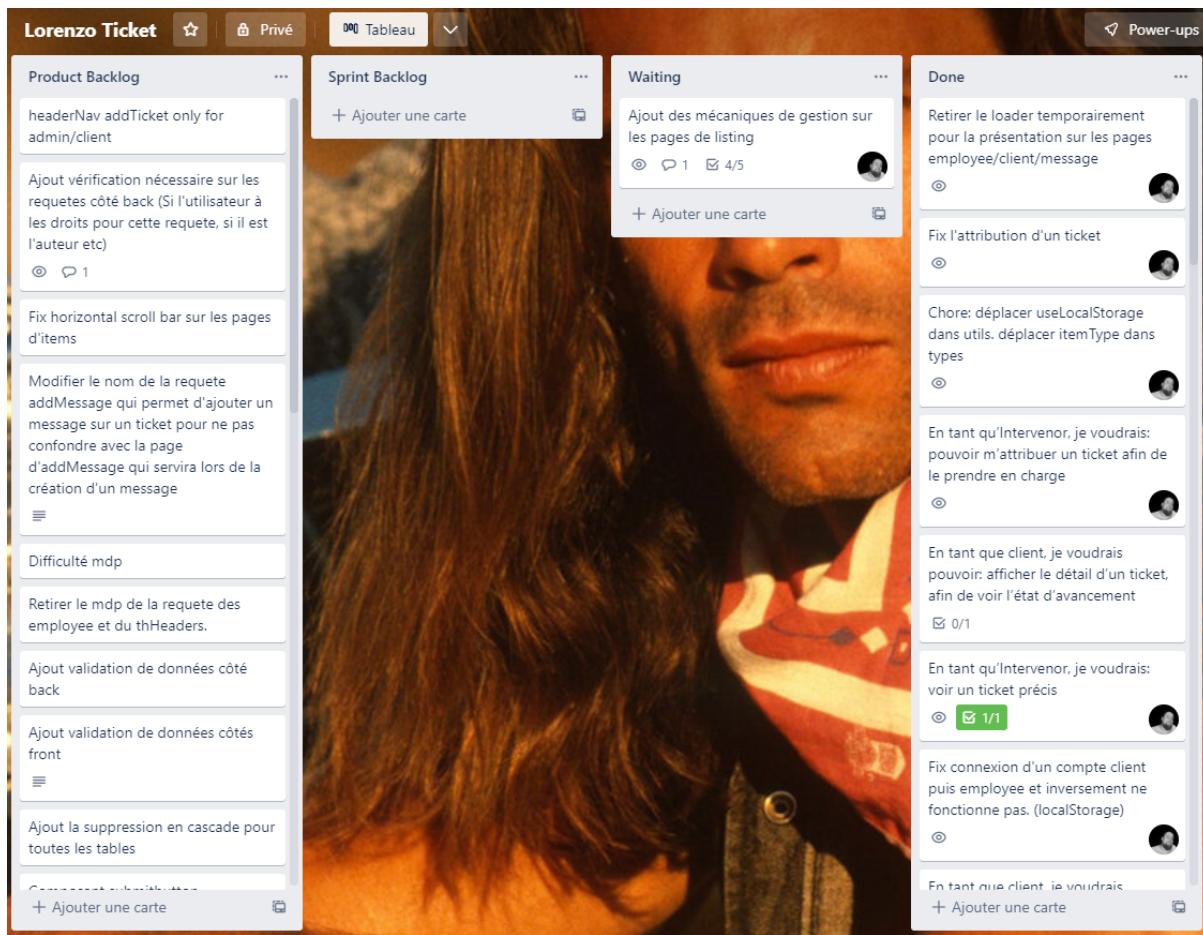
Nous étions trois personnes pour la réalisation de ce projet et avons choisis de tous travailler sur le fullstack lors du développement.

Comme nous étions une petite équipe, les rôles n'étaient pas absous, nous prenions l'avis de chacun et avons travaillé ensemble sur tous les aspects du projet, que ce soit la conception ou la réalisation du front, du back ou du style.

- El Bouanani Fidia, Lead Back
  - Ayant le plus d'expérience côté back end et ayant déjà travaillé avec GraphQL, Fidia a assumé le rôle de Lead Back et nous a apporté support et explications sur le sujet. En effet, contrairement à nous, elle avait déjà travaillé avec GraphQL en dehors des cours.
- Delisle Nicolas, Scrum Master
  - Il devait rédiger le journal de bord de l'équipe et y noter l'avancée du projet ainsi qu'animer les réunions d'équipe appelées daily scrum pour faciliter la transmission d'informations au sein du groupe de travail.
- Herbet Le Faucheur Tony, Lead Front et Git Master
  - Ayant plus d'expérience côté front-end, principalement concernant React, j'ai été choisi pour prendre le rôle de Lead Front. J'avais en charge le développement de la partie front-end et devait veiller à son bon fonctionnement ainsi qu'apporter soutien et conseil sur celle-ci.
  - Je me suis également occupé de la partie git, notamment les CI (Intégration Continue) et les conventions de commit.
- Product Owner et Designer, toute l'équipe.

### 5b - Outils :

- Nous avons utilisé git pour la gestion des versions du projets et la plateforme github pour héberger le code du projet.
- Pour aider à l'organisation du travail en équipe, nous avons utilisé l'outil en ligne Trello qui permet d'organiser et décomposer le projet en tâches.  
Les tâches peuvent être assignées à des membres du projet, elles peuvent avoir des étiquettes qui aide à visualiser le(s) domaine(s) auxquelles elles appartiennent, des commentaires et peuvent être déplacées selon leurs avancements et/ou la phase de développement suivant la méthodologie de travail utilisée (je reviendrais sur ce point plus bas).



Comme nous avons travaillé en étant ensemble sur discord, nous n'avons pas ressenti le besoin d'utiliser les étiquettes.

Nous avons commencé par ajouter les User Stories aux tâches à faire dans le Product Backlog et, quand cela était nécessaire, nous avons ajouté des checklist et des commentaires.

Nous avons également créé de nouvelles tâches lorsqu'un besoin ou bug était découvert.

*Exemple de checklist et commentaire :*

## 💻 Ajout des mécaniques de gestion sur les pages de listing

Dans la liste Waiting ⓘ

Membres



+

Notifications

ⓘ Suivie



### ☰ Description

Ajouter une description plus détaillée...

#### Redirection pour "voir"

Masquer les tâches cochées

Supprimer

80%

Redirection pour "voir"

Modifier la table en input pour "modifier" + requêtes

Requête pour "supprimer"

Page d'un ticket

Ajout message sur la page d'un ticket

Ajouter un élément

### 🕒 Activité

Afficher les détails



Écrivez un commentaire...



**Tony Herbet Le faucheur** 31 oct. 2022 à 10:13 (modifié)

Discussion sur la modification d'un item, choix:

- modale
- input dans la Table

Le comportement devrait être mutualisé avec celui de la création d'un item ce qui demanderait de changer le workflow ou son implémentation (page Add deviendrait Add/Update).

Nécessitera une maj screenshot du workflow

- Comme annoncé plus tôt, nous avons également utilisé discord pour communiquer en vocal et à l'écrit pour garder une trace de nos échanges.
- Pour le partage et la centralisation des fichiers, nous avons utilisé google drive
- Les schémas ont été réalisés sur différents outils :
  - Lucidchart pour les UML
  - Excalidraw pour faire des dessins lors des discussions et la réalisation des schéma du Workflow et de l'Architecture
  - Whimsical pour les wireframes
- Pour créer notre palette de couleurs, nous avons utilisé le site Coolors.

## 5c - Méthodologie :

Nous avons travaillé selon la méthodologie Scrum qui se base sur les principes de la méthode Agile.

Elle consiste à diviser le développement du projet en “sprint” qui peuvent avoir une durée variable.

Pour l’Apothéose, nous avons décidé que la durée d’un sprint serait d’une semaine de “cours” donc de deux jours. Cependant, de temps en temps, nous avons travaillé le week-end ou lors de nos congés ce qui a rallongé certains sprint.

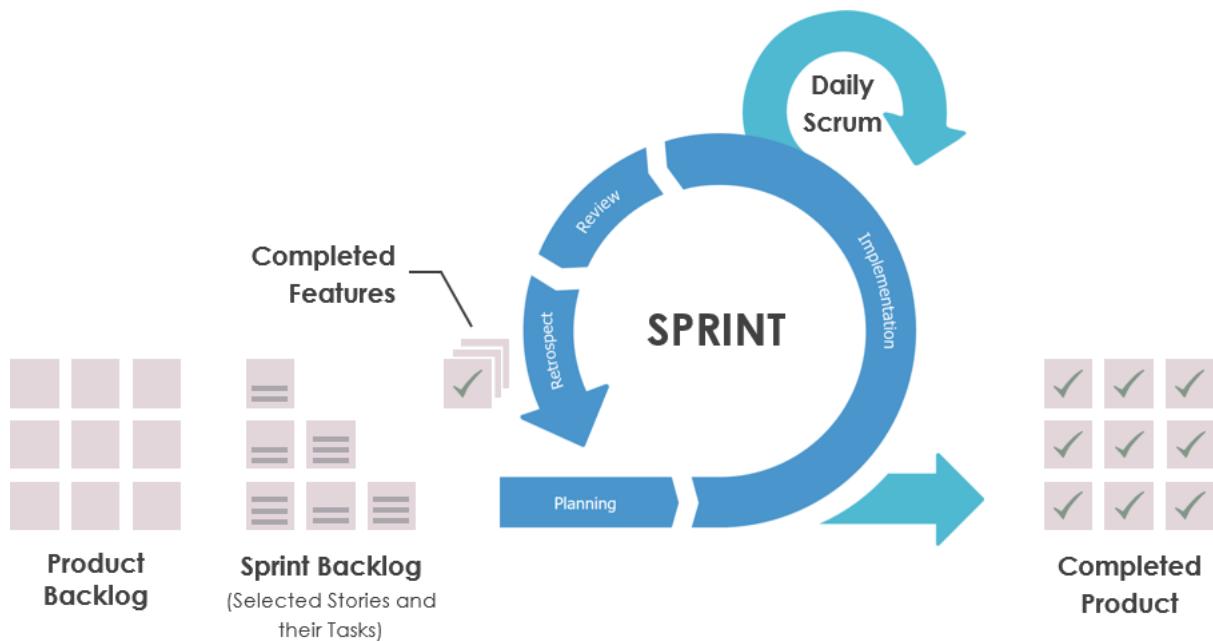
Le but de cette méthodologie est de travailler de concert avec le client, en l’impliquant dans chaque évolution du projet et en accueillant ses remarques et demandes de modification de façon positive. Cela a pour effet d’avoir un projet qui évolue rapidement et régulièrement en ajoutant des fonctionnalités à grande valeur ajoutée.

Lors de notre première réunion, nous avons défini toutes les tâches que nous aurions à réaliser lors du développement, le Product backlog. Il est amené à changer au fur et à mesure de l'avancement du développement et des demandes reçues.

Au début de chaque sprint, nous avions un Sprint Planning pour définir les objectifs du sprint et remplir sur le trello le Sprint Backlog via les tâches du Product Backlog.

Tous les matins, nous avions une réunion appelée Daily Scrum, animée par le Scrum Master afin de transmettre toutes les informations nécessaires au bon déroulement du projet. Nous discutions des tâches que chacun allait effectuer dans la journée, des problèmes que nous avions rencontrés la veille, demander de l'aide ou poser des questions si nécessaire.

Au vu du rythme de travail, nous avons estimé qu'il n'était pas nécessaire de faire de Sprint review ou Sprint retrospective car l'alternance entre cours et travail nous demandait un temps de réadaptation entre les deux et nous discutions déjà des problèmes et amélioration possible lors du Sprint Planning.



Nous avons fait huit petits sprints plus un “bonus” :

- Du sprint 1 à 3 nous avons travaillé sur la conception et à la réalisation du cahier des charges ainsi que les conventions.
- Sprint 4 à 7, développement du front et du back.
- Sprint 8 et bonus, finition des tâches commencé et peaufinage de l'existant.

Une journée de travail type commençait à 9h par un Daily Scrum pour l'organisation de la journée.

Une pause déjeuner d'environ une heure avait lieu aux alentours de midi puis nous continuons de coder en étant sur discord jusqu'à 17h.

Avant de se déconnecter, nous faisions une dernière réunion pour revenir très rapidement sur la journée et annoncer le cas échéant ce que nous allions faire dans la soirée afin que tout le monde puisse travailler sur quelque chose de différent.

C'est souvent lors de ces heures de travail en soirée ou le week-end que nous testions des designs, modifications ou fonctionnalités qui n'avaient jusqu'alors pas été envisagés et que nous présentions lors du prochain Daily scrum.

Nous avons également fait le choix de faire beaucoup de pair programming. Bien que nous étions une petite équipe de trois, plus que de “finir” le projet, nous voulions apprendre et approfondir nos connaissances sur les différentes technologies.

## 5d - Conventions et règles :

Voici les différentes conventions et règles que nous avons essayées au mieux de respecter lors du développement :

Git :

- Ne pas coder sur la branche main.
- Créer une branche pour chaque tâche, puis faire une pull request.
- Faire un seul commit par PR.
- Utilisation de [git-cz](#) pour une uniformisation des messages de commit.
- Review des autres développeur obligatoire avant de pouvoir merge une PR.
- Uniformisation du projet Git par des choix stylistiques et structurels.
- Fragmentation des tâches par branches avec un ticket ou issue pour chaque tâche.
- L'organisation de la documentation du projet permet à un nouvel acteur de trouver plus facilement des informations pertinentes du projet ( README.md ; INSTALL.md ; CONVENTIONS.md ; LICENSE.md ; etc. ).
- La branche “MAIN” est la branche principale sur laquelle nous allons merge.
- Depuis la branche “MAIN”, créer une branche “RELEASE/VX.X” pour chaque version qui sera déployée.

Code :

- Écrire le code en ANGLAIS
- Écrire les commentaires en FRANÇAIS
- Utilisation d'un Linter :
  - Un linter est un outil qui analyse statiquement du code et vérifie que celui-ci respecte un certain nombre de règles.
  - Intérêt du linter :
    - Vous assurer de la constance du code, qu'il s'agisse de bonnes pratiques ou de considérations plus esthétiques.

- Être toujours à jour sans avoir à faire d'effort, les mises à jour du Linter prenant en considération les évolutions de bonnes pratiques de développement.
  - En cas d'erreur de syntaxe dans votre code, l'analyse statique du Linter échouera et l'erreur en question vous sera remontée : c'est un garde fou supplémentaire.
  
- Convention de nommage :
  - Les variables du type Boolean et les fonctions retournant un Booléen doivent commencer par "is", "has" ou "should".
  - ex: const isComplete = true; // const hasMessages = data.length > 0.
  - Utiliser le PascalCase pour les noms de fichiers(ex: Login.js).
  - Utiliser les noms de dossiers comme nom de composant.
  - Les événements React sont nommés à l'aide de camelCase et commencent par "on" (ex: onSubmitButtonClick").
  - Les gestionnaires d'événements doivent commencer par "handle"(ex: handleSubmitButtonClicked).
  
- Les commentaires et documentation du code source.
  
- Utilisation de TypeScript :
  - L'utilisation de TypeScript améliore la qualité du code et permet d'identifier d'éventuelles erreurs dans notre code en amont.
  
- DRY : factoriser et simplifier tant que possible.
  - Si vous faites la même chose à plusieurs endroits, consolidez votre code et supprimer les duplications.
  
- La déstructuration :
  - La déstructuration est à prioriser pour faciliter l'accès direct à un élément de tableau plutôt que par sa position ou sa clé.
  
- Utilisation de Jest :
  - Pour les mêmes raisons que nous utiliserons TypeScript, nous utiliserons Jest afin d'améliorer la qualité du code en écrivant des tests pour détecter les erreurs le plus rapidement possible.

## 5e - Github :

En ayant mon rôle de Git Master en tête, j'ai décidé de créer un fichier avec quelques informations utiles à propos de git et github, notamment les lignes de commandes qui pourraient être utilisées sur le projet et la configuration de github que j'ai mise en place.

### Que faire pour commit ? :

Si c'est le premier commit, utiliser "git cz" (depuis la racine du projet)

Si ce n'est pas le premier commit, il faut utiliser "amend"

Après avoir push, si la branche de la PR est en retard sur la branche main il faut "rebase"

Avant de faire une nouvelle branche, toujours aller sur main et faire un "git pull"

### Rebase :

1. git checkout <main>
2. git pull
3. git checkout <branch>
4. git rebase <main>
  - 4.1. si il y a des conflits, les résoudre puis git add .
  - 4.2 git rebase --continue
5. git push <remote> <branch> --force

Le nom du <remote> de base est *origin*

<main> est la branche principale

<branch> la branche sur laquelle vous travaillez

S'il y a un problème avec la résolution des conflits lors d'un rebase, il est possible d'annuler le rebase avec un "git rebase --abort".

Si la résolution des conflits est possible, la faire puis "git add ." suivi d'un "git rebase --continue"

### Reset last commit :

Reset le dernier commit non push et remet le commit dans les changements en cours.

"git reset --soft HEAD~1"

### Squash :

1. Prendre le SHA du premier commit de la branche
2. git rebase -i SHA
3. écrire squash devant les commits que je veux squasher

4. git push <remote> <branch> --force

Delete branch :

local:

git branch -D <branch>

remote:

git push <remote> --delete <branch>

Amend :

Change le dernier commit et y ajoute les nouvelles modifications sans changer le message

1. git add .
2. git commit --amend --no-edit
3. git push --force

Possible d'utiliser *-m "a commit message"* à la place de *--no-edit* pour ajouter un nouveau message de commit (git --amend -m "a commit message")

Stash:

<https://git-scm.com/docs/git-stash/fr>

- git stash

Retire tous les changements et les mets en mémoire

- git stash list

Liste tous les stash en mémoire

- git stash pop stash@{i}

Applique les changements disponibles dans le stash et les supprime de la mémoire.  
*stash@{i}* est optionnel, s'il n'est pas utilisé la commande sera faite sur le dernier stash

- git stash apply stash@{i}

Même fonctionnement que *pop* mais le stash n'est pas supprimé de la mémoire. (A utiliser de préférence)

- git stash clear

Supprime tous les stash de la mémoire

Github :

Dans Settings -> Branches -> Branch protection rules:

Sur main:

- Require a pull request before merging:  
When enabled, all commits must be made to a non-protected branch and submitted via a pull request before they can be merged into a branch that matches this rule.
- Require approvals (1)  
When enabled, pull requests targeting a matching branch require a number of approvals and no changes requested before they can be merged.
- Require conversation resolution before merging  
When enabled, all conversations on code must be resolved before a pull request can be merged into a branch that matches this rule.
- Do not allow bypassing the above settings  
The above settings will apply to administrators and custom roles with the "bypass branch protections" permission.

## 6 - Réalisations :

### 6a - Journal de bord :

Voici mon journal de bord pour ce projet, j'y ai noté un résumé de ce que j'ai pu faire chaque jour.

#### Journée 1 - Sprint 0

Cockpit (le cockpit est le site de cours en ligne que nous utilisons chez O'clock) explication de l'apothéose le matin.

Après midi discussion et débat sur la conception, choix des technos.

Début du cahier des charges et définition du MVP et de ses User stories

#### Journée 2 - Sprint 0

Nous avons continué de débattre sur les technos, nous avons validé l'utilisation de React et WP.

Discussion des User Stories et affinement de celles-ci, changement du système de connexion pour les "clients".

Réalisation du MCD

Avancement sur l'écriture du cahier des charges

## Journée 3 - Sprint 1

Retour en arrière sur le choix de WP (pas adapté au CDA après discussion avec Fanny)

Journée UML, nous avons fait :

- Use Case Diagram
- System Sequence Diagram
- Entity Relationship Diagram
- Physical Data Model (MPD en fr)

Modification des User Stories

## Journée 4 - Sprint 1

Discussion des choix possibles

Réalisation :

- Workflow
- Diagramme de navigation
- Refonte MCD
- Débuts des wireframes

## Journée 5 - Sprint 2

Rédaction des conventions.

Discussion/veille et réalisation :

- Finition des wireframes
- Schéma de l'architecture
- Mise à jour du Workflow

Discussion sur les technos et bibliothèque

## Journée 6 - Sprint 2

Choix des technos et bibliothèque et explication du choix

Discussion autour de l'analyse de risque

Liste des routes

Renommage sur les différents schéma de User en Employee

Ajout dans le workflow des pages manquantes (404 & 403)

Charte graphique (choix typographies, discussion sur les couleurs)

## Journée 7 - Sprint 3

Ajout du gros des tickets sur Trello

Création du repo github et début de l'installation des dépendances

Rédaction d'un document Git avec les lignes de commandes qui seront sûrement nécessaire

Configuration de github pour les Pull Requests et ajout des règles dans le document Git  
Création du premier composant React et résolution des problèmes de config  
ESLint/Prettier/TypeScript

## Journée 8 - Sprint 3

Ajout des bibliothèques et de la config pour les tests.  
Ajout d'un petit test pour vérifier.  
Ajout de CI pour lancer les tests au push.  
Ajout des bibliothèques et configs pour eslint/typescript pour voir les erreurs.  
Ajout d'un parsing eslint dans les CI au push.  
Discussion sur l'architecture.

## Journée 9 - Sprint 4

Sprint planning  
Explication du pattern scss par Fidia  
Ajouts des composants pour les pages et leurs routes  
Ajouts du Header et du Footer avec la base du css et la navigation

## Journée 10 - Sprint 4

Review et discussion sur le code  
Recherche sur le .env  
Pair programming sur la structure côté back et les schémas/types/scalars et debug  
Pair programming côté front sur la page de connexion, les composants nécessaire, le state et le début du css

## Journée 11 - Sprint 5

Review  
Réécriture des tests pour le composants de Connexion et refactorisation  
Page des tickets, ajout d'un composant Table qui sera réutilisable pour les autres tables (Messages, Clients, Users)

## Journée 11 - Sprint 5

Review  
Aide debug côté back  
Cockpit explication TP et fin d'apothéose  
Ajout de commentaire dans le code  
Modification de types  
Base page messages

Base page clients  
Fix type et self closing FieldLongText  
Ajout texte et titre sur la page d'accueil  
Page 404 contenu  
Page 403 contenu

## Journée 12 - Sprint 6

Absent

## Journée 13 - Sprint 6

Debug eslint/typescript  
Mob programming sur la connexion d'un employé  
Tentative de debug des tests  
Début de debug de la connexion qui ne fonctionnait plus

## Journée 14 - Sprint 7

Debug de la semaine dernière sur la connexion d'un employée  
Debug des tests du composant Connexion  
Refactor de signin pour l'utiliser en client ou employée  
Gestion connexion côté front pour le client/employée  
"Redirection" pour les pages 404 & 403  
Gestion de la déconnexion  
Affichage conditionnel du header selon le rôle

## Journée 15 - Sprint 7

Rename ctx employee côté back en user  
Génération des types depuis les schémas  
Ajout condition sur les requêtes back si le token est valide  
Fix envoie du token côté front (soucis condition causé par une proposition quick fix de l'ide)  
Ajout requêtes "tous les tickets" et "tous les tickets d'un client" côté front  
Modification du composant Table pour correspondre aux différences entre le mock et ce que renvoie le back  
Affichage conditionnel des boutons sur le composant Table en fonction du type d'utilisateur

## Journée 16 - Sprint 7

Nettoyage du localstorage au montage de la page de connexion afin d'éviter les soucis s'il n'y a pas eu de déconnexion.

Modification du schéma pour Message et ajout de resolver pour avoir les informations nécessaire à l'affichage des messages pour la page d'un ticket.

Génération des nouveaux types.

Nouveau composant Message pour afficher tous les messages sur la page d'un ticket

Page d'un ticket fini

Requête back createMessage et implémentation côté front.

Ajout de moment et react moment pour la gestion des dates et utilisation sur la page de ticket.

Fonctionnalité suppression d'un ticket côté back & front

## Journée 17 - Sprint 7

Ajout d'un enum ItemType qui servira sur les pages d'affichages à :

- faire une condition pour le bouton pour "voir" un item
- définir l'url param utilisé par le bouton "voir" selon l'item (ticket ou message)
- utiliser les bonnes variables pour delete

Grosse peur au moment de rebase, des erreurs en rapport avec le cherry pick (que je n'avais pas utilisé), j'ai tout perdu en local et la PR a également disparu. Le souci a finalement été réglé (plus d'informations dans la partie Recherches de ce dossier).

## Journée 18 - Sprint 8

Discussion sur la modification d'un item, choix :

- modale
- input dans la Table

Le comportement devrait être mutualisé avec celui de la création d'un item ce qui demanderait de changer le workflow ou son implémentation (page Add deviendrait Add/Update).

Discussion sur l'affichage du mot de passe pour un admin, à ne pas renvoyer lors de la requête et à retirer du thHeaders.

Aide debug update profil (modification de la requête et du return pour avoir le nouveau Employee directement)

Ajout prise en charge d'un ticket par un employee (back & front) et un peu de css

Chore: déplacer les types dans le bon fichier, déplacer hook dans utils

## Journée 19 - Sprint bonus

Ajout de validation pour le changement de mot de passe côté front et back.

Ajout de messages pour afficher une erreur ou un succès en fonction du résultat.

## 6b - Tests :

Pour ce projet j'ai mis en place une CI (Intégration Continue) en utilisant les Github Actions. Github cherche si un dossier ".github/workflow" existe et exécute les actions en fonction des déclencheurs.



```
1 # This is a basic workflow to help you get started with Actions
2 name: CI
3 # Controls when the workflow will run
4 on: push
5 # A workflow run is made up of one or more jobs that can run sequentially or in parallel
6 jobs:
7   # This workflow contains a single job called "build"
8   build:
9     # The type of runner that the job will run on
10    runs-on: ubuntu-latest
11    # Steps represent a sequence of tasks that will be executed as part of the job
12    steps:
13      # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
14      - uses: actions/checkout@v3
15      # Runs a single command using the runners shell
16      - name: Install modules inside front folder
17        run: cd front && npm ci
18      # Runs a set of commands using the runners shell
19      - name: Run tests inside front folder
20        run: cd front && npm test
21      # Runs a set of commands using the runners shell
22      - name: Run eslint inside front folder
23        run: cd front && npm run lint
```

Dans mon fichier CI, j'ai défini l'action "on" à "push" ce qui permet de déclencher les actions lorsqu'un push est détecté. Le job aurait dû être renommé car ce n'est pas un "build" mais un job pour lancer les tests et un check du linter (j'ai remarqué cette erreur lors de la rédaction de ce dossier).

Voici les différentes étapes importantes (elles commencent toutes par "cibler le dossier front") :

- Installation des modules
- Exécuter tous les tests
- Exécuter un check du linter

Ces actions permettent donc lors d'un push de vérifier que les tests passent toujours et qu'il n'y a pas d'erreurs de linter.

Cela nous assure que le nouveau code de la PR ne contient pas de régression par rapport aux parties du code déjà présent sur la branche principale pour lesquelles les tests sont écrits. Cela permet aussi de vérifier que les conventions configurées sur le linter sont respectées.

Voici un exemple de test qui a été mis en place :

```
● ● ●

1 describe('Connection', () => {
2   it('is an employee', () => {
3     render(
4       <ApolloProvider client={apolloClient}>
5         <Router>
6           <Routes>
7             <Route path="/" element={<Connection logout={jest.fn()} />} />
8           </Routes>
9         </Router>
10      </ApolloProvider>
11    );
12
13   const buttons = screen.getAllByRole('button');
14   const clientButton = buttons[0];
15   const employeeButton = buttons[1];
16
17   fireEvent.click(employeeButton);
18
19   expect(screen.getByText(employeeText)).toBeInTheDocument();
20   expect(screen.queryByText(clientText)).not.toBeInTheDocument();
21   expect(screen.queryByText('Mot de passe')).toBeInTheDocument();
22
23   expect(employeeButton).toHaveClass(selected);
24   expect(employeeButton).not.toHaveClass(notSelected);
25   expect(clientButton).not.toHaveClass(selected);
26   expect(clientButton).toHaveClass(notSelected);
27 });
28 });
```

● ● ●

```
1 export enum ButtonsClassNames {
2   selected = 'role-choice-button-selected',
3   notSelected = 'role-choice-button',
4 }
5 export enum RoleText {
6   clientText = 'Vous êtes un client?',
7   employeeText = 'Vous êtes un employé(e)?',
8 }
```

d'éviter des régressions lors du développement et des modifications de la page de connexion.

En effet grâce au CI, nous avons tout de suite vu que les tests ne passaient plus et avons pu modifier notre code en conséquence.

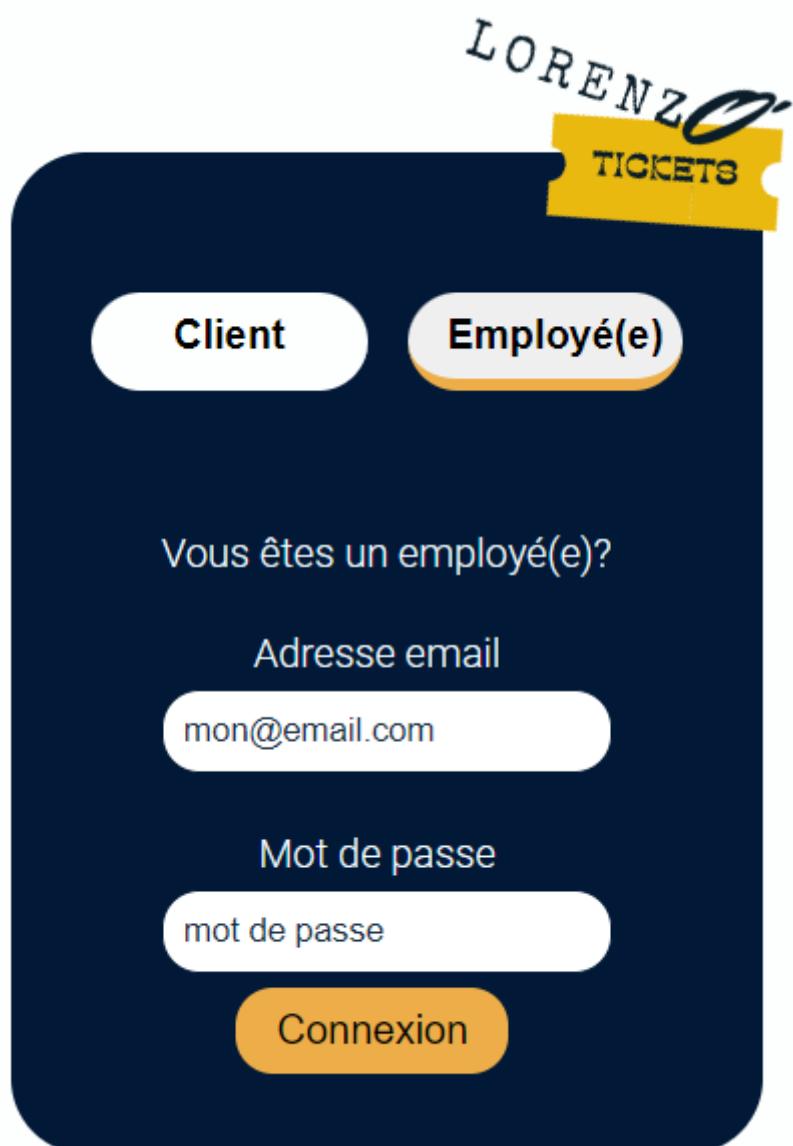
Ce type de tests permettrait également lors d'une évolution de l'application pour l'internationalisation de vérifier que les textes sont bien ceux prévus.

En utilisant Jest et React Testing Library, j'ai pu réaliser des tests sur l'affichage.

Ce test simule l'affichage de la page de connexion puis un clique sur le bouton "Employé(e)" et vérifie ensuite que le texte présent est bien celui voulu et que les classes css ont bien été modifiées.

Bien que ce test ne soit pas pour un élément critique de l'application, il nous a permis

Image de la page de connexion :



## 6c - Validation des données :

Lors d'un changement de mot de passe par un employé, une validation est effectuée côté front et back. La validation des données envoyées par l'utilisateur est importante car nous ne pouvons jamais lui faire confiance. Cette démarche de validation devrait être faite chaque fois que l'utilisateur envoie des données.



```
1 const handleSubmit = (event: FormEvent): void => {
2   event.preventDefault();
3   // Mot de passe validation
4   if (confirmNewPassword !== newPassword) {
5     setNewPassword('');
6     setConfirmNewPassword('');
7     setErrorMessage('Vos mot de passe ne correspondent pas');
8   } else if (confirmNewPassword === newPassword) {
9     // Entre 8 et 128 caractères
10    const hasRightLength = newPassword.length >= 8 && newPassword.length <= 128;
11    // A une lettre majuscule
12    const hasUpperCase = /[A-Z]/.test(newPassword);
13    // A une lettre minuscule
14    const hasLowerCase = /[a-z]/.test(newPassword);
15    // A un chiffre
16    const hasNumbers = /\d/.test(newPassword);
17    // A un caractère spécial
18    const hasNonAlphas = /\W/.test(newPassword);
19
20    if (hasRightLength && hasUpperCase && hasLowerCase && hasNumbers && hasNonAlphas) {
21      // Mot de passe valide, requête envoyé
22      void updatePassword();
23    } else {
24      setNewPassword('');
25      setConfirmNewPassword('');
26      setErrorMessage(
27        'Votre mot de passe doit être entre 8 et 128 caractères de long et doit contenir une majuscule, une
28        minuscule, un chiffre et un caractère spécial'
29      );
30    }
}
```

Côté front, lorsque le formulaire de changement de mot de passe est envoyé, plusieurs vérification sont faites.

Premièrement, une comparaison entre les deux champs de mot de passe qui ont été remplis est faite pour s'assurer que le mot de passe est bien le même et celui souhaité.

Si ce n'est pas le cas, un message d'erreur adapté est affiché et les champs sont vidés.

Si cette première vérification passe, on teste si le mot de passe est "fort" et suit bien les règles imposées. Tout comme pour la première vérification, si ce n'est pas le cas, on vide les champs et affiche un message d'erreur adapté.

Si la vérification passe, on fait notre requête au back.



```
1 // Ce trouve dans le fichier de validation
2
3 // Schema pour la validation du changement de mot de passe d'un employee
4 const schemaUpdateEmployeePassword = Joi.object({
5   password: Joi.string().min(8).max(128).regex(/[A-Z]/).regex(/[a-z]/).regex(/\d/).regex(/\W/).required(),
6 });
7
8 // Ce trouve dans le fichier de mutation
9 async updateEmployeePassword(_, args, { dataSources, user }) {
10   if (!user) {
11     throw new AuthenticationError('Vous devez être connecté pour supprimer un ticket');
12   }
13
14   const data = args.input;
15   const password = data.password;
16
17   // Joi validation des données
18   const validatedPassword = schemaUpdateEmployeePassword.validate({ password });
19
20   // Si la validation ne passe pas
21   if (validatedPassword.error !== undefined) {
22     throw new Error('Non valid password');
23   }
24
25   // Si la validation passe on hash le mot de passe et le met à jour dans la bdd
26   const employeeNewPwdCrypt = await bcrypt.hash(password, 10);
27
28   const updatedPasswordInput = { ...data, password: employeeNewPwdCrypt };
29
30   const response = await dataSources.employee.update(args.id, updatedPasswordInput);
31
32   return response;
33 },
34
35 // Ce trouve dans le fichier sql
36 async update(id, inputData) {
37   const result = await this.knex(this.tableName)
38     .connection(this.establishedConnection)
39     .where({ id })
40     .update({ ...inputData, updated_at: new Date() })
41     .returning('*');
42
43   return result[0];
44 }
```

Côté back, nous utilisons la bibliothèque Joi afin de valider les données. Joi permet de définir des contraintes à respecter pour nos objets.

Pour cet exemple de changement de mot de passe, nous avons dans le fichier de validation, son schéma Joi qui permet de définir les contraintes.

Ces contraintes sont les mêmes que côté front (lignes 4 à 6).

Dans la requête qui se trouve dans le fichier de mutation, nous avons une première vérification si l'utilisateur est bien connecté (lignes 10 à 12).

Puis, via le schéma de validation, on vérifie que le nouveau mot de passe respecte bien les contraintes prévues. Si ce n'est pas le cas, on renvoie un message d'erreur (lignes 14 à 23).

Si le mot de passe est validé, on le hash avant de faire notre requête afin qu'il ne soit pas en "clair" dans notre base de données (lignes 25 à 32).

La requête préparée est ensuite effectuée dans le fichier sql (lignes 35 à 44).

## 6d - Explication d'une fonctionnalité :

Je vais vous expliquer une des fonctionnalités que j'ai réalisée, le système de message sur un ticket.

Je ne dirais pas que cette fonctionnalité est principale, la création d'un ticket ayant plus d'importance mais c'est une fonctionnalité que j'ai principalement réalisée seul (contrairement à la majorité du projet sur laquelle nous avons travaillé en pair programming) et cela me permettra également d'aborder la page d'un ticket.

The screenshot shows a dark-themed web application interface. At the top, there is a navigation bar with links for "Accueil", "Créer un ticket", "Tickets", and a "Déconnexion" button. The main title "Un ticket de test pour la démo des messages" is displayed prominently. Below the title, ticket details are listed: "Numéro du ticket: 18", "Bénéficiaire: client1@test.test", "Entreprise: Company 1", "Statut: ouvert", "Prise en charge:", "Date de création: 30/01/2023 à 18:40", and "Description: La description". A large central area titled "Messages:" contains a text input field labeled "Ajouter un message" and an "Envoyer" button. At the bottom, a footer bar includes the text "LorenzO'tickets © 2023", links for "CGU" and "Mentions légales", and a note stating: "Le lien de la page d'un ticket est disponible sur la page Tickets. On y accède automatiquement via une redirection après la création du ticket." There are also three small colored circles (red, yellow, green) at the very bottom left.

```
1 <Route path="/ticket/:id" element={user?.logged ? <Ticket /> : <Error403 />} />
```

Notre composant Ticket, qui est le composant React pour la page d'un ticket, est affiché uniquement si l'utilisateur est connecté sinon la page 403 est affichée à la place.

1 / 3

```
1 const Ticket: FunctionComponent = () => {
2   const { id } = useParams();
3   const idAsNumber = Number(id);
4
5   const { user } = useUserContext();
6
7   (...)

8
9   const [triggerGetTicketById] = useLazyQuery<GetTicketById, GetTicketByIdVariables>(GET_TICKET_BY_ID, {
10     variables: { ticketId: idAsNumber },
11     fetchPolicy: 'no-cache', // Pas de cache afin que la page soit à jour lors de l'envoie d'un nouveau message
12     onCompleted: data => {
13       if (data !== null) {
14         const { getTicketById } = data;
15         // On met à jour le ticket
16         setTicket(getTicketById as GetTicketById_getTicketById);
17         // On met à jour l'info si l'employee traite le ticket
18         if (
19           (getTicketById as GetTicketById_getTicketById).employees.find(employee => employee.email ===
user.email) !=
20           null
21         ) {
22           setIsEmployeeHandlingTicket(true);
23         } else {
24           setIsEmployeeHandlingTicket(false);
25         }
26       }
27       setLoading(false);
28     },
29     onError: error => {
30       console.log(error);
31       setLoading(false);
32     },
33   });
34
35   const [addMessage] = useMutation<CreateMessage, CreateMessageVariables>(ADD_MESSAGE, {
36     variables: {
37       input: {
38         content: newMessageText,
39         ticket_id: idAsNumber,
40         client_id: user.userType === 'client' ? user.id : null,
41         employee_id: user.userType === 'employee' ? user.id : null,
42       },
43     },
44     onCompleted: data => {
45       setNewMessageText('');
46       void triggerGetTicketById(); // On refait la requête pour avoir les infos à jour
47     },
48     onError: error => {
49       console.log(error);
50     },
51   });
52
53   (...)

54   const handleSubmit = (event: FormEvent): void => {
55     event.preventDefault();

56     void addMessage();
57   };
58
59   useEffect(() => {
60     void triggerGetTicketById();
61   }, []);
62
63   (...)
```

*(Voici le début du code, j'ai retiré ce qui n'était pas nécessaire à cette explication et je l'ai remplacé par (...) )*

Dans notre composant Ticket, on commence par identifier l'id utilisé dans l'url (lignes 2 à 3). Ensuite, on accède au contexte pour retrouver les informations de l'utilisateur (ligne 5).

Un useEffect est utilisé (ligne 61) afin de recharger la page et ainsi d'exécuter la méthode "triggerGetTicketById" qui va nous permettre de faire notre requête à l'API afin d'avoir les informations du ticket de la page.

```
1 import { gql } from '@apollo/client';
2 export const GET_TICKET_BY_ID = gql` 
3   query GetTicketById($ticketId: Int!) {
4     getTicketById(id: $ticketId) {
5       id
6       title
7       content
8       status
9       client {
10         id
11         email
12         company
13       }
14       messages {
15         id
16         content
17         client_id
18         client {
19           email
20           id
21         }
22         employee_id
23         employee {
24           id
25           firstname
26           lastname
27         }
28         created_at
29         updated_at
30       }
31       employees {
32         id
33         email
34         firstname
35         lastname
36       }
37       created_at
38       updated_at
39     }
40   }
41 `;
```

Cette méthode (lignes 9 à 33) va exécuter notre requête à l'API. Cette requête de type query se trouve dans le dossier apollo et permet de recevoir les informations souhaitées relatives à un ticket.

Pour fonctionner, cette requête nécessite l'id d'un ticket en variable, chose que nous avons via l'url param.

A la ligne 10, le deuxième argument, fetchPolicy, permet de ne pas utiliser le système de cache pour cette requête. Cette façon de faire a été choisie pour pouvoir afficher les nouveaux messages sans forcer l'utilisateur à recharger la page.

Si la requête est complétée (ligne 12), on vérifie tout de même de bien avoir reçu des données et si c'est bien le cas, on sauvegarde ces données dans le state (lignes 13 à 16). On ajoute ensuite un traitement pour vérifier si l'utilisateur est un employé et s'il est chargé de ce ticket. On enregistre l'information dans le state également pour l'utiliser plus tard lors de l'affichage (lignes 17 à 24).

Ensuite, que la requête soit complétée ou non, on désactive le Loader. Le cas d'une erreur n'est pour le moment pas traité.

Lorsqu'un utilisateur écrit puis envoie un message, la méthode "handleSubmit" est

exécutée. Cette méthode arrête le comportement par défaut de l'envoi d'un formulaire puis exécute la méthode "addMessage" (lignes 55 à 59).

```
1 import { gql } from '@apollo/client';
2 export const ADD_MESSAGE = gql` 
3   mutation CreateMessage($input: MessageInput) {
4     createMessage(input: $input) {
5       content
6       ticket_id
7       client_id
8       employee_id
9     }
10   }
11 `;
```

Comme pour la requête d'un ticket, cette méthode va exécuter notre requête à l'API qui se trouve également dans le dossier apollo. Elle est cependant de type mutation et non querle. Ces deux types de requêtes graphQL ont des objectifs différents. Une querle servira à chercher des données alors qu'une mutation servira à écrire ou modifier des données.

Cette requête CreateMessage, a besoin de plusieurs variables (lignes 36 à 41) :

- content : le texte du message
- ticket\_id : l'id du ticket sur lequel on veut ajouter un message
- client\_id : l'id du client qui ajoute le message le cas échéant
- employee\_id: l'id de l'employé qui ajoute le message le cas échéant

Un message pouvant provenir soit d'un client soit d'un employé, on vérifie le type de l'utilisateur via les informations dans le contexte et utilise son id ou null en fonction (lignes 40 à 41).

Si la requête est complétée, on efface le message du state et on exécute la requête pour avoir les informations du ticket afin d'avoir le message (lignes 44 à 47).

Ici aussi, il n'y a pour le moment pas de gestion des erreurs.

Voyons le fonctionnement côté back pour cette requête :

```
● ● ●  
1 type Message {  
2   id: PositiveInt!  
3   content: String!  
4   ticket_id: PositiveInt!  
5   client_id: PositiveInt  
6   client: Client  
7   employee_id: PositiveInt  
8   employee: Employee  
9   created_at: DateTime!  
10  updated_at: DateTime  
11 }  
12  
13 input MessageInput {  
14   content: String!  
15   ticket_id: PositiveInt!  
16   client_id: PositiveInt  
17   employee_id: PositiveInt  
18 }
```

Voici les schémas graphQL d'un Message.  
Le type est la “forme” de notre objet et contient toutes les propriétés de l'entité Message de notre base de données ainsi que les informations en plus que nous souhaitons ajouter comme “employee”.

Les points d'exclamation indiquent que la valeur est obligatoire.

“MessageInput” est la “forme” de l'objet passé en argument lors d'une requête pour l'ajout d'un message.

Pour les propriétés comme “employee”, nous avons besoin d'un resolver pour que graphQL puisse interpréter cette valeur.

La valeur “Employee” fait référence au type Employee qui se trouve dans son fichier de schéma de la même façon que pour Message.

```

1 module.exports = {
2   employee(parent, _, { dataSources }) {
3     if (parent.employee_id == null) {
4       return null;
5     }
6     return dataSources.employee.findByPk(parent.employee_id);
7   },
8
9   client(parent, _, { dataSources }) {
10    if (parent.client_id == null) {
11      return null;
12    }
13    return dataSources.client.findByPk(parent.client_id);
14  },
15};

```

Voici le code qui se trouve dans le resolver pour Message.

Comme un message peut être soit d'un employé soit d'un client, nous avons besoin de vérifier si un id pour l'un ou l'autre est présent (lignes 3 et 10).

Lorsque le message n'est pas celui d'un employé (ligne 3) ou d'un client (ligne 10), nous retournons "null" sinon nous utilisons la requête "findByPk" du dataSource approprié (respectivement lignes 6 et 13).

Puisque la requête pour un id est quelque chose de commun et souvent utilisé, une méthode mutualisée a été écrite. Elle se trouve dans le dossier de dataSource commun "core".

```

1  async findByPk(id) {
2    /*
3      Ici plutôt que de faire la requête directement
4      On passe l'id au DataLoader qui va le stocker
5      Et décharger toute la liste d'id au moment approprié
6      et redistribuer à chaque appelant les données demandées.
7    */
8
9    if (process.env.DATALOADER_ENABLED) {
10      return this.idLoader.load(id);
11    }
12
13    const query = this.knex(this.tableName).connection(this.establishedConnection).select('*').where({ id });
14
15    const result = await (process.env.CACHE_ENABLED ? query.cache(SECONDS) : query);
16
17    return result[0];
18  }

```

Cette méthode va, dans un premier temps, si le système de Batching est activé, l'utiliser (lignes 9 à 11) pour garder en mémoire la requête désirée avec l'id puis faire les requêtes par lot.

Si ce n'est pas le cas, nous construisons la requête directement en utilisant knex (ligne 13), qui pour rappel est la bibliothèque de construction de requête que nous utilisons.

"this.tableName" fait par exemple référence au "dataSources.client...." qui a été utilisé dans

le résoudre et permet d'identifier la table de la base de données sur laquelle faire la requête.

Cette requête préparée est l'équivalent SQL de :



```
1 SELECT * FROM client WHERE client.id = id
```

Où "id" est la valeur passée en argument.

Ensuite, ligne 15, si le système de Caching est activé et le temps imparti est défini, on utilise les données reçues lors du dernier appel de cette requête.

Si ce n'est pas le cas, nous envoyons la requête à notre base de données.

Puis, on renvoie la réponse (ligne 17). Cette réponse est celle reçue côté front lors du "onCompleted" ou "onError".

Voyons maintenant le fonctionnement de l'affichage :

```
● ● ●

1  return (
2    <div className="ticket-container">
3      <h1>{ticket.title}</h1>
4      {loading ? (
5        <Loader />
6      ) : (
7        <>
8          <div className="ticket-infos">
9            <div>
10              <span className="ticket-subtitle">Numéro du ticket:</span> {ticket.id}
11            </div>
12            <div>
13              <span className="ticket-subtitle">Bénéficiaire:</span> {ticket.client?.email}
14            </div>
15            <div>
16              <span className="ticket-subtitle">Entreprise:</span> {ticket.client?.company}
17            </div>
18            <div className="ticket-status-container">
19              <span className="ticket-subtitle">Statut:</span>{' '}
20              <span className={ticketStatusClassName(ticket.status)}>{ticketStatusTraduction(ticket.status)}</span>
21            </div>
22            <div>
23              <span className="ticket-subtitle">Prise en charge:</span>{' '}
24              {isEmployeeHandlingTicket && user.userType !== 'client' && (
25                <FontAwesomeIcon icon={faMinus} onClick={handleMinus} className="ticket-minus" />
26              )}
27              {isEmployeeHandlingTicket && user.userType === 'client' && (
28                <FontAwesomeIcon icon={faPlus} onClick={handlePlus} className="ticket-plus" />
29              )}
30              {ticket.employees.map(employee => (
31                <span key={`employees${employee.id as number}`}>{employee.lastname} {employee.firstname}</span>
32              )));
33            )}
34          </div>
35        <div>
36          <span className="ticket-subtitle">Date de création:</span>{' '}
37          {{ticket.created_at}}
38        </div>
39        {Boolean(ticket.updated_at) && (
40          <div>
41            <span className="ticket-subtitle">Dernière modification:</span>{' '}
42            {{ticket.updated_at}}
43          </div>
44        )}
45        <div>
46          <span className="ticket-subtitle">Description:</span> {ticket.content}
47        </div>
48      </div>
49      <div className="messages-ctn">
50        <h1>Messages:</h1>
51        {ticket.messages.map(message => (
52          <Message
53            key={`${message.id as number}`}
54            message={message}
55            clientId={ticket.client?.id as number}
56          />
57        ))}
58      </div>
59      <div className="messages-form-container">
60        </div>
61      </div>
62    );
63  </div>
64);
```

Lorsque la requête pour le ticket est complétée, nous enregistrons les informations dans le state de React ce qui nous permet ensuite de faire un affichage dynamique.

Je ne vais pas rentrer trop dans les détails, mais nous avons un Loader qui permet d'informer l'utilisateur que la requête est en cours. Une fois que la requête est effectuée, l'affichage de la page est fait (ligne 4). Le reste du code s'occupe de l'affichage dynamique et conditionnel pour certaines informations.

Pour l'affichage des messages sur la page des tickets (lignes 52 à 57), un map est effectué afin d'afficher tous les messages que pourrait avoir ce ticket en utilisant le composant

Message que j'ai créé.

Ce composant à trois arguments :

- key : un identifiant unique pour le bon fonctionnement de React
- message : toutes les données du message
- clientId : l'id du client à qui appartient le ticket

Voici le contenu important du composant Message :

```
● ● ●

1 const Message = ({ message, clientId }: MessageProps): ReactElement => {
2   // eslint-disable-next-line @typescript-eslint/no-unsafe-assignment
3   const author =
4     message.client_id != null
5       ? message.client?.email
6       : message.employee_id != null
7       ? // eslint-disable-next-line @typescript-eslint/restrict-template-expressions
8         `${message.employee?.lastname} ${message.employee?.firstname}`
9       : "Problème lors de la récupération de l'auteur";
10
11  return (
12    <div className={message.client_id === clientId ? 'message message-client-container' : 'message message-container'}>
13      <div className="infos-container">
14        <span className="author-infos">{author}</span>
15        <span className="date-infos">
16          <Moment format="DD/MM/YYYY à HH:mm">{message.created_at}</Moment>
17        </span>
18      </div>
19      <div>{message.content}</div>
20    </div>
21  );
22};
```

Premièrement, une condition permet de déterminer l'auteur du message (lignes 3 à 9) et ce afin d'afficher l'email pour le cas d'un client ou le nom et prénom pour un employé. Si on ne se trouve pas dans un de ces deux cas, cela veut dire qu'il y a eu un problème et on affiche un texte en conséquence.

Une condition dans la className permet de changer le style css en fonction des messages du client ou d'un employé (ligne 12).

Le reste du code est simplement un affichage dynamique avec les informations contenues dans l'argument message reçu par le composant Message.

Pour l'affichage de la date, nous avons utilisé la bibliothèque Moment qui permet rapidement et simplement de formater le texte d'une date.

Voici un exemple du résultat :

Accueil    Créeer un ticket    Tickets

Déconnexion

## Un ticket de test pour la démo des messages

**Numéro du ticket:** 18  
**Bénéficiaire:** client1@test.test  
**Entreprise:** Company 1  
**Statut:** ouvert  
**Prise en charge:**  
**Date de création:** 30/01/2023 à 18:40  
**Description:** La description

### Messages:

client1@test.test 30/01/2023 à 19:43  
Un premier message du client

Nom employee 3 Prénom employee 3 30/01/2023 à 19:45  
Une réponse d'un employee

Ajouter un message

## 7 - Recherches :

### 7a - Veilles sur la sécurité :

Lors de la phase de conception, nous avons discuté du sujet de la sécurité.

Pour nous aider dans cette démarche, nous avons utilisé le top dix des failles de sécurité OWASP (Open Web Application Security Project).

Pour chacune de ces failles, nous avons noté des informations afin d'avoir une base de réflexion lors du développement et pouvoir tenter de nous en prémunir.

Voici un exemple concernant l'authentification de mauvaise qualité : l'usage d'un mot de passe faible.

L'utilisation des mots de passe trop faibles ou communs et donc facilement découverts lors d'une attaque par force brute est une situation très commune.

Alors qu'il était initialement demandé de ne pas avoir de système d'authentification pour les clients, nous avons trouvé que cela était un trop grand risque et avons décidé, comme compromis, de n'utiliser que l'adresse email pour les clients.

Pour les employés, lors d'un changement de mot de passe, afin d'avoir un mot de passe robuste, il est demandé d'utiliser :

- un minimum de huit caractères
- au moins une majuscule
- au moins une minuscule
- au moins un chiffre
- au moins un caractère spécial.

Cette vérification n'est pour le moment utilisée que lors du changement de mot de passe car la création d'un compte employé n'a pas été réalisée et doit se faire directement sur la base de données.

Une première vérification côté front est effectuée puis une validation côté back, en utilisant la bibliothèque Joi, nous permet également de vérifier que cette règle est respectée et nous prémunir contre les injections.

Ensuite, un hachage est effectué pour ne pas avoir de mot de passe dit "en clair" sur la base de données.

Il faut noter que pour la présentation de ce projet et une facilité de démonstration, des comptes ne respectant pas ces règles ont été prévus.

Voici la liste du top 10 des failles et les informations que nous avions notées les concernant dans le cahier des charges lors de la phase de conception :

- **Contrôle d'accès défaillants :**

Nous devons impérativement ajouter un contrôle d'accès (côté serveur) sur les ressources qui n'ont pas vocation à être publiques.

Ce contrôle d'accès doit idéalement être géré par des ACL centralisées dans un seul fichier. Il faut ensuite tester si le contrôle d'accès fonctionne correctement et si une

personne non autorisée à accéder à certaines ressources voit bien son accès restreint comme prévu.

Redirection des url avec un .htaccess

- **Défaillances cryptographiques :**

Les mots de passe doivent être hachés avec une fonction de hachage solide (SHA256/512, bcrypt, argon, etc.)

Pour se protéger au mieux des attaques, les hash générés peuvent être salés : on ajoute une donnée supplémentaire au mot de passe avant le hachage. Le sel utilisé devra également être stocké en BDD.

- **Injection :**

Les injections XSS & SQL ne se préviennent pas de la même façon, mais la logique est la même : on ne peut pas faire confiance à l'utilisateur.

- Remédiation XSS : Pour protéger notre application d'éventuelles failles XSS, plusieurs possibilités s'offrent à nous :
  - On peut "nettoyer"/valider les données saisies par les utilisateurs côté serveur.
  - On devrait utiliser textContent plutôt que innerHTML dans notre code JS côté front.
  - Utiliser textContent dans notre JS côté client quand on veut modifier le contenu texte d'un nœud du DOM permet d'éviter que du code JS contenu dans le texte à afficher soit exécuté. Avec innerHTML, le code JS sera exécuté (il faut donc éviter de l'utiliser, surtout quand on doit afficher des données qui ont été saisies par des utilisateurs).
- Remédiation SQL
  - Contre l'injection SQL, une solution : les requêtes préparées. Dès que nous allons devoir faire une requête qui contiendra des paramètres saisis par les utilisateurs de notre site, il faudra utiliser une requête préparée.

- **Conception non sécurisée :**

L'utilisation de questions secrètes pour récupérer un mot de passe perdu : c'est une façon de faire qui doit être bannie de nos pratiques de développement, il est en effet trop simple pour un individu mal intentionné de récupérer des informations sur un utilisateur pour répondre à ces questions.

Une réduction qui serait appliquée sur des réservations de places de cinéma sans demander d'acompte : quelqu'un pourrait potentiellement réserver un grand nombre de places dans le seul objectif de faire perdre des revenus au cinéma.

Cette menace est particulièrement difficile à contrer puisqu'elle implique de devoir penser à tous les cas de figure possibles. Dans cette situation, modifier le code pour chacun des cas possibles ne sera pas une réponse efficace, il faut donc y penser dès la phase de recueil du besoin / rédaction des users stories afin de prévoir une architecture adaptée en amont.

- **Mauvaise configuration de sécurité :**

- Divulgation d'informations : 404 & headers

- **Composants vulnérables/obsoletes :**

Cette catégorie est assez claire : il faut faire attention à ne pas utiliser de composants (logiciels, bibliothèques, frameworks, etc.) vulnérables (pour lesquels des failles sont connues, exemple : Log4j) ou obsolètes (exemple : vieilles versions de PHP, Symfony, etc.).

Pour nous aider à vérifier les dépendances utilisées par notre application, OWASP met à disposition un outil : [dependency check](#).

- **Authentification de mauvaise qualité :**

Ce qu'OWASP appelle une authentification de mauvaise qualité est un système qui n'est pas protégé contre les attaques par force brute (avec ou sans dictionnaire), qui utilise des mots de passe par défaut / très faibles (admin/admin).

- **Manque d'intégrité des données & du logiciel :**

Cette faille concerne les scripts récupérés depuis Internet ou les mises à jour effectuées à l'intérieur d'une application. Par exemple : vous n'avez pas besoin de mettre à jour votre application Netflix, celle-ci se met à jour automatiquement sans nécessiter aucune action de la part de l'utilisateur. Il faut vérifier l'intégrité des scripts récupérés de la sorte, pour s'assurer qu'ils n'aient pas été altérés par une personne malveillante.

Il est par exemple nécessaire d'utiliser l'attribut HTML integrity pour vérifier que les scripts JS hébergés sur des CDNs n'aient pas été altérés.

- **Carence de contrôle & de journalisation :**

Cette catégorie nous indique qu'il faut mettre en place des mécanismes de journalisation sur nos applications : enregistrer toutes les tentatives de connexions (réussies ou échouées), toutes les actions critiques (ajout/modification d'un compte utilisateur par exemple).

Les journaux ne doivent pas être stockés en local, afin d'éviter qu'ils soient effacés/altérés.

Certaines actions doivent déclencher des alertes, afin de pouvoir réagir au plus vite en cas d'attaque.

- **Falsification de requête côté serveur :**

Nos serveurs web sont parfois à l'intérieur de réseaux comprenant d'autres machines ou services, protégés derrière des pare-feu et normalement inaccessibles depuis Internet. Les attaques

ont pour objectif de permettre d'effectuer des requêtes vers ces serveurs par le biais du serveur web, accessible depuis Internet, et ainsi de contourner d'éventuels pare-feu et autres protections.

Pour s'en protéger, il faut segmenter : notre serveur web ne doit pas pouvoir communiquer avec d'autres serveurs sur le réseau de l'entreprise. Si certaines ressources internes doivent être accessibles, il faut impérativement mettre en place des règles de pare-feu & de filtrage permettant de bloquer toute requête illégitime.

## 7b - Recherches :

Lors du développement du projet, j'ai dû faire beaucoup de recherches seul ou en groupe, que ce soit pour trouver comment résoudre un problème ou trouver la réponse à un objectif.

Pendant la phase de conception, nous avons effectué des recherches dans le but de trouver de l'inspiration et voir ce qui était possible.

Pendant les autres sprints, j'ai rencontré divers problèmes que j'ai pu régler généralement en relisant ce que nous avions fait en cours, la documentation appropriée ou bien évidemment mon code.

Voici quelques exemples de problèmes rencontrés qui ont nécessité des recherches.

1. Je vais commencer par le plus gros problème :

Lors d'une rebase, la PR a disparu de github et tout le code qui allait avec n'était plus présent en local également.

Lors de la finalisation de ce dossier, je ne sais toujours pas ce qui c'est passé.

J'ai l'habitude de faire des rebase et j'ai dû en faire plusieurs fois par jour pendant plus d'un an au travail.

Dans la panique qui a suivi, je n'ai pas pris de note de mes recherches mais une fois le problème résolu, j'ai noté quelques informations.

J'avais le message d'erreur suivant :

*"Another git process seems to be running in this repository, e.g. an editor opened by 'git commit'. Please make sure all processes are terminated then try again. If it still fails, a git process may have crashed in this repository earlier : remove the file manually to continue."*

Traduction :

*"Un autre processus git semble être en cours d'exécution dans ce dépôt (le repository git donc), un éditeur a été ouvert par un "git commit". Assurez-vous que tous les processus sont terminés puis réessayez. S'il échoue toujours, un processus git peut avoir planté dans ce dépôt plus tôt : supprimez le fichier manuellement pour continuer."*

J'avais également des messages à propos d'un cherry-pick.

N'ayant jamais utilisé cette commande git, je suis allé lire la documentation sur le cherry-pick et fais quelques recherches sur le sujet. J'ai également trouvé un post sur stackoverflow où les commentaires m'ont aidé à identifier le problème.

(<https://stackoverflow.com/questions/12397752/how-to-conclude-a-git-cherry-pick>).

J'ai également lu les documentations sur le rebase et fais des recherches comme "revert commit" ou "revert rebase".

J'ai trouvé un autre post stackoverflow

(<https://stackoverflow.com/questions/4114095/how-do-i-revert-a-git-repository-to-a-previous-commit>) et via les commentaires j'ai trouvé la commande "git reflog".

Via toutes ces informations, j'ai pu retrouver le commit qui contenait tout le code de la PR et en supprimant des fichiers CHERRY\_PICK\_HEAD, CHERRY\_PICK\_HEAD.lock et COMMIT\_EDITMSG le problème était résolu.

Je n'ai malheureusement pas plus d'informations à partager sur ce problème. C'était la première fois que je perdais du code et je n'avais absolument aucune idée de pourquoi et j'étais vraiment paniqué. Résoudre le problème a dû me prendre une à deux heures, le temps de faire des recherches, lire de la documentation, check les logs etc.

J'ai pensé qu'il était intéressant de partager cette expérience malgré le manque d'information car c'était la recherche la plus importante du projet pour moi et je regrette de ne pas avoir pris plus de note sur le sujet.

## 2. J'ai eu des difficultés à mettre en place les CI de github.

Après avoir revu ce que nous avions appris en cours et lu la documentation de github sur les actions, je me suis rendu compte via les exemples que j'oubliais d'aller dans le dossier "front" avant d'utiliser la commande pour lancer les tests. Une erreur très simple mais qui m'a malgré tout pris plus de temps que je ne suis prêt à l'admettre.

Cette erreur m'a cependant permis d'apprendre quelque chose, en lisant les exemples de la documentation

(<https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-nodejs>) j'ai remarqué l'utilisation de la commande "npm ci" plutôt que "npm install".

Voici le passage de la documentation :

*"Using npm ci installs the versions in the package-lock.json or npm-shrinkwrap.json file and prevents updates to the lock file. Using npm ci is generally faster than running npm install. For more information, see [npm ci](#) and "[Introducing npm ci for faster, more reliable builds](#)."*

Traduction :

*"L'utilisation de npm ci installe les versions présentes dans le fichier package-lock.json (...) et empêche les mises à jour du fichier .lock. Utilisé npm ci est généralement plus rapide que d'utiliser npm install. (...)"*

Je suis donc aller lire le lien d'introduction de cette commande et j'en ai compris que npm ci n'utilisait pas le package.json pour installer les modules mais le package-lock.json ce qui permet d'être sur d'avoir les versions qui sont attendu et permet ainsi une meilleur reproductibilité. Il y avait également un passage sur le fait que cette commande était plus rapide mais il n'y avait pas d'explication du pourquoi et je n'ai pas continué mes recherches sur le sujet.

## 8 - Conclusion :

### 8a - LorenzO'Ticket :

Nous avons pris beaucoup de temps sur la conception et sommes plusieurs fois revenus en arrière pour essayer de faire de notre mieux, notamment sur les différents schémas UML. Ils ont été un véritable challenge car ce n'est pas quelque chose pour lequel nous avions beaucoup d'expérience hormis les cours qui remontaient à pratiquement un an.

Pour les points négatifs, je n'ai pas assez commenté mon code à mon goût. C'est un défaut que j'ai et dont je suis conscient et que j'essaye de corriger.

Le plus gros regret est bien sûr de ne pas avoir pu finir le MVP entièrement.

J'aurais également aimé avoir plus de temps pour écrire des tests et de la validation de données.

Le rythme de l'alternance était de deux jours en cours et trois jours en entreprise. Ce rythme était très difficile pour bien se concentrer sur les projets, un temps d'adaptation était nécessaire et cela a impacté la productivité.

Pour continuer sur le sujet de la productivité et du temps, notre choix de faire beaucoup de pair programming a comme prévu également joué un grand rôle sur la quantité de code que nous avons été à même de produire.

Cependant, ce n'est pas un choix que je changerais.

En effet, cela a été très utile pour la compréhension et l'approfondissement des technologies et c'est un des points positifs. La bonne humeur au sein de l'équipe est également un des points forts.

Étant tous pris par le travail, nous avons décidé de ne pas ajouter trop de code après la fin de l'Apothéose afin que tout le monde soit bien au courant de ce qui a été fait et puisse préparer le dossier dans les meilleures conditions.

## 8b - Améliorations :

Au-delà de finir le MVP et d'ajouter les fonctionnalités secondaires, il y a différentes améliorations et potentiels problèmes auxquels nous n'avions pas pensé.

Le principe de Lead nous semblait être une bonne idée, mais après avoir discuté avec un ami qui travaille dans le domaine du SAV avec un site de gestion de ticket comme celui-ci, il y a un problème auquel nous n'avions pas pensé, le débit de ticket. En effet, si l'agence web à beaucoup de clients, il ne serait pas viable que seulement les Lead filtrent les tickets. Nous avions noté en fonctionnalité secondaire, l'ajout d'un champ lors de la création d'un ticket afin que les clients puissent eux même définir leur problème. Cette fonctionnalité est donc importante et nécessiterait un temps de réflexion supplémentaire afin de réduire la charge de travail un maximum pour les Lead.

Un système de fermeture automatique des tickets devrait être envisagé si aucune nouvelle n'a été donnée par le client dans un laps de temps prédéfini.

Sur la page de connexion nous avons mis "employé(e)", il faudrait donc aussi "un(e)".

Pour le moment nous affichons le nom et prénom des employées sur les tickets. Un système d'anonymat devrait être envisagé.

Nous avons pour le moment une fonctionnalité qui permet de fermer un ticket. Il faudrait ajouter un système de formulaire de satisfaction à celle-ci.

## 8c - Remerciements :

Bien évidemment, je commence par remercier O'clock et toute son équipe (j'ai trop peur d'oublier des noms donc je m'abstiens d'en mettre) chez qui la bonne humeur et l'humour règnent.

J'avais fais une première formation pour passer le DWWM. Cela a été une magnifique expérience que j'ai pu suivre dans les meilleures conditions possibles malgré ma situation (j'ai un handicap au genou qui m'empêche de me déplacer facilement) et cela a changé ma vie.

Merci à ma première promotion, Excalibur, où l'entraide était le mot d'ordre.

Un grand merci aux "Ambulancières", un groupe soudé de cette promotion avec qui j'échange encore tous les jours.

Merci encore à O'clock d'avoir recommencé les formations en alternance et m'avoir permis d'approfondir mes connaissances développeur et concepteur.

Merci à Canal+ de m'avoir donné une chance et l'opportunité de travailler sur un super projet et de pouvoir suivre cette formation en alternance.

Merci à toutes les personnes qui travaillent au sein de l'équipe FRONT TV et

particulièrement à mon équipe FULL OTT pour la bonne humeur et les connaissances que vous m'avez partagées.

Merci à tous mes amis qui m'ont aidés pour la réalisation des dossiers pour le titre professionnel, que ce soit pour de la relecture, des explications, des avis ou les projets sur lesquels nous avons travaillés ensemble. EL BOUANANI Fidia, DELISLE Nicolas, Maxence ROYER, Damien TOSCANO, Tomy BLONDEAU et Farid AIT GACEM.

Et un dernier merci à Lorenzo Lamas pour être un rebelle.

