CAUDRON CEDRIC

DOSSIER PROJET

Réalisation d'une application de gestion de chantiers

Dossier de fin d'année dans le cadre de la formation Développeur Web-WebMobile à Valarep Valenciennes.

Réalisée dans le cadre des stages dans l'entreprise Nortec Ingéniérie.

Ce dossier vaudra également rapport desdits stages.

ANNEE 2020

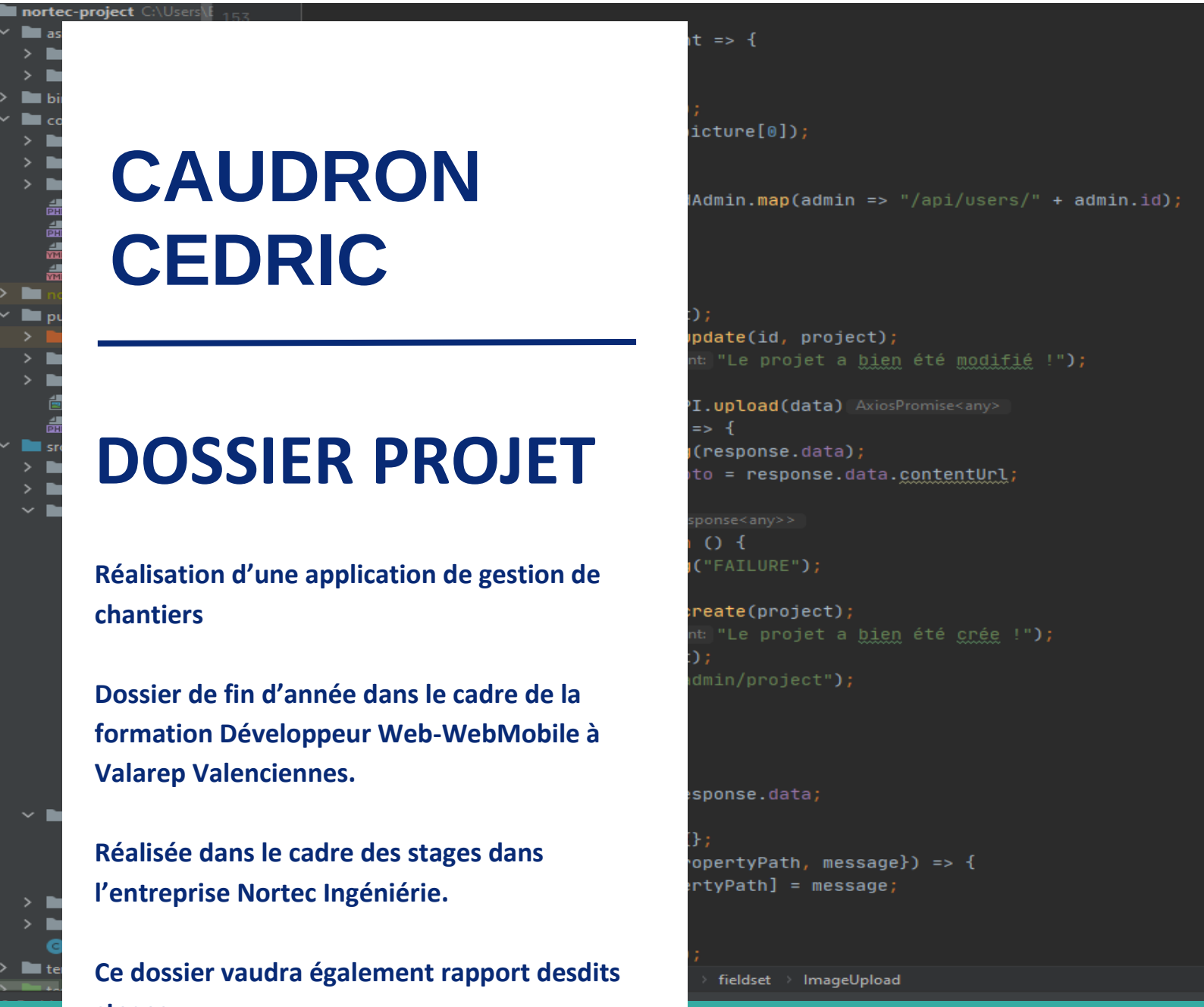


Table des matières

I : LA GENÈSE

1 : Attribution du projet

Début d'année 2020, Mr Arnaud Harbonnier, formateur référent de notre session de formation, a eu une demande de création d'une application de la part de l'entreprise Nortec Ingénierie.

Dans ce cadre, il m'a proposé, ainsi qu'à Vincent Brocheton, de participer à une réunion avec les représentants de l'entreprise afin de faire un point sur leur besoins, et la réalisabilité de leur projet dans le cadre de nos deux stages restant à effectuer dans notre parcours de formation.

Afin de préparer la réunion, les représentants de l'entreprise ont fait un schéma préliminaire, avec leur vision de l'application qu'ils désiraient.

Nous avons donc eu copie de cette ébauche, jointe dans les pages suivantes, avant la réunion afin que nous puissions commencer à comprendre le but recherché par l'application. (Figures 1-1, 1-2 et 1-3)

Avec ces documents, nous commençons à nous faire une idée du but de l'application. Qui sera de générer des rapports PDF automatiquement en fonction de ce que les utilisateurs rentreront, et la possibilité de les envoyer par mail. Mais il est vrai qu'à ce stade, le fonctionnement général restait très nébuleux, d'où la nécessité de la réunion afin d'éclaircir précisément les attentes du client.

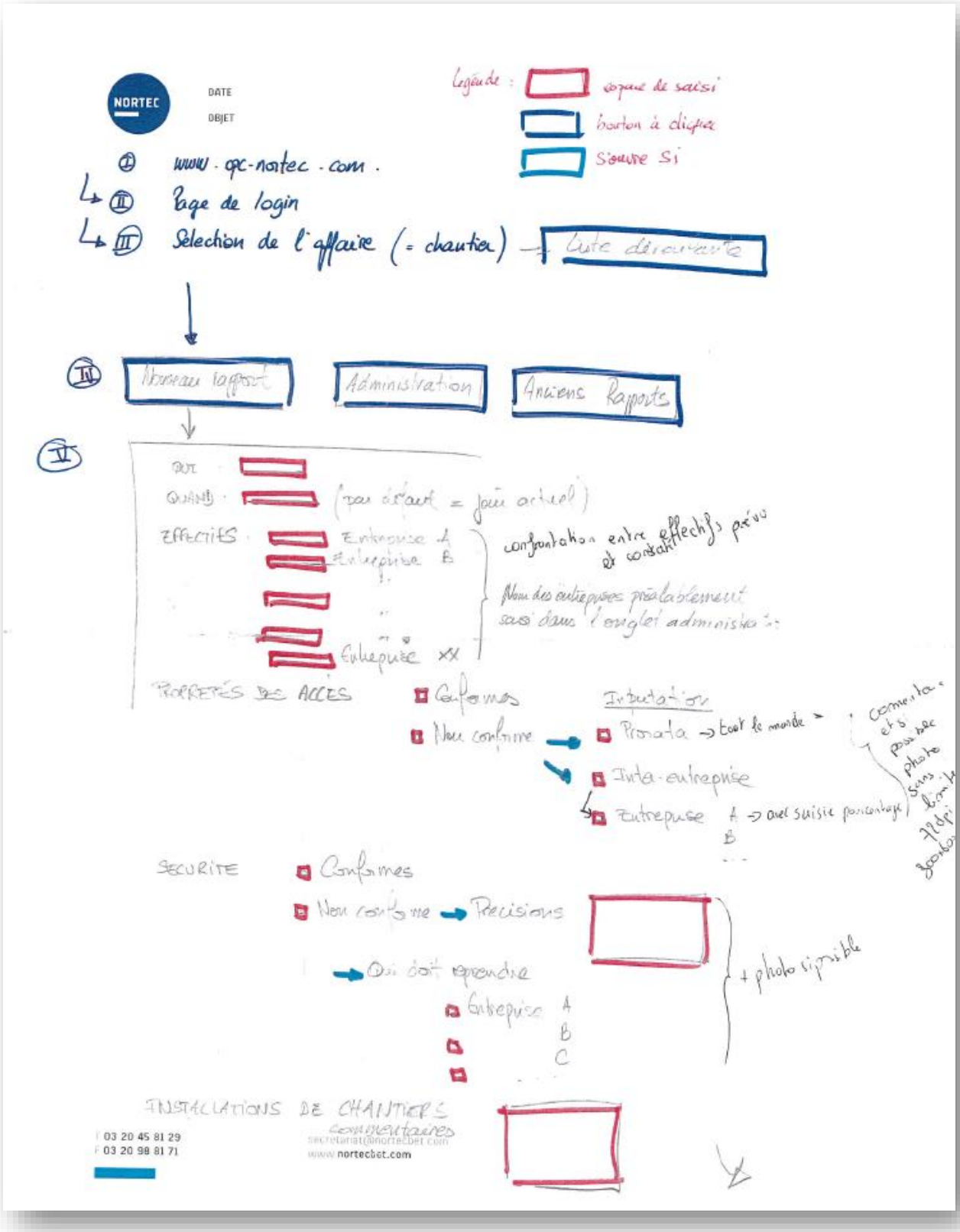

NORTEC
DATE
OBJET
Légende :
espace de saisi
bouton à cliquer
Sélection Si
www.qc-nortec.com
Page de login
Selection de l'affaire (= chantier)
Liste déroulante
Nouveau rapport
Administration
Anciens Rapports
QUI
QUAND
(par défaut = jour actuel)
EFFECTIFS
Entreprise A
Entreprise B
Entreprise XX
comparaison entre effectifs prévu et constatés
Nom des entreprises préalablement saisi dans l'onglet administration
PROPRETÉS DE ACCES
Conformes
Non conforme
Intempéries
Pluvia → tout le monde
Inter-entreprise
Entreprise A → avec saisie pourcentage
B
SECURITE
Conformes
Non conforme → Precisions
+ photo si possible
Entreprise A
B
C
INSTALLATIONS DE CHANTIERS
Commentaires
T 03 20 45 81 29
F 03 20 98 81 71
secretariat@nortecbet.com
www.nortecbet.com

Figure 1

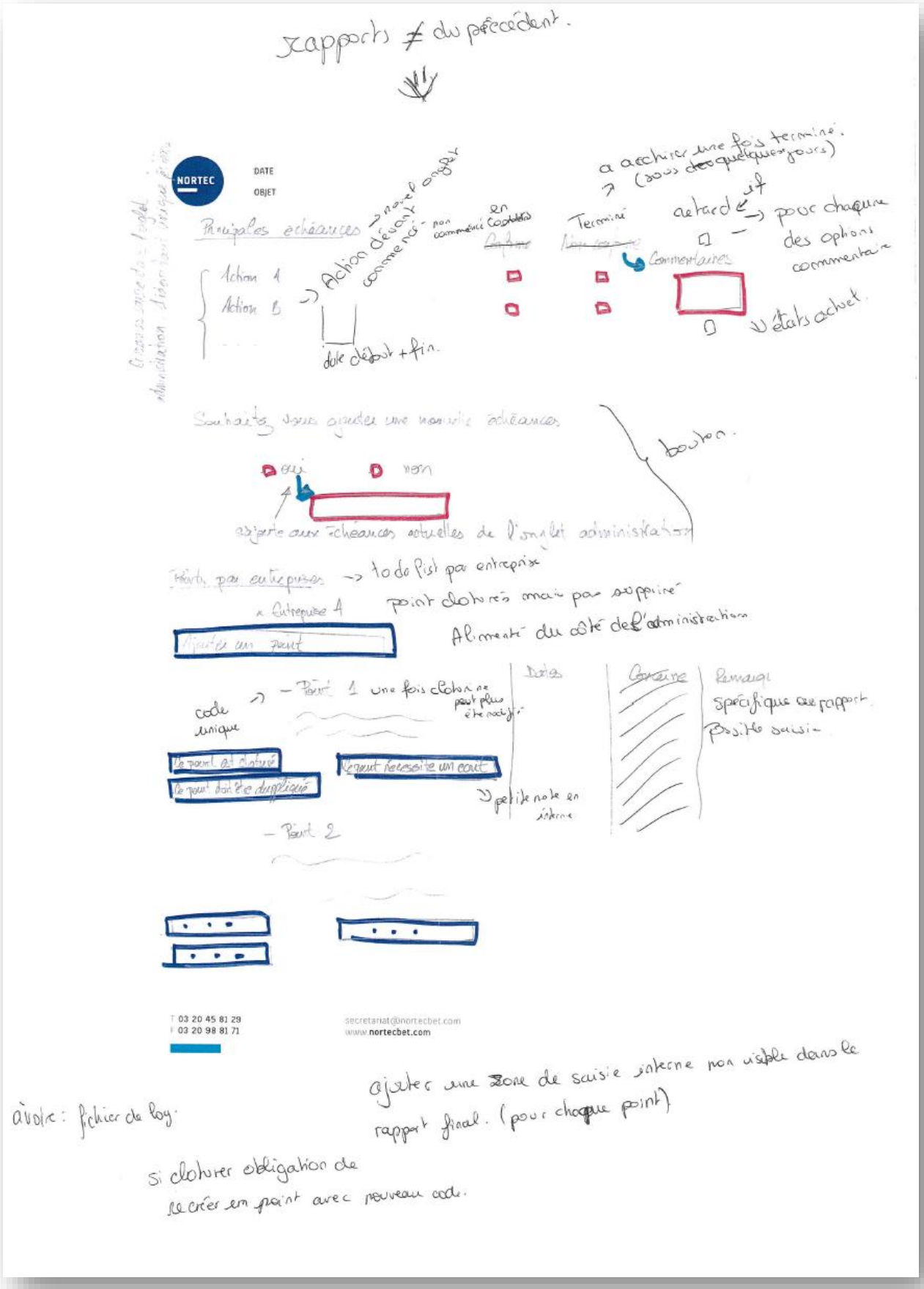

rapports ≠ du précédent.
NORTEC
DATE
OBJET
Principales échéances
Action 1
Action B
Action devant commencer
non commencé
en cours
Terminé
retard
Commentaires
pour chaque
des options
commentaire
états achet
date début + fin.
Souhaitez vous ajouter une nouvelle échéances
bouton.
oui
non
ajoute aux échéances actuelles de l'onglet administration
Point par entreprise
-> todo list par entreprise
point clotures mais pas supprimé
Alimenté du côté de l'administration
+ Entreprise A
Ajouter un point
code unique
- Point 1 une fois cloturé ne peut plus être modifié
Le point est clôturé
Le point doit être dupliqué
Le point nécessite un ecart
petite note en interne
- Point 2
Date
Commentaire
Remarque
spécifique au rapport
Possible saisie
T 03 20 45 81 29
F 03 20 98 81 71
secretariat@nortecbet.com
www.nortecbet.com
ajouter une zone de saisie interne non visible dans le rapport final. (pour chaque point)
aide : fichier de log.
Si cloturer obligation de
le créer en point avec nouveau code.

Figure 2

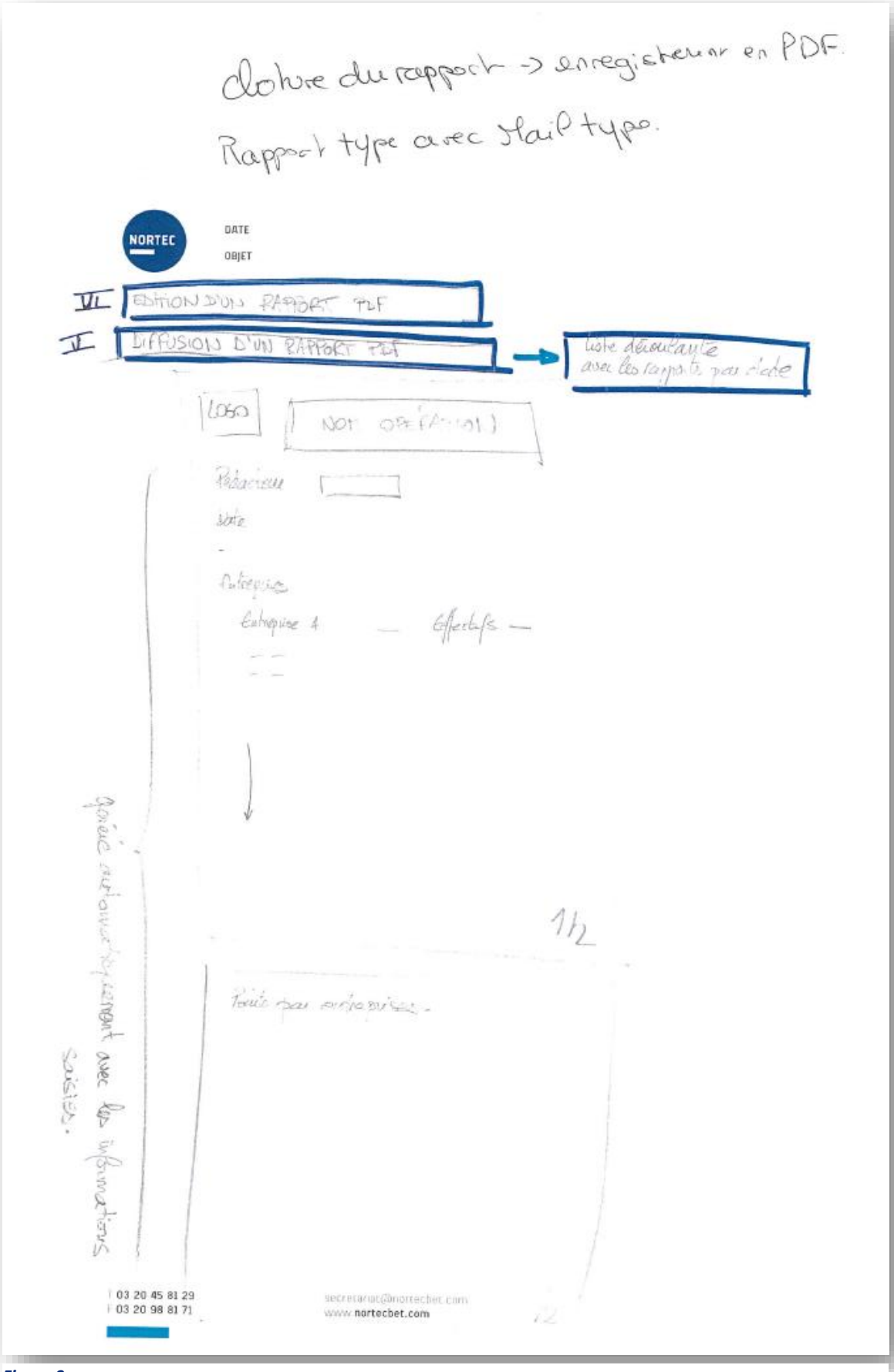

cloture du rapport -> enregistrement en PDF.
Rapport type avec Mail type.
NORTEC
DATE
OBJET
EDITION D'UN RAPPORT PDF
DIFFUSION D'UN RAPPORT PDF
liste déroulante avec les rapports par ride
Logo
NOT OPTIONAL
Rédacteur
Date
Entreprise
Entreprise 1
Effectifs
Triés par entreprises.
Triés automatiquement avec les informations saisies.
1/2
03 20 45 81 29
03 20 98 81 71
secretariat@nortecbet.com
www.nortecbet.com

Figure 3

2 : La réunion avec les représentants de Nortec

Mi-janvier, nous avons donc eu cette réunion avec :

- Mr Florent Courbez, chargé d'affaire chez Nortec
- Mme Hélène Verdrot, chargée d'affaire également
- Mr Arnaud Harbonnier, formateur référent de notre session
- Mr David Riehl, ingénieur développeur et formateur de notre session
- Mr Vincent Brocheton, stagiaire de la formation
- Et moi-même.

Le but de la réunion a été de définir avec plus de précision les réels besoins du client, la faisabilité de l'application à notre niveau dans un délai correct, et là déjà nous avons rencontré une problématique qui nous suivra tout le long du développement : la complexité de dialoguer entre deux métiers différents.

En effet, Mr Courbez avait déjà une idée précise de comment il voyait l'application, comment elle l'aiderait au quotidien dans l'exercice de sa profession, et avec le langage propre à celle-ci. La grande problématique aura donc été de se comprendre, de bien mettre les choses à plat, afin que plus tard nous ne nous lancions pas dans le développement en pensant avoir compris les attentes, mais en étant « à côté de la plaque » et produire une application qui ne répondrait pas aux besoins.

Ainsi, pour l'anecdote, Mr Courbez parlait de « base de données », ce qui nous de notre côté nous parlait en tant que développeur comme système de base de données, alors que pour lui il s'agissait de la liste des utilisateurs. Ce n'est qu'un exemple parmi d'autres sur lesquels nous avons dû faire attention au langage utilisé.

C'est pourquoi par ailleurs, pour éviter ce genre de désagréments, que, suivant le conseil de Mr Riehl, nous avons décidé dès cette réunion, de mettre en place un système AGILE afin d'avoir des échanges réguliers entre nous et le client, pour pouvoir rectifier le tir si jamais le développement partait dans une mauvaise direction.

A l'issue de la réunion la décision a finalement été prise que nous effectuerions nos deux derniers stages dans l'entreprise, à la réalisation de l'application, de la conception au développement, avec l'aide si besoin de Mr Riehl si jamais nous avions des difficultés.

L'aventure a donc commencé ainsi…

II : CAHIER DES CHARGES

1 : Présentation de l'entreprise

NORTEC est un Bureau d'Etudes Techniques pluridisciplinaire indépendant qui agit dans le bâtiment depuis 1995 :

- Secteur d'activité : Bâtiment
- Nombre d'employés : ± 40
- Gérant : Mr Benoit Petit
- Un bureau à Villeneuve d'Ascq, un à Paris
- Chiffre d'affaire : 4,5M€

Il faut savoir que l'entreprise ne possède pas de secteur développement informatique. Nous avons donc opéré en totale autonomie en relation avec eux pour le développement de l'application.

Nous avons été en contact régulier avec Mr Florent Courbez, responsable pôle travaux MOE Paris, en tant que référent dans l'entreprise.

Nous avons aussi présenté le projet d'application à Mr Benoit Petit, gérant de l'entreprise lors d'une réunion improvisée dans les locaux de celle-ci.

Nos relations générales avec l'entreprise auront finalement été similaires à celles qu'ont des développeurs freelance avec leur client.

2 : Objectifs de l'application

L'objectif de cette application est de créer un écosystème inédit dans la gestion de chantiers de construction.

Elle permettra :

- Le suivi des entreprises intervenant sur un chantier
- Le suivi des effectifs réellement présents ou non par semaine
- Le suivi des différentes tâches et éventuels retards
- La génération de rapport de conformité en pdf
- L'envoi automatique par mail des rapports aux entreprises concernées

3 : Cibles de l'application

L'application est destinée à être utilisée par :

- Les chargés d'affaires qui inspecteront les chantiers
- Les responsables de l'entreprise
- Les responsables des différentes entreprises intervenantes

Les droits devront être ajustables suivant les différentes responsabilités des différents intervenants sur chaque chantier.

4 : Périmètre du projet

L'application sera sous la forme d'un site internet. Idéalement consultable de façon ergonomique à la fois sur PC et sur IPad.

Une évolution envisagée sera la mise en place à terme d'une application mobile IOS pour utilisation sur IPad.

La création de compte ne pourra être possible que par les administrateurs. L'application n'est pas publique.

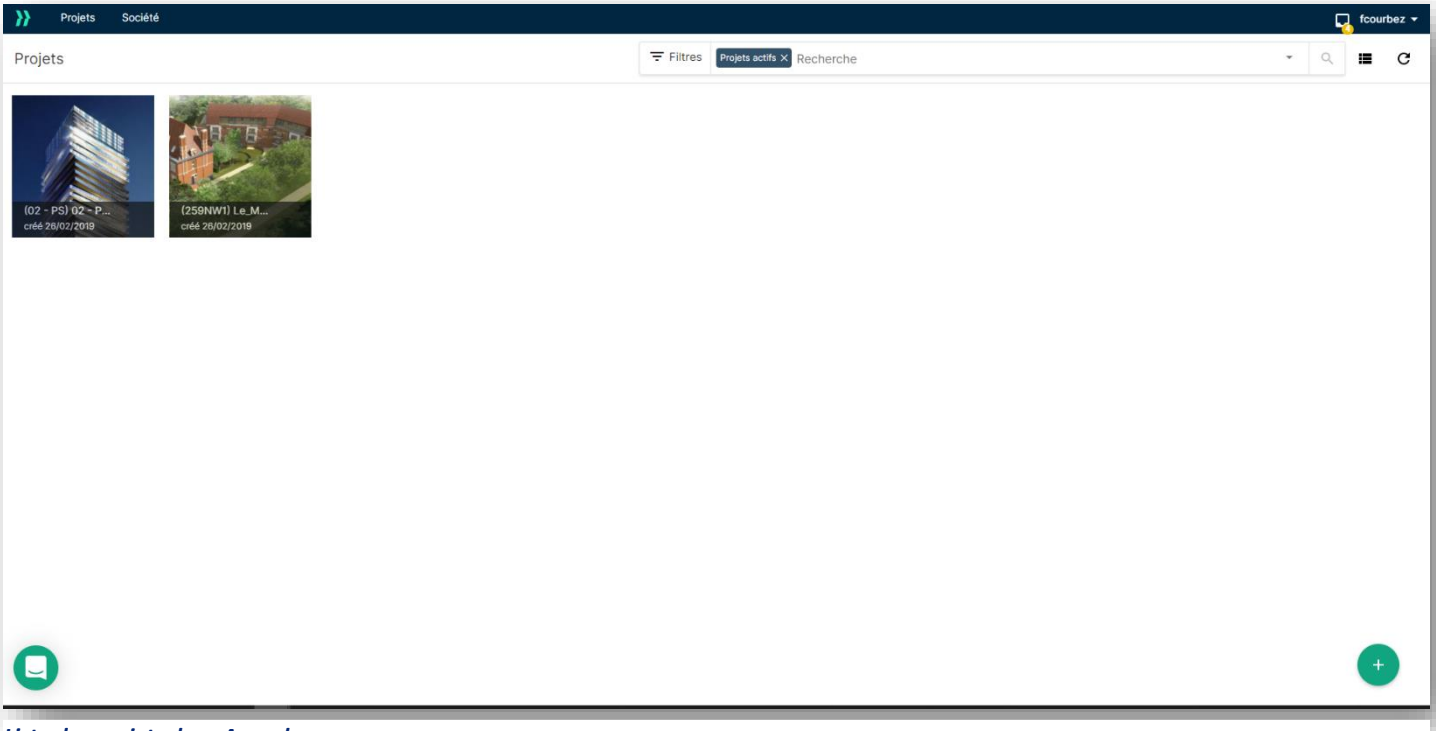
L'application, véhiculant des informations sensibles internes à l'entreprise devra être la plus sécurisée possible.

5 : La charte graphique

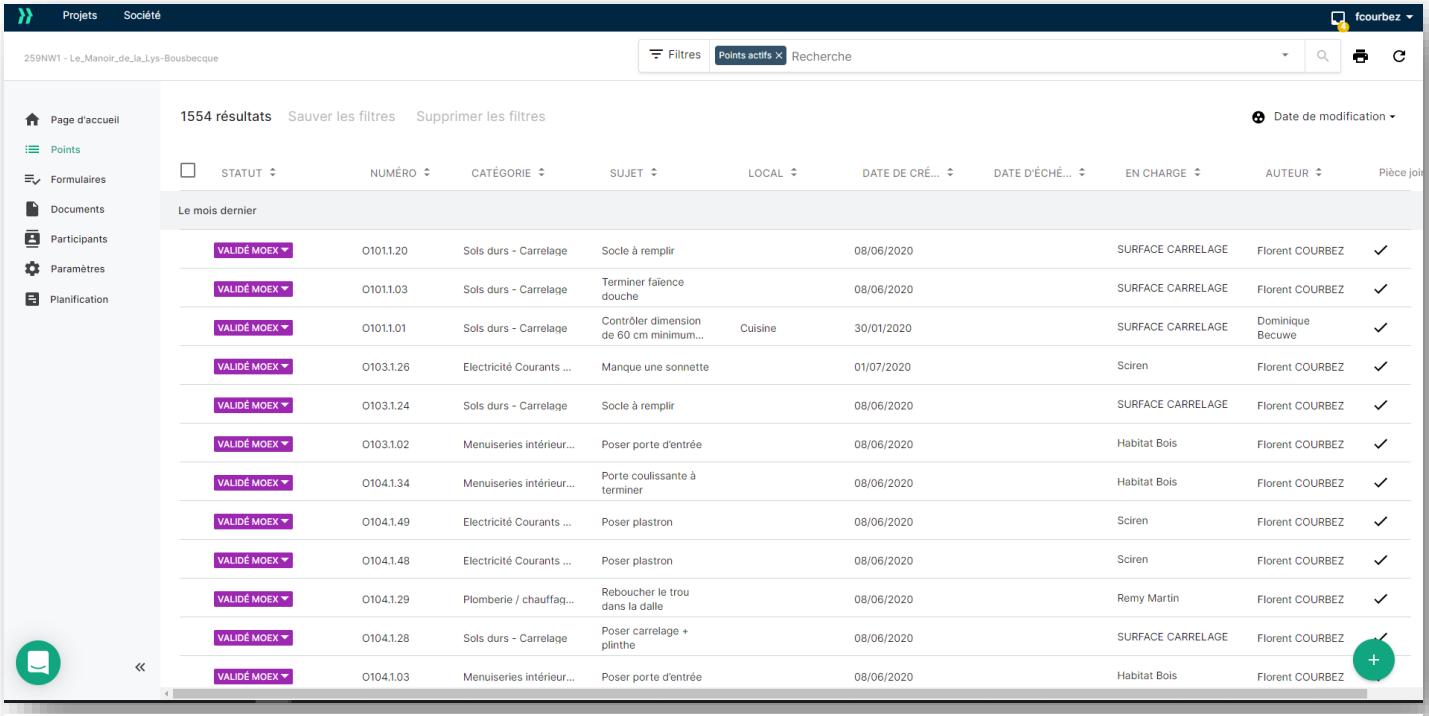
Nous avons toute latitude quant à la charte graphique du site, en respectant quelques contraintes néanmoins :

- Une UI simple et intuitive
- Un aspect moderne
- De manière générale, il faut donner envie aux utilisateurs de réellement utiliser l'application. Qu'ils y trouvent un gain de temps, sans que ce soit trop compliqué pour autant.

Pour cela, Mr Courbez nous a donné quelques exemples d'applications utilisées dans le domaine afin de nous inspirer. Comme ici avec l'application Aproplan.



Liste des projets dans Aproplan



Liste des tâches dans Aproplan

6 : Contraintes techniques

Il n'y a eu aucune contrainte technique spécifiquement évoquée. L'application n'existant tout simplement pas, nous avons toute latitude dans les technologies à utiliser, l'hébergement. Tout doit être fait de zéro !

La seule contrainte, et qui aura été primordiale dans le choix des technologies, c'est évolutivité de l'application. Avec l'éventuel développement, plus tard, d'une application mobile en plus de la version Web.

Plus tard il aura été fait mention de la maintenance, et dans quelles conditions… Avec l'éventualité que nous effectuions notre contrat d'alternance chez eux, afin que nous développions une application vraiment fonctionnelle, ainsi que l'application mobile qui s'y rattacherait. Mais finalement, la situation a fait que ce ne sera pas possible.

7 : Livraison

Aucun délai de livraison n'a été donné à ce stade.

Une estimation a été faite par Mr David Riehl qui, vue la complexité du travail à effectuer, a estimé la livraison possible d'une version fonctionnelle à partir de décembre 2020.

III : DÉROULEMENT DES STAGES

1 : Introduction

Dans cette partie, je parlerai essentiellement, sans rentrer dans la technique, du déroulé plus ou moins chronologique de la conception, du développement, et des relations avec l'entreprise dans le cadre des stages, et en dehors.

Je parlerai des moyens mis en place pour la communication, des difficultés rencontrées lors de cette communication. Toute la partie relationnelle de la gestion de projet en somme.

J'adopte ce plan par soucis de synthèse pour éviter que l'ensemble du dossier soit trop indigeste. Donc ici je parlerai vraiment de ce qui a été fait de façon succincte, toutes les explications arrivant par la suite, par thématique, et de façon beaucoup plus technique.

2 : La période pré-stages

Dès la fin de la réunion, nous savions que seuls les deux mois de stage restant, même en binôme, ne seraient bien évidemment pas suffisants pour le développement d'une telle application.

Nous avons donc commencé bien avant la première période le maquettage (que je détaillerai plus tard), la réflexion sur le choix des technologies, et notre formation en autodidacte sur les technologies retenues (qui ne faisaient pas partie du programme de la formation), le choix s'étant porté sur une API Rest en Symfony et un Front en React.

Ainsi, dans la chronologie, avant d'entamer la première période de stage, le travail de maquettage était bouclé, validé par le client, et notre formation sur les technologies choisies quasiment terminée. Cette période préliminaire d'un mois environ aura été bénéfique afin que nous puissions nous lancer réellement dans la conception et le travail de développement dès la première période.

De plus, pendant cette période nous avons également mis en place un groupe WhatsApp afin de simplifier les échanges le client.

3 : Le stage du 17/02/2020 au 13/03/2020

3.1 : La réflexion sur la base de données

La première étape pour nous, et non des moindres, aura été une grande réflexion sur la structure de la base de données de l'application. Etape cruciale afin d'arriver à une architecture solide, nous nous sommes basés sur le travail de maquettage précédemment établi et validé par le client pour déterminer le dictionnaire de données, ainsi que le modèle relationnel.

La tâche étant assez ardue, et après plusieurs ébauches via JMerise, finalement nous avons eu l'aide bienvenue de Mr Riehl (spécialiste dans le domaine) afin de nous épauler.

Avec lui nous avons donc établi un diagramme de classe, ce qui correspondait d'ailleurs plus logique afin que, plus tard, dans Symfony, ce soit plus naturel pour la création des entités avec Doctrine.

Nous avons par ailleurs choisi une approche du Front vers le Back. Mais l'établissement de ce diagramme de classes nous paraissait absolument nécessaire avant tout début de code.

Cette première période nous aura pris quand même une semaine complète, en plus de terminer notre formation sur Symfony et React.

3.2 : Le développement de la partie Front

À la suite de l'établissement du diagramme de classe, nous nous sommes enfin lancés dans le développement.

La première chose aura été de mettre en place tout l'écosystème de développement. La création de la base Symfony, avec ApiPlatform, la configuration de la partie React, et le branchement entre les deux avec WebPack.

Nous avons en outre mis en place un dépôt GitHub, défini les règles pour son utilisation afin d'éviter au maximum les conflits de branche.

Et nous avons donc le reste du temps commencé réellement le développement de la partie React de l'application.

C'est à ce moment-là, lorsque les premiers visuels furent prêts, que nous avons mis en place une méthode « pseudo-Agile » afin de transmettre à l'entreprise le plus fréquemment possible les avancées au niveau de l'interface. C'était nécessaire vu que, la plupart du temps, nous étions en télétravail. Car il a été convenu que nous déplacer tous les jours dans les bureaux de Villeneuve d'Ascq n'était pas utile, car de toute façon personne n'aurait pu chapeauter notre travail sur place.

Le développement s'est donc fait sans Back fonctionnel. Mais à l'aide d'un Mock, d'où l'importance d'avoir fait au préalable le diagramme de classes afin d'avoir la bonne structure de données.

Néanmoins, je nuancerai le fait que cette partie ait été full-Front car, même si cela n'a pas été implémenté dès le départ, la partie authentification a été faite également pendant cette période.

3.3 : Les déplacements sur chantier

Dès le départ il a été également conclu que nous aurions des déplacements à faire sur site, afin de voir et de comprendre in situ le fonctionnement que devra avoir l'application.

Ce fut une étape très importante et très instructive ! En effet, comme je l'ai déjà mentionné, nos métiers sont très différents et il arrivait fréquemment que nous parlions la même langue, mais pas le même langage. Ainsi une immersion sur les futurs sites d'utilisation ont permis de lever des incertitudes, de préciser certains fonctionnements, et de pouvoir s'approprier nous-même les contraintes du métier pour lequel nous développons l'application et en quoi elle pourra leur simplifier le travail.

Nous avons donc effectué deux déplacements :

A Bousbecque en métropole lilloise, sur un chantier de logements HLM en voie de finalisation.

A Plaisir, en région parisienne, sur un chantier également de logements HLM mais beaucoup plus important et moins avancé.

Ainsi, en suivant le client dans ses inspections sur chantier, il nous notifiait par moment certaines tâches que devrait automatiser l'application.

Par exemple, pour la création des rapports, il existe une partie nettoyage, car si l'entreprise ayant précédemment travaillé dans le secteur ne l'a pas correctement effectué, il est à sa charge de le faire, ou de payer pour le faire faire.

La photo ci-contre est un exemple de non-respect du nettoyage, et l'application devra permettre de le notifier dans le rapport qu'elle générera, en y adjoignant la photo et en précisant l'endroit.

3.4 : Présentation du projet au gérant

À la suite de notre visite du site de Bousbecque, nous sommes allés au siège de l'entreprise à Villeneuve d'Ascq.

Sur place, nous avons donc rencontré le gérant de l'entreprise, Mr Benoit Petit, et lui avons présenté le travail effectué à ce stade, en petit comité. Il s'est montré d'ailleurs très intéressé par le projet porté par Mr Courbez, son employé, qui lui faisait également la liaison pour lui donner sa vision de l'application définitive dans le contexte de leur activité.

C'est d'ailleurs à ce moment là qu'on été posé les questions sur la future maintenance, sur la propriété intellectuelle du produit, d'une éventuelle alternance dans l'entreprise pour 2020-2021.

Cette réunion aura été également une bonne expérience pour nous présenter le produit de notre travail, et de tenter de lui prouver la valeur ajoutée pour son entreprise.

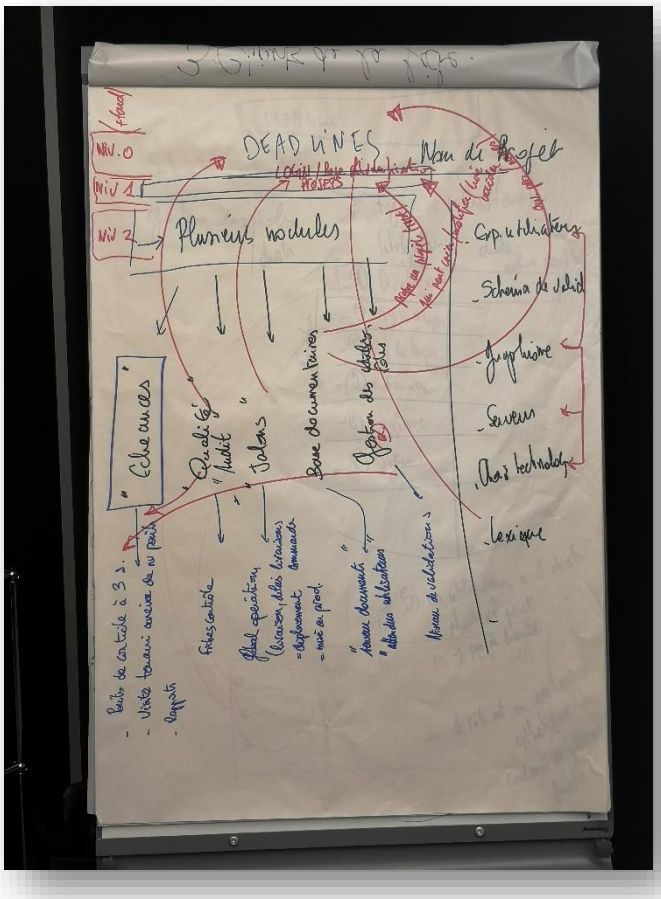
3.5 : Les « joies » de la relation clientèle

A la vue de notre travail à ce stade, et au retour apparemment positif du gérant, Mr Courbez a finalement eu d'autres nouvelles idées à implémenter dans l'application… Avec la question « est ce que c'est possible ? ». Possible oui… Mais faisable à deux dans un délai raisonnable, pas vraiment…

Il a donc fallu à ce moment vraiment préciser les limites de l'application, du moins dans sa première version. Ensuite rien n'empêcherait qu'elle puisse évoluer en y adjoignant des fonctionnalités supplémentaires.

En effet, il s'est mis à espérer pouvoir faire en sorte que l'application devienne à terme une solution complète de gestion de projet (et non plus uniquement limitée aux chantiers de construction). A ce moment nous avons dû vraiment limiter et cadrer les choses, car on sortait totalement du cahier des charges initial…

A ce moment également un nom provisoire a été trouvé pour l'application : Deadlines.



La vision de Mr Courbez sur le futur de l'application est réaliste… Mais non réalisable pour le moment… Nous avons donc dû recadrer les priorités…

3.6 : Conclusion de la période

Ce mois de stage fut donc très prolifique, autant en nous familiarisant de façon bien plus concrète avec React et le développement de toute la partie Front, qui en grande partie était achevée à ce moment (ne manquait que la résolution de certains bugs gênants et la réalisation de la partie panneau d'administration), autant sur le plan personnel en découvrant le secteur du bâtiment et la relation client/développeur qui peut être parfois compliquée à gérer.

4 : La période inter-stages

Cette période fut très particulière… Comprenant le confinement dû au Coronavirus, les cours ont été, après un temps d'organisation, dispensés en télétravail.

Concernant le projet, ces événements auront marqué un point d'arrêt dans les relations que nous avions avec l'entreprise. En effet, durement touché par cette crise, Nortec a dû s'organiser comme toute autre entreprise. Et l'application est donc tombée en quelque sorte dans les oubliettes… Ce qui aura des répercussions sur la dernière période de stage, où des choses envisagées n'auront finalement pas eu lieu.

En ce qui concerne son développement, la toute nouvelle organisation, ainsi que les difficultés personnelles, l'auront fortement ralenti. Mais même dans ces conditions, il a continué, corrigeant surtout des bugs coté Frontend.

Le plus gros soucis à ce moment fut donc la grande raréfaction des retours du client que nous avions auparavant…

5 : Le stage du 02/06/2020 au 29/06/2020

Nouvelle période de stage officielle, nouvelles difficultés.

En effet, durant cette période, il nous a été demandé à Vincent et moi, en plus du stage pour Nortec, de dispenser les cours HTML et CSS à Valarep à la nouvelle session qui venait de débuter, à raison de deux jours et demi par semaine. Diminuant ainsi notre temps disponible à consacrer au développement de l'application.

Néanmoins, pendant cette période nous avons enfin commencé le Backend, avec la création de toutes les entités avec Doctrine, et nous avons véritablement commencé le travail de branchement entre l'application React et l'API Symfony.

Il était prévu que nous fassions une présentation PowerPoint de l'application devant toute l'entreprise… Ce qui n'aura jamais eu lieu finalement…

6 : La période post-stages

Le développement aura finalement repris sur des chapeaux de roue pendant l'été, malgré nos autres obligations comme par exemple la recherche de notre contrat d'alternance pour l'année prochaine.

Durant cette période, et ce même si les rapports avec le client ont continué d'être quasi-inexistants, nous avons finalisé toute la partie administration, création-modification des projets, création-modification des utilisateurs, implémentation des rôles (que nous avons finalement réduit à deux dans un premier temps), branchement de l'API avec l'application React pour la liste des projets, la modification de ceux-ci, création du système d'upload des images, et un début de l'adaptation responsive de l'application etc.…

Durant cette période, j'ai également mis en production une version que je qualifierai pré-alpha sur un serveur (pas forcément adapté mais c'est fonctionnel sur les fonctionnalités implémentées malgré quelques bugs notamment au niveau du lien des images).

Enfin, pour illustrer le désintérêt du client, que je comprends vu la situation, c'est qu'un message a été envoyé avec le lien de cette version en production, et un accès pour qu'il puisse tester… Au moment où j'écris ces lignes, trois semaines ont passé et nous n'avons eu aucun retour, malgré que le message a été bel et bien reçu et lu…

6 : Conclusion

Ces différentes périodes auront donc été inégales en qualité, mais auront été très instructives sur différents plans.

Elle nous aura initié à la gestion de projet avec la relation clientèle, ce qui n'est clairement pas évident, entre la compréhension des attentes, des besoins, et leur formalisation afin de faire le développement il y a un grand travail de réflexion (transposition des besoins en un développement réaliste), relationnel pour mettre également des limites suivant les contraintes de temps (ici il n'y avait pas de contrainte de budget mais ce serait aussi à prendre en compte dans le cas d'un développement commercial)…

IV : REFLEXIONS PRELIMINAIRES

1 : Le choix des technologies

1.1 : La réflexion

Avant de débuter tout projet, il faut déterminer avec quelles technologies il sera le plus adapté. C'est une étape primordiale et très importante car elle constitue déjà un point de non-retour. En effet, cette étape doit être faite avec soin, car commencer à développer avec des technologies inadaptées est une perte de temps, et de plus le résultat pourrait ne jamais être à la hauteur des attentes du client. Et en cas de changement… Il faudra reprendre l'intégralité du développement…

Nous avons fait notre choix en fonction de deux critères primordiaux :

- Des technologies toujours maintenues et libres !
- Une communauté conséquente et active.

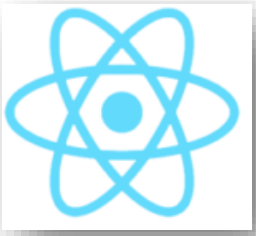
1.2 : Une application Symfony pour le Back

Symfony est un puissant Framework MVC libre de droit écrit en PHP, développé par la société française SensioLabs d'abord pour ses propres besoins, et qui l'a partagé avec la communauté PHP par la suite. Il est devenu le plus populaire au fil du temps. Avec une très solide communauté. Il est actuellement en version 5.

Dès l'issue de la réunion préliminaire, nous étions d'accord sur un point : ne pas faire le back end en PHP pur. Et ayant déjà commencé à toucher à Symfony, ce choix de Framework nous a semblé évident.

En effet, grâce à lui, nous pouvons déjà penser à une gestion « simplifiée » de la base de données grâce à l'ORM Doctrine, pouvoir profiter de son système puissant d'injection de dépendances, de toutes les librairies existantes, et nous appuyer sur la communauté en cas de soucis rencontré pendant le développement.

1.3 : Une application React pour le Front



React est une bibliothèque JS libre développée par Facebook. Elle permet de faciliter la création d'application Web monopage via la création de composants dépendant d'un état et générant une page HTML à chaque changement d'état.

Pour la partie Front, nous avons hésité entre Angular et React. Nous voulions de toute façon utiliser une technologie Front réactive, qui permet l'actualisation des informations sans pour autant rafraichir la page afin de donner une meilleure expérience utilisateur.

Pourquoi React et pas Angular ? Ce choix en revanche s'est fait de façon pragmatique… En effet, avec mes recherches, je suis tombé sur une formation de Lior Chamla ayant pour but la création d'une application React, avec un back Symfony en passant par une API Rest. Ainsi, et bien que, contrairement à Angular, React n'était pas au programme de cette année, nous avons décidé de suivre cette formation et d'utiliser cette architecture pour le projet.

1.4 : Une API Rest pour dans la pratique les lier

Le choix d'une API Rest finalement s'est avéré naturel pour lier le Back et le Front. En effet, le principe même d'une API Rest est son indépendance. Elle répond aux requêtes et envoie les données demandées sous une forme standardisé (dans notre cas ce sera en JSON) et cela correspond bien à la volonté du client d'évolution de l'application avec l'éventualité de développement d'une application mobile dédiée. Ainsi, dans cet optique, il n'y aurait que le client à développer, le back étant indépendant et renvoyant toujours les mêmes informations dans le même format suivant les requêtes effectuées, ce qui diminue donc le futur temps de développement, permet une grande modularité en ayant la possibilité que plusieurs clients se connectent à l'API.

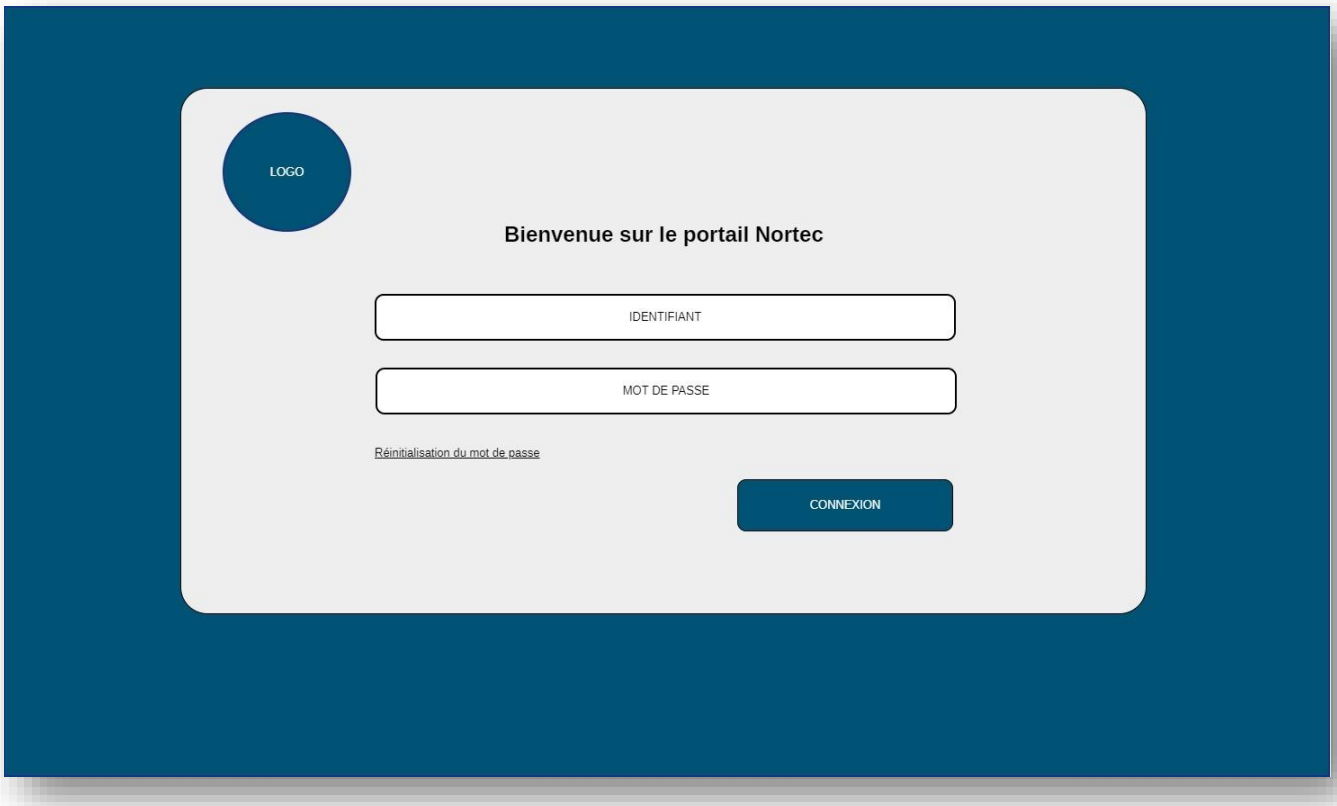
2 : Les maquettes

À la suite du choix des technologies, nous avons pu commencer le travail sur l'interface utilisateur avec le maquettage graphique de l'application. Le faire de façon précoce nous a été très utile car c'est à partir de ces maquettes que nous avons pu déterminer le dictionnaire de données afin d'élaborer ensuite le modèle logique de celles-ci.

Ce travail a été réalisé avec le logiciel Pencil, permettant la réalisation de maquette dans un style « fil de fer ».

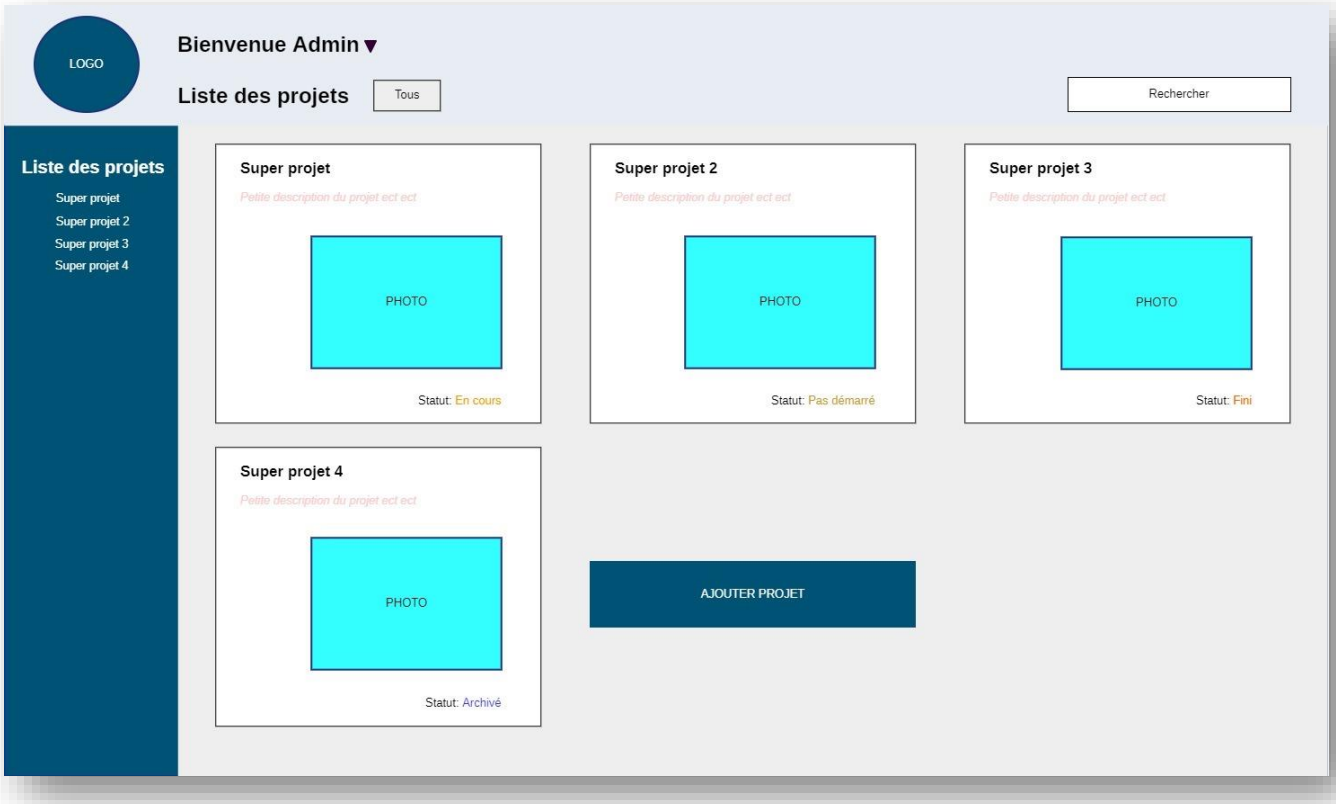
A chaque réalisation, nous envoyions, Vincent et moi, une version chacun au client pour validation, avec pour lui le choix de la version qu'il préférait. Ici je ne mettrai que les maquettes que j'ai personnellement réalisées.

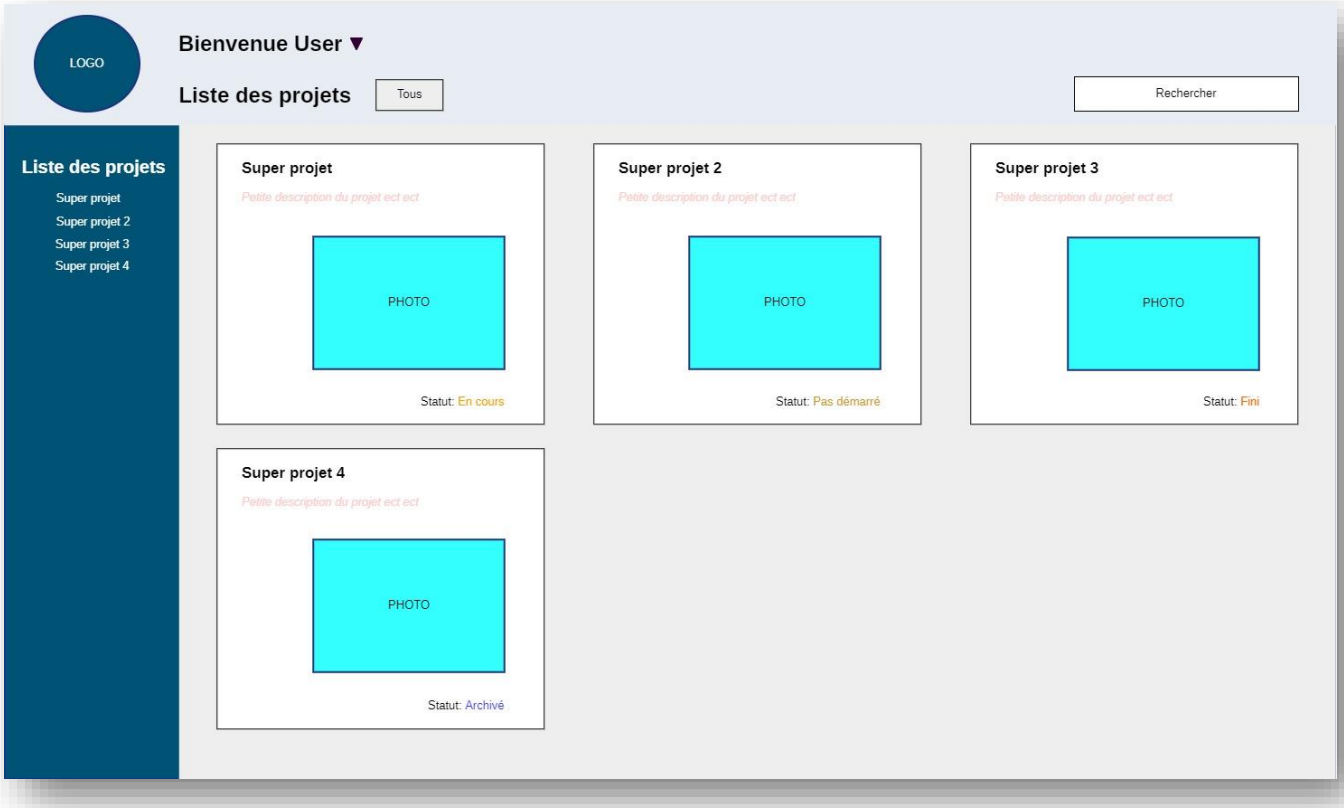
2.1 : La page de Login



La page de Login reste donc très simple. J'ai repris les couleurs du logo de l'entreprise pour lui donner une identité visuelle propre. Pas de lien pour l'inscription vu que l'application ne le permettra pas.

2.2 : La page de la liste des projets





La page de la liste des projets est celle sur laquelle nous arrivons juste après l'authentification dans l'application.

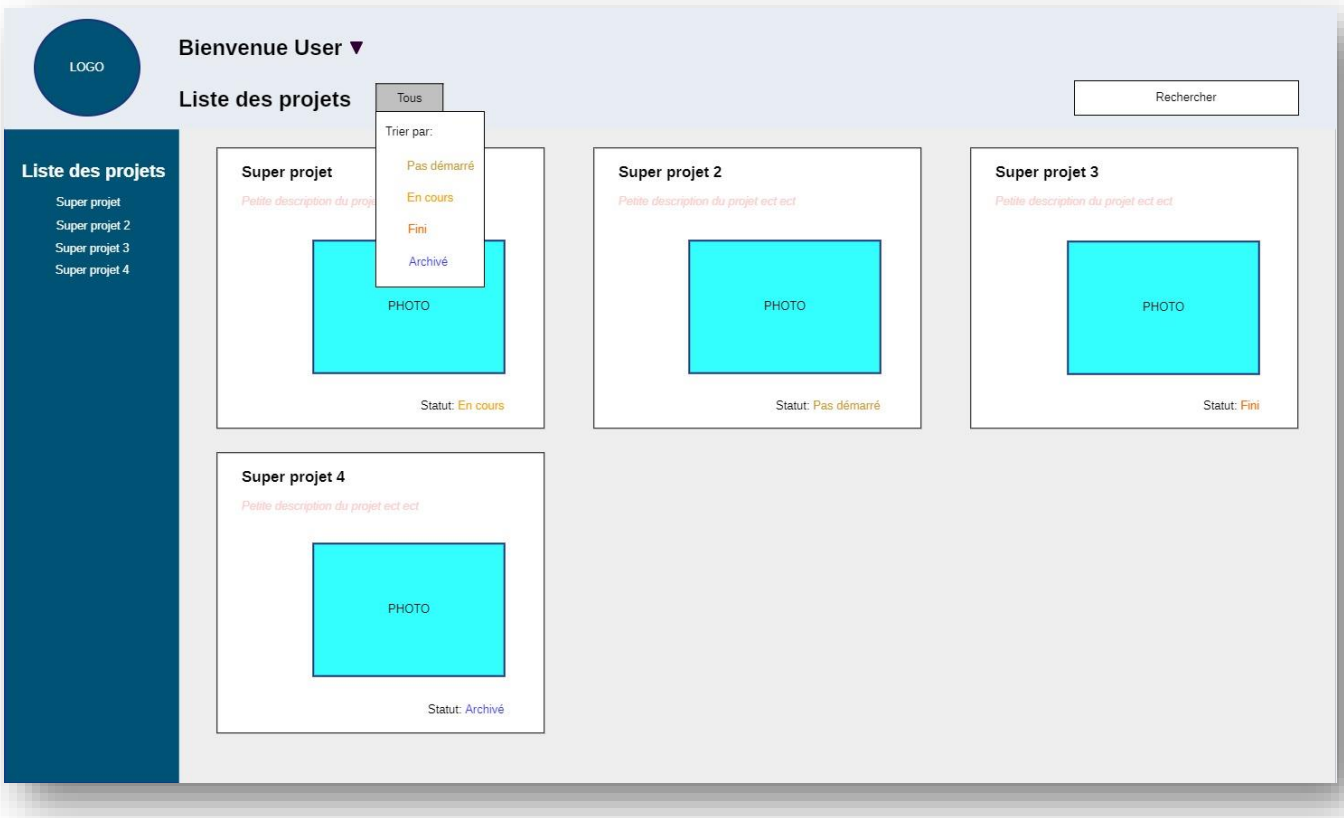
Inspirée de la disposition d'Aproplan, elle consiste en l'affichage des projets qui sont disponibles pour l'utilisateur authentifié. Avec un titre, une description, une photo et un statut de chaque projet.

Elle comporte également un champ de recherche pour retrouver facilement un ou plusieurs projets particuliers.

Un menu vertical à droite, jugé redondant par la suite, sera finalement abandonné.

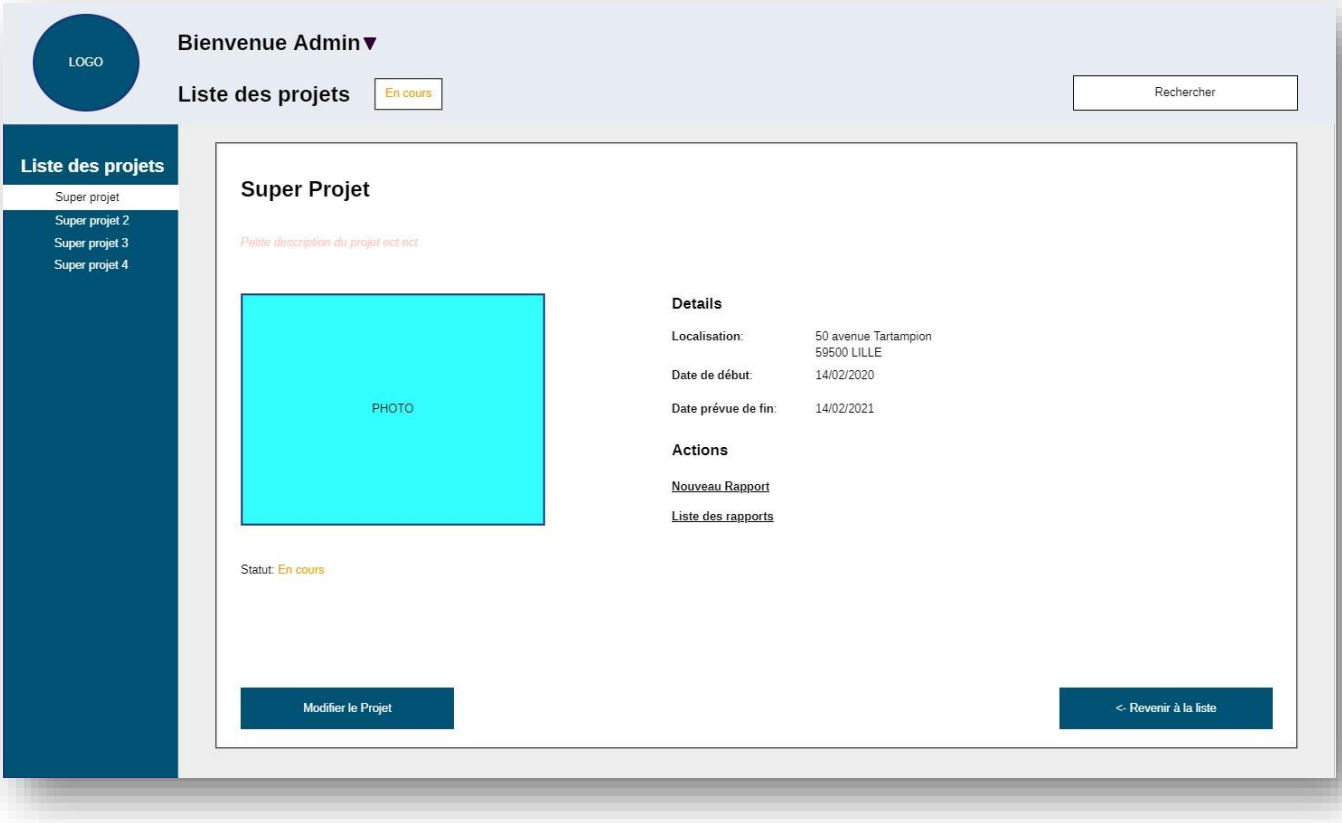
Elle existe en deux versions :

- La version pour un administrateur qui peut ajouter un projet
- La version pour un utilisateur



Un menu déroulant pour filtrer les projets suivant son statut a aussi été pensé, mais abandonné par la suite car redondant avec la recherche qui le permettra. De plus, en retour nous avons eu la demande suite à ces maquettes de ne pas afficher par défaut les projets archivés, mais qu'un bouton le permette.

2.3 : La page de détail d'un projet



En cliquant sur un projet dans la liste, nous arrivons donc sur une page avec des informations plus détaillées sur le projet sélectionné.

On y retrouve les informations générales, mais en plus nous avons toutes les informations de localisation du projet, les dates de début, fin prévue et fin réelle si celle-ci a été renseignée.

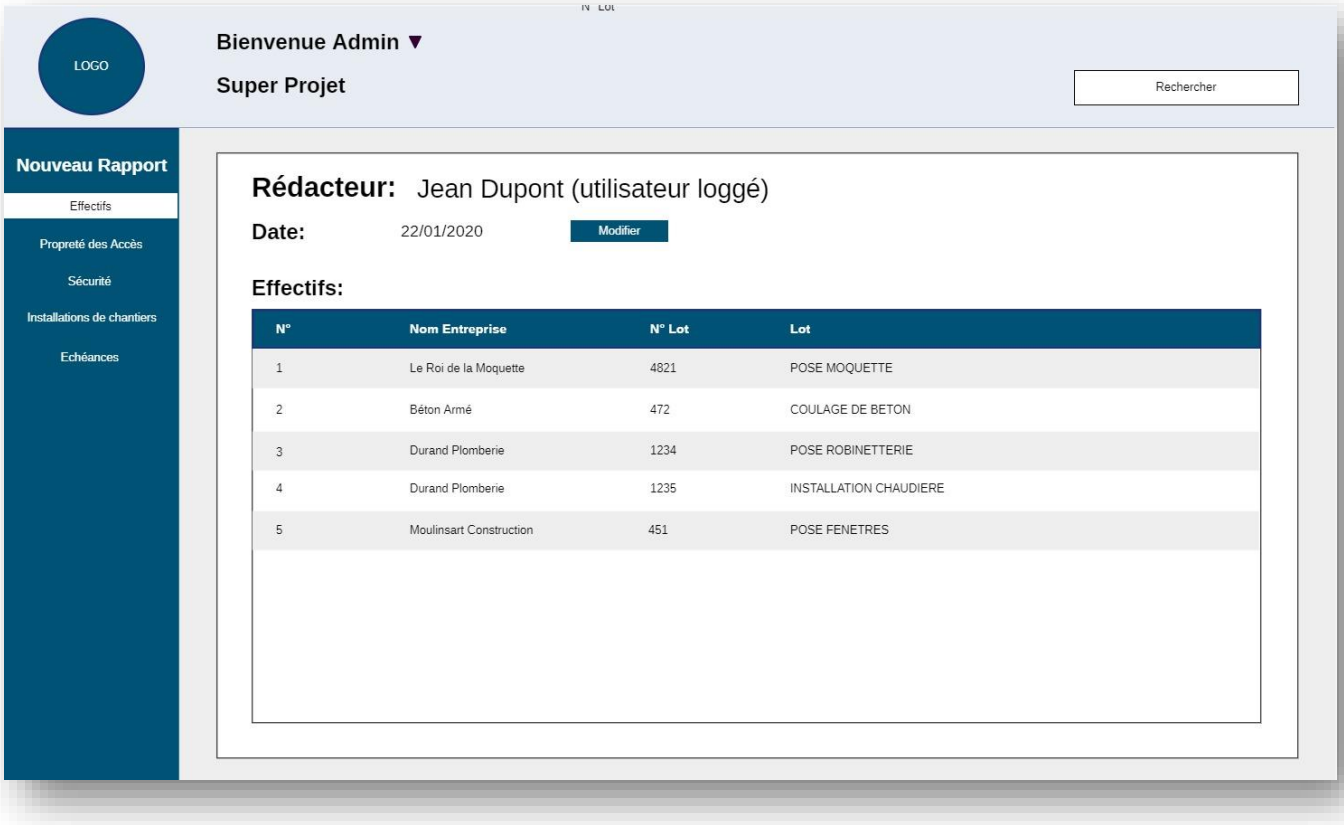
On y trouve aussi des liens pour les futures actions comme la création d'un rapport, voir la liste des rapports pour ce projet, et un bouton pour modifier le projet (uniquement visible pour les administrateurs).

Il nous sera fait la demande à ce stade la possibilité d'ajouter plusieurs dates de fin prévue (dans le cas où elle serait repoussée plusieurs fois), ainsi que l'ajout du nom de l'administrateur en charge du projet, et les coordonnées du client (une ou deux adresses mail).

Ici également à terme le menu de gauche sera supprimé.

2.4 : Les pages de création de rapport

2.4.1 : Page des effectifs



Pour chaque projet on a la possibilité de créer des rapports. Les rapports sont créés par n'importe quel utilisateur de l'application.

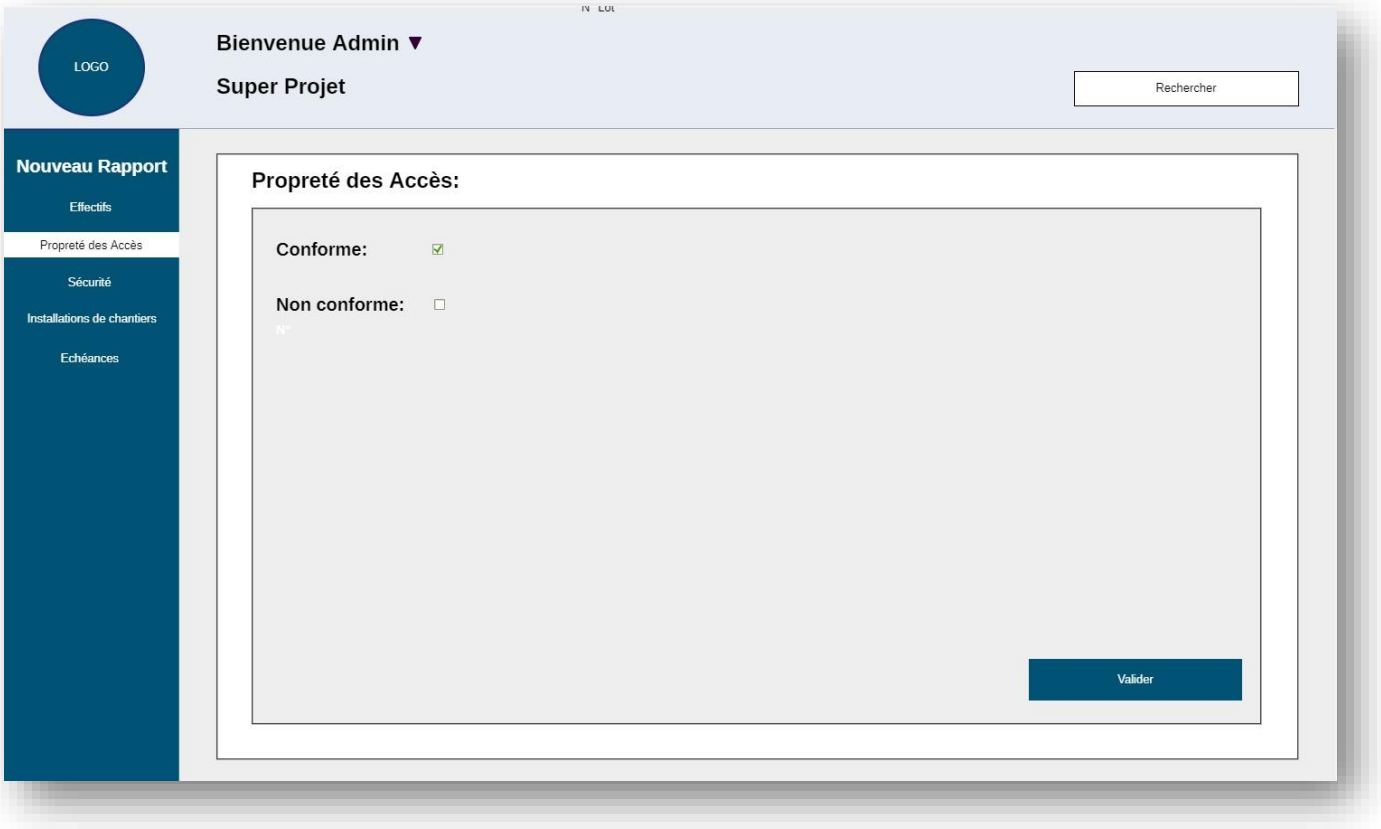
Lors de la création du rapport, la première page est un récapitulatif des effectifs présent cette semaine sur le chantier. Chaque entreprise y est donc représentée, avec son ou ses lots (lot = tâche à effectuer).

Il nous a donc été demandé par la suite d'afficher les effectifs prévu par lot et la possibilité de rajouter les effectifs réellement constatés lors de l'inspection du chantier. Les effectifs prévu par lot il est à la charge de l'administrateur de les rajouter chaque semaine.

En outre il nous a été demandé que le rédacteur du rapport soit automatiquement renseigné et non modifiable comme la personne qui créé le rapport (donc qui est loggée) dans l'application. La date de création automatiquement renseignée également mais modifiable.

2.4.2 : Page Propreté des accès

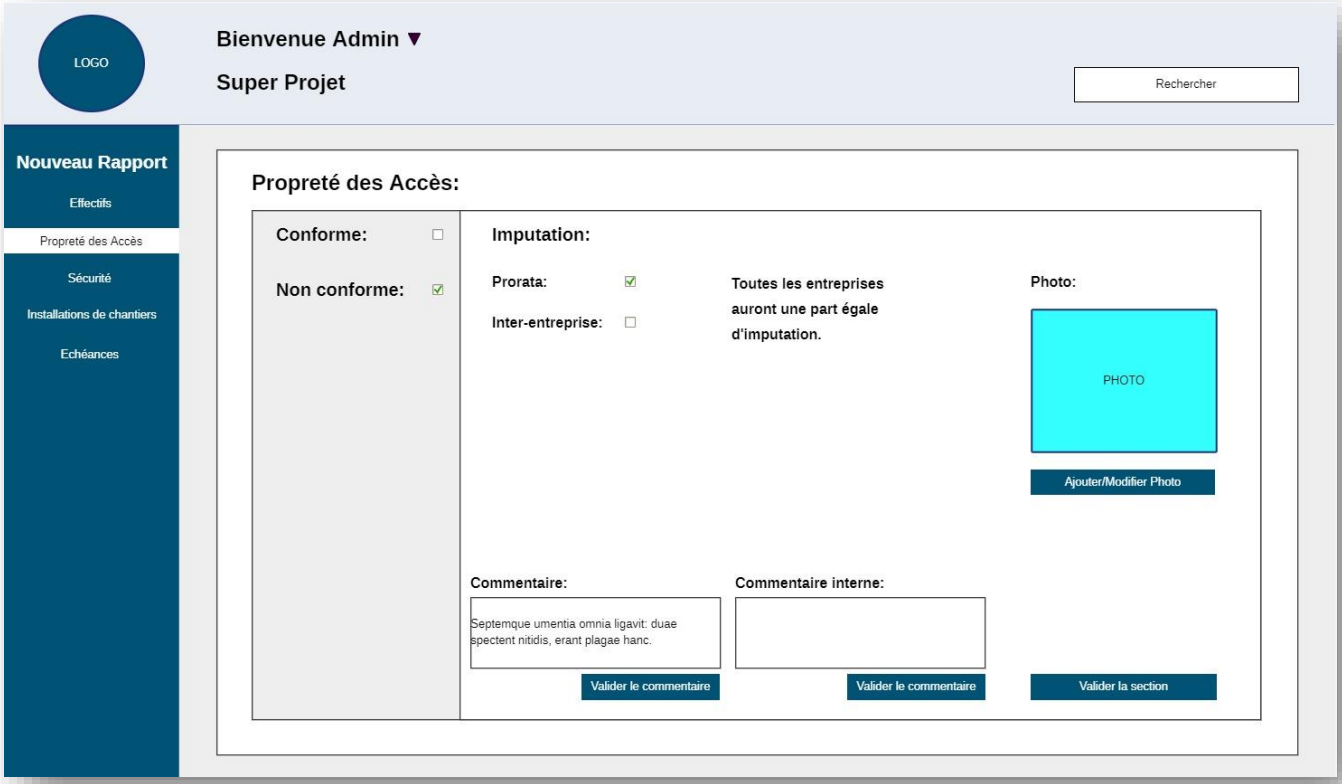
Cette page se divise en trois parties avec des designs différents selon les cas.



Dans le cas où la propreté serait conforme aux attentes, il faudra juste enregistrer son état.

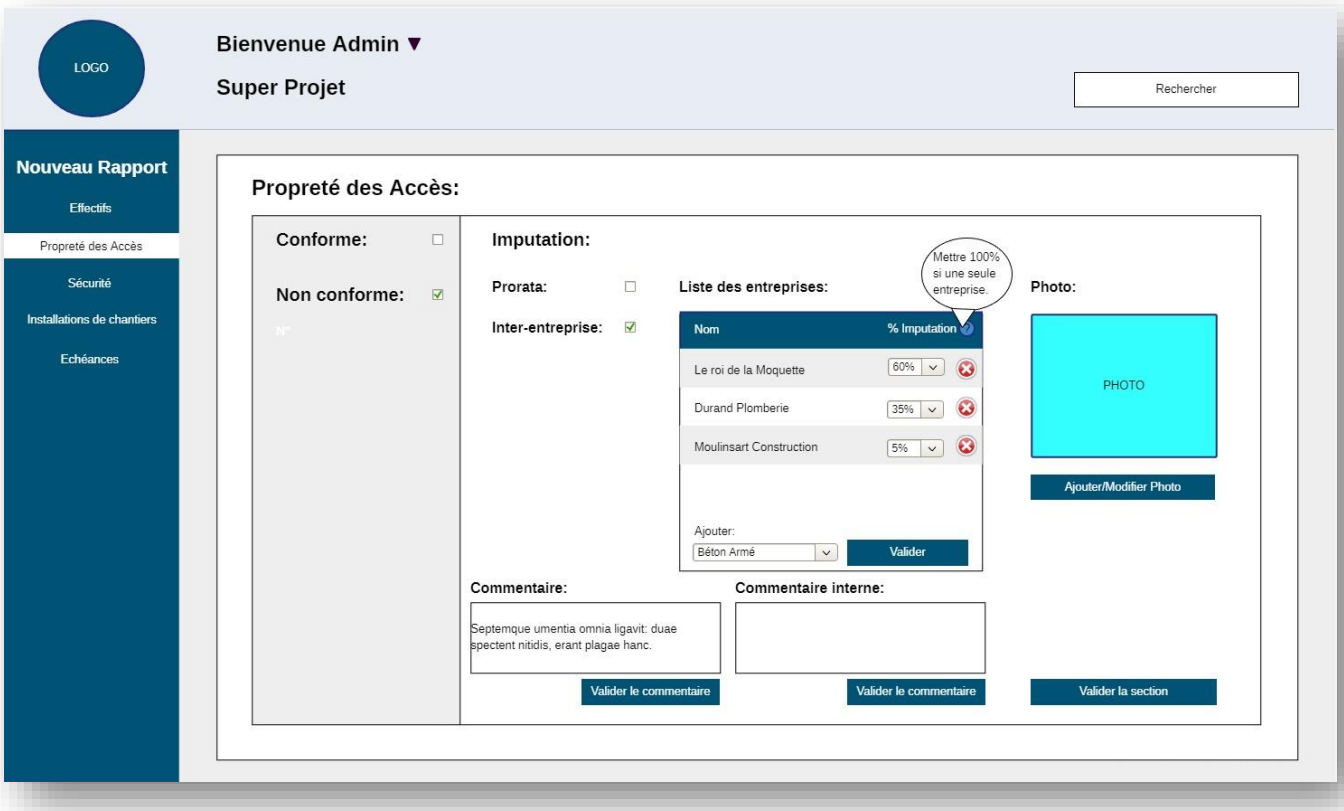
Dans le cas où il y aurait des problèmes au niveau propreté, deux choix s'offriront à l'utilisateur qui rédige le rapport. Soit il donnera une imputation dite « au prorata », soit il déterminera un pourcentage d'imputation entre les entreprises qui devront payer le nettoyage par la suite.

Nouvel exemple d'incompréhension de langage que nous avons eu ici. Nous avions compris que l'imputation au « prorata » était une imputation égale entre chaque entreprise, et donc que nous n'aurions qu'à calculer un pourcentage équitable entre elles. Mais en fait, le « prorata » signifie dans le domaine du bâtiment une caisse externe dans laquelle cotise toutes les entreprises du projet. Donc une imputation au « prorata » signifie qu'à terme les coûts de nettoyage seront prélevés dans cette caisse.



Donc ici nous nous trouvons dans le cas où l'imputation serait au « prorata ». Avec la possibilité de rajouter une ou plusieurs photos, et des commentaires, pour illustrer et justifier l'imputation.

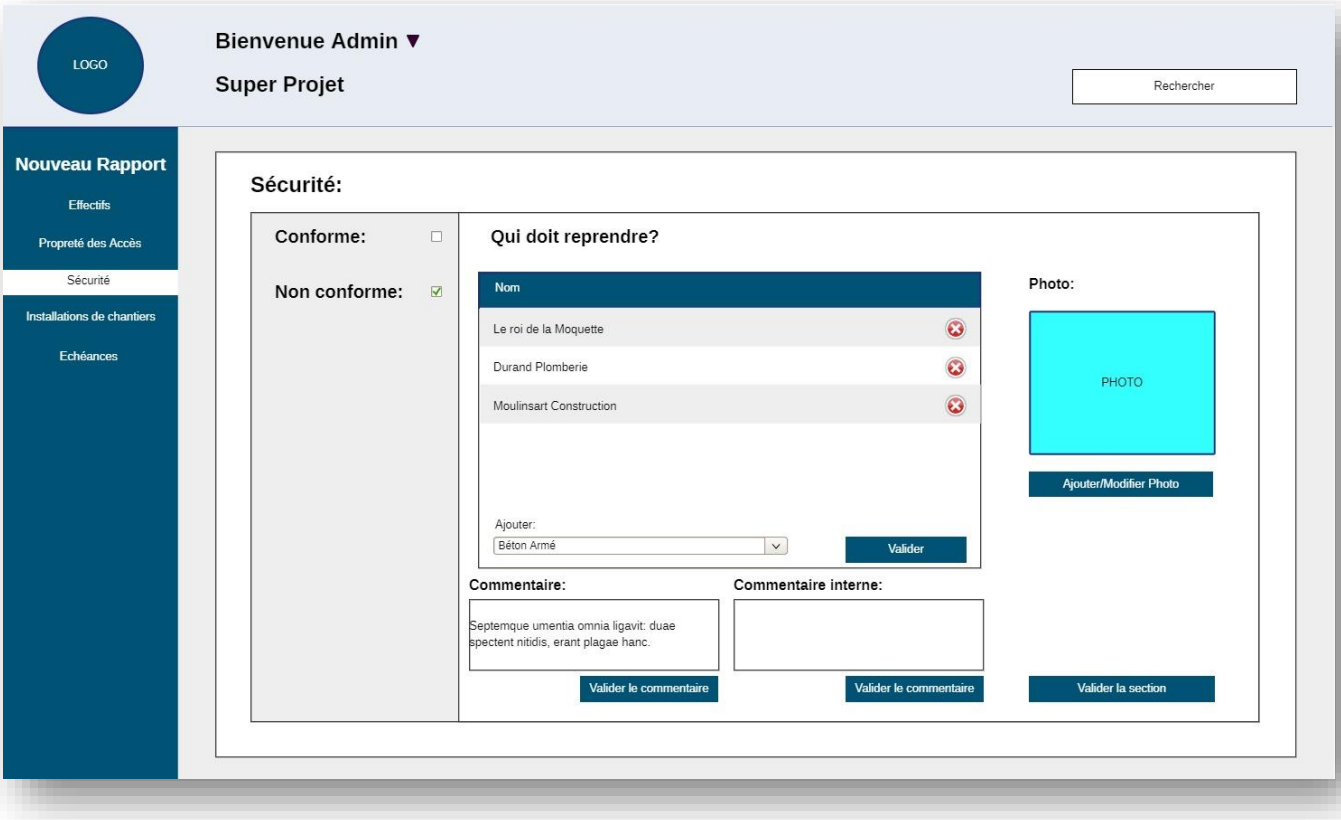
Il nous a été demandé de rajouter deux champs de commentaires. Un qui figurera sur le rapport final, et l'autre non qui restera enregistré dans l'application et donc qui restera interne.



Et enfin, dans le cas d'une propreté des accès non conforme et d'une imputation personnalisée par entreprise, on rajoute un champ avec la liste des entreprises qui seront concernées par l'imputation, ainsi qu'un champ pour déterminer le pourcentage d'imputation de chaque entreprise.

Les autres informations, comme les commentaires ou les photos sont toujours à rajouter, pour justifier du pourquoi de l'imputation.

2.4.3 : Page Sécurité

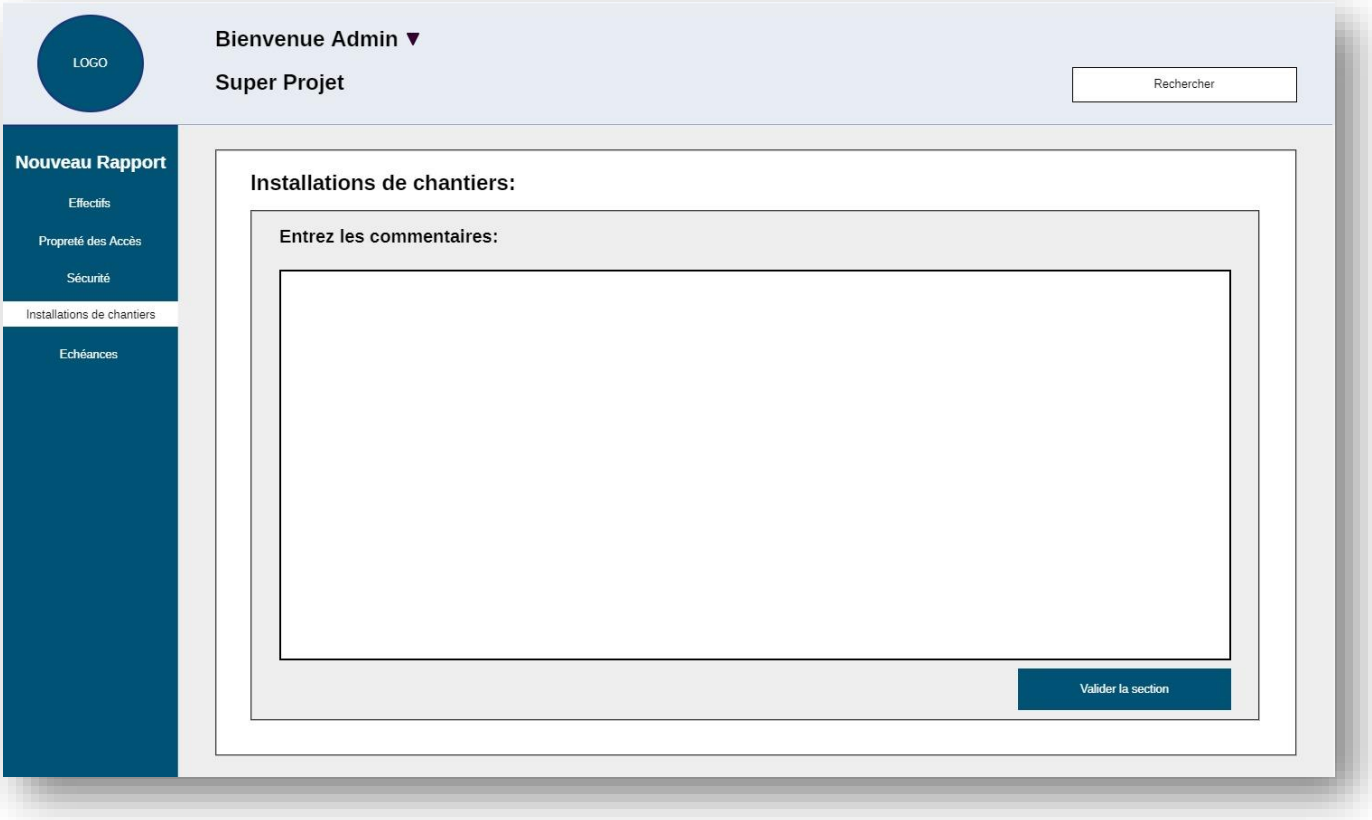


La page sécurité du rapport se présente exactement de la même façon que la page propreté. Mais il n'y a pas de prorata ni d'imputation au pourcentage par entreprise.

Ainsi nous restons sur un design très similaire, avec juste une section où l'on rajoute les entreprises qui doivent reprendre leurs problèmes de sécurité.

Entre ce travail de maquettage sur ces deux pages et ce qu'il en sera par la suite, bien après que nous ayons établi le modèle relationnel, cette partie a subi de gros changements à la demande du client… Faisant en sorte que nous avons dû reprendre une bonne partie du travail… C'est à ce moment là également que nous avons dû mettre les holà, pour éviter que nous n'ayons à reprendre toute la structure de l'application…

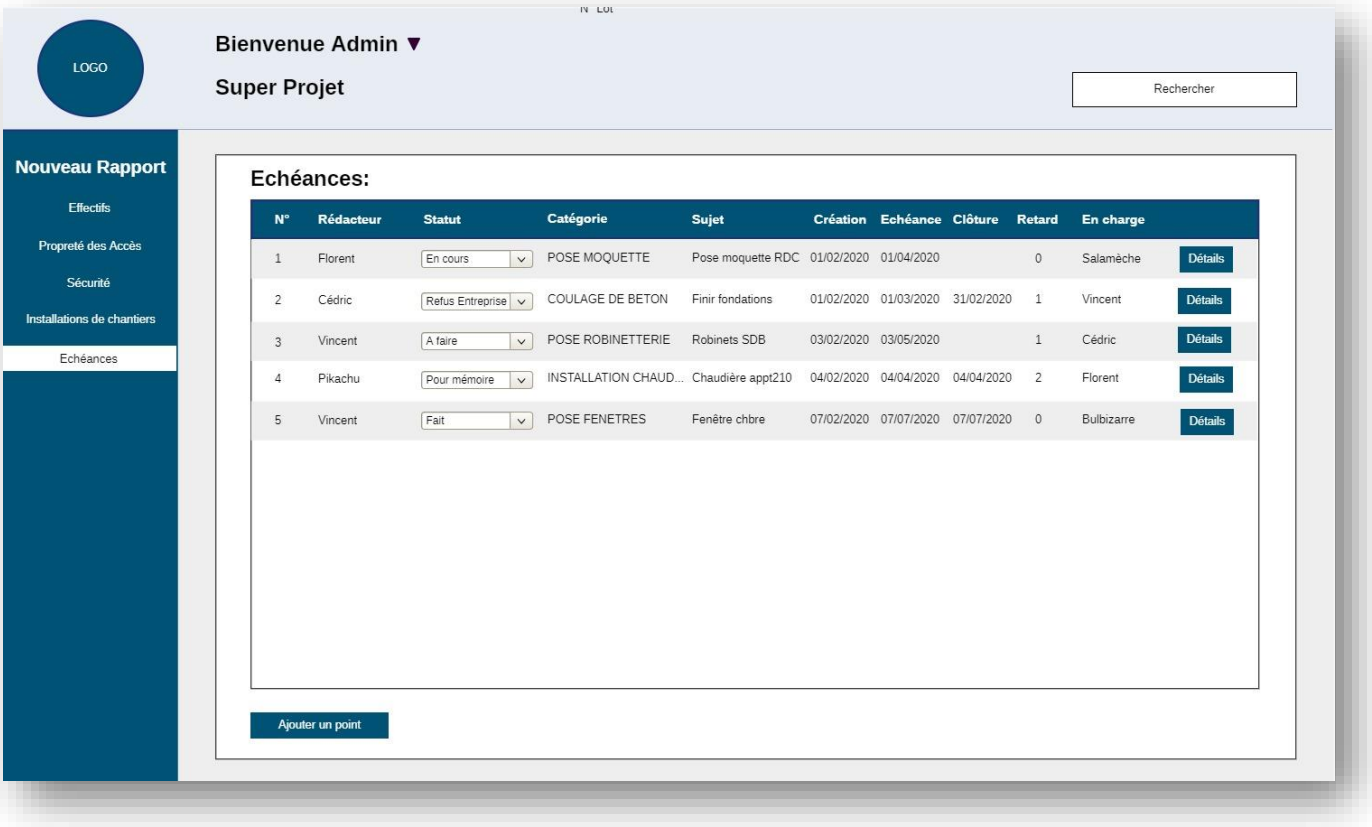
2.4.4 : Page Installations de chantiers



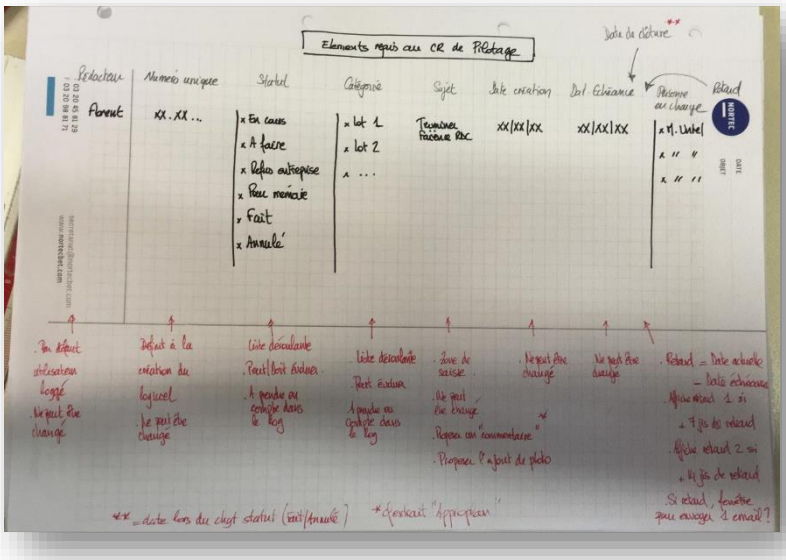
Cette page consiste simplement en l'enregistrement d'un commentaire à intégrer dans le rapport…

Plus tard cette page sera totalement supprimée à la demande du client, avec une refonte générale de la propreté et de la sécurité. Ce dont je parlais précédemment.

2.4.5 : Page des échéances



Enfin, la dernière page concernant la création des rapports est celle des échéances. Les échéances sont les tâches de la semaine, en fonction des lots (catégories), avec leurs date et leur statut afin de vérifier s'ils sont en retard ou non.



Pour la réaliser, nous nous sommes basés sur un nouveau document envoyé par le client précisant les informations qu'il voulait y voir apparaitre.

2.5 : Conclusion sur le maquettage

Cette partie, le maquettage, fut donc une étape primordiale dans la compréhension des attentes du client. Avec une communication régulière entre nous, nous avons pu lever le doute sur la plupart des fonctionnalités encore nébuleuses qui étaient attendues.

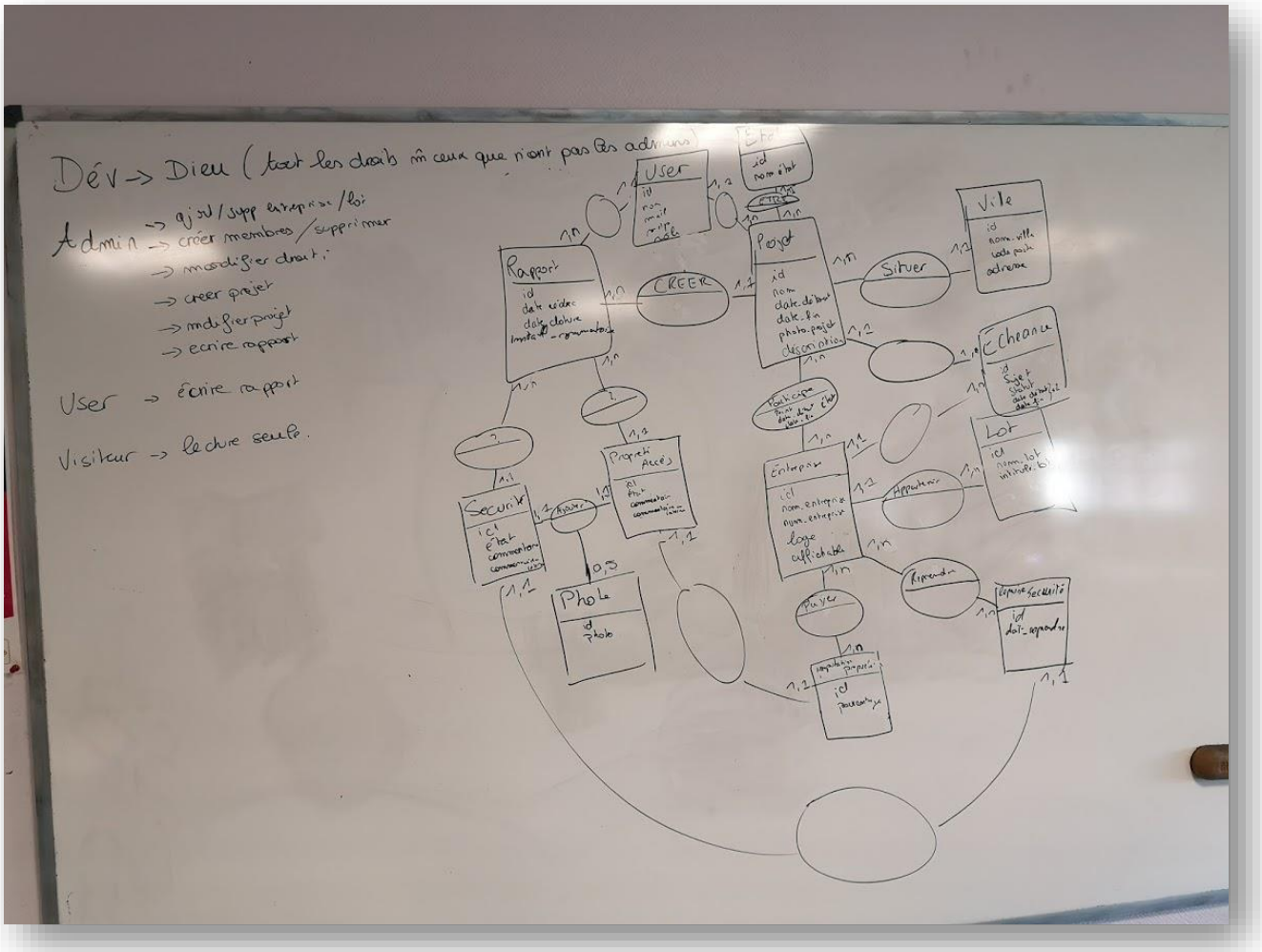
De plus, en procédant de la sorte, le dictionnaire de données de l'application est apparu naturellement, au fur et à mesure des corrections faites en fonction des remarques du client.

Après ce travail, il restait encore quelques fonctionnalités pas totalement comprises, mais qui se sont éclaircies plus tard lors des visites de chantier. Et également il y a eu beaucoup de changements de la part du client en cours de développement au fur et à mesure que de nouvelles idées lui venaient… Donc, même si la maquette reste proche du résultat final, beaucoup de changements sont intervenus en aval de ce travail.

Néanmoins, pour donner suite à celui-ci, nous avons pu commencer à nous pencher sur la structure de la base de données, ce que nous verrons dans la prochaine partie.

3 : La création du modèle logique de données

3.1 : Tentatives d'élaboration du MCD

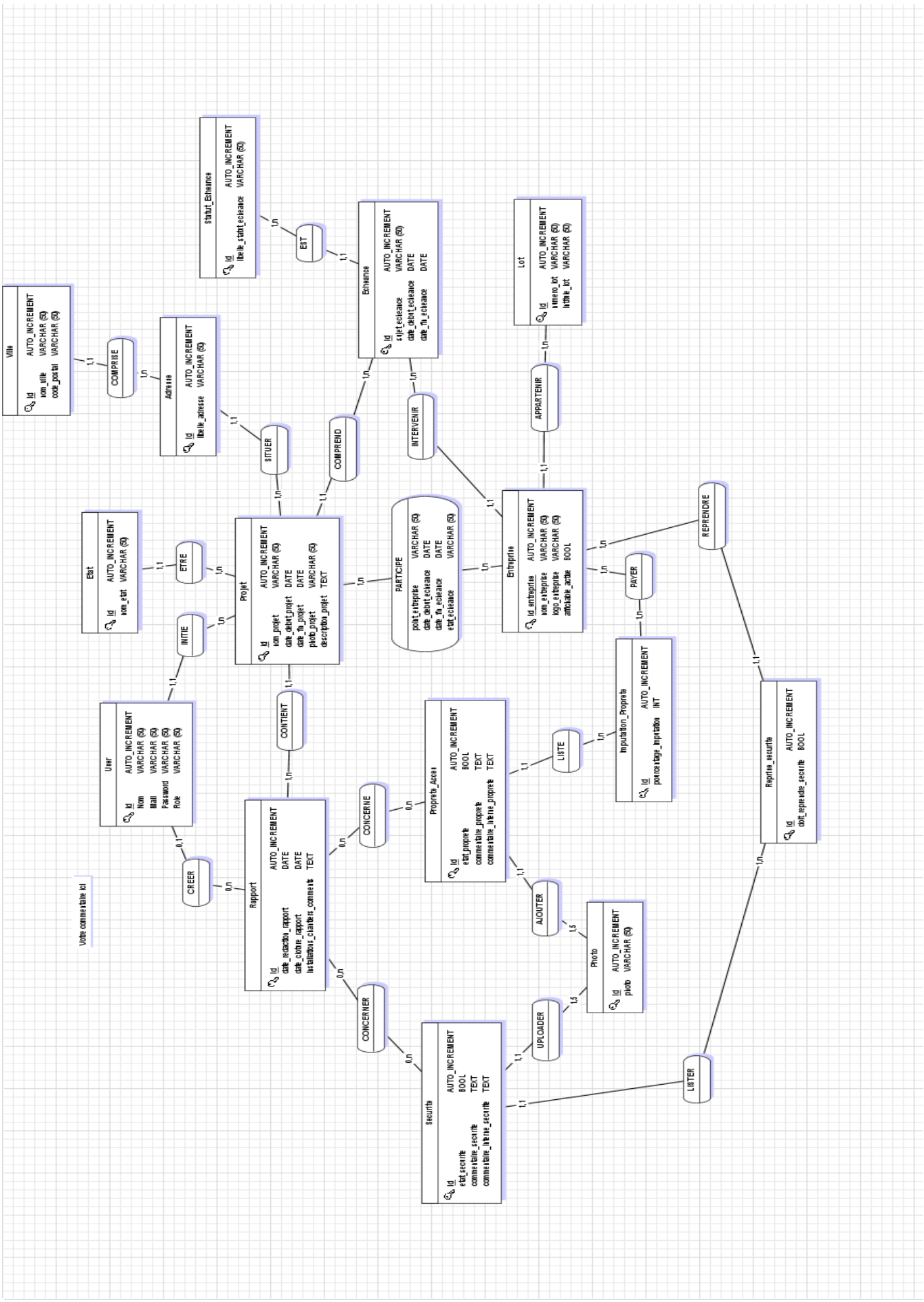


A partir de là, nous avons tenté de réaliser le modèle conceptuel de données de l'application. Il y a beaucoup d'interactions possibles entre chaque partie, et ce fut une tâche particulièrement ardue…

Ainsi nous avons commencé, au tableau, à en faire une ébauche, que nous soumettions à Mr Riehl pour être sûr de ne pas avancer dans la mauvaise direction.

Nous avons également commencé à lister les différents rôles et responsabilités de chacun d'entre eux que l'application devrait avoir.

Sur la page suivante, se trouve la version la plus aboutie du MCD que nous avons faite, et formalisée avec JMerise.


Ville
id
nom_ville
code_postal
COMPRISE
Adresse
id
libelle_adresse
Statut_Echeance
id
libelle_statut_echeance
EST
Echeance
id_pret_echeance
date_debut_echeance
date_fin_echeance
Lot
id
numero_lot
libelle_lot
APPARTENIR
SITUER
COMPREND
INTERVENIR
Etat
id
nom_etat
ETRE
Projet
id
nom_projet
date_debut_projet
date_fin_projet
photo_projet
description_projet
PARTICIPE
point_entreprise
date_debut_echeance
date_fin_echeance
etat_echeance
Entreprise
id_entreprise
nom_entreprise
logo_entreprise
affichable_active
INITIE
CONTIENT
PAYER
REPRENDRE
User
id
Nom
Mail
Password
Role
CREER
Rapport
id
date_redaction_rapport
date_cloture_rapport
Invitations_citoyens_commente
CONCERNE
Propriete_Acces
id
etat_propriete
commentaire_propriete
commentaire_interne_propriete
LISTE
Imputation_Propriete
id
pourcentage_imputation
AJOUTER
Photo
id
photo
UPLOADER
CONCERNER
Securite
id
etat_securite
commentaire_securite
commentaire_interne_securite
LISTER
Reprise_securite
id
doit_reprendre_securite
Votre commentaire ici

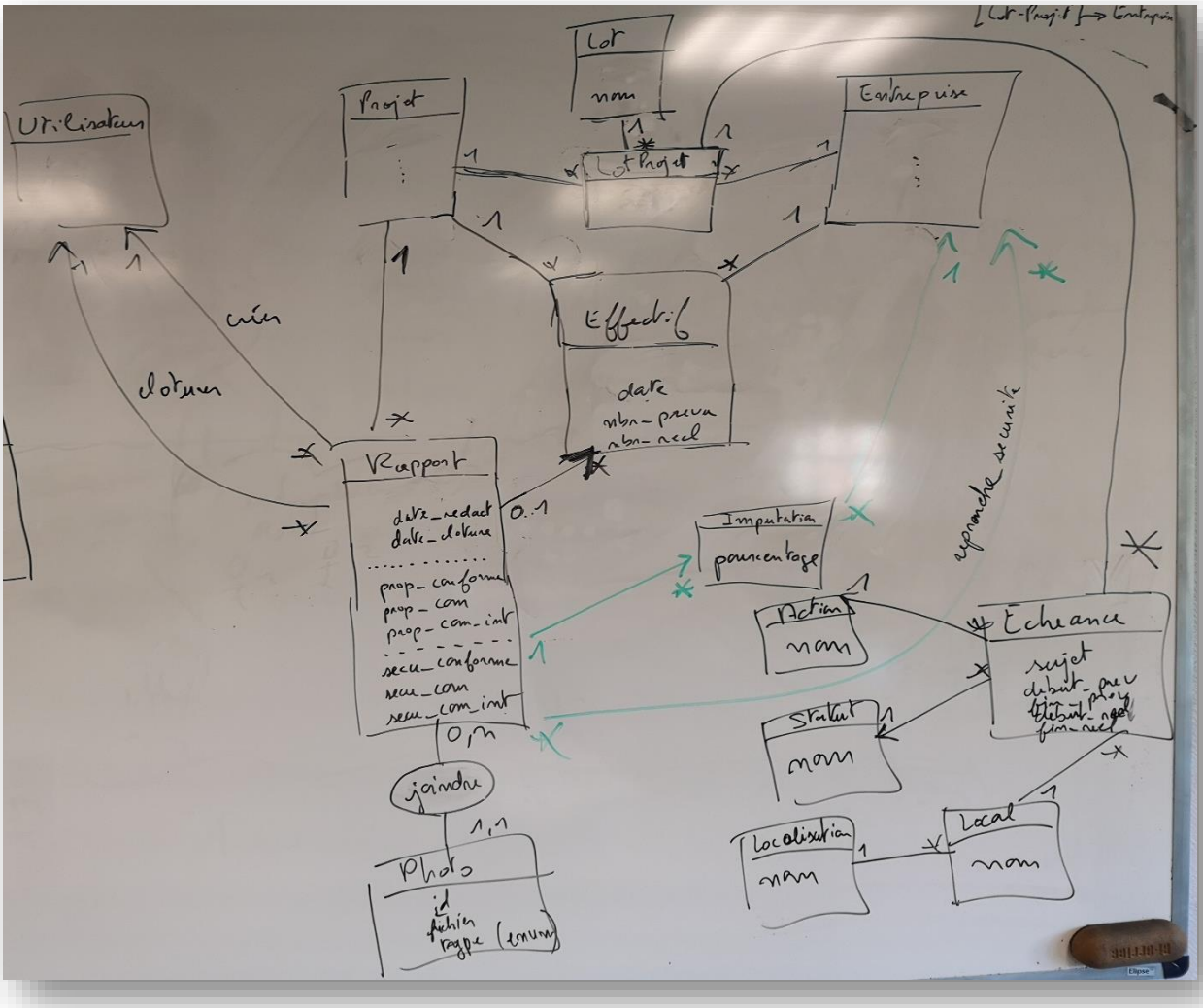
3.2 : Elaboration du diagramme de classe avec Mr Riehl

Après quelques échanges avec Mr Riehl, il nous a finalement proposé de venir nous aider à le refaire de façon plus simple.

Ainsi, après un échange à trois, est ressorti le diagramme de classe ci-dessous, qui aura servi de base à toute l'application.

Le choix pour finir de faire un diagramme de classe plutôt qu'un MCD/MLD aura été régi par le fait que nous allions utiliser Doctrine comme ORM, et ainsi il correspondait plus exactement aux entités qui plus tard seront créées. Sans le problème d'inversion des cardinalités.

Il subira quelques modifications mineures par la suite en fonction des changements que le client nous aura demandé…



V : MISE EN PLACE DE L'ENVIRONNEMENT

1 : Git, GitHub et convention d'utilisation

1.1 : Git

Git est un logiciel de gestion de version. Il permet de stocker tout l'historique du code écrit sur un projet, de visualiser les modifications faites au cours du temps, de revenir en arrière en cas de problème.

Ce n'est pas le seul logiciel de versioning existant, mais il est de loin le plus répandu dans le développement informatique. Son utilisation s'est donc avérée naturelle.

En outre, vu que notre projet s'est conçu et développé en équipe, il permet une meilleure organisation avec son système de branches et la détection automatique des conflits dans le cas où nous aurions travaillé sur le même fichier.

1.2 : GitHub

GitHub est un service Web d'hébergement et de gestion de développement logiciel utilisant Git. Développé initialement par la société éponyme, auteure également de l'éditeur de texte Atom ou du Framework Electron, l'ensemble fut acquis par Microsoft en 2018.

L'utilisation de cette plateforme s'est avérée également naturelle au vu du caractère collaboratif du projet. En effet, grâce à elle, nous avons pu partager instantanément notre code même à distance afin de toujours travailler sur la dernière version.

1.3 : Convention d'utilisation

La mise en place de ces outils est une chose. Mais il a fallu définir une convention d'utilisation entre nous afin que nous ne nous marchions pas sur les pieds, d'avoir un ensemble cohérent, et de pouvoir à terme se retrouver dans un projet qui s'annonçait déjà comme conséquent.

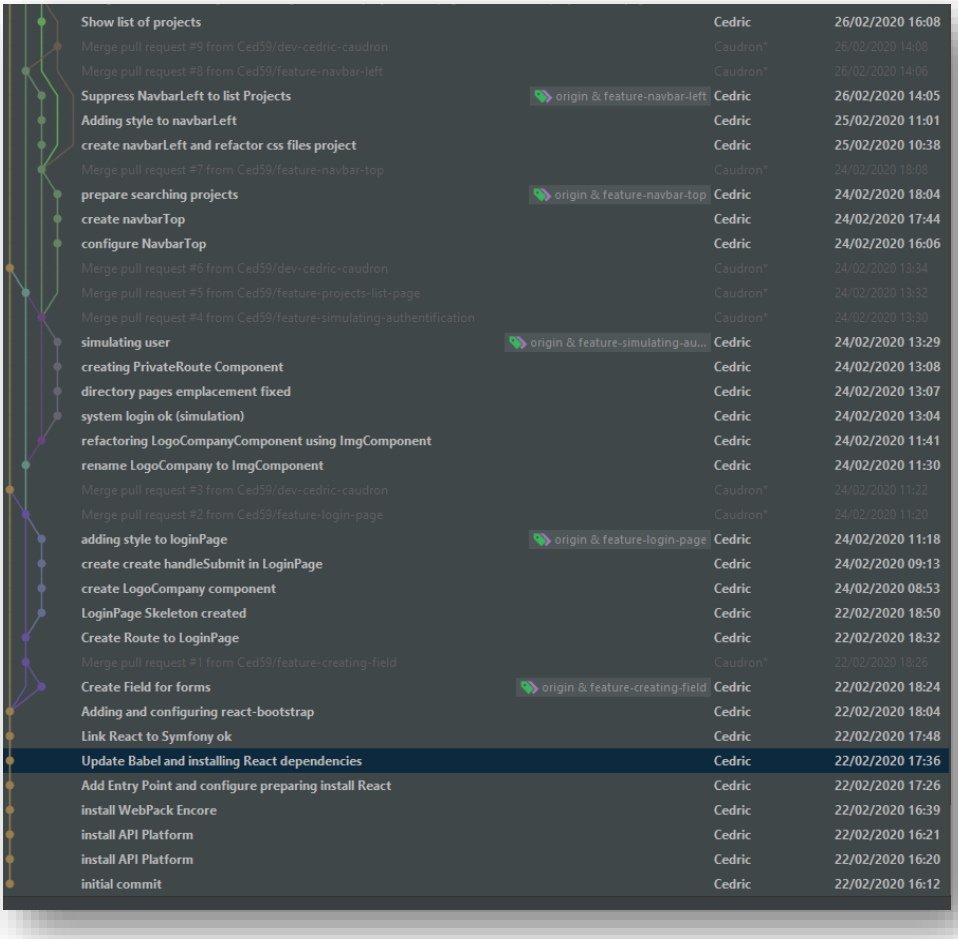
Ainsi, il a été convenu que la branche principale « master » ne servirait qu'à la mise en production. Elle fut donc très peu utilisée, mis à part pour la création du projet, et pour la première mise en production bien plus tard.

Il a été défini une branche nommée « release », qui a servi à la préproduction. Lorsqu'une fonctionnalité était terminée, sans bug, elle était partagée sur cette branche, qui servait également de branche de partage de l'état courant de l'évolution du projet. Il s'agit de la seule branche commune sur laquelle nous travaillions tout les deux.

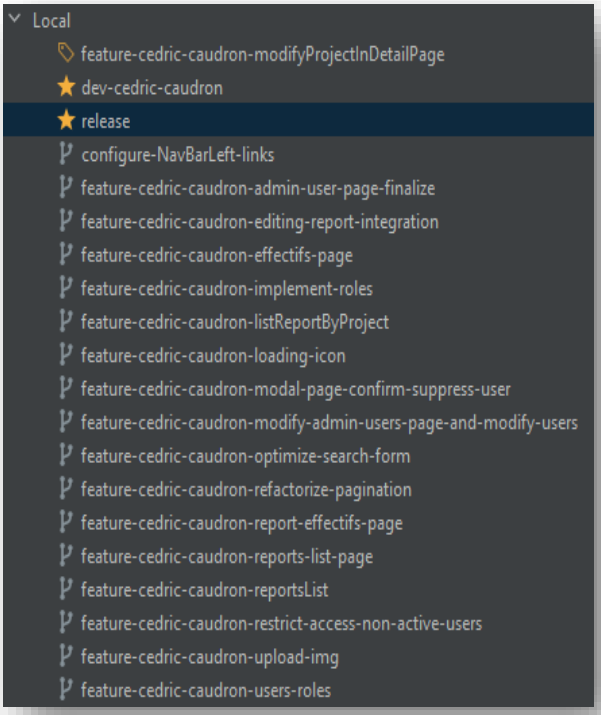
Nous avons ensuite défini chacun une branche de développement personnelle. Qui comprenait notre version personnelle du projet avec nos nouvelles fonctionnalités développées par chacun. Qui servait de base avant chaque fusion avec la branche release. Elle servait également pour apporter les modifications mineures qui ne justifiait pas à elles seules la création d'une nouvelle branche. Nous avons convenu d'une convention de nommage étant : « dev-<prénom>-<nom> ».

Enfin, pour le développement de chaque nouvelle grosse fonctionnalité, nous avons défini une convention comme quoi elles seraient développées sur une nouvelle branche basée sur notre branche dev personnelle. Nous avons également défini une convention de nommage de chacune d'entre elles afin de pouvoir s'y retrouver facilement : « feature-<prénom>-<nom>-<nom de la feature> ».

Sur la page suivante, vous trouverez donc des exemples de l'organisation de ces branches et de l'arborescence ainsi générée.



Ci-contre, l'arborescence et la chronologie des commits du début du projet.



Ici, nous trouvons une sélections des branches du projet. Avec la convention de nommage définie.

Enfin, dans ce dernier exemple, nous pouvons avoir un aperçu de l'arborescence lorsque nous travaillions effectivement ensemble.

La mise en place de cette convention nous aura permis de ne pas nous perdre dans l'ensemble de la chronologie du projet, de permettre la séparation de nos codes respectifs afin d'éviter les conflits, et de pouvoir partager notre travail facilement lorsque les fonctionnalités étaient correctement implémentées.

2 : La création de la base Symfony et de l'API

2.1 : Vérification de l'environnement

La première chose à faire, est de vérifier dans le terminal si tous les éléments sont bien installés et configurés afin de faire tourner Symfony.

Tout d'abord vérifier que PHP est bien installé dans une version assez récente.

```
C:\Users\BIENVENUE\PhpstormProjects\nortec-project>php -v
PHP 7.4.1 (cli) (built: Dec 17 2019 19:24:02) ( ZTS Visual C++ 2017 x64 )
Copyright (c) The PHP Group
Zend Engine v3.4.0, Copyright (c) Zend Technologies
```

Ensuite, il faut vérifier l'installation de Composer.

Composer est un outil de gestion de dépendances pour PHP. Il permet l'automatisation des tâches de déclaration et d'installation de librairies externes dans notre projet.

Symfony utilise pleinement Composer. Déjà pour sa propre installation et de ses propres dépendances, c'est donc un outil indispensable pour tout développeur sur ce Framework.

```
C:\Users\BIENVENUE\PhpstormProjects\nortec-project>composer -V
Composer version 1.9.2 2020-01-14 16:30:31
```

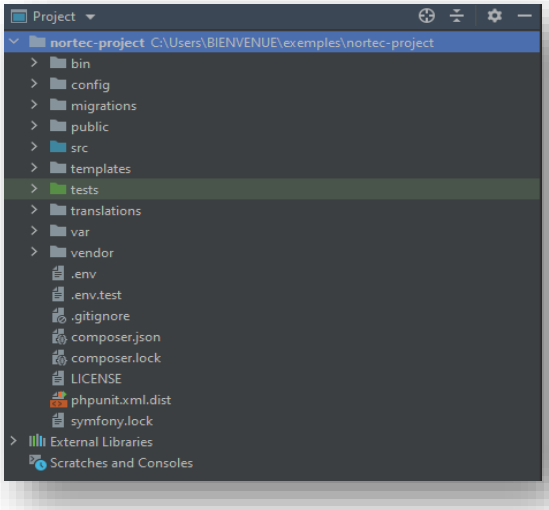
De plus, il faut vérifier la bonne installation de WampServer. WampServer est une plateforme de développement Web comprenant trois serveurs (Apache, MySQL et MariaDB). Il nous sera utile dans notre projet pour faire fonctionner localement la base de données qui sera par conséquent en MySQL.

Il y a d'autres environnements de développement similaire à WampServer. Ou nous pouvions même uniquement installer le moteur MySQL qui nous était le seul utile, mais nous l'avons utilisé par habitude et de part sa simplicité d'utilisation, notamment l'installation comprise de phpMyAdmin.

2.2 : Installation de Symfony

Nous en arrivons donc à l'installation de Symfony lui-même, qui se fait via Composer.

```
C:\Users\BIENVENUE\exemples>composer create-project symfony/website-skeleton nortec-project
```



Nous avons donc notre projet Symfony installé et prêt à être utilisé !

2.3 : Installation d'ApiPlatform

2.3.1 : Qu'est-ce qu'une API ?

Une API, ou Application Programming Interface, est une architecture applicative qui permet la communication entre différents logiciels ou services.

Ainsi, dans le cadre d'une API, plusieurs applications peuvent avoir accès à ses données et/ou services, la plupart du temps via le protocole http (ce qui sera notre cas). Elle est donc adaptée à nos besoins dans le cadre de cette application et de sa future évolution.

2.3.2 : Qu'est-ce que l'architecture REST ?

Architecture REST, ou **RE**presentational **S**tate **T**ransfert, est un standard créé en 2000 par Roy Fielding, informaticien américain.

Son but est de faire transiter une représentation de l'état (les données) de notre application vers d'autres applications (ou inversement). En utilisant différents formats de données possible (JSON (ce qui sera notre cas), XML, CSV, …).

On accède aux ressources via des URLs uniques. Et l'on peut appliquer des modifications aux ressources, ajouter une nouvelle ressource, ou supprimer des ressources.

Le standard REST répond donc à plusieurs critères :

- Client – Server : Un mode de communication avec séparation de rôles entre client et serveur
- Stateless Server : Les requêtes doivent contenir toutes les informations nécessaires au traitement. Le serveur ne connait donc pas l'état du client entre deux requêtes.
- Cache : La réponse du serveur doit être cacheable coté client.
- Uniform Interface : La méthode de communication entre client et serveur doit être uniforme avec des ressources identifiables. Ainsi, en http, la requête est identifiée avec une URL, et la réponse comprend un body et une entête.
- Layered System : Le système doit permettre le rajout de couches intermédiaires (proxy, firewall, CDN, etc.…).
- Code-on-Demand Architecture (optionnel) : L'architecture doit permettre d'exécuter du code coté client.

2.3.3 : Pourquoi utiliser ApiPlatform ?

ApiPlatform est une librairie permettant la création très simplement d'une API en bootstrappant une application Symfony.

Elle intègre entièrement les composants de Symfony, permettant l'accès aux bases de données avec Doctrine, de gérer la validation avec Symfony etc….

Elle est en outre très flexible avec énormément de paramétrages possibles, et peux exposer les ressources dans différents formats.

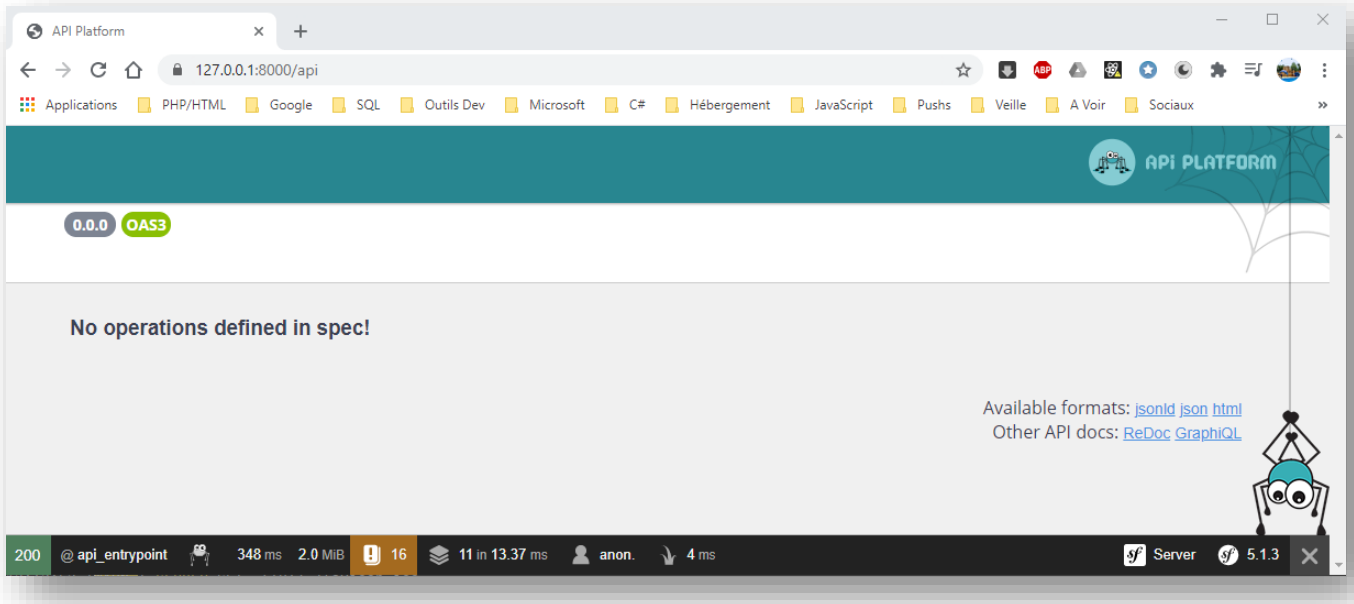
Enfin tient compte des derniers standards et spécifications sur le format que doit avoir les réponses de l'API au client, en fait avoir une représentation standardisée de nos ressources exposées.

2.3.4 : Installation

L'installation se fait tout simplement par une ligne de commande dans le terminal, qui sera géré par Symfony Flex, qui comprendra qu'il faut installer toutes les dépendances de ApiPlatform.

```
C:\Users\BIENVENUE\exemples\nortec-project>composer require api
```

Notre architecture API REST est donc déjà prête à être configurée et utilisée. Nous pouvons d'ores et déjà accéder à la page où toutes ses « entry points » (les différentes URLs et actions) seront bientôt disponibles.



3 : La configuration de WebPack

3.1 : Qu'est-ce que WebPack ?

WebPack est un projet permettant l'organisation d'une application en modules JS.

Ainsi, avec son système de loader, il permet de transformer tout et n'importe quoi en JS, et de les organiser en modules.

Les intérêts sont multiples :

- Dans le cas de notre application React donc écrite en JSX, il permet de compiler ce langage incompréhensible pour les navigateurs modernes en JS natif (grâce au loader Babel).
- Il permet de découper le build en plusieurs morceaux, chargeables à la demande, ce qui permet un énorme gain de performance.
- Il permet de disposer de toutes les ressources statiques (images, CSS, polices etc…) en tant que module.

C'est un outil populaire dans la communauté React, permettant une bien meilleure optimisation des applications.

3.2 : Configuration avec la librairie WebPack Encore

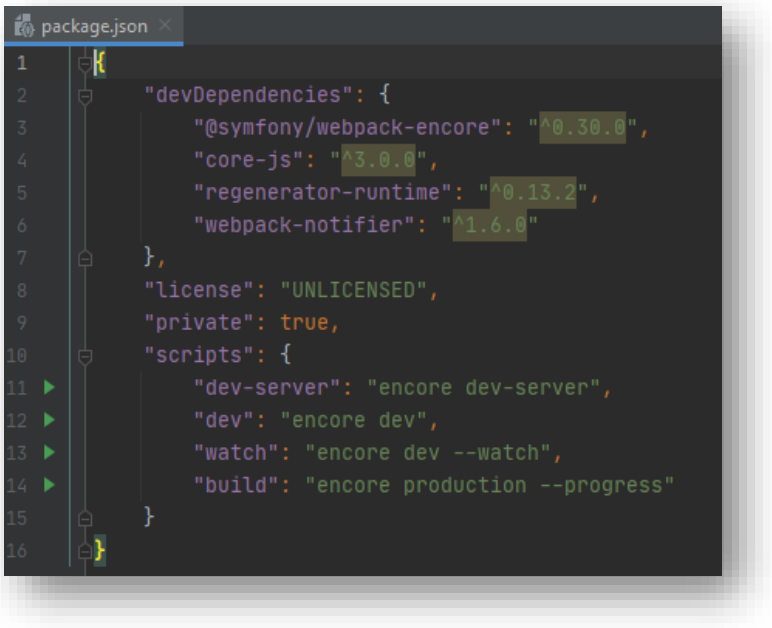
Afin de simplifier la configuration de WebPack, nous allons utiliser la librairie « WebPack Encore ».

```
C:\Users\BIENVENUE\exemples\nortec-project>composer require encore
```

Avec cette commande, nous installons les dépendances PHP nécessaires au fonctionnement de WebPack sur notre projet. Ainsi, plusieurs fichiers importants se sont créés :

- webpack.config.js : qui comprendra la configuration de WebPack
- un dossier assets : qui comprendra à terme toute l'application React
- package.json : équivalent du composer.json mais pour la partie Front, il listera toutes les dépendances de celle-ci

Maintenant, justement, nous devons installer toutes les dépendances Front de WebPack listées dans le package.json.



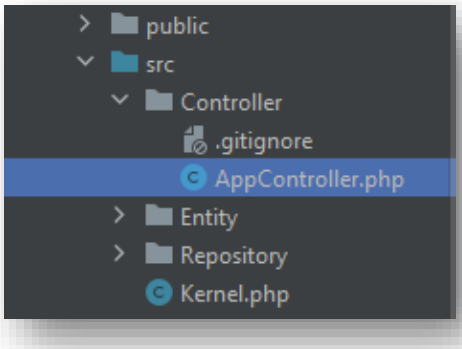
Pour se faire, il suffit d'exécuter la commande npm install dans le terminal (après bien sûr avoir vérifié qu'outil de gestion de dépendance npm était bien installé).

Les dépendances installées permettront le lancement de la compilation des fichiers en fonction de la configuration du webpack.config.js. Il va en outre, en mode production, minifier les fichiers pour un chargement plus rapide, changer leur noms et faire les liens nécessaires afin que les bons fichiers soient chargés au bon moment.

3.3 Création de la page d'accueil

Nous devons maintenant créer la page d'accueil, qui servira de point d'entrée de notre application React. Pour ce faire, nous allons créer un Controller avec Symfony.

```
C:\Users\BIENVENUE\exemples\nortec-project>php bin/console make:controller AppController
```



AppController.php

```php
<?php

namespace App\Controller;

use ...

class AppController extends AbstractController
{
    /**
     * @Route("/", name="app")
     */
    public function index()
    {
        return $this->render( view: 'app/index.html.twig', []);
    }
}
```

Nous créons donc un Controller Symfony classique. On adapte sa route pour qu'il s'active sur la page d'accueil. Et il n'a pas besoin de variable pour fonctionner, on retourne donc le template Twig 'app/index.html.twig' avec un tableau vide comme paramètre.

Enfin, il faut maintenant configurer le template Twig pour qu'il pointe vers les fichiers JS et CSS ainsi compilés par WebPack. Le problème étant qu'à chaque compilation, ceux-ci changent de nom, ainsi des liens en dur sont à proscrire. Heureusement, WebPack Encore propose des fonctions pour les retrouver dynamiquement en fonction des informations comprises dans le fichier entrypoints.json.

entrypoints.json

```json
{
  "entrypoints": {
    "app": {
      "js": [
        "http://localhost:8080/build/runtime.js",
        "http://localhost:8080/build/vendors~app.js",
        "http://localhost:8080/build/app.js"
      ],
      "css": [
        "http://localhost:8080/build/vendors~app.css",
        "http://localhost:8080/build/app.css"
      ]
    }
}
```

Les fonctions ainsi configurées vont aller chercher dans entrypoints.json les bons chemins et noms de fichiers pour faire les liens correctement au chargement de l'application.

Un dernier problème subsiste donc ici. Car Symfony tourne sur le port 8000, et nous remarquons que WebPack configure l'ensemble des liens sur le port 8080. Pour corriger cela, il suffit donc de configurer le bon port dans le fichier package.json dans la configuration du dev-server.

WebPack est donc enfin correctement configuré, et nous allons pouvoir nous atteler à l'installation de React, ainsi que s'occuper de son branchement sur le template Twig que nous avons créé ici.

4 : La création de l'application React

4.1 : Vérification de l'environnement

Tout d'abord, nous devons vérifier que Node est bien installé. Node est une plateforme logicielle libre qui permet l'exécution de JS coté serveur. Créé par Ryan Dahl en 2009, son développement et sa maintenance sont effectués actuellement par l'entreprise Joyent.

L'installation de Node est impérative afin d'exécuter des applications React ou même Angular.

```
C:\Users\BIENVENUE\PhpstormProjects\nortec-project>node -v
v12.16.1
```

Il faut également vérifier que npm, gestionnaire de paquets officiel de Node, est bien installé. Il est grosso modo l'équivalent de Composer dans l'environnement Node. Je précise que cette vérification devait être fait avant les étapes de la partie précédentes, mais elles sont nécessaires lors de la création de n'importe quelle application React, d'où le fait que je l'ai placée dans cette partie.

```
C:\Users\BIENVENUE\PhpstormProjects\nortec-project>npm -v
6.13.6
```

Une fois ces vérifications effectuées, nous pouvons nous lancer dans la création réelle de l'application React.

4.2 : Installation de React et de ses dépendances

4.2.1 : Dernière configuration de WebPack

```
// enables Sass/SCSS support
//.enableSassLoader()

// uncomment if you use TypeScript
//.enableTypeScriptLoader()

// uncomment if you use React
.enableReactPreset()
```

La dernière petite configuration à apporter à WebPack, est qu'il faut le « prévenir » que nous allons utiliser React. Ainsi, dans le fichier webpack.config.js, il nous suffit juste de décommenter la ligne qui activera la fonction.

On peut donc remarquer également que différentes configurations sont possibles suivant que l'on utilise du SCSS/SASS ou du TypeScript.

De plus, il faudra installer le preset React afin que Babel puisse traduire le JSX en JS natif.

```
C:\Users\BIENVENUE\PhpstormProjects\nortec-project>npm install @babel/preset-react@^7.0.0 --save-dev
```

4.2.2 : Installation de React et de ses dépendances

Nous pouvons maintenant installer la librairie React elle-même. Ainsi que quelques-unes de ses dépendances qui nous seront très utiles par la suite.

Nous n'utilisons pas les toutes dernières versions, ayant appris sur celles qui seront installées par la suite, et par gain de temps, nous avons préféré rester dessus afin de ne pas avoir de travail d'adaptation des concepts ou de syntaxe qui pourraient être survenues entre temps. Néanmoins, cela reste des versions récentes.

```
C:\Users\BIENVENUE\PhpstormProjects\nortec-project>npm install react@16.8.6 react-dom@16.8.6 react-router-dom@5.0.0 axios@0.18.0
```

Nous avons donc ici plusieurs packages qui vont s'installer :

- React : package de base de la librairie React
- React DOM : package qui permettra de brancher React à notre DOM, donc notre document html qui est le fichier index.html.twig que nous avons configuré précédemment.
- React Router DOM : package qui permettra la gestion des routes dans notre application React, et ainsi d'avoir plusieurs pages.
- Axios : package qui permettra d'effectuer des requêtes http très facilement, et qui permettra d'échanger avec notre API.

4.2.3 : Branchement de l'application React à la page d'accueil

La dernière chose à faire avant de se lancer dans le développement proprement dit, est donc de brancher l'application React à la page d'accueil que nous avons configuré précédemment.

Pour ce faire, dans le fichier index.html.twig, nous allons tout simplement rajouter une dans le block <body>.

Et enfin, dans le fichier app.js, qui est le point d'entrée de l'application React, grâce à React DOM, nous allons faire le lien avec cette , pour que le rendu se fasse sur cette page d'accueil.

```
1   import React, {useState} from 'react';
2   import ReactDOM from "react-dom";
```

On importe React ainsi que React DOM.

```
108
109 const rootElement = document.querySelector( selectors: '#app');
110 ReactDOM.render(<App/>, rootElement);
111
```

Puis enfin, nous allons chercher la qui a pour id 'app', et nous demandons à React DOM de faire le rendu de l'application à cet emplacement.

5 : Création de la base de données

Enfin, la dernière chose à faire est de configurer et créer la base de données de notre application.

Nous avons choisi le SGBD MySQL, qui est libre et le plus utilisé dans le monde du développement Web.

Pour le développement nous utilisons WampServer. C'est lui qui fera tourner le serveur de base de données.

Ainsi, dans le fichier .env de Symfony, nous devons définir la chaine de connexion à la base de données.

```
.env
16
17  # IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
18  DATABASE_URL=mysql://root:@127.0.0.1:3306/deadlines?serverVersion=5.7
19  ###< doctrine/doctrine-bundle ###
20
```

Enfin, grâce à Doctrine, nous pouvons simplement faire la création de la base de données via une ligne de commande.

```
C:\Users\BIENVENUE\exemples\nortec-project>php bin/console doctrine:database:create
Created database `deadlinesExample` for connection named default
```

6 : Conclusion

Après avoir mis en place tout cet environnement de développement, assez complexe, comprenant donc plusieurs technologies, nous allons enfin pouvoir commencer à se lancer dans le développement de l'application proprement dit.

Nous avons pour cela appliqué une approche du Front vers le Back afin de donner rapidement des visuels et des mécaniques au client, dans le but de pouvoir apporter des modifications facilement en cas de changement.

VI : L'AUTHENTIFICATION

1 : Introduction

A partir de maintenant, nous entrons dans le développement à proprement parler.

Par soucis de synthèse et de compréhension, je n'aborderai pas les différents sujets par ordre chronologique, mais par thèmes. En effet, beaucoup de notions sont redondantes dans différentes partie du code, et cette approche me permettra de les expliquer de manière globale sans pour autant avoir à y revenir plusieurs fois.

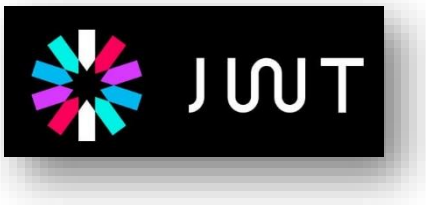
Pourquoi commencer par l'authentification ? Car la gestion des comptes utilisateurs est un point très important de l'application, et qui imprégnera l'ensemble des pages. De plus, faisant intervenir des notions Front comme Back, je me devais de faire une partie distincte pour aborder ce système.

2 : JWT

2.1 : Définition

Contrairement à une application qui aurait eu une architecture plus traditionnelle, le serveur, dans le cas d'une API REST, ne doit pas connaitre le statut du Front entre deux requêtes. Ainsi, utiliser des sessions en PHP serait à l'encontre du standard ainsi établi.

La solution à ce problème réside dans l'utilisation d'un système de token. Ainsi, toutes les informations permettant de savoir si l'utilisateur est autorisé à accéder à telle ou telle page ou fonctionnalité sera transmise à l'intérieur de la requête sous la forme d'un token.



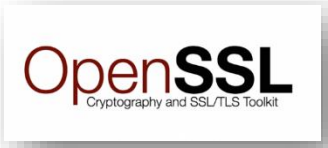
JWT (pour JSON Web Token), est un standard ouvert permettant l'échange sécurisé de tokens entre plusieurs parties. Cette sécurité se traduit par la vérification de l'intégrité des données via une signature numérique.

2.2 : Implémentation

Pour notre application, afin d'implémenter JWT, nous avons utilisé la bibliothèque LexikJWTAuthentificationBundle.

Pour l'installation, rien de plus simple qu'une ligne de commande avec Composer.

```
C:\Users\BIENVENUE\PhpstormProjects\nortec-project>composer require lexik/jwt-authentification-bundle
```



Pour la suite de la configuration, il faut avoir installé OpenSSL. Il s'agit d'une boite à outils de chiffrement comportant deux bibliothèques : libcrypto et libssl. Elles fournissent respectivement une implémentation des algorithmes cryptographiques et du protocole de communication SSL/TLS.

L'implémentation de cet outil nous permettra d'utiliser un système de cryptographie asymétrique pour encoder et décoder le token, avec une clé publique et une clé privée, augmentant ainsi la sécurité générale de du système d'authentification de l'application.

J'ai par ailleurs rencontré un problème étrange pour mettre en œuvre ces outils. En effet, malgré une installation correcte (en rajoutant OpenSSL dans les variables d'environnement du système également), les commandes d'OpenSSL n'étaient pas reconnues dans le terminal intégré de PHPStorm (l'IDE que j'utilise pour le développement Web), j'ai pu contourner le problème en utilisant le terminal Git Bash afin d'effectuer les opérations nécessaires à l'obtention des fameuses clés.

Mais tout d'abord, pour comprendre le fonctionnement général de ce système de cryptographie, nous pouvons déjà nous intéresser au fichier de configuration du bundle, qui s'est ajouté lors de son installation.

```yaml
# lexik_jwt_authentication.yaml
lexik_jwt_authentication:
    secret_key: '%env(resolve:JWT_SECRET_KEY)%'
    public_key: '%env(resolve:JWT_PUBLIC_KEY)%'
    pass_phrase: '%env(JWT_PASSPHRASE)%'
```

Ce fichier nous indique tout simplement où se trouvent les composants nécessaires au bon fonctionnement du système de cryptographie.

On y apprend donc que l'on peut trouver les clés ainsi que la « pass phrase » dans les variables d'environnements de l'application. En développement, ces variables sont dans le fichier .env de Symfony. Allons donc voir ce qu'il s'y trouve.

Dans le fichier environnement de Symfony, nous trouvons donc des nouvelles informations rajoutées lors de l'installation du bundle.

```
# .env
###> lexik/jwt-authentication-bundle ###
JWT_SECRET_KEY=%kernel.project_dir%/config/jwt/private.pem
JWT_PUBLIC_KEY=%kernel.project_dir%/config/jwt/public.pem
JWT_PASSPHRASE=8bf4597390f0931925cc4eef873af82d
###< lexik/jwt-authentication-bundle ###
```

Ont été rajouté les chemins des fichiers où l'on trouvera la clé publique ainsi que la clé privée. Et enfin une « pass phrase » a automatiquement été défini. En revanche les fichiers private.pem et public.pem n'existent pas. Et c'est à ce moment qu'OpenSSL va être nécessaire afin de les générer.

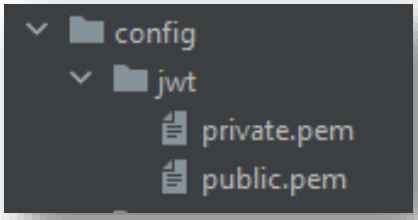
```
BIENVENUE@226114-18021 MINGW64 ~/exemples/nortec-project
$ mkdir -p config/jwt

BIENVENUE@226114-18021 MINGW64 ~/exemples/nortec-project
$ openssl genrsa -out config/jwt/private.pem -aes256 4096
Generating RSA private key, 4096 bit long modulus (2 primes)
......++++
..............++++
e is 65537 (0x010001)
Enter pass phrase for config/jwt/private.pem:
Verifying - Enter pass phrase for config/jwt/private.pem:

BIENVENUE@226114-18021 MINGW64 ~/exemples/nortec-project
$ openssl rsa -pubout -in config/jwt/private.pem -out config/jwt/public.pem
Enter pass phrase for config/jwt/private.pem:
writing RSA key
```

En ligne de commande (dans Git Bash donc), nous effectuons la génération.

La première commande sert à créer le dossier dans lequel seront stockés les clés.

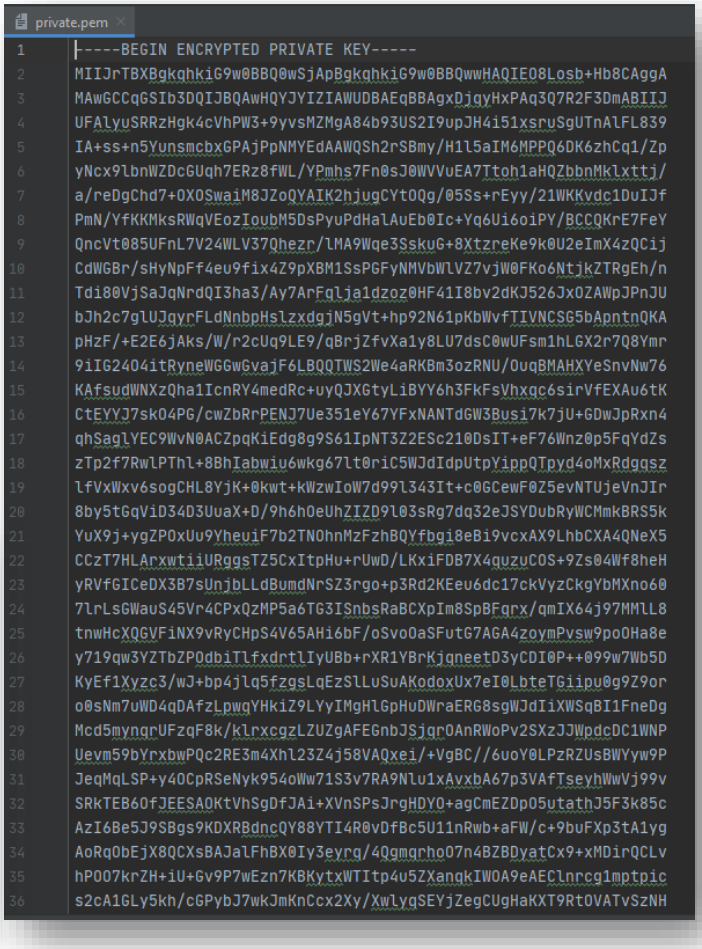
La seconde sert à générer la clé privée dans un fichier private.pem suivant le standard AES (Advanced Encryption Standard). Il demande la « pass phrase » qui servira de base au chiffrement.

Enfin, la dernière ligne permet de générer la clé publique à partir de la clé privée, car elle en est dépendante, et demande également la « pass phrase » afin de vérifier que nous sommes autorisés à l'utiliser.



Nous nous retrouvons donc bien avec les deux fichiers supplémentaires dans le dossier config/jwt, comprenant chacune la clé générée par OpenSSL.

Ci-contre, une partie de la clé privée générée.



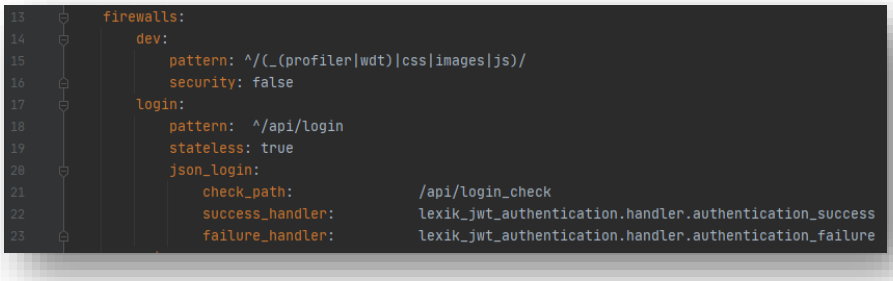
Ci-dessous, la clé publique générée à partir de la clé privée.

C’est à partir de ces deux clés que le Bundle JWT pourra chiffrer et déchiffrer les informations du token, qui sera transmis dans chaque requête qui nécessitera une authentification sur l’application.

2.3 : Configuration de Symfony pour l'authentification

L'application est donc théoriquement capable de générer des JWT. Mais il faut encore ajouter différentes configurations dans Symfony pour qu'il prenne tout cela en compte.

En effet, il va falloir rajouter un firewall dans le security.yaml de Symfony, ce qui déclenchera l'utilisation du JWT. Mais également configurer la route qui servira de cible pour le login.



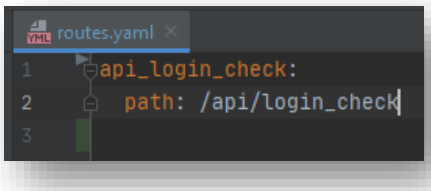
Dans le fichier security.yaml, nous créons donc un nouveau firewall que nous nommons « login ».

Toutes les URLs commençant par « /api/login » seront gérées et protégées par ce firewall (pattern).

La partie « json_login » est dans notre cas propre au bundle que nous venons d'installer. Il dit simplement que si l'on veut s'authentifier, nous pouvons envoyer toutes les informations de login à l'adresse « /api/login_check ».

Et par la suite, suivant le cas où l'authentification est couronnée de succès ou non, il appellera le service du bundle correspondant.

Il reste néanmoins une dernière chose à faire pour que Symfony comprenne, ce qu'il faut faire. Car pour le moment aucune route n'a été configurée et si nous allons sur l'URL « api/login_check », nous aurons une erreur. Il faut donc la configurer dans le fichier de configuration routes.yaml.



2.4 : Conclusion

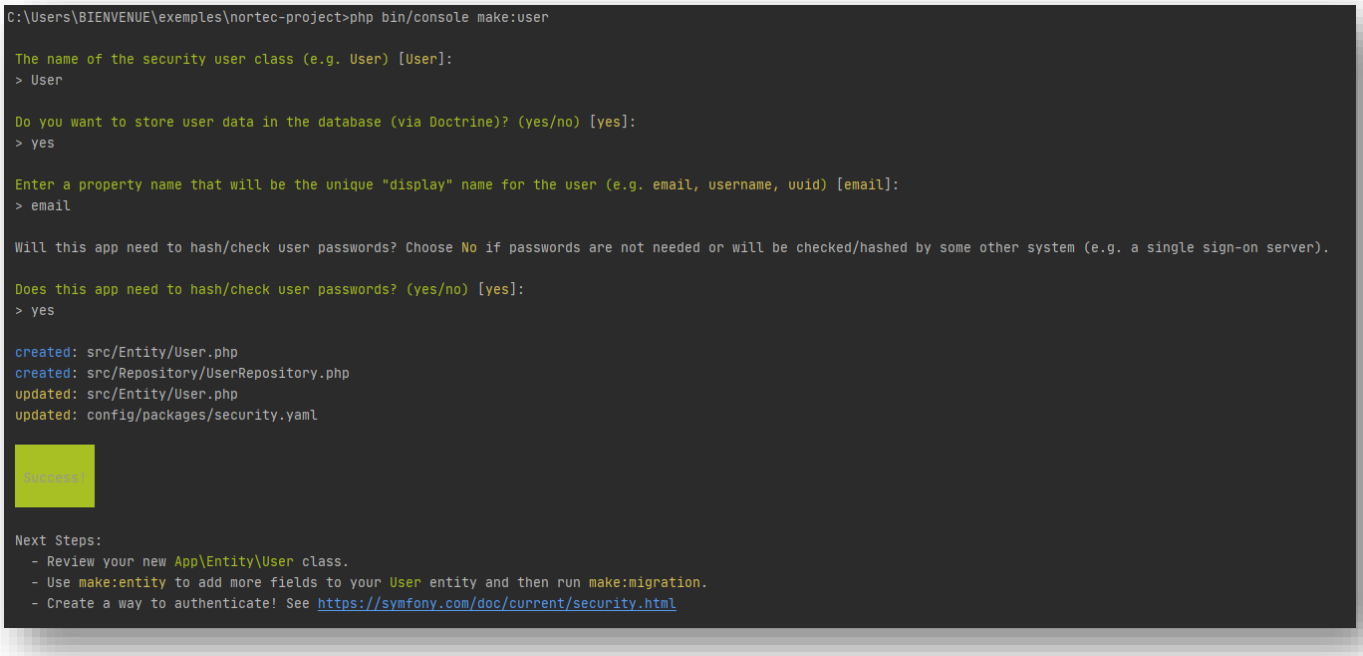
Pour conclure, nous avons donc maintenant configuré notre API de manière qu'elle puisse encoder et décoder des JWT. Bien sûr, à ce stade, rien n'est fonctionnel car il manque beaucoup d'éléments. Mais nous disposons d'une base solide niveau sécurité qu'il nous faudra utiliser par la suite.

3 : Création de l'entité User

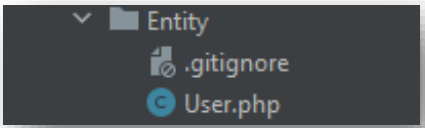
Afin de pouvoir s'authentifier, il faut bien sûr des utilisateurs autorisés à le faire, avec des informations de login/mot de passe.

Ceci sera implémenté niveau Back dans Symfony, avec enregistrement des informations en base de données.

Pour cela, nous allons créer une entité User dans Symfony.



Nous créons donc l'entité User via la commande « make:user ». Nous laissons le nom « User » pour le nom de classe. Nous définissons également que l'email sera l'identifiant pour l'authentification. Que les informations soient enregistrées en base de données et enfin que le mot de passe devra être encodé pour plus de sécurité.



Symfony nous a donc créé l'entité correctement qui apparait dans l'arborescence des fichiers du projet.

```php
namespace App\Entity;

use App\Repository\UserRepository;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Security\Core\User\UserInterface;

/**
 * @ORM\Entity(repositoryClass=UserRepository::class)
 */
class User implements UserInterface
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=180, unique=true)
     */
    private $email;

    /**
     * @ORM\Column(type="json")
     */
    private $roles = [];

    /**
     * @var string The hashed password
     * @ORM\Column(type="string")
     */
    private $password;
```

La classe User a été créée automatiquement par Symfony, implémentant la UserInterface (ce qui lui donne un statut différent dans l'application comme quoi elle représente des utilisateurs), avec les différentes propriétés, les getters et setters.

Il a également rajouté toutes les annotations qui serviront à Doctrine afin de créer la table et les champs nécessaires en base de données.

security.yaml

```yaml
security:
    encoders:
        App\Entity\User:
            algorithm: auto


    providers:
        app_user_provider:
            entity:
                class: App\Entity\User
                property: email
```

Il a en outre rajouté dans le security.yaml un encoder associé à la classe User. C'est ce qui définit l'algorithme d'encodage du mot de passe. Le paramètre est défini ici en « auto », et dans Symfony 5, celui utilisé par défaut est Argon2i.

Il a également ajouté un provider, également rattaché à la classe User, et définissant que l'authentification se fera à partir de l'email.

```
C:\Users\BIENVENUE\exemples\nortec-project>php bin/console make:entity User

Your entity already exists! So let's add some new fields!

New property name (press <return> to stop adding fields):
> firstName

Field type (enter ? to see all types) [string]:
> string

Field length [255]:
> 255

Can this field be null in the database (nullable) (yes/no) [no]:
> no

updated: src/Entity/User.php

Add another property? Enter the property name (or press <return> to stop adding fields):
> lastName

Field type (enter ? to see all types) [string]:
> string

Field length [255]:
> 255

Can this field be null in the database (nullable) (yes/no) [no]:
> no

updated: src/Entity/User.php

Add another property? Enter the property name (or press <return> to stop adding fields):
>


 Success!


Next: When you're ready, create a migration with php bin/console make:migration
```

Ensuite, nous aurons besoin d'autres informations dans l'entité User. Comme le prénom et le nom de famille… Ainsi nous pouvons les rajouter en faisant un « make:entity User ».

L'entité aura besoin d'autres informations plus tard mais qui ne concerne pas l'authentification.

Enfin, pour mettre à jour la base de données, nous pouvons d'ores et déjà utiliser Doctrine pour créer le fichier de migration, et ensuite effectuer réellement cette migration.

```
C:\Users\BIENVENUE\exemples\nortec-project>php bin/console make:migration


 Success!


 Next: Review the new migration "migrations/Version20200821115201.php"
 Then: Run the migration with php bin/console doctrine:migrations:migrate
 See https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html

C:\Users\BIENVENUE\exemples\nortec-project>php bin/console doctrine:migrations:migrate

 WARNING! You are about to execute a database migration that could result in schema changes and data loss. Are you sure you wish to continue? (yes/no) [yes]:
 > yes

[notice] Migrating up to DoctrineMigrations\Version20200821115201
[notice] finished in 115.3ms, used 24M memory, 1 migrations executed, 1 sql queries
```

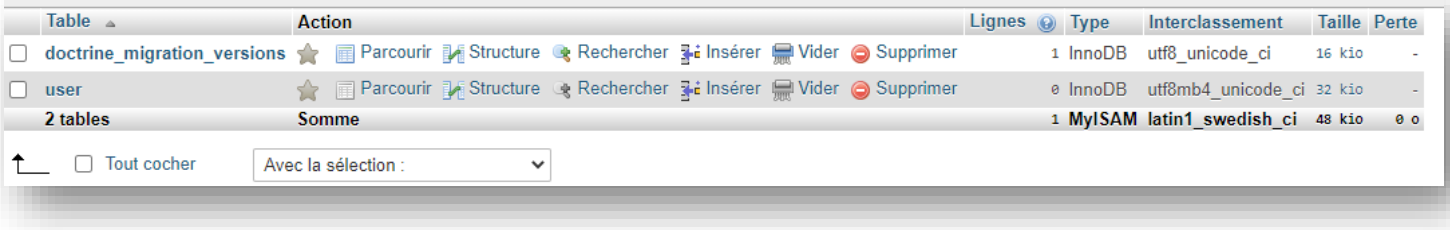
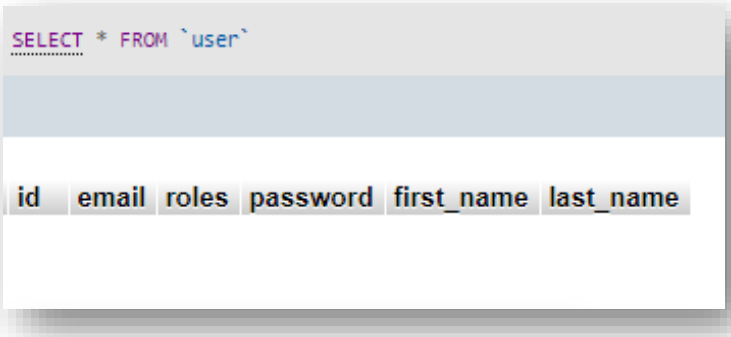


Table	Action	Lignes	Type	Interclassement	Taille	Perte
doctrine_migration_versions	Parcourir Structure Rechercher Insérer Vider Supprimer	1	InnoDB	utf8_unicode_ci	16 kio	-
user	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_unicode_ci	32 kio	-
2 tables	Somme	1	MyISAM	latin1_swedish_ci	48 kio	0 o

Tout cocher Avec la sélection :



```
SELECT * FROM `user`
```

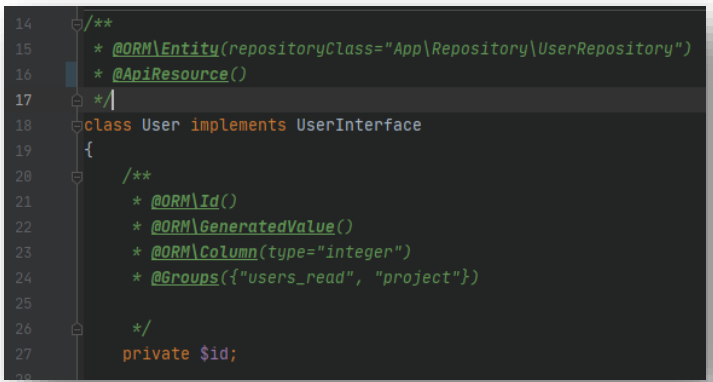
id	email	roles	password	first_name	last_name

En base de données nous pouvons donc observer que la table user a bien été créée, ainsi que les champs dont nous aurons besoin par la suite.

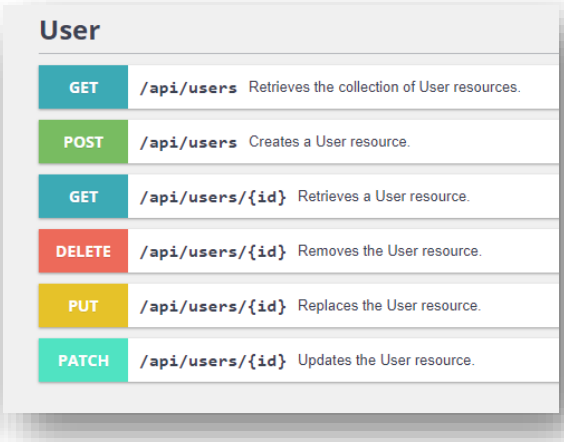
4 : Reconnaissance de l'entité User par ApiPlatform

Il faut maintenant notifier à ApiPlatform que les Users seront une ressource que nous voulons exposer.

Pour se faire, rien de plus simple. Il suffit de rajouter une annotation sur la classe afin qu'il la mappe et la rende exploitable dans l'API.



```
/**
 * @ORM\Entity(repositoryClass="App\Repository\UserRepository")
 * @ApiResource()
 */
class User implements UserInterface
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     * @Groups({"users_read", "project"})

     */
    private $id;
```



User

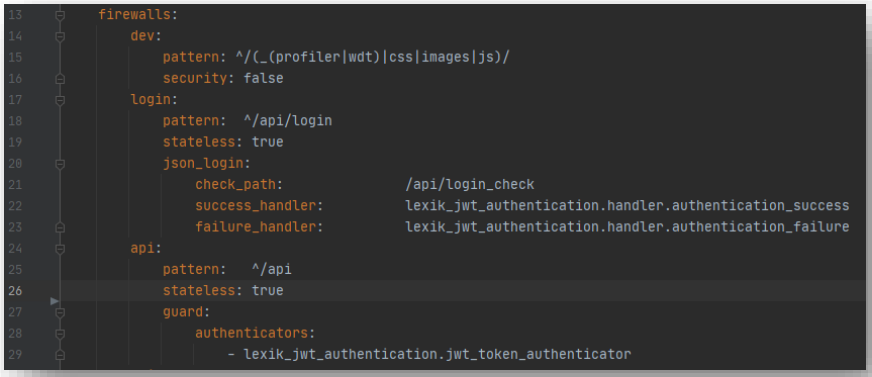
GET	/api/users	Retrieves the collection of User resources.
POST	/api/users	Creates a User resource.
GET	/api/users/{id}	Retrieves a User resource.
DELETE	/api/users/{id}	Removes the User resource.
PUT	/api/users/{id}	Replaces the User resource.
PATCH	/api/users/{id}	Updates the User resource.

Le simple ajout de l'annotation nous permet d'avoir la ressource User exposée dans l'API, avec son lien. Avec toutes les actions possibles suivant le type de requête http qui viendra l'attaquer.

5 : Rendre privée l'accès aux ressources de l'API

Nous avons vu comment exposer les ressources de l'API au public grâce à l'annotation « ApiResource() ». Mais le soucis est que tout le monde peut y accéder. Ce qui n'est pas le but recherché.

Néanmoins, nous voulons les exposer aux utilisateurs authentifiés uniquement. Pour cela, nous allons rajouter une configuration dans le security.yaml.



```yaml
firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false
    login:
        pattern:  ^/api/login
        stateless: true
        json_login:
            check_path:               /api/login_check
            success_handler:          lexik_jwt_authentication.handler.authentication_success
            failure_handler:          lexik_jwt_authentication.handler.authentication_failure
    api:
        pattern:   ^/api
        stateless: true
        guard:
            authenticators:
                - lexik_jwt_authentication.jwt_token_authenticator
```

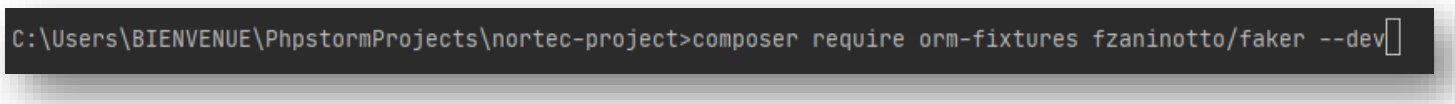
Nous rajoutons donc un nouveau firewall qui protégera toutes les URLs commençant par « /api ». Ainsi, seuls les requêtes comprenant un token valide pourront accéder à ces ressources. Il est important de mettre ce firewall après celui qui garde le « login », sinon l'authentification ne sera plus du tout accessible.

6 : Ajout de faux utilisateurs

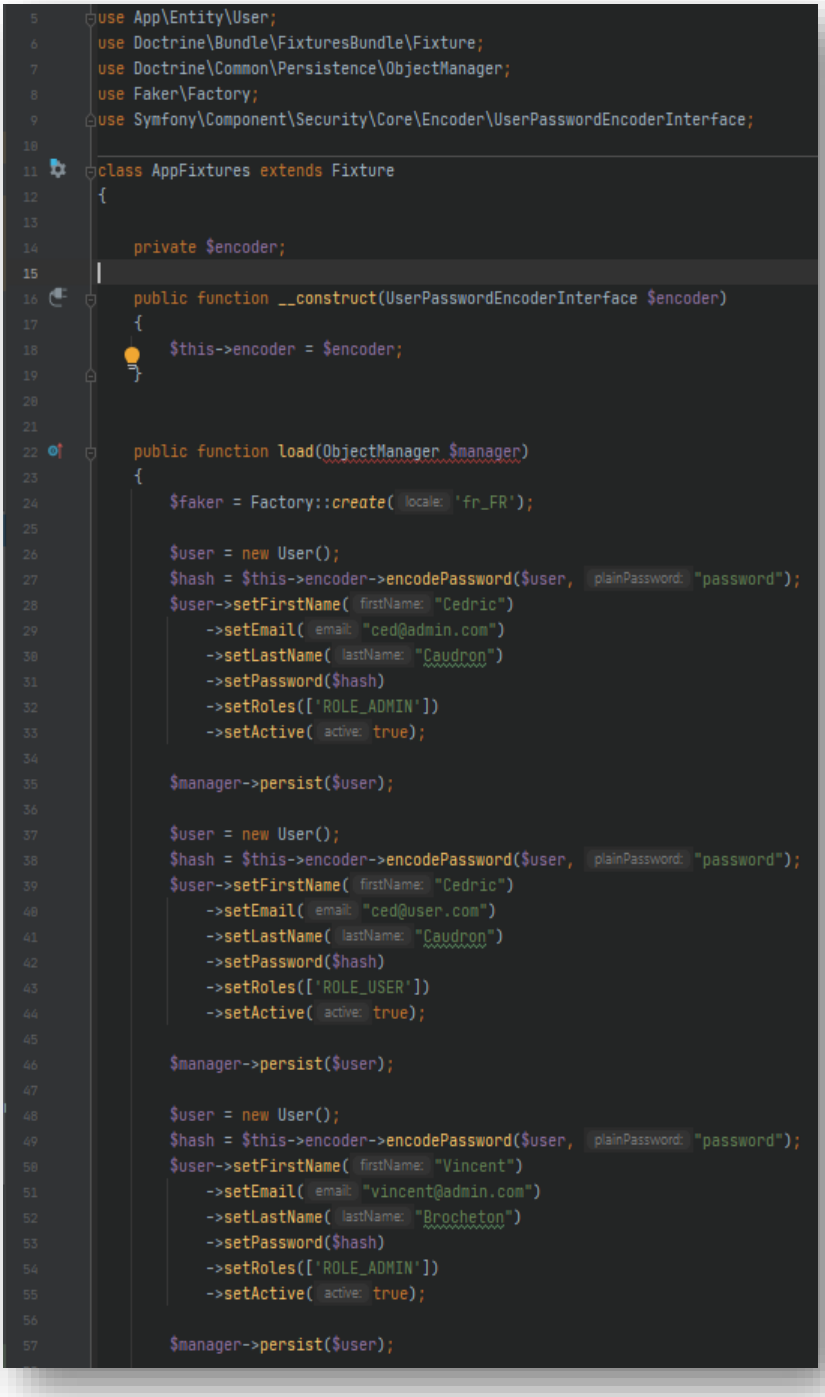
Pendant le développement, et pour vérifier le bon fonctionnement général de l'application, nous avons utilisé des Fixtures afin de créer de faux utilisateurs avec leurs informations propres.

Pour mettre ce système en place, nous avons besoin de deux dépendances :

- Orm-fixtures : qui permet la création de fausses données
- Fzaninotto/faker : qui génère aléatoirement de fausses données réalistes



```
C:\Users\BIENVENUE\PhpstormProjects\nortec-project>composer require orm-fixtures fzaninotto/faker --dev
```



```php
use App\Entity\User;
use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Common\Persistence\ObjectManager;
use Faker\Factory;
use Symfony\Component\Security\Core\Encoder\UserPasswordEncoderInterface;

class AppFixtures extends Fixture
{

    private $encoder;

    public function __construct(UserPasswordEncoderInterface $encoder)
    {
        $this->encoder = $encoder;
    }


    public function load(ObjectManager $manager)
    {
        $faker = Factory::create( locale: 'fr_FR');

        $user = new User();
        $hash = $this->encoder->encodePassword($user,  plainPassword: "password");
        $user->setFirstName( firstName: "Cedric")
            ->setEmail( email: "ced@admin.com")
            ->setLastName( lastName: "Caudron")
            ->setPassword($hash)
            ->setRoles(['ROLE_ADMIN'])
            ->setActive( active: true);

        $manager->persist($user);

        $user = new User();
        $hash = $this->encoder->encodePassword($user,  plainPassword: "password");
        $user->setFirstName( firstName: "Cedric")
            ->setEmail( email: "ced@user.com")
            ->setLastName( lastName: "Caudron")
            ->setPassword($hash)
            ->setRoles(['ROLE_USER'])
            ->setActive( active: true);

        $manager->persist($user);

        $user = new User();
        $hash = $this->encoder->encodePassword($user,  plainPassword: "password");
        $user->setFirstName( firstName: "Vincent")
            ->setEmail( email: "vincent@admin.com")
            ->setLastName( lastName: "Brocheton")
            ->setPassword($hash)
            ->setRoles(['ROLE_ADMIN'])
            ->setActive( active: true);

        $manager->persist($user);
```

Un nouveau fichier est donc créé où l'on peut configurer les futures fausses données.

Dans un premier temps, quatre comptes ont été créés avec des informations bien définies qui serviront de compte à Vincent et moi-même pour nous identifier sur l'application. Un compte avec le rôle Admin et l'autre avec le rôle User.

Pour encoder le mot de passe, nous devons importer dans le constructeur, par injection de dépendance, un encoder qui implémente UserPasswordEncoderInterface. Cet encoder possède une fonction « encodePassword » qui prend en paramètre l'User en question et le mot de passe en clair. Ainsi, en base de données, le mot de passe sera directement hashé en Argon2i.

Après la création de chaque User, nous faisons appel à la fonction persist() du manager de Doctrine. Elle permet de préparer l'objet pour ensuite un enregistrement en base de données.

De plus, nous préparons aussi Faker en appelant sa méthode statique « create » avec la configuration en français. Pour qu'il nous donne des données cohérentes avec la langue française.

```php
for ($u = 0; $u < 100; $u++) {
    $user = new User();

    $hash = $this->encoder->encodePassword($user, plainPassword: "password");

    $firstName = $faker->firstName;
    $lastName = $faker->lastName;

    $mail = str_to_noaccent(mb_strtolower($firstName . "." . $lastName));
    $mail = $mail . "@fake.com";

    $user->setFirstName($firstName)
        ->setLastName($lastName)
        ->setPassword($hash)
        ->setRoles(['ROLE_USER'])
        ->setActive( active: true)
        ->setEmail($mail);


    $manager->persist($user);
}

$manager->flush();

}
```

Nous créons ensuite 100 faux utilisateurs de manière aléatoire.

Ce processus est géré dans une boucle for.

Nous récupérons donc un nom et un prénom grâce à Faker.

Faker est capable de nous donner des emails aléatoires. Mais en le faisant comme cela nous avions des noms et prénoms totalement incohérents avec l'adresse mail, ce qui dans la pratique ne fait pas très réaliste. C'est pourquoi je me suis amusé à reconstituer des adresses mails reprenant le prénom et le nom des utilisateurs, avec le suffixe @fake.com commun à tout le monde.

Pour rendre cela encore plus réaliste, j'ai mis les noms et prénoms en minuscule, et enfin j'ai retiré tous les accents et caractères spéciaux grâce à une petite fonction qui les remplace par leur version normale.

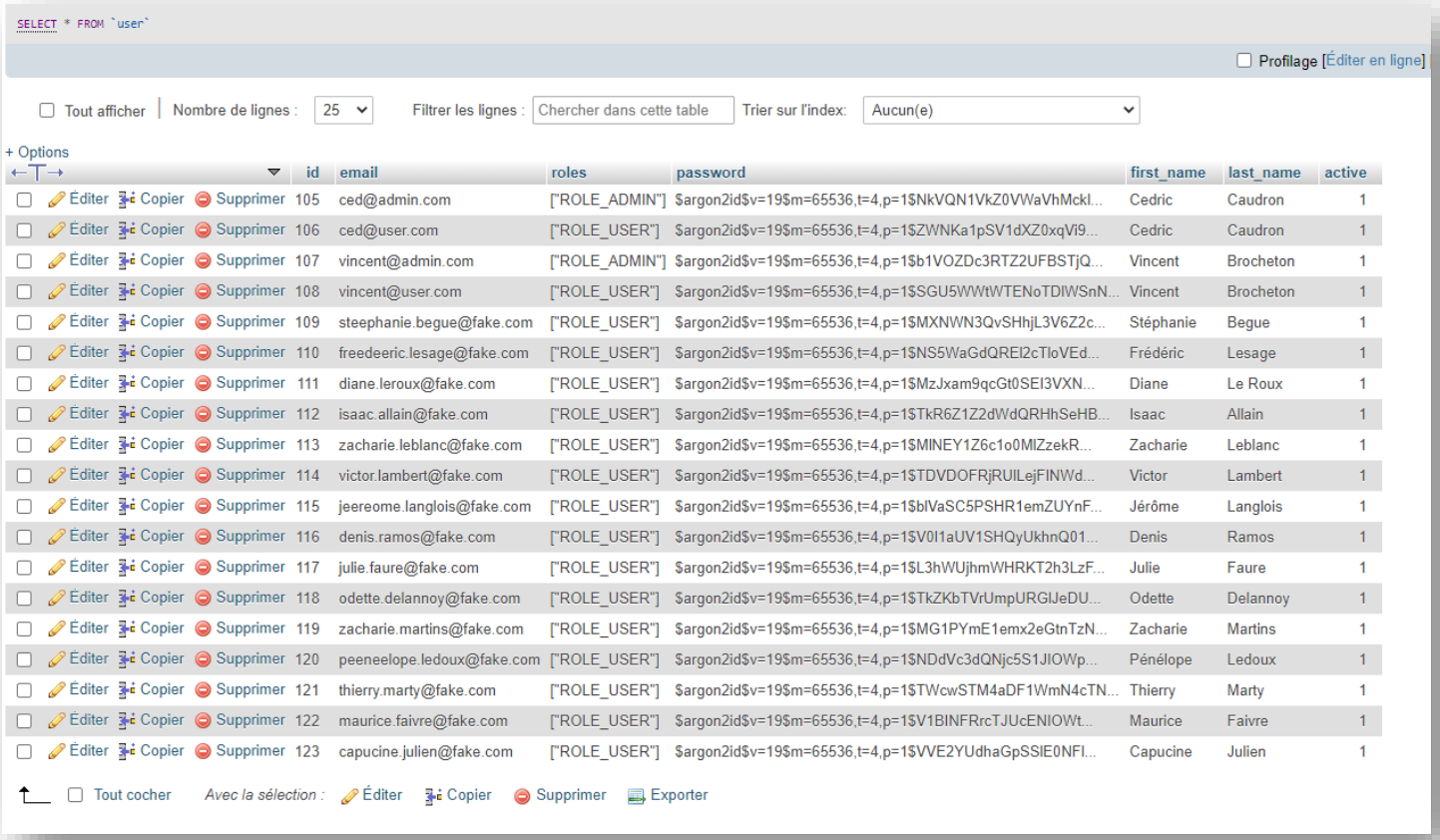
```php
function str_to_noaccent($str)
{
    $url = $str;
    $url = preg_replace( pattern: '#Ç#', replacement: 'C', $url);
    $url = preg_replace( pattern: '#ç#', replacement: 'c', $url);
    $url = preg_replace( pattern: '#[èéêë]#', replacement: 'e', $url);
    $url = preg_replace( pattern: '#[ÈÉÊË]#', replacement: 'E', $url);
    $url = preg_replace( pattern: '#[àáâãäå]#', replacement: 'a', $url);
    $url = preg_replace( pattern: '#[@ÀÁÂÃÄÅ]#', replacement: 'A', $url);
    $url = preg_replace( pattern: '#[ìíîï]#', replacement: 'i', $url);
    $url = preg_replace( pattern: '#[ÌÍÎÏ]#', replacement: 'I', $url);
    $url = preg_replace( pattern: '#[ðòóôõö]#', replacement: 'o', $url);
    $url = preg_replace( pattern: '#[ÒÓÔÕÖ]#', replacement: 'O', $url);
    $url = preg_replace( pattern: '#[ùúûü]#', replacement: 'u', $url);
    $url = preg_replace( pattern: '#[ÙÚÛÜ]#', replacement: 'U', $url);
    $url = preg_replace( pattern: '#[ýÿ]#', replacement: 'y', $url);
    $url = preg_replace( pattern: '#Ý#', replacement: 'Y', $url);
    $url = preg_replace( pattern: '# #', replacement: '', $url);

    return ($url);
}
```

On fait ensuite appel à la fonction persist() à chaque fin d'itération de la boucle. Et enfin, quand celle-ci est terminée, on appelle la fonction flush() qui enregistrera effectivement les données persistées dans la base de données.

Enfin, pour exécuter ce code, nous faisons appel à une simple ligne de commande.

```
C:\Users\BIENVENUE\PhpstormProjects\nortec-project>php bin/console doctrine:fixtures:load
```

Et enfin, nous pouvons aller voir le résultat dans la base de données, et nous constatons que tous les Users définis ont bien été créés, avec leur mot de passe encodé, et que leur adresse mail est bien formatée et réaliste.



```
SELECT * FROM `user`
```

Profilage [Éditer en ligne]

Tout afficher | Nombre de lignes : 25 | Filtrer les lignes : Chercher dans cette table | Trier sur l'index : Aucun(e)

+ Options

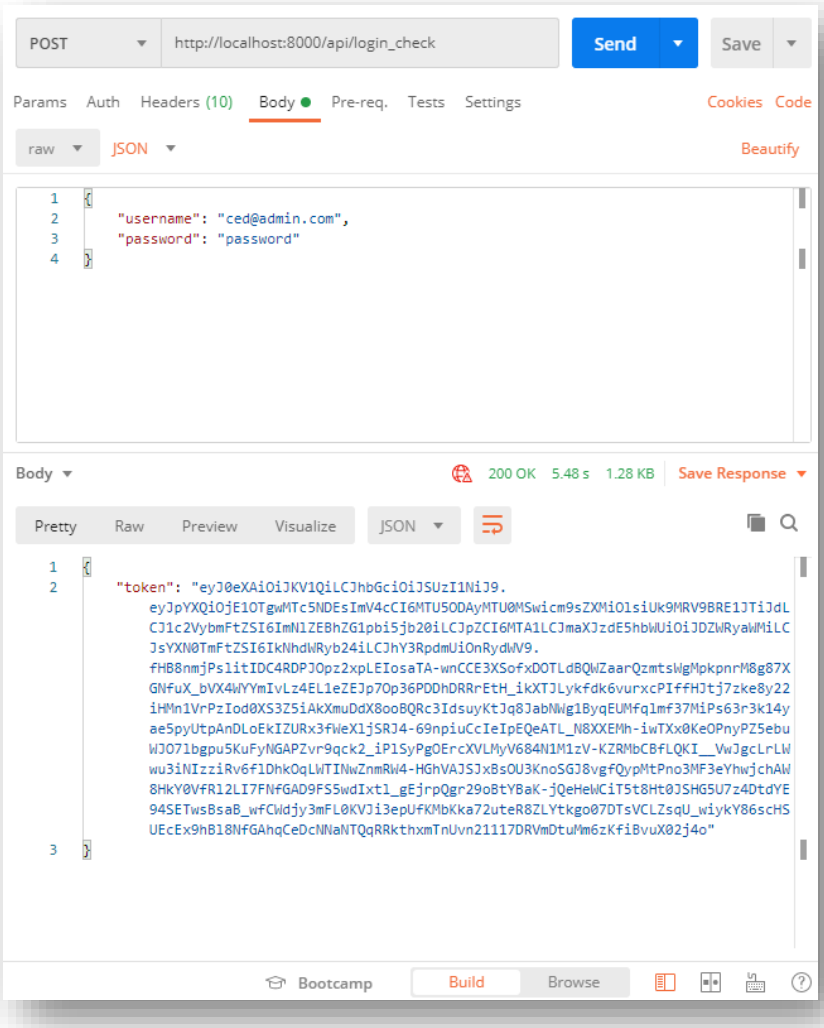
	id	email	roles	password	first_name	last_name	active
Éditer Copier Supprimer	105	ced@admin.com	["ROLE_ADMIN"]	$argon2id$v=19$m=65536,t=4,p=1$NkVQN1VkZ0VWaVhMckl...	Cedric	Caudron	1
Éditer Copier Supprimer	106	ced@user.com	["ROLE_USER"]	$argon2id$v=19$m=65536,t=4,p=1$ZWNKa1pSV1dXZ0xqVi9...	Cedric	Caudron	1
Éditer Copier Supprimer	107	vincent@admin.com	["ROLE_ADMIN"]	$argon2id$v=19$m=65536,t=4,p=1$b1VOZDc3RTZ2UFBSTjQ...	Vincent	Brocheton	1
Éditer Copier Supprimer	108	vincent@user.com	["ROLE_USER"]	$argon2id$v=19$m=65536,t=4,p=1$SGU5WWtWTENoTDlWSnN...	Vincent	Brocheton	1
Éditer Copier Supprimer	109	steephanie.begue@fake.com	["ROLE_USER"]	$argon2id$v=19$m=65536,t=4,p=1$MXNWN3QvSHhjL3V6Z2c...	Stéphanie	Begue	1
Éditer Copier Supprimer	110	freedeeric.lesage@fake.com	["ROLE_USER"]	$argon2id$v=19$m=65536,t=4,p=1$NS5WaGdQREl2cTloVEd...	Frédéric	Lesage	1
Éditer Copier Supprimer	111	diane.leroux@fake.com	["ROLE_USER"]	$argon2id$v=19$m=65536,t=4,p=1$MzJxam9qcGt0SEI3VXN...	Diane	Le Roux	1
Éditer Copier Supprimer	112	isaac.allain@fake.com	["ROLE_USER"]	$argon2id$v=19$m=65536,t=4,p=1$TkR6Z1Z2dWdQRHhSeHB...	Isaac	Allain	1
Éditer Copier Supprimer	113	zacharie.leblanc@fake.com	["ROLE_USER"]	$argon2id$v=19$m=65536,t=4,p=1$MlNEY1Z6c1o0MlZzekR...	Zacharie	Leblanc	1
Éditer Copier Supprimer	114	victor.lambert@fake.com	["ROLE_USER"]	$argon2id$v=19$m=65536,t=4,p=1$TDVDOFRjRUlLejFINWd...	Victor	Lambert	1
Éditer Copier Supprimer	115	jeereome.langlois@fake.com	["ROLE_USER"]	$argon2id$v=19$m=65536,t=4,p=1$blVaSC5PSHR1emZUYnF...	Jérôme	Langlois	1
Éditer Copier Supprimer	116	denis.ramos@fake.com	["ROLE_USER"]	$argon2id$v=19$m=65536,t=4,p=1$V0l1aUV1SHQyUkhnQ01...	Denis	Ramos	1
Éditer Copier Supprimer	117	julie.faure@fake.com	["ROLE_USER"]	$argon2id$v=19$m=65536,t=4,p=1$L3hWUjhmWHRKT2h3LzF...	Julie	Faure	1
Éditer Copier Supprimer	118	odette.delannoy@fake.com	["ROLE_USER"]	$argon2id$v=19$m=65536,t=4,p=1$TkZKbTVrUmpURGlJeDU...	Odette	Delannoy	1
Éditer Copier Supprimer	119	zacharie.martins@fake.com	["ROLE_USER"]	$argon2id$v=19$m=65536,t=4,p=1$MG1PYmE1emx2eGtnTzN...	Zacharie	Martins	1
Éditer Copier Supprimer	120	peeneelope.ledoux@fake.com	["ROLE_USER"]	$argon2id$v=19$m=65536,t=4,p=1$NDdVc3dQNjc5S1JlOWp...	Pénélope	Ledoux	1
Éditer Copier Supprimer	121	thierry.marty@fake.com	["ROLE_USER"]	$argon2id$v=19$m=65536,t=4,p=1$TWcwSTM4aDF1WmN4cTN...	Thierry	Marty	1
Éditer Copier Supprimer	122	maurice.faivre@fake.com	["ROLE_USER"]	$argon2id$v=19$m=65536,t=4,p=1$V1BINFRrcTJUcENlOWt...	Maurice	Faivre	1
Éditer Copier Supprimer	123	capucine.julien@fake.com	["ROLE_USER"]	$argon2id$v=19$m=65536,t=4,p=1$VVE2YUdhaGpSSlE0NFl...	Capucine	Julien	1

Tout cocher *Avec la sélection :* Éditer Copier Supprimer Exporter

7 : Test de l'authentification au niveau de l'API

Avant de passer aux fonctionnalités Front de l'authentification, nous devons nous assurer que le login fonctionne bien avec les utilisateurs enregistrés dans la base de données.

Pour ce faire, vu qu'il n'y a aucune interface Front, nous utilisons Postman. Un logiciel permettant d'effectuer des requêtes et de visualiser les réponses obtenues . Il est donc devenu incontournable pour tester les API.



Nous envoyons donc une requête POST vers le lien servant à vérifier l'authentification, avec dans la requête un JSON comprenant un « username » et un « password » corrects.

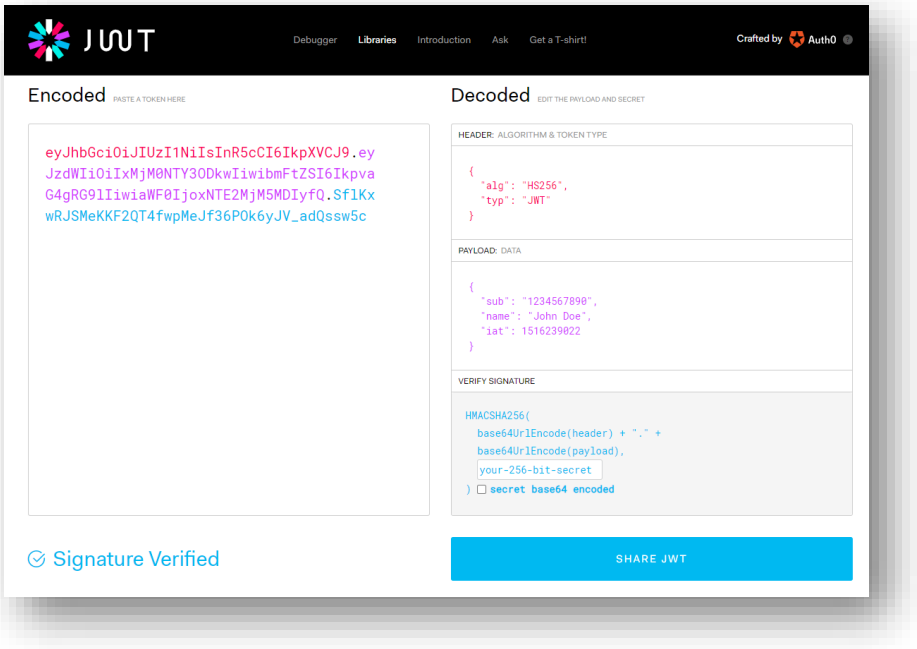
Et nous pouvons remarquer que l'API nous renvoie une réponse avec le token à l'intérieur. L'authentification a réussie, l'utilisateur a droit d'accéder aux ressources de l'API !

Et pour se faire, il devra donc, à chaque requête, rajouter une Authorization avec ce token pour prouver qu'il est bien authentifié. Et c'est grâce à cette méthode que nous restons conformes au standard REST. Le serveur n'a aucune idée de l'état d'authentification de l'utilisateur entre deux requêtes ! Toutes les infos sont comprises dans chaque requête.

8 : Le token

8.1 : Comment se présente-t-il ?

Le token, comme nous l'avons vu précédemment, est une chaine de caractère cryptée **qui transporte des informations.**



En effet, en nous rendant sur le site Web jwt.io, nous avons un exemple.

Le token se décompose en trois parties séparées par un point. Un entête (header) qui est utilisé pour décrire le jeton. Une charge utile (payload) qui représente les informations embarquées dans le jeton. Ces deux parties sont représentées par des objets JSON. Et enfin, la troisième partie représentant une signature numérique.

Dans l'exemple ci-dessus, nous avons donc le token crypté sur la gauche, et à droite les informations qu'il comprend.

De base, dans notre application, nous ne récupérons dans le token que le username, le tableau des rôles, la date d'émission et la date d'expiration (en Timestamp). Or il serait intéressant pour nous pour pouvoir travailler plus facilement de récupérer des informations supplémentaires. Ainsi, nous allons pouvoir créer un service dans Symfony afin d'y enrichir ses informations.

8.2 : Intervenir sur la création du JWT pour enrichir ses données

```php
namespace App\Events;

use Lexik\Bundle\JWTAuthenticationBundle\Event\JWTCreatedEvent;

class JwtCreatedSubscribers {
    public function updateJwtData(JWTCreatedEvent $event) {

        // Récupérer le user
        $user = $event->getUser();

        $data = $event->getData();
        $data['id'] = $user->getId();
        $data['firstName'] = $user->getFirstName();
        $data['lastName'] = $user->getLastName();
        $data['active'] = $user->getActive();

        $event->setData($data);

    }
}
```

Nous créons donc une nouvelle classe prenant en paramètre via injection de dépendance un événement propre à la librairie LexikJWT (évènement qui n'est pas à confondre avec ceux du Kernel de Symfony).

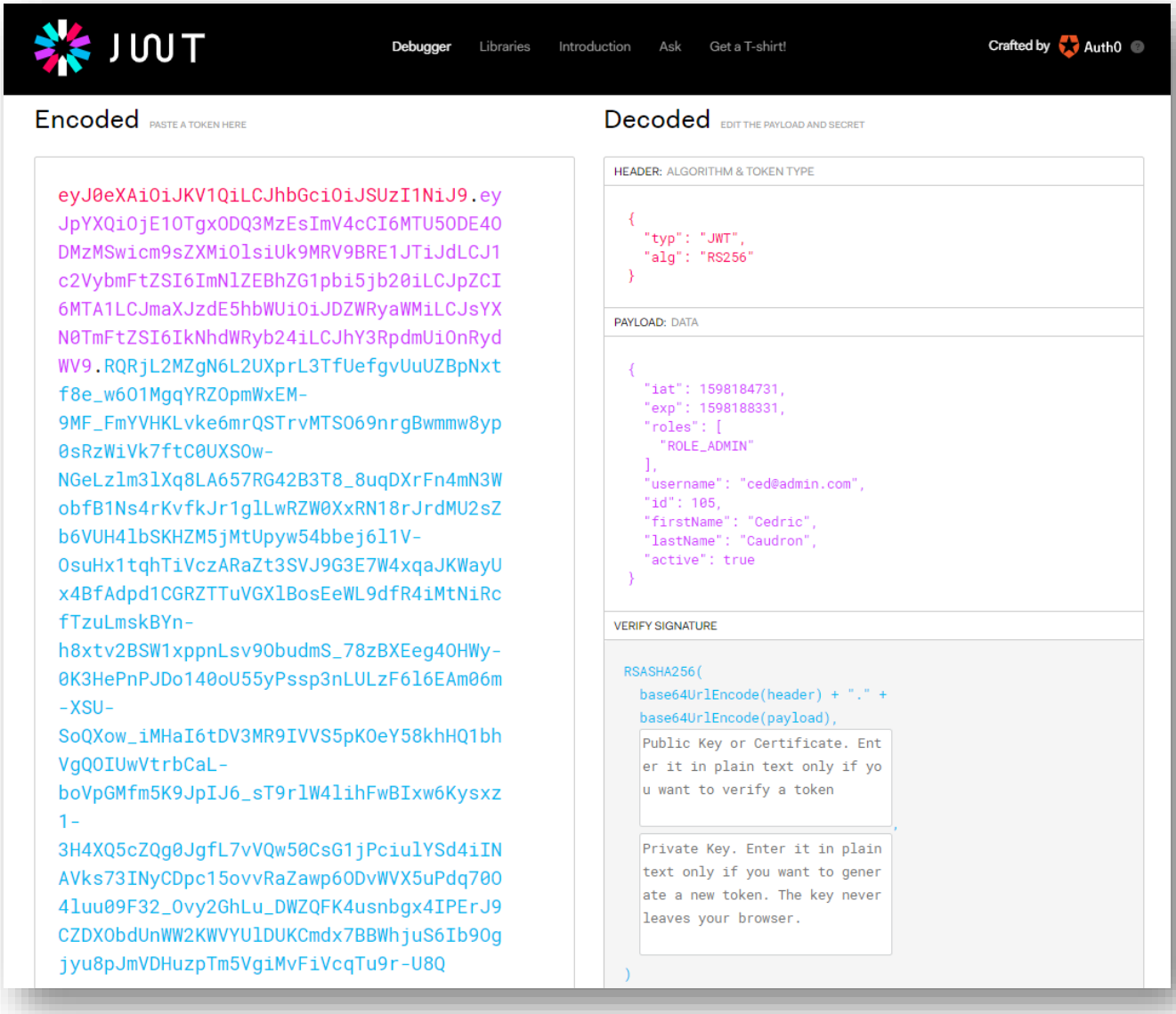
Nous récupérons donc l'utilisateur qui s'authentifie, les données de l'événement, et nous l'enrichissons avec les données de l'utilisateur qui nous intéressent.

Donc ici, dans le tableau (sous la forme clé / valeur), nous ajoutons l'id, le firstname, le lastname et le statut « active » de l'utilisateur.

```yaml
services:
    App\Events\JwtCreatedSubscribers:
        tags:
            - {name: kernel.event_listener, event: lexik_jwt_authentication.on_jwt_created, method: updateJwtData}
```

Il faut en outre, pour que ce soit reconnu, rajouter le service dans la configuration de Symfony dans le fichier services.yaml, afin que Symfony sache à quel moment appeler cette classe et la fonction qu'elle contient : à la création du JWT.

Nous nous retrouvons donc avec un token enrichi des informations de l'utilisateur authentifié. Il était très intéressant de les faire apparaitre dans le token car elles seront pour la plupart constante tout le temps de la connexion de l'utilisateur, et ainsi, au lieu de multiplier les requêtes, nous pouvons donc travailler directement sur le token pour utiliser ces informations.



Notre API est donc enfin prête pour gérer l'authentification des utilisateurs, ainsi que leur statut ou leur rôle. Nous allons donc aborder maintenant les systèmes que nous avons mis en place côté Front afin de gérer et automatiser au maximum celle-ci.

9 : Fonctionnement de la page de Login

9.1 : Template

Nous allons aborder maintenant l'authentification du coté de l'application React. Ici je ne parlerai que des éléments spécifiques à cette fonctionnalité. Mais certaines notions de base de React devront inévitablement être abordées… Néanmoins, je reviendrai dans la partie Front sur l'organisation générale d'une application React.

```
<form className="login-style form card p-3 m-5" onSubmit={handleSubmit}>
    <div className="card-title">
        <LogoCompanyComponent style={{width: "150px"}}/>

        <h1 className="login-style title text-center mb-3">Bienvenue sur le portail Nortec</h1>
    </div>


    <div className="card-body">
        <Field
            label="Adresse Email"
            value={credentials.username}
            placeholder="Adresse mail de connexion"
            type="email"
            name="username"
            error={error}
            onChange={handleChange}
        />

        <Field
            label="Mot de passe"
            value={credentials.password}
            type="password"
            name="password"
            onChange={handleChange}
        />

        <div className="mt-3">
            <Link to="/reinitialisation">Réinitialisation du mot de passe</Link>
        </div>

        <div className="text-right mt-3 mb-3">
            <button
                type="submit"
                className="btn btn-login"
            >
                Connexion
            </button>
        </div>
    </div>
</form>
```

Au niveau du template, nous avons ici créé deux champs ainsi qu'un bouton. Le tout agrémenté d'un peu de Bootstrap.

Les champs ont été refactorisé dans un component Field, auxquels nous passons les paramètres nécessaires, que nous avons nous-même défini.

Le formulaire, de type submit, lors de la validation appellera une fonction handleSubmit.

Field.jsx

```jsx
import React from 'react';

const Field = ({name, label, value = "", onChange = "", placeholder, className = "", type = "text", error = "", noLabel = false}) => {
    return (
        <div className="form-group">
            {!noLabel &&
            <label htmlFor={name}>
                {label}
            </label>
            }
            <input
                value={value}
                onChange={onChange}
                type={type}
                placeholder={placeholder || label}
                name={name}
                id={name}
                className={"form-control" + (error && " is-invalid") || className}
            />
            {error && <p className="invalid-feedback">{error}</p>}
        </div>
    );
};

export default Field;
```

Le component Field (ci-dessus), a été conçu pour répondre au maximum de possibilités que puisse offrir une balise <input>. Il est entièrement paramétrable.

Ainsi, lors de l'appel, nous pouvons définir :

- Son nom
- L'absence ou non de label (avec la propriété booléenne noLabel, défini par défaut sur false)
- Son label le cas échéant
- Sa value
- La fonction appelée lorsqu'il y a un changement de valeur (onChange)
- Son placeholder
- Ses classes
- Son type, avec la valeur text par défaut
- Les éventuelles erreurs. Dans le cas où les erreurs existent, elles seront mises en forme grâce aux classes Bootstrap « is-invalid » et « invalid-feedback ».

9.2 : La fonction HandleChange et notion de State

Afin de gérer les champs du formulaire, nous allons introduire ici la notion de State dans React.

Un State représente un état de la page à un moment donné. Dont les informations peuvent être mises à jour en temps réel dans la page React. Nous l'utilisons sous la forme d'un Hook, notion sur laquelle je reviendrai plus tard.

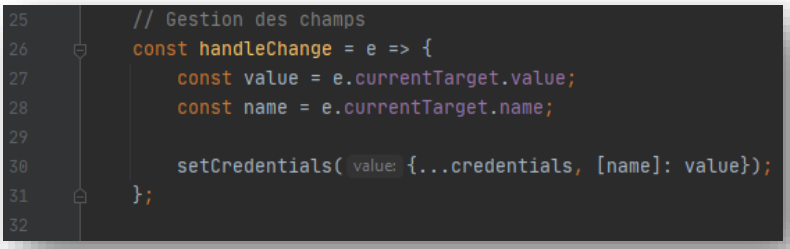
Le State se présente sous la forme d'une variable couplée à une fonction « set » qui permettra de la modifier.

Ici nous définissons donc un State représentant chaque champ du formulaire de Login, avec un username et un password donc, et dont la valeur initiale sera pour chacun une chaine de caractère vide.

En regardant plus haut dans le template, nous voyons donc que la valeur credentials.username est affecté au premier Field, et credentials.password au deuxième.

Le soucis, c'est que si nous laissons en l'état, il n'y aura pas de modification possible par l'utilisateur de ces champs. D'où l'appel à la fonction handleChange dans la propriété onChange de chaque Field.

La fonction handleChange va donc récupérer les values et names courantes de chaque Field, de concaténer cette valeur avec celle présente dans la variable credentials, et enfin de mettre à jour cette variable. Ce qui mettra mécaniquement à jour la value de chaque Field.



9.3 : La fonction HandleSubmit

```
35      // Gestion du Submit
36      const handleSubmit = async event => {
37          event.preventDefault();
38
39          try {
40              await AuthAPI.authenticate(credentials);
41              setError( value: "");
42
43              if (AuthAPI.getUserActiveStatus())
44              {
45                  setIsAuthenticated(true);
46                  toast.success( content: "Vous êtes connnecté en tant que " + AuthAPI.getUserFirstNameLastName() + ". Vous avez le statut d'" + AuthAPI.isRole() + ".");
47                  history.replace("/projects");
48              }
49              else
50              {
51                  setIsAuthenticated(false);
52                  toast.error("Votre compte a été désactivé. Veuillez contacter un administrateur!");
53                  AuthAPI.logout();
54              }
55
56          } catch {
57              setError( value: "Les informations de login/mot de passe sont incorrectes!");
58              toast.error("Les informations de login/mot de passe sont incorrects!");
59          }
60      };
```

La fonction handleSubmit, qui est appelée lors de la soumission du formulaire, permet le déclenchement de la requête vers l'API, vérifier si l'utilisateur est autorisé à accéder à l'application ou non, et de définir le statut s'il est authentifié ou non. Il s'agit d'une fonction asynchrone, nous reviendrons sur ce concept plus tard.

Tout d'abord, nous appelons la fonction JS preventDefault() contenue dans l'évènement déclenché par le submit, afin d'éviter le rechargement de la page qui est le comportement par défaut du navigateur dans ce cas.

Dans un « try catch », nous lançons donc la requête. Nous verrons son fonctionnement dans la partie suivante.

De plus, nous faisons également la vérification du statut actif ou non de l'utilisateur dans le cas où la requête est correcte.

```
23      const [error, setError] = useState( initialState: "");
```

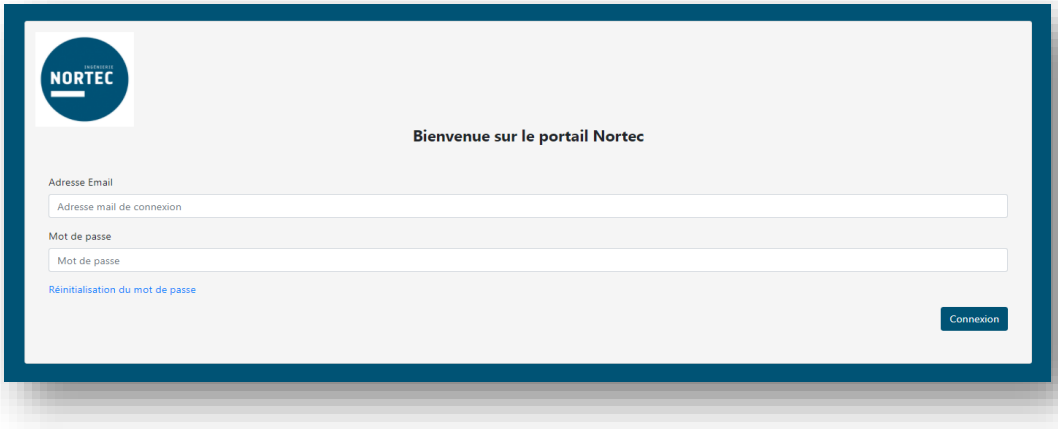
Dans le cas d'une requête qui retournerait une erreur (mauvaises informations de login/mdp par exemple), là nous peuplons la variable error qui a été préalablement déclarée dans un State. Ce qui déclenchera la classe « is-invalid » du Field, et affichera le texte de l'erreur dans la classe « invalid-feedback » du Field.

Nous déclenchons aussi des notifications que nous aborderons plus tard.

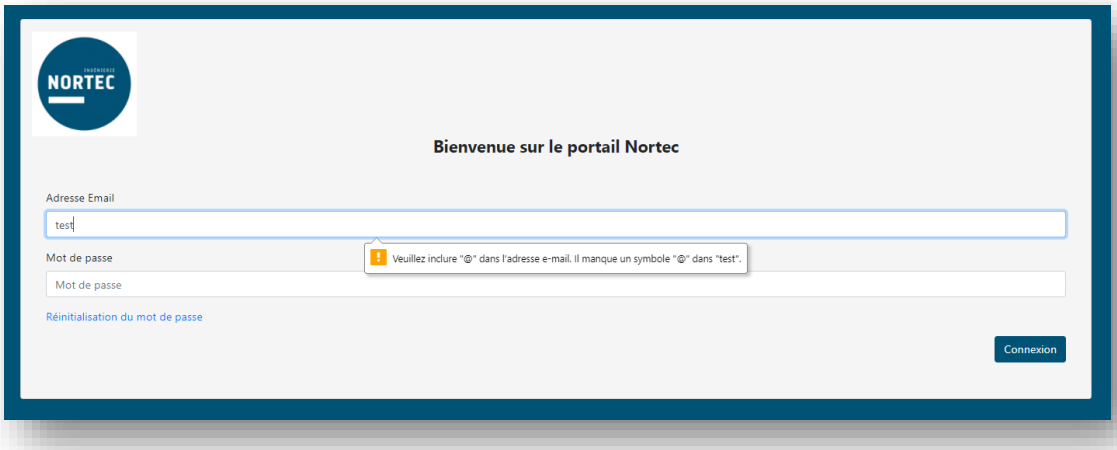
9.4 : Résultat visuel

Nous nous retrouvons donc avec ce résultat pour la page de Login :

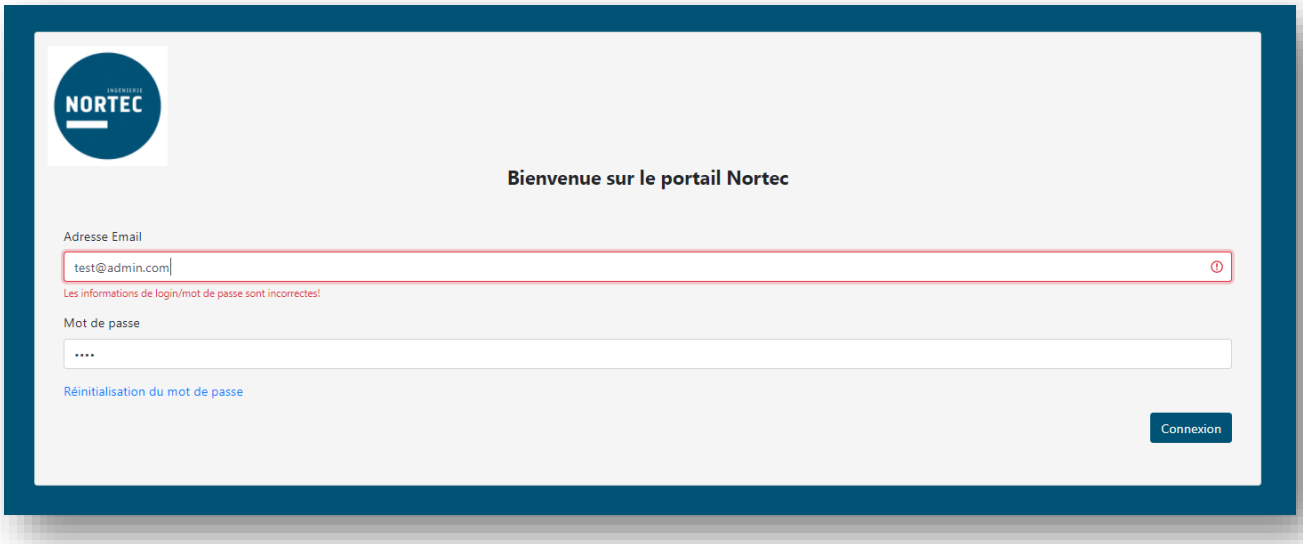
- Design général :



- Erreur de format grâce au type de l'input :



- En cas d'échec d'authentification :



10 : La requête d'authentification

Parlons désormais de la requête d'authentification, qui, souvenons-nous, est appelée lors du submit du formulaire de Login.

Premièrement, par soucis de factorisation, toutes les requêtes concernant l'authentification ont été délocalisées dans un fichier nommé AuthAPI.jsx.

api_links_config.js

```
export const API_URL = "https://localhost:8000/api/";

export const LOGIN_API = API_URL + "login_check";
```

Les liens ont également été délocalisés dans un fichier à part afin de pouvoir les modifier simplement en cas de besoin.

```
function authenticate(credentials) {
    return axios
        .post(LOGIN_API, credentials) AxiosPromise<any>
        .then(response => response.data.token) Promise<AxiosResponse<any>>
        .then(token => {

            //on stocke le token dans le localstorage
            window.localStorage.setItem("authToken", token);

            //on met un header par défaut sur les future requêtes
            setAxiosToken(token);

            return true;
        })

}
```

Grace à Axios, nous pouvons exécuter simplement ces requêtes avec ses fonctions intégrées.

Nous effectuons donc une requête en POST vers le lien d'authentification, en lui passant en paramètre l'objet JSON credentials qui, rappelons-le, contient les données de login entrées par l'utilisateur dans le formulaire.

Nous récupérons le token dans la réponse de la requête, nous le stockons dans le localstorage du navigateur pour éviter de devoir effectuer la requête à chaque fois qu'on en a besoin, et enfin, nous définissons un header par défaut sur les futures requêtes.

```
function setAxiosToken(token) {
    axios.defaults.headers["Authorization"] = "Bearer " + token;
}
```

Cette méthode de stockage du token est fonctionnelle mais, d'après mes recherches, n'est pas très sécurisée ! En effet, il faut éviter de stocker des informations sensibles dans le localstorage car il est accessible à n'importe quel script JS ! Ainsi il est vulnérable à des attaques de type XSS (Cross Site Scripting).

La méthode pour éviter ce problème pourrait être d'utiliser un cookie en « httpOnly » pour le stocker. Malheureusement, par manque de temps, nous n'avons pu changer ce mode de fonctionnement pour le moment.

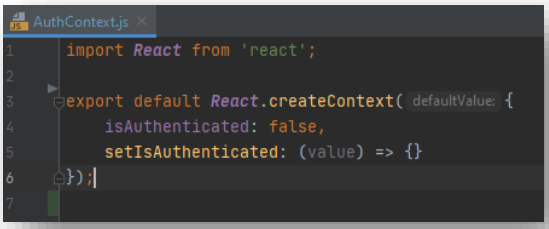
11 : Le « Context » d'authentification

11.1 : Définition

En React, il existe la notion de « Context », qui permet de définir des variables et des fonctions de manières générales, sans que nous ayons à les passer en paramètre à nos Components.

Ainsi, avec ce système, nous évitons la répétition de code et la refactorisation est plus simple.

11.2 : Création du Context



La première chose à faire est de définir la forme que l'on veut donner à notre contexte (un peu à la façon d'une Interface en C# ou JAVA)

Dans un nouveau fichier, nous définissons donc une fonction appelant le createContext de React, auquel nous passons le « contrat » auquel le contexte doit répondre. Ainsi, il doit contenir une variable isAuthenticated qui sera un booléen, et une fonction setIsAuthenticated qui prendra une valeur.

11.3 : Détermination des limites du contexte

Il faut maintenant déterminer les limites à l'intérieur desquelles le contexte s'appliquera, c'est-à-dire quels components pourront accéder à ses informations.

En ce qui concerne l'authentification, dans notre application, ces informations devront être accessibles partout. Ainsi, dans le point d'entrée de l'application (app.js), nous allons pouvoir englober l'ensemble de nos composants avec les balises du contexte. Nous lui passons de plus une variable contextValue qui est de la forme définie dans le contexte. Ainsi, tous les components compris entre les balises du contexte pourront accéder au State ainsi défini, sans avoir à les passer en paramètres.

11.4 : Récupération et utilisation des valeurs du contexte dans les components

Enfin, dans chaque component qui aura besoin d'utiliser les valeurs du contexte, nous devons faire appel au Hook « useContext ».

Nous pouvons donc récupérer la valeur de isAuthenticated, ou l'accès à la fonction setIsAuthenticated, ou même aux deux si besoin.

```
21      const {setIsAuthenticated} = useContext(AuthContext);
```

Et c'est pourquoi, si l'on revient à notre page de Login, que la fonction setAuthenticated est disponible alors qu'elle n'est pas formellement définie dans cette page, mais qu'elle l'est au niveau de la racine de l'application.

```
35    // Gestion du Submit
36    const handleSubmit = async event => {
37        event.preventDefault();
38
39        try {
40            await AuthAPI.authenticate(credentials);
41            setError( value: "");
42
43            if (AuthAPI.getUserActiveStatus())
44            {
45                setIsAuthenticated(true);
46                toast.success( content: "Vous êtes connnecté en tant que " + AuthAPI.getUserFirstNameLastName() + ". Vous avez le statut d'" + AuthAPI.isRole() + ".");
47                history.replace("/projects");
48            }
49            else
50            {
51                setIsAuthenticated(false);
52                toast.error("Votre compte a été désactivé. Veuillez contacter un administrateur!");
53                AuthAPI.logout();
54            }
55
56        } catch {
57            setError( value: "Les informations de login/mot de passe sont incorrectes!");
58            toast.error("Les informations de login/mot de passe sont incorrects!");
59        }
60    };
```

12 : Déterminer si l'utilisateur est authentifié

Nous avons donc mis en place tout le système pour que les components puissent avoir accès à l'information si l'utilisateur est authentifié où non. Mais nous n'avons pas encore parlé de la récupération de cette information proprement dit dans le cas où elle existerait déjà.

```
30  //Permet de voir si on est authentifié ou pas
31  function isAuthenticated() {
32
33      const token = getToken();
34
35      // voir si token encore valide
36      if (token) {
37
38          const jwtData = jwtDecode(token);
39          return jwtData.exp * 1000 > new Date().getTime();
40
41      }
42      return false;
43  }
```

Nous allons pouvoir le faire via une fonction dans AuthAPI.js, qui a été appelée dans l'état initial du State dans app.js (voir page 80), appelée isAuthenticated().

```
129 function getToken() {
130     return window.localStorage.getItem( key: "authToken");
131 }
```

Cette fonction va donc récupérer le token, vérifier qu'il existe, le décoder, et vérifier qu'il n'a pas expiré. Et enfin retourner un booléen en fonction du résultat.

La vérification de l'expiration du token se fait avec une des propriétés de celui-ci. En effet, il a par défaut une validité au bout de laquelle les requêtes ne seront plus acceptées par le serveur. Il faut donc vérifier que cette date d'expiration n'est pas postérieure à la date actuelle.

La propriété « exp » du JWT est exprimée en Timestamp, temps passé depuis le 1er janvier 1970 en secondes. Or la fonction getTime() de Date est exprimée en milliseconde. D'où la multiplication par 1000 du premier pour pouvoir effectuer correctement la comparaison.

13 : Charger le JWT au démarrage de l'application

Enfin, il reste à gérer les cas où l'utilisateur aurait quitté l'application sans se délogger, ou celui où il actualiserait tout simplement la page, et que le token est encore valable. En effet, nous risquons des soucis car nous ne chargerions pas le token pour vérifier l'authentification dans ce cas.

Ainsi, nous pouvons le charger automatiquement au démarrage de l'application, vérifier sa validité, et ainsi éviter qu'il ne doive se réauthentifier le cas échéant.

```
95  function setup() {
96      const token = getToken();
97
98      if (token) {
99          const jwtData = jwtDecode(token);
100         if (jwtData.exp * 1000 > new Date().getTime()) {
101             setAxiosToken(token);
102             return true;
103         } else {
104             return false;
105         }
106     } else {
107         return false;
108     }
109 }
```

Nous faisons ce traitement dans une fonction, toujours dans AuthAPI.js, avec pour nom setup().

Etant très similaire à la fonction isAuthenticated() précédemment évoquée, il y a possibilité de refactoriser.

```
35  AuthAPI.setup();
36
37
38  const App = () => {
39
```

Et nous l'implémentons au tout début du fichier racine app.js, avant même le chargement de l'application.

14 : La déconnexion

```
<button onClick={handleLogout} className="btn btn-danger ml-3">
    Déconnexion
</button>
```

```
const handleLogout = () => {
    AuthAPI.logout();
    setIsAuthenticated(false);
    toast.info( content: "Vous êtes déconnecté ");
    history.push("/");
};
```

Enfin, il nous reste à aborder le Logout de l'utilisateur, lorsqu'il clique sur le bouton associé.

Lors du clic sur le bouton, nous appelons la fonction handleLogout, qui elle-même appelle la fonction logout du fichier AuthAPI. Le rôle de cette dernière étant de retirer le token du localstorage.

```
function logout() {
    window.localStorage.removeItem( key: "authToken");
    delete axios.defaults.headers["Authorization"];
}
```

15 : Conclusion

Nous arrivons enfin à la fin de cette longue et complexe partie traitant de l'authentification.

Nous avons donc posé les bases d'une application avec des comptes utilisateurs pouvant avoir différents rôles, une connexion sécurisée (malgré le problème du localstorage), et différentes vérifications afin que nous ayons le moins de problème possible par la suite.

VII : DEVELOPPEMENT FRONT END

1 : Un Mock pour les données

1.1 : Définition

Pour le développement Front, qui est celui que nous avons effectué en premier après l'élaboration du diagramme de classe, nous avions besoin d'un jeu de donnée s'écrivant facilement, dans un format qui se rapprochait le plus possible de ce que l'API nous renverrait à terme, et qui soit bien plus facilement modifiable qu'une base de données.

Le choix d'utiliser un Mock, c'est-à-dire un jeu de faux objets simulés s'est donc avéré indispensable. Au fur et à mesure des modifications du client, nous pouvions donc l'adapter sans avoir besoin de faire de lourdes modifications, ce qui aurait été le cas si nous avions directement implémenté une base de données et l'API en elle-même.

1.2 : Ecriture

Nous nous sommes basés sur le diagramme de classes existant afin de déterminer la structure du Mock, en JSON, et dans lequel nous avons pu rajouter de fausses informations pour tester le Front de l'application au fur et à mesure de son développement.

Pour éviter au maximum la redondance dans l'écriture, gagner du temps, et également éviter les erreurs, j'ai imaginé un système de génération via des fonctions JS qui appelle les informations redondantes et les place dans certains endroits du JSON.

1.3 : Structure

Il se décompose donc en trois fonctions JS principales qui représentent les principales données dont nous aurons besoin. Et en quatre autres fonctions qui s'occuperont d'effectuer le tri dans les précédentes.

1.3.1 : Les projets

Une fausse liste de projets.

```
fakeData.js
function fakeData() {
    return [
        {
            id: 0,
            name: "La grande flèche qui pique le cul du ciel",
            description: "Aussi appelée la dame de fer!",
            photo: "../img/projects-img/projects-general-img/0-project-img.jpg",
            adresse1: "Champ de Mars",
            adresse2: "",
            codePostal: "75000",
            ville: "Paris",
            dateDebut: "2015-02-27",
            dateFinReelle: "2020-02-29",
            date_fin_prevues:
                [
                    {
                        id: 0,
                        date: "2017-03-27"
                    },
                    {
                        id: 1,
                        date: "2017-04-27"
                    }
                ],
            nomMOEX: "Capitaine Haddock",
            nomOPC: "Tintin",
            contactClient: "moulinsart@herge.com"
        },
```

1.3.2 : Les rapports

Une fausse liste de rapports. Nous pouvons voir que la partie projet est générée par une fonction qui va venir rajouter les informations du projet auquel est rattaché le rapport au bon endroit, de même que pour les entreprises.

```
function fakeListReports() {
    return [
        {
            id: 0,
            project: projectById( id: 7),
            redacteur: "Ced",
            dateRedaction: "2020-03-03T18:25:43",
            status: "in-progress",
            proprete_conformity: true,
            proprete_imputation: [], //pas implémenté
            proprete_comment: "Tout est très propre !",
            proprete_comment_intern:
                "Ils ont pas été contents mais je m'en fous ils ont nettoyé!!",
            securityConformity: true,
            propreteAccessComment: "RAS",
            propreteAccessCommentIntern:
                "Y'a fallu leur botter les fesses pour qu'ils mettent cette fichue barrière!",
            installations: "18 échafaudages, 5 tracto-pelles",
            lots: [
                {
                    id: 0,
                    numeroLot: "101ABC674",
                    libelleLot: "FOURNISSEUR SPECIAL",
                    entreprise: companyById( id: 3),
                    effectifPrevu: 5,
                    effectifConstate: 2
                },
                {
                    id: 1,
                    numeroLot: "10242342DEF",
                    libelleLot: "INSTALLATION TELEPHONE",
                    entreprise: companyById( id: 4),
                    effectifPrevu: 5,
                    effectifConstate: 4
                },
                {
                    id: 2,
                    numeroLot: "102AZZZAA1FEF",
                    libelleLot: "INSTALLATION FIBRE",
                    entreprise: companyById( id: 4),
                    effectifPrevu: 5,
                    effectifConstate: 7
                },
                {
                    id: 3,
                    numeroLot: "102EEZDSS",
                    libelleLot: "MISE EN PLACE ANTENNES SPECIALES",
                    entreprise: companyById( id: 4),
                    effectifPrevu: 5,
                    effectifConstate: 8
                },
                {
                    id: 4,
                    numeroLot: "10QSSQSDSSDHJH",
                    libelleLot: "FOURNITURE ELECTRICITE",
                    entreprise: companyById( id: 2),
                    effectifPrevu: 5,
                    effectifConstate: 5
                }
            ],
```

```
                    effectifConstate: 5
                }
            ],
            photos: [
                {
                    id: 0,
                    link: "../img/projects-img/projects-general-img/0-project-img.jpg",
                    type: "security"
                },
                {
                    id: 1,
                    link: "../img/projects-img/projects-general-img/0-project-img.jpg",
                    type: "proprete"
                }
            ],
            echeances: []
        },
```

```
function fakeListCompanies() {
    return [
        {
            id: 0,
            nom: "Le roi de la moquette",
            adresse1: "36 quai des Orfèvres",
            adresse2: "",
            codePostal: "75000",
            ville: "Paris",
            mail1: "moquetteman@raslamoquette.com",
            mail2: "razmoket@cesttouffu.org"
        },
        {
            id: 1,
            nom: "Béton Armé",
            adresse1: "56 rue betonnée",
            adresse2: "Dans le béton",
            codePostal: "56123",
            ville: "Béton-City",
            mail1: "ilestdurmonbeton@arme.com",
            mail2: ""
        },
        {
            id: 2,
            nom: "Centrale nucléaire de Springfield",
            adresse1: "777 rue de l'atome",
            adresse2: "Derrière Chez Mr Burns",
            codePostal: "78451",
            ville: "Springfield",
            mail1: "montyburns@enfoire.com",
            mail2: "wellonsmithers@larbin.com"
        },
```

1.3.3 : Les entreprises

Une fausse liste d'entreprises

1.3.4 : Les fonctions de tri

Et enfin les fonctions de tri qui permettent de sélectionner un élément par son id dans les trois fonctions que l'on trouve plus haut.

```
function companyById(id) {
    const companies = fakeListCompanies();
    const companyById = filterById(companies, id);

    return companyById[0];
}

function projectById(id) {
    const projects = fakeData();
    const projectById = filterById(projects, id);

    return projectById[0];
}

function reportById(id) {
    const reports = fakeListReports();
    const reportById = filterById(reports, id);

    return reportById[0];
}

function filterById(tab, id) {
    return tab.filter(
        t => t.id === id
    );
}
```

1.4 : Implémentation

Il ne reste plus qu'à implémenter le Mock dans l'application React, ce qui se fait facilement par appel des fonctions désirées dans une autre fonction qui se trouve dans les components qui en ont besoin. Fonction qui elle-même est appelée dans le Hook useEffect, sur lequel je reviendrai dans la partie suivante. Ici par exemple dans le component de la liste des projets.

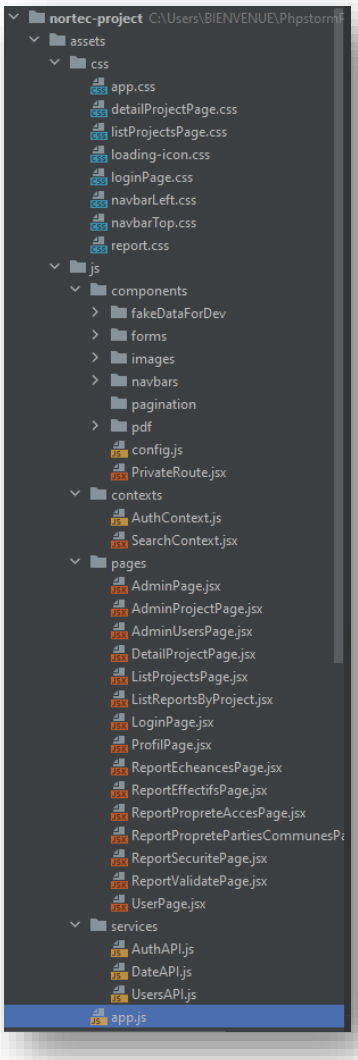
```
useEffect( effect: () => {
    fetchProjects().then(r => "");
}, deps: []);

// -------------------------- Récupérer tous les projets de l'utilisateur -----

const fetchProjects = async () => {
    try {
        const data = await fakeData.fakeListProjects();
        setProjects(data);
        setLoading( value: false);

    } catch (error) {
        toast.error("Erreur lors du chargement de la liste des projets");
    }
}
```

1.5 : Conclusion

L'utilisation du Mock nous aura été donc d'une grande aide pour le développement de la partie Front, et ensuite de la partie Back. Néanmoins, dans la suite de cet exposé, les extraits de code ne feront pour la plupart pas référence à celui-ci, mais aux véritables requêtes vers l'API qui ont été implémentées. Exception faite de la partie de l'application qui est toujours en cours de développement, et par conséquent qui repose toujours sur ce Mock.

2 : Structure d'une application React

2.1 : Structure générale



Ci-contre, nous pouvons voir l'arborescence générale de notre application React.

Une application React est une Single Page Application, ce qui veut dire qu'elle est accessible via une page Web unique. Le but étant d'éviter le chargement d'une nouvelle page à chaque action, et ainsi de fluidifier l'expérience utilisateur. C'est d'ailleurs une des raisons principales pour laquelle nous avons choisi cette technologie pour notre application.

Ainsi, nous avons le fichier racine app.js qui sera le point d'entrée de celle-ci et en gros le chef d'orchestre de l'application, car c'est lui qui déterminera quelle page charger à quel moment.

D'ailleurs, la notion de page peut être contradictoire avec ce que je viens d'expliquer. Mais il faut savoir que grâce à React Router DOM, nous pouvons faire du routing et définir différentes URLs et pages dans l'application. Mais le changement de page se fera de toute façon sans avoir à rafraichir le contenu du navigateur.

Enfin, les différentes pages feront appel à différents components, que ce soient les fichiers css spécifiques à chacune, les services que nous avons mis en place, ou même aux components que nous avons spécifiquement créé pour l'application.

Les components sont l'unité de base d'une application React, et c'est vraiment la brique de son architecture.

2.2 : Structure d'un Component

Ci-contre se trouve donc la structure de base d'un Component React. Avec la plupart des éléments que nous avons utilisé pour le développement de l'application.

Tout en haut, nous trouvons donc les imports des différentes dépendances.

Dans cet exemple j'importe React lui-même ainsi que deux de ses utilitaires. Et également un fichier css.

Le corps du code se compose finalement d'une fonction en JSX, qui porte le nom du Component. Et à la fin que nous exportons afin qu'elle soit accessible de l'extérieur du fichier.

Dans cette fonction, nous trouvons donc la déclaration du State, notion dont nous avons déjà parlé plus haut, qui représente l'état d'un élément. Il s'agit d'un Hook de React.

La fonction useEffect ensuite, un Hook également, qui s'exécutera au chargement du Component. Généralement c'est ici que nous appellerons la fonction qui déclenchera les requêtes vers l'API.

Ensuite nous avons une déclaration de fonction, comme nous pouvons en avoir plusieurs par Component.

Et enfin, la fonction retourne, et c'est là l'originalité de React et qui simplifie énormément de choses quant à l'écriture des pages, du html ! Html qui sera l'affichage du Component en lui-même.

3 : Le Routing

3.1 : Fonctionnement

```
return (
    <AuthContext.Provider value={contextValue}>

            <HashRouter>

                <SearchContext.Provider value={searchContextValue}>
                    {isAuthenticated && <NavbarTopWithRouter/>}
                </SearchContext.Provider>

                <Switch>
                    {!isAuthenticated && <Route path="/" component={LoginPage}/>}
                    <PrivateRoute path="/profil/:id" component={ProfilPage}/>
                    <PrivateRoute path="/project/:id/:idReport/effectifs" component={ReportEffectifsPage}/>
                    <PrivateRoute path="/project/:id/:idReport/propreteacces" component={ReportPropreteAccesPage}/>
                    <PrivateRoute path="/project/:id/:idReport/securite" component={ReportSecuritePage}/>
                    <PrivateRoute path="/project/:id/:idReport/propretepartiescommunes"
                                  component={ReportPropretePartiesCommunesPage}/>
                    <PrivateRoute path="/project/:id/:idReport/echeances" component={ReportEcheancesPage}/>
                    <PrivateRoute path="/project/:id/:idReport/validate" component={ReportValidatePage}/>

                    <PrivateRoute path="/project/:id/listReports" component={ListReportsByProject}/>
                    <PrivateRoute path="/project/:id" component={DetailProjectPage}/>

                    <PrivateRoute path="/admin/userslist" component={AdminUsersPage}/>
                    <PrivateRoute path="/admin/user/:id" component={AdminUserPage}/>
                    <PrivateRoute path="/admin/project/:id" component={AdminProjectPage}/>
                    <PrivateRoute path="/admin/project" component={AdminProjectsPage}/>
                    <PrivateRoute path="/admin/company/:id" component={AdminCompanyPage}/>
                    <PrivateRoute path="/admin/company" component={AdminCompaniesPage}/>
                    <PrivateRoute path="/admin/:id" component={AdminPage}/>


                    <SearchContext.Provider value={searchContextValue}>
                        <PrivateRoute path="/projects" component={ListProjectsPage}/>
                    </SearchContext.Provider>

                </Switch>

            </HashRouter>

            < ToastContainer
                position={toast.POSITION.BOTTOM_LEFT}
            />

    </AuthContext.Provider>

);
```

Le routing est géré par la bibliothèque React Router DOM. C'est elle qui va permettre de faire matcher les routes que nous aurons dans l'URL avec un Component React en particulier.

Dans notre cas, c'est le fichier app.js, qui va porter la responsabilité de celui-ci. D'où le fait que je l'ai nommé plus haut le « chef d'orchestre » de l'application.

Il comprend trois balises, deux qui englobent et une, <Route> qui prendra comme paramètre un chemin et le component à charger. L'important ici est de toujours déclarer les chemins du plus spécifique au moins spécifique.

Le Component PrivateRoute a quant à lui été créé par nos soins pour qu'il gère en interne l'authentification et ainsi protéger les routes.

3.2 : Le Component PrivateRoute

Afin de protéger les routes et d'éviter qu'un utilisateur non authentifié n'y accède, nous avons créé ce Component par soucis de lisibilité et de refactorisation.

Ainsi, il prend comme paramètre la route et le Component associé, comme le Component Route de React Router DOM.

A l'intérieur, nous allons donc faire appel au AuthContext, qui, rappelons-nous, porte en lui l'état d'authentification de l'utilisateur dans la variable isAuthenticated.

Ainsi, nous vérifions que l'utilisateur est bien authentifié grâce à elle, et nous retournons la Route paramétrée de React Router DOM le cas échéant. Dans le cas contraire, nous faisons une redirection vers la page d'accueil qui n'est tout simplement que la page de login de l'application.

Notre application maintenant comporte donc toutes les routes nécessaires, protégées des utilisateurs non authentifiés.

4 : L'implémentation de Bootstrap et du CSS

4.1 : Bootstrap

Bootstrap est un Framework css très populaire dans le développement Web développé par Twitter. Il permet de mettre en forme très simplement par son système de class css prédéfinies une page Web ainsi que son responsive design.

L'implémentation dans notre application s'est faite de façon très simple. Juste en important les fichiers nécessaires à son fonctionnement dans le fichier app.js. Et bien sûr après les avoir téléchargé et copié dans le dossier correspondant dans notre arborescence.

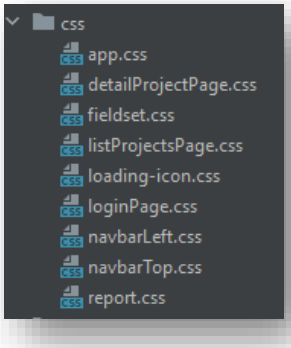
Nous utilisons la version 4 du Framework qui est la dernière en date.

```
12  import 'bootstrap/dist/js/bootstrap.bundle.min';
13  import 'bootstrap/dist/css/bootstrap.css';
```

Il existe également une librairie Bootstrap React, avec ses propres Components mais nous avons préféré ne pas l'utiliser, la trouvant moins intuitive que le faire soit même…

4.2 : Les fichiers CSS personnalisés

Une des forces de React pour le design de mon point de vue, est la possibilité d'utiliser autant de fichier CSS que l'on désire, très simplement, pour les moments où Bootstrap seul ne s'avère pas suffisant ou inadapté.



Nous avons donc créé autant de fichier CSS que nous importons à la demande dans les Components où ils sont nécessaires.

Par exemple, à droite, j'ai même surchargé le container de Bootstrap car il ne nous laissait pas assez de place pour arriver à un design correct sur l'ensemble de l'application.

```
app.css
1   main {
2       margin-top: 150px
3   }
4
5
6   @media (min-width: 1400px) {
7       .container {
8           max-width: 1360px;
9       }
10  }
11
12  @media (min-width: 1700px) {
13      .container {
14          max-width: 1660px;
15      }
16  }
```

4.3 : Cas particulier

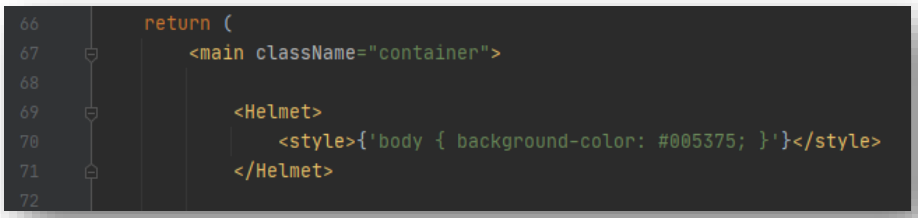
Nous avons cependant dû faire face à un soucis qui provient des limitations de la prise en charge du CSS par React.

En effet, il nous était impossible de faire un changement de couleur de fond sur une page en particulier, la propriété background-color sur le body n'était pas prise en compte dans le fichier concerné.

Nous avons trouvé une parade en utilisant une bibliothèque React qui permet d'injecter du CSS sur une page particulière : Helmet.

Son utilisation est par ailleurs très simple en rajoutant dans le template sa balise, ainsi que la balise style et la propriété css que nous voulons voir appliquer à cet endroit.

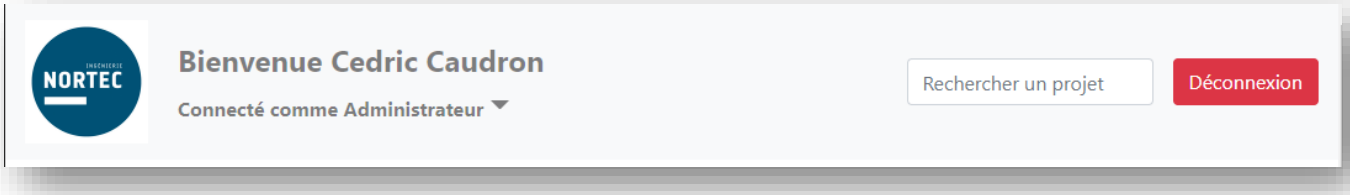
Ainsi nous avons pu changer le background-color de la page de Login et uniquement sur celle-ci.



5 : Les menus

5.1 : Le menu du haut

Dans le design défini, nous avons prévu une barre de menu en haut de l'application, accessible partout sauf sur la page de login, et qui contient les informations de l'utilisateur ainsi qu'un champ pour effectuer des recherches et le bouton de déconnexion.



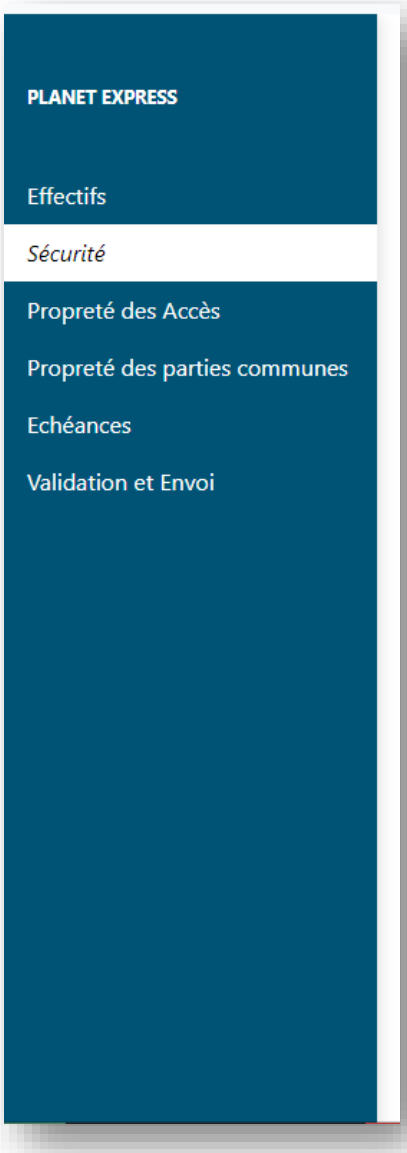
Dans la réalisation, rien d'exceptionnel. Il s'agit d'une navbar bootstrap fixée en haut. Avec les informations que l'on récupère du token pour le nom, prénom et le rôle. L'affichage du logo se fait dans un Component à part, et est un lien pour revenir à la liste des projets. Et la barre de recherche utilise un Context afin que sa valeur puisse être accessible par d'autres Components.

Voici un extrait du code de la navbar.

```
return (

    <nav className="navbar navbar-expand-lg navbar-light bg-light lighten-5 mb-4 fixed-top">

        <NavLink className="navbar-brand" to="/projects"><LogoCompanyComponent style={{width: "100px"}}/></NavLink>

        <div className="collapse navbar-collapse" id="navbarSupportedContent">

            <ul className="navbar-nav mr-auto">

                <li className="nav-item dropdown">
                    <a className="nav-link dropdown-toggle navbar-top-style navbar-top-text-style"
                       id="navbarDropdownMenuLink" data-toggle="dropdown"
                       aria-haspopup="true" aria-expanded="false">Bienvenue {completeNameUser} <br/>
                        <span className="text-statut">Connecté comme {AuthAPI.isAdmin() ?
                            'Administrateur'
                            :
                            'Utilisateur'}
                        </span>
                    </a>
                    <div className="dropdown-menu dropdown-primary" aria-labelledby="navbarDropdownMenuLink">
                        <NavLink className="dropdown-item" to={"/profil/" + userId}>Votre profil</NavLink>
                        {AuthAPI.isAdmin() &&
                        <NavLink className="dropdown-item" to={"/admin/" + userId}>Panneau Administration</NavLink>
                        }
                        <a className="dropdown-item" href="#">Something else here</a>
                    </div>
                </li>
            </ul>
```

5.2 : Le menu de gauche

Sur certaines pages de l'application, soit sur celles de la génération des rapports, il y a une autre navbar, verticale cette fois, afin de permettre une navigation facile entre les différents thèmes du rapport qui est en cours de rédaction.



Une fois encore, pour la réalisation, rien de bien exceptionnel. Bootstrap nous simplifiant grandement la tâche.

Elle est composée du titre du projet sur lequel le rapport est généré, ainsi que la série de liens pointant vers les différentes parties du rapport.

Il y a une notion par ailleurs que nous n'avions pas abordé, qui est la récupération des paramètres de l'url. Afin de récupérer l'id du projet.

Pour cela nous utilisons la props « match ». Et nous récupérons donc un tableau se trouvant dans la propriété params. Qui contient donc tous les paramètres de l'url définis dans le app.js.

```
10  const NavbarLeft = ({match, selected}) => {
11
12      const id = match.params;
```

```
<PrivateRoute path="/project/:id/:idReport/propreteacces" component={ReportPropreteAccesPage}/>
```

Avec la propriété :id dans la chaine de caractère et match.params, nous récupérons pour cette page un tableau avec deux clés valeurs. Une id et un idReport.

Nous avons besoin de ces paramètres afin d'établir les liens corrects pour la navigation du menu.

```
return (
    <div className="vertical-nav" id="sidebar">

        <p className="text-white font-weight-bold text-uppercase px-3 small pb-4 mt-5">{project.name}</p>

        <ul className="nav flex-column mb-0">

            <li className="nav-item">
                <NavLink to={"/project/" + project.id + "/" + id.idReport + "/effectifs"}
                         className={"nav-link font-italic" + (selected === 'effectifs' && " selected")}>
                    Effectifs
                </NavLink>
            </li>
            <li className="nav-item">
                <NavLink to={"/project/" + project.id + "/" + id.idReport + "/securite"}
                         className={"nav-link font-italic" + (selected === 'securite' && " selected")}>
                    Sécurité
                </NavLink>
            </li>
            <li className="nav-item">
                <NavLink to={"/project/" + project.id + "/" + id.idReport + "/propreteacces"}
                         className={"nav-link font-italic" + (selected === 'proprete' && " selected")}>
                    Propreté des Accès
                </NavLink>
            </li>
            <li className="nav-item">
                <NavLink to={"/project/" + project.id + "/" + id.idReport + "/propretepartiescommunes"}
                         className={"nav-link font-italic" + (selected === 'propetepartiecommune' && " selected")}>
                    Propreté des parties communes
                </NavLink>
            </li>
```

De plus, nous pouvons noter que les balises pour configurer les liens sont ici NavLink. Il s'agit en fait d'un Component intégré à React Router DOM qui évite le rechargement de la page à chaque fois que l'on suit un lien.

6 : Les pages de l'application

6.1 : Généralités concernant les pages comprenant un formulaire

Nous allons nous intéresser maintenant aux pages de l'application telles qu'elles ont été développées.

Sur les pages comprenant un formulaire, nous avons utilisé le même template pour gérer les cas de création ou de modification. Ainsi, dans le cas de la modification, les informations préalablement enregistrées s'affichent donc dans les champs du formulaire.

```
55      const [edit, setEdit] = useState( initialState: false);
```

```
const {id = "new"} = match.params;
```

Le système qui a été mis en place se base sur un State nommé edit, booléen, dont on détermine la valeur en fonction de l'URL sur laquelle nous nous trouvons. En effet, dans le cas d'une création, en lieu et place de l'id de l'élément, nous avons configuré l'URL pour qu'elle contienne le mot clé « new » par défaut, remplacé par la valeur de l'id s'il existe.

```
useEffect( effect: () => {
    if (id !== "new") {
        setEdit( value: true);
        fetchUser(id).then(r => "");
        fetchProjects().then(r => "");

    } else {
        setLoading( value: false);
    }

}, deps: [id]);
```

Dans le useEffect, nous effectuons donc la vérification, et nous mettons à jour le state en fonction du résultat.

Et enfin, dans le template nous gérons les deux cas, si l'on est en mode édition ou création.

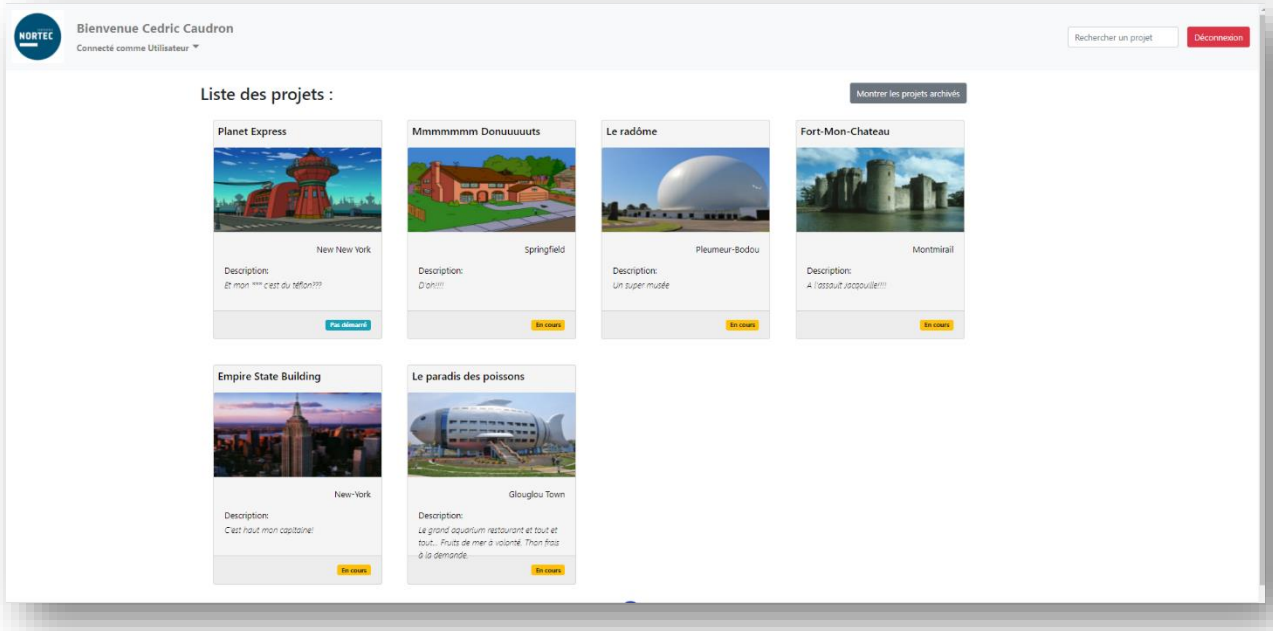
```
260             {!edit && <h1>Création d'un Utilisateur</h1> || <h1>Modification de l'utilisateur</h1>}
```

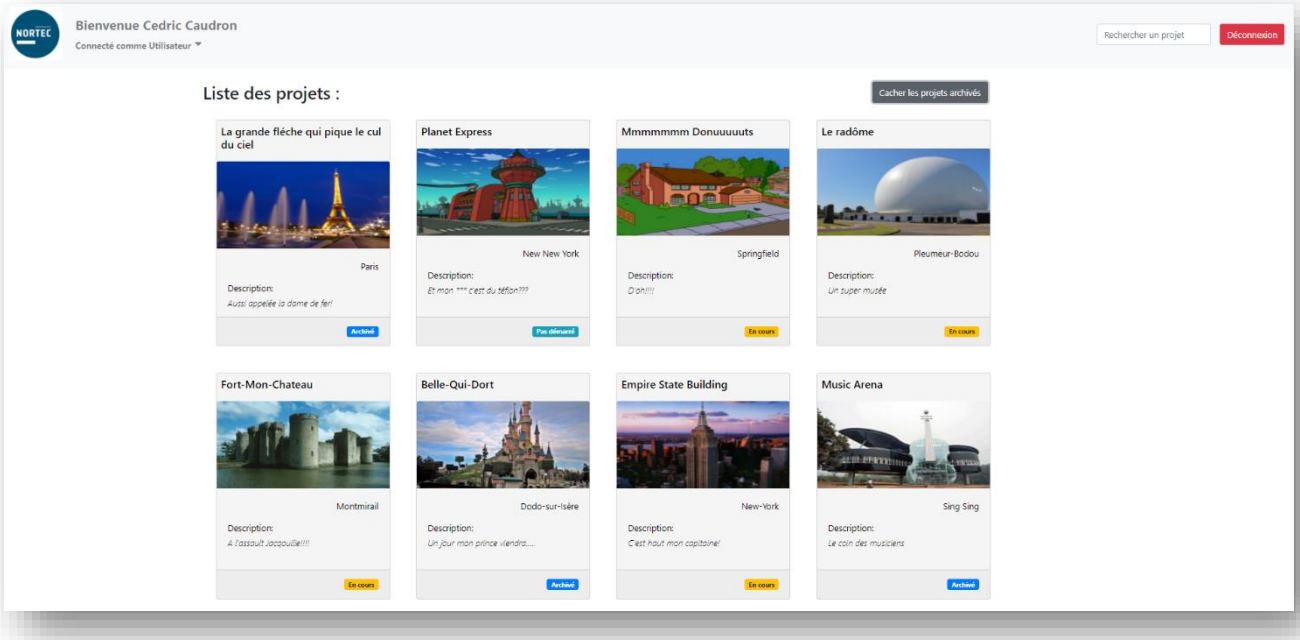
6.2 : La page de Login

Je mets une partie ici tout de même pour la page de Login car elle y a sa place. Néanmoins, l'ayant déjà détaillée dans la partie Authentification, je vous invite à y retourner si besoin est.

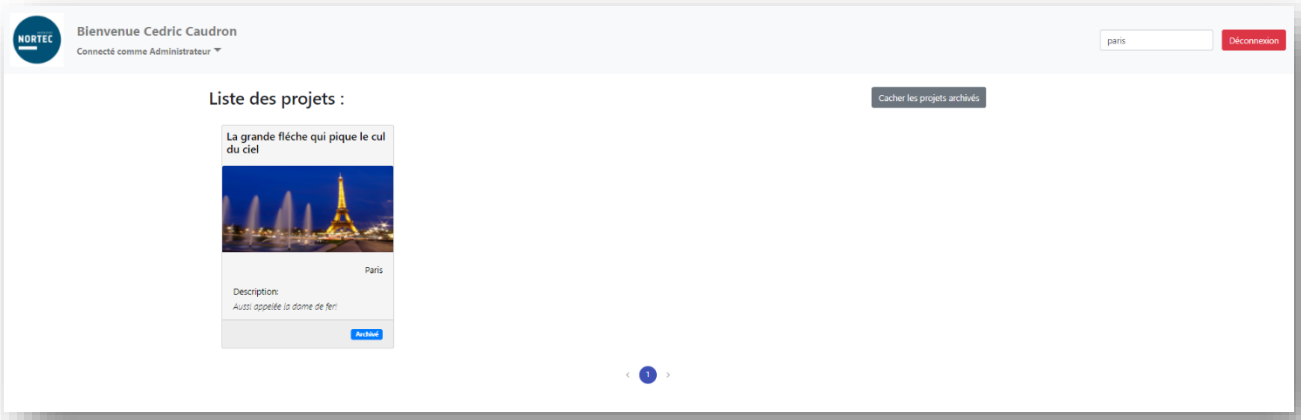
6.3 : La liste des projets

6.3.1 : Design

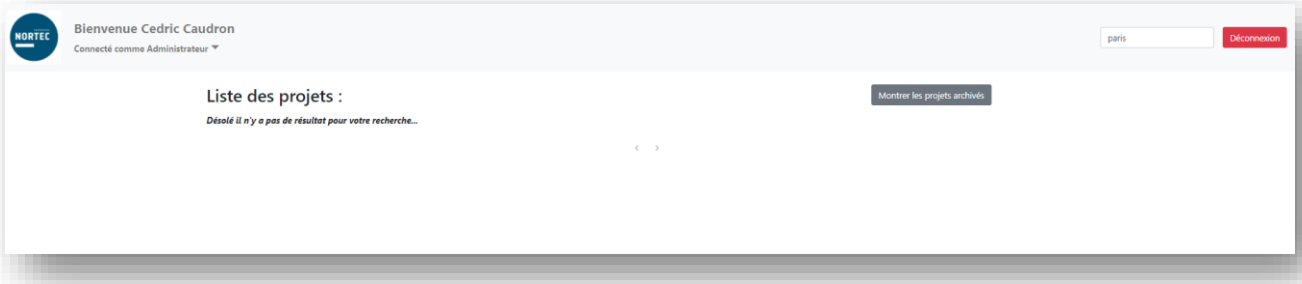




Deux aspects, en montrant ou non les projets archivés.



La recherche est fonctionnelle. Nous pouvons rechercher par nom de projet, par ville, ou par statut. Ici une recherche pour la ville de Paris, parmi l'ensemble des projets, même ceux archivés.



La même recherche sans inclure les projets archivés, elle ne remonte aucun résultat et affiche donc un message.

La réalisation niveau design a encore une fois été assez simple grâce à Bootstrap. Chaque projet étant inclus dans une Card.

```
<div className="card-group">

    {!loading && <>

        {paginationConfig.paginatedItems.length === 0 ?
            <p className="mt-2 font-weight-bold font-italic">Désolé il n'y a pas de résultat pour votre
                recherche...</p>
            :
            paginationConfig.paginatedItems.map(project =>

                <Link key={project.id} to={'/project/' + project.id} style={{textDecorationLine: "none", color: "black"}} >

                    <div className="card m-4" style={{width: '20rem', height: '26rem'}}>

                        <h5 className="card-title p-2">{project.name}</h5>

                        <ImgWithStyleComponent
                            className="card-img-top"
                            src={project.photo}
                            alt={project.name}
                            style={{height: "10rem"}}
                        />
                        <div className="card-body">
                            <p className="card-text text-right mb-3">{project.ville}</p>
                            <p className="card-text mb-1">Description:</p>
                            <p className="font-weight-light font-italic card-text">{project.description}</p>
```

6.3.2 : La recherche

La mise en place de la fonction de recherche aura nécessité l'utilisation d'un Context, afin de pouvoir accéder aux données du formulaire qui se trouve dans un Component différent.

```
28        const {searchValue} = useContext(SearchContext);
```

Je ne réexpliquerai pas le fonctionnement des contextes en React vu que je l'ai déjà fait. Il faut juste savoir que sur cette page nous récupérons la valeur entrée par l'utilisateur dans le formulaire dans la variable searchValue.

Ensuite, nous trions la liste complète des projet suivant son statut archivé ou non (je reviendrai dessus dans la partie de gestion des dates car le statut est déterminé automatiquement).

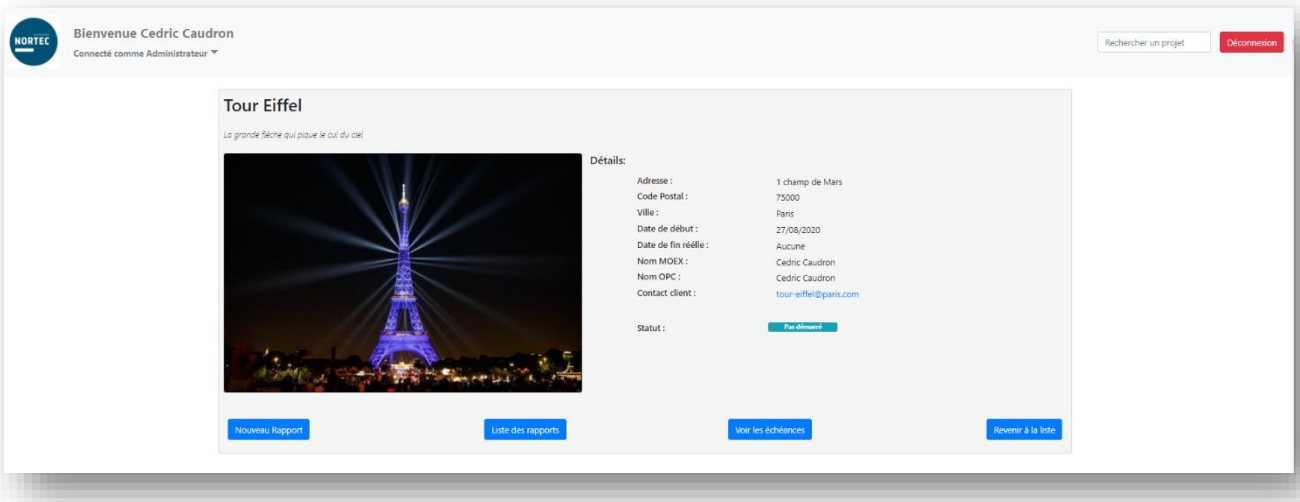
```
const filteredArchivedProjects = projects.filter(
    archivedProjects
        ?
        p =>
            DateAPI.determineStatus(p.dateDebut, DateAPI.verifyDateExist(p.dateFinReelle)) === 'archived' ||
            DateAPI.determineStatus(p.dateDebut, DateAPI.verifyDateExist(p.dateFinReelle)) === 'no_start' ||
            DateAPI.determineStatus(p.dateDebut, DateAPI.verifyDateExist(p.dateFinReelle)) === 'in_progress' ||
            DateAPI.determineStatus(p.dateDebut, DateAPI.verifyDateExist(p.dateFinReelle)) === 'finished'
        :
        p =>
            DateAPI.determineStatus(p.dateDebut, DateAPI.verifyDateExist(p.dateFinReelle)) === 'no_start' ||
            DateAPI.determineStatus(p.dateDebut, DateAPI.verifyDateExist(p.dateFinReelle)) === 'in_progress' ||
            DateAPI.determineStatus(p.dateDebut, DateAPI.verifyDateExist(p.dateFinReelle)) === 'finished'
);  //TODO Trouver façon de refactoriser cette condition... C'est moche...
```

Enfin, nous trions donc cette nouvelle liste de projet en fonction de la valeur de la recherche. Pour éviter les soucis, nous appliquons la méthode toLowerCase afin que la casse n'entre pas en jeu.

```
81      const filteredProjects = filteredArchivedProjects.filter(
82          p =>
83              p.name.toLowerCase().includes(searchValue.toLowerCase()) ||
84              STATUS_LABEL[DateAPI.determineStatus(p.dateDebut, DateAPI.verifyDateExist(projects.dateFinReelle))].toLowerCase().includes(searchValue.toLowerCase()) ||
85              p.ville.toLowerCase().includes(searchValue.toLowerCase())
86      );
```

Nous filtrons donc le résultat en fonction du nom de projet, de la ville et du statut qui lui également est déterminé automatiquement, j'y reviendrai plus tard.

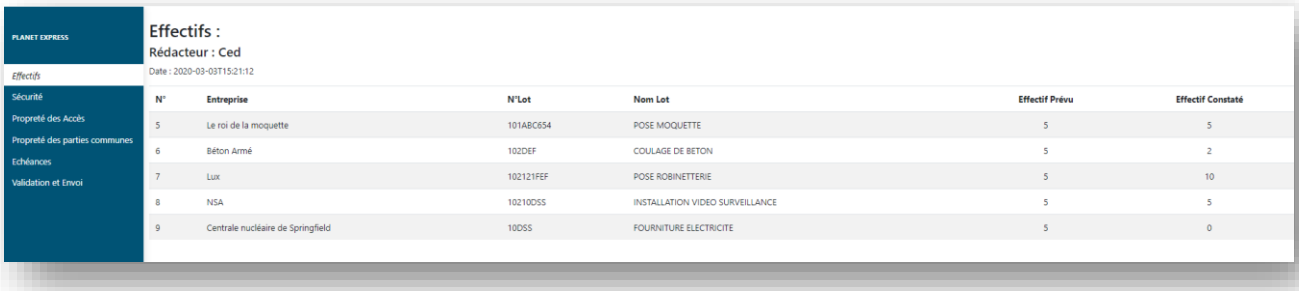
6.4 : La page de détail d'un projet



Cette page n'a pas de mécanique particulière à décrire. Elle a subi quelques changements par rapport à la maquette, mais son utilité reste de montrer des détails pour chaque projet, et de fournir les liens vers les différentes actions en lien avec ce projet.

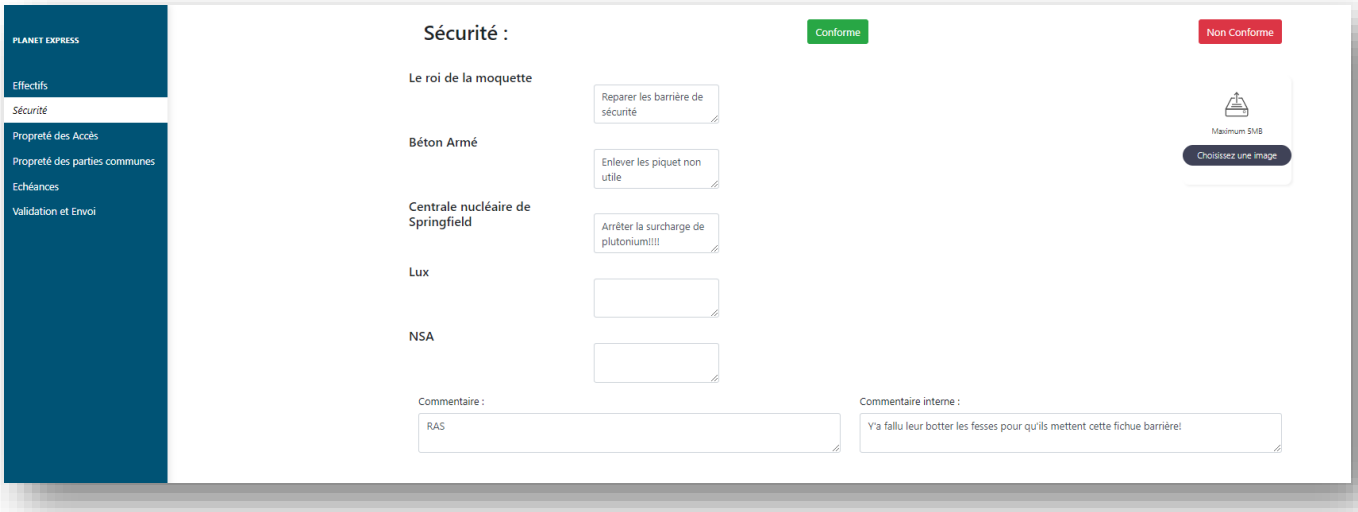
6.5 : La page des effectifs

Attention, toutes les pages qui suivent qui concernent la génération des rapports, à l'heure actuelle, travaillent encore sur le Mock.



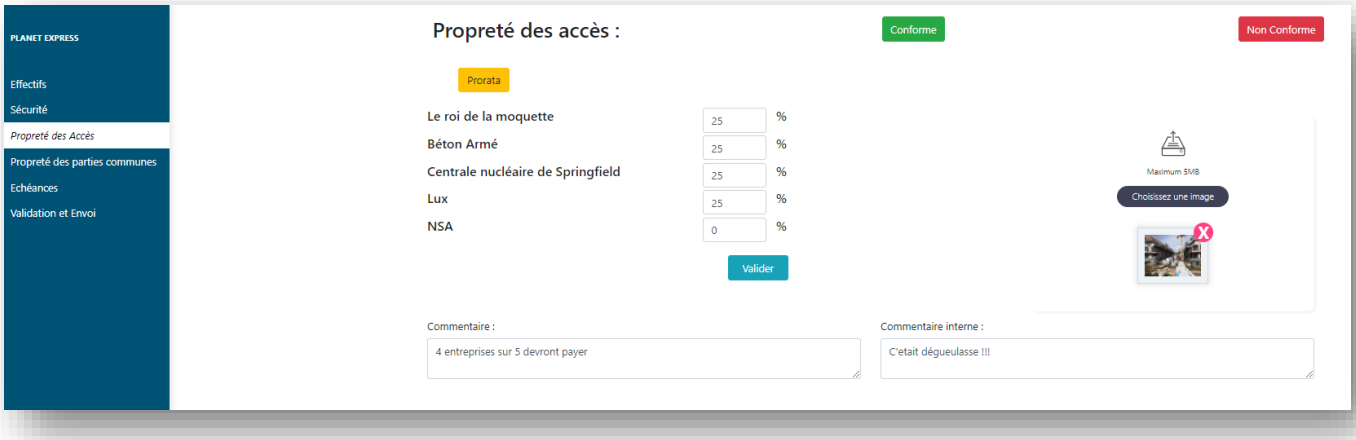
Cette page est une des moins avancée dans l'état actuel du développement. Car à terme elle devra comprendre une gestion poussée des dates semaine par semaine, ce que nous n'avons pas eu le temps d'implémenter.

6.6 : La page Sécurité



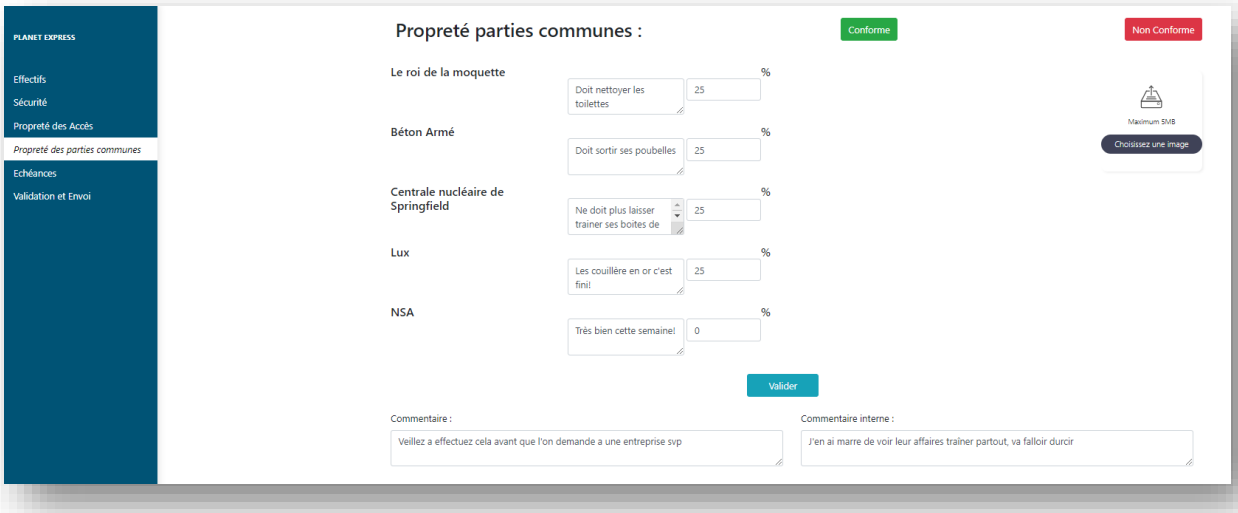
Encore beaucoup de travail sur cette page, notamment niveau design. De plus nous avons un gros bug concernant la mise à jour des textarea, la fonction handleChange ne fonctionne pas correctement… Ceci est dû au fait que le nombre de champ est généré automatiquement en fonction du nombre d'entreprise.

6.7 : La page Propreté des Accès



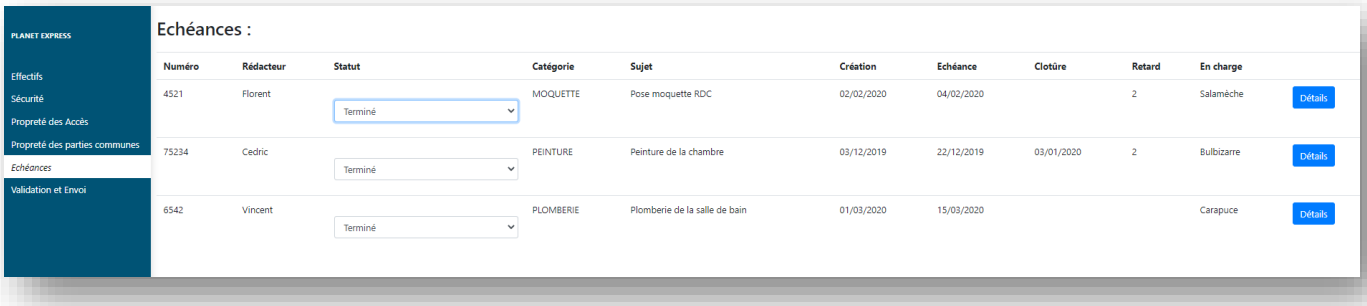
Même soucis sur cette page. Et notons qu'il y a le bouton pour définir l'imputation au prorata en plus, et que l'utilitaire d'upload d'image nous donne une miniature de l'image sélectionnée.

6.8 : La page Propreté des Parties Communes



Cette page n'existait pas dans le cahier des charges initial… Elle vient en lieu et place de la page Installations de chantier. Et son fonctionnement est similaire à la précédente, mais avec un champ commentaire supplémentaire pour chaque entreprise.

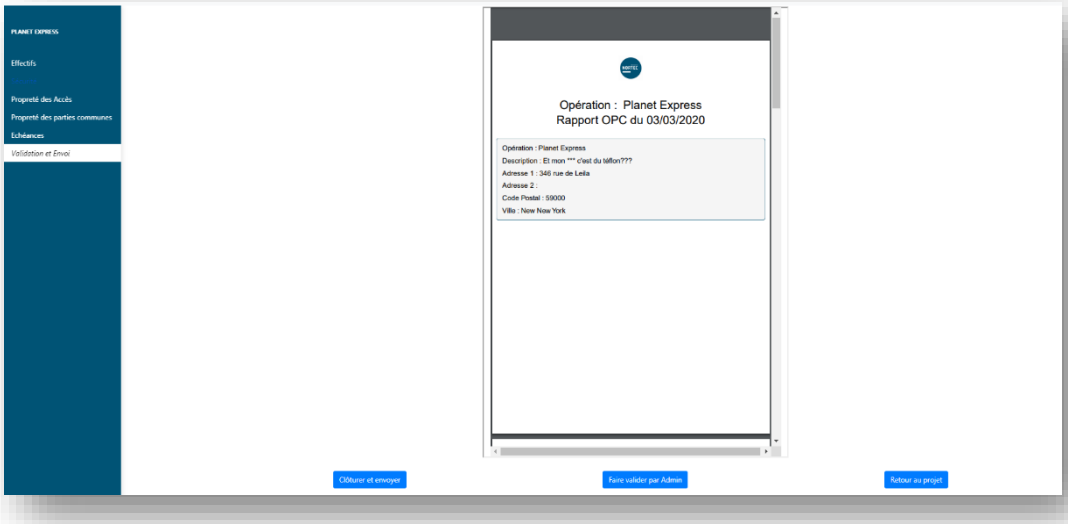
6.9 : La page des échéances



Cette page également en est au tout début de son développement. Dans l'attente d'autres informations que nous aurions dû y rajouter, nous l'avons mise de côté pour le moment.

6.10 : La page de validation, de visualisation et d'envoi du pdf

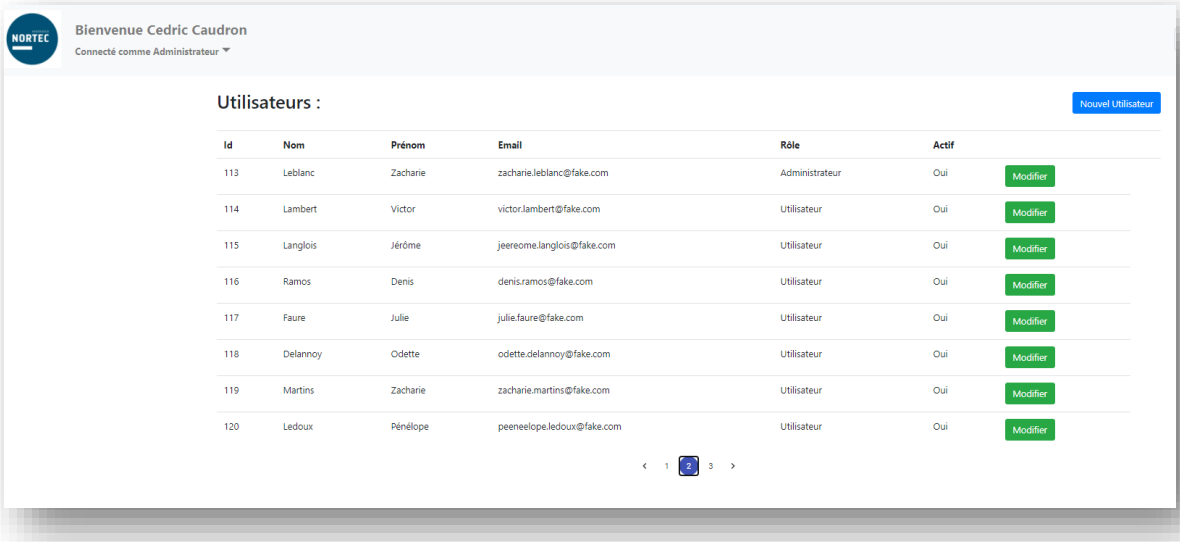
Cette page comprend en première partie un résumé exhaustif des informations du rapport.



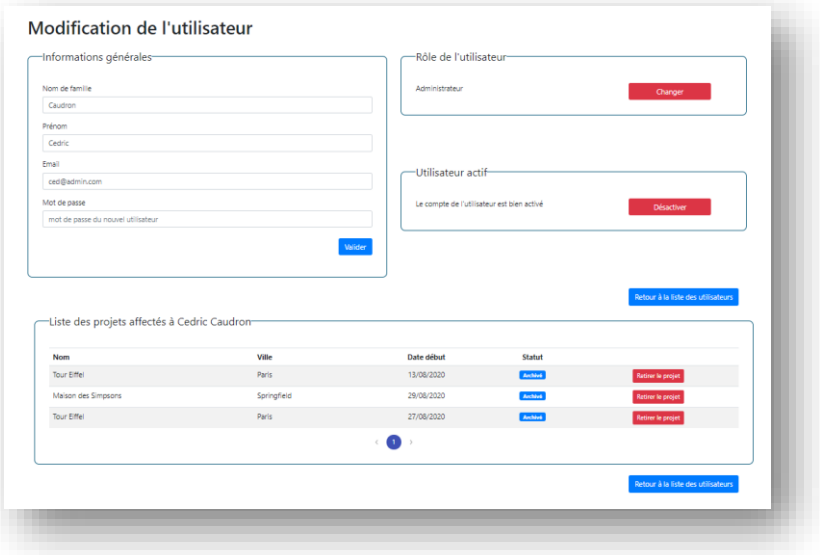
Et enfin une prévisualisation du fichier pdf, et des boutons pour les différentes options s'offrant à l'utilisateur.

Une amélioration de l'ergonomie de cette page serait nécessaire à mon avis. En externalisant par exemple la visualisation du pdf, pourquoi pas dans une fenêtre modale.

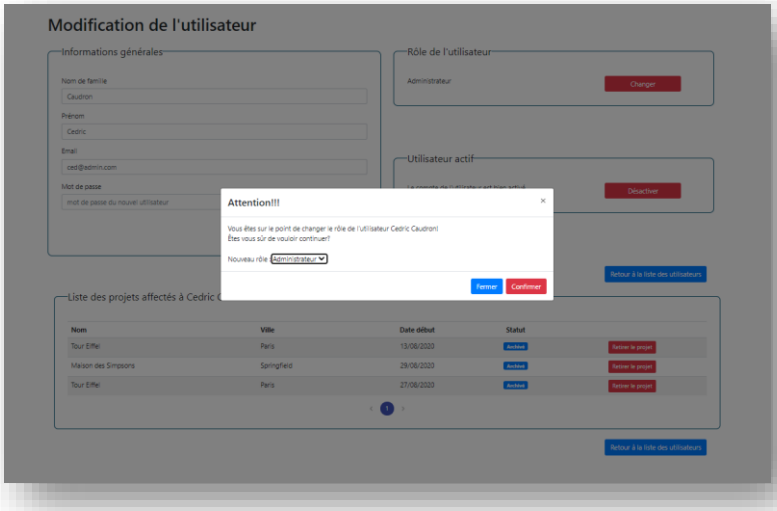
6.11 : Interface d'administration des utilisateurs



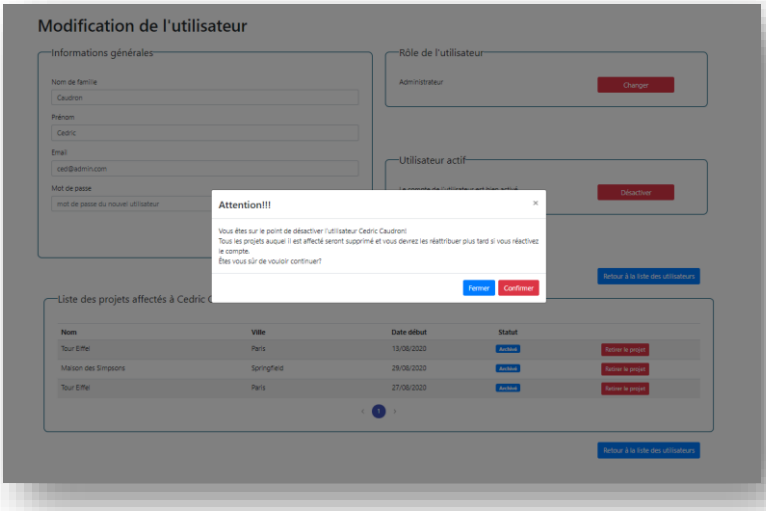
Dans l'interface Administrateur, qui est entièrement fonctionnel, nous avons en premier lieu une liste des utilisateurs de l'application.



Sur la page servant à les modifier, nous avons un récapitulatif de toutes ses informations, avec la possibilité de les modifier.

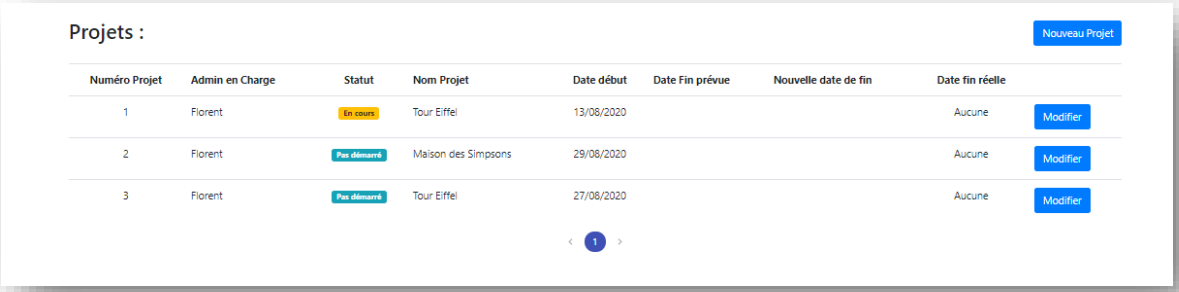


Pour la modification de leur rôle, il est géré dans une fenêtre modale avec un avertissement.



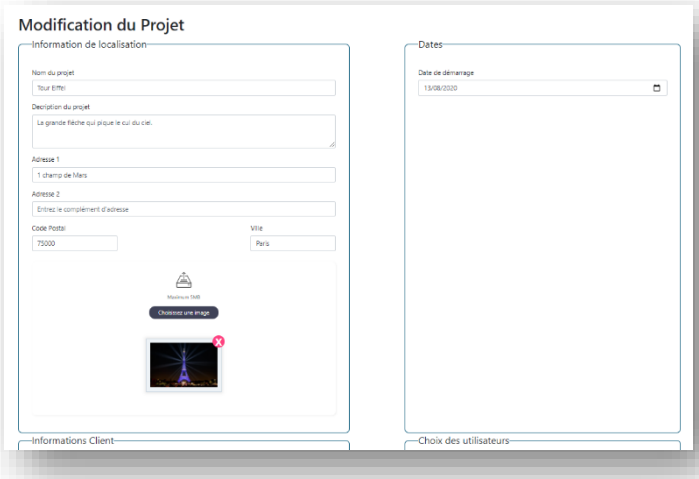
Et nous avons la même chose en ce qui concerne la modification de son statut.

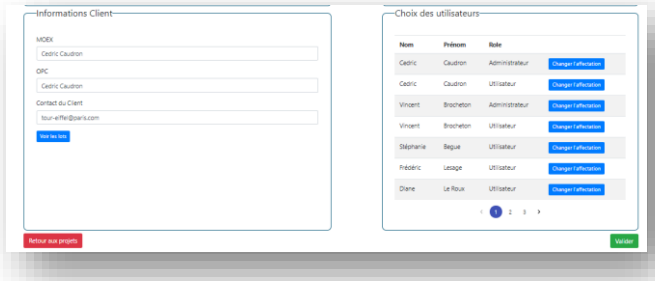
6.12 : Interface d'administration des projets

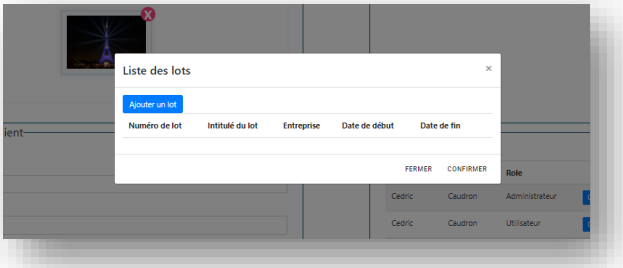


De la même façon que précédemment, nous avons une liste des projets dans leur administration.

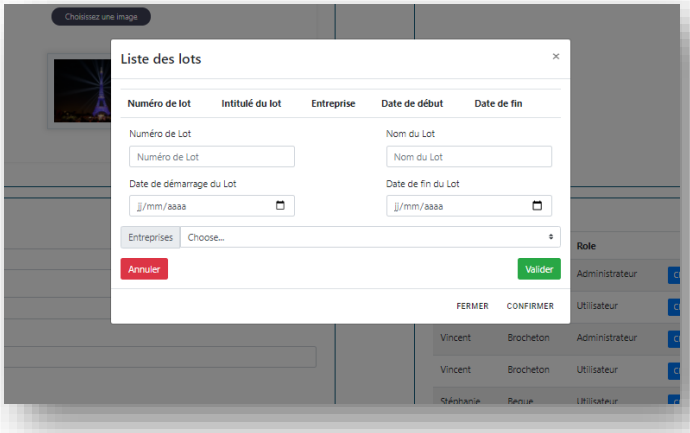
Une interface de création / modification du projet sélectionné.



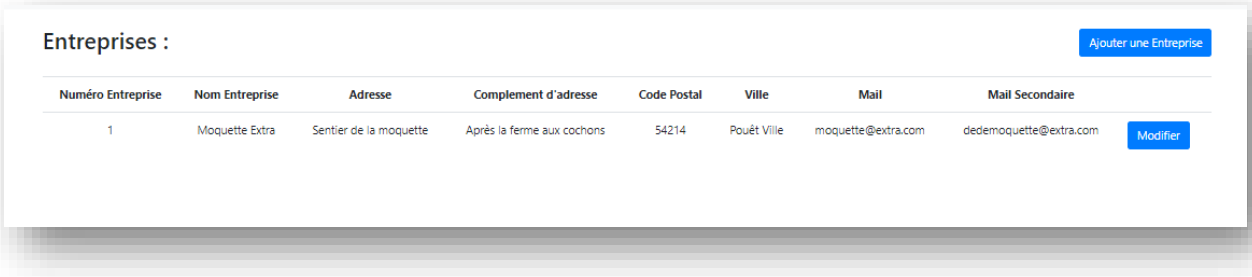




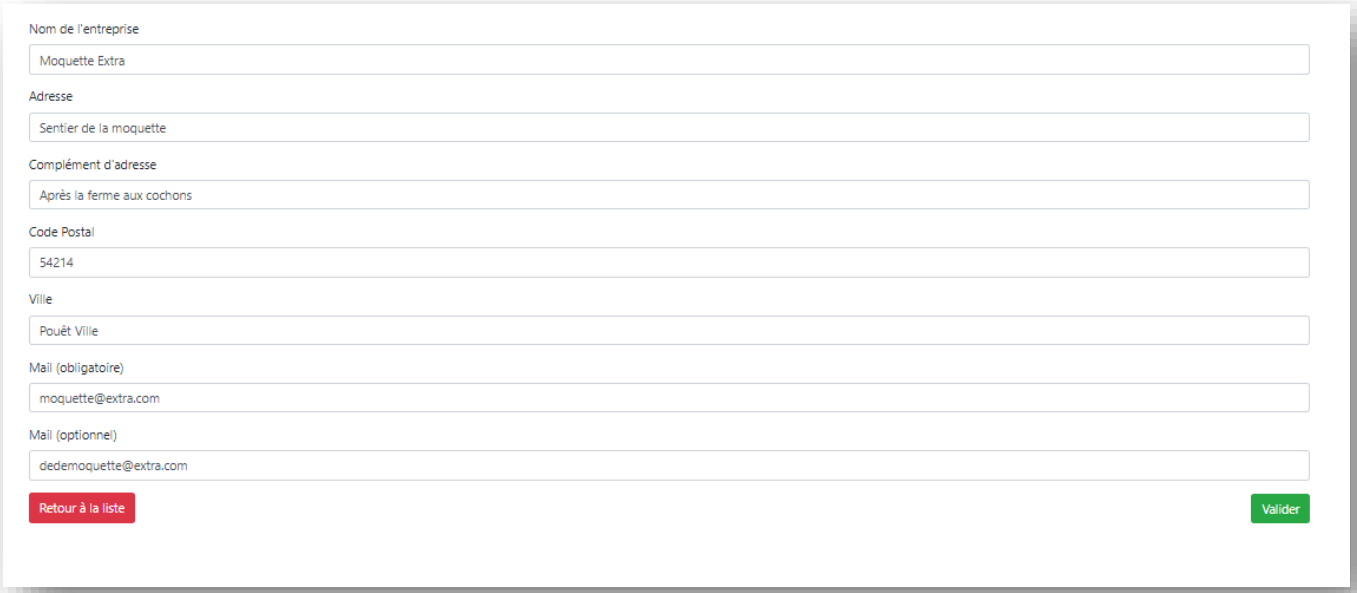
La possibilité de voir la liste des lots qui font partie du projet, ainsi que d'en rajouter.



6.13 : Interface d'administration des entreprises



Enfin, sur le même modèle, nous avons la gestion des entreprises. Avec la possibilité d'en ajouter et de les modifier.



6.14 : Conclusion

J'ai volontairement montré très peu de code dans cette partie. Car d'une part je n'en voyais pas la pertinence, il y aurait eu beaucoup de choses très similaire et la plupart du temps cela ne reste que du HTML.

L'intérêt ici se trouvait plutôt dans le fait de montrer l'évolution de l'ergonomie de l'application entre les maquettes et maintenant. Et aussi de montrer l'interface administrateur qui n'avait pas été évoqué jusque maintenant.

En ce qui concerne certaines mécaniques, je vais maintenant vous les expliquer, comme le système de pagination et d'autres.

7 : La gestion des dates

Notre application gère beaucoup de dates. Que ce soit la création des projets, ou la détermination des statuts de ceux-ci… Ainsi, par soucis de factorisation toujours, nous avons décidé d'externaliser toute cette gestion dans un fichier à part : dateAPI.js .

7.1 : La librairie moment.js

Moment.js est une librairie Javascript visant à simplifier la gestion du temps et des dates.

En effet, elle propose pléthore de fonctions qui automatisent les traitements, que ce soit pour les comparaisons, les formatages, ou autre.

7.2 : Les fonctions de notre gestion des dates

Notre fichier dateAPI.js est donc composé de plusieurs petites fonctions réutilisables selon nos besoin.

```
import moment from "moment";


function determineStatus(dateDebut, dateFinReelle) {
    let status = "";

    if (dateFinReelle === "")
    {
        if (moment().isBefore(dateDebut))
        {
            status = "no_start";
        }
        else
        {
            status = "in_progress";
        }
    }
    else
    {
        if (moment().diff(dateFinReelle, unitOfTime: 'days') > 7)
        {
            status = "archived";
        }
        else
        {
            status = "finished";
        }
    }
    return status;
}
```

Cette première fonction sert à déterminer le statut d'un projet. A savoir s'il est démarré ou pas, s'il est terminé ou archivé.

La fonction est très simple je ne vais pas la décrire outre mesure.

Il faut juste savoir que le statut d'un projet archivé doit intervenir 7 jours après sa date de fin réelle, ce qui était une demande du client.

```
function formatDate(date)
{
    return moment(date).format( format: 'DD/MM/YYYY');
}

function formatDateForm(date)
{
    return moment(date).format( format: 'YYYY-MM-DD');
}

function formatDateHours(date)
{
    return moment(date).format( format: 'DD/MM/YYYY à h:mm:ss' );
}

function verifyDateExist(date) {
    if (moment(date).isSame( inp: "1900-01-01T00:00:00+00:00"))
    {
        return "";
    }else{
        return date;
    }
}
```

Les autres fonctions du fichier sont juste à caractère utilitaire afin de nous simplifier la vie. Elles servent à formater les date sous différents format.

La dernière en revanche est intéressante.

En effet, nous avons été confrontés au problème qu'en base de donnée nous ne pouvions enregistrer de date nulle.

Ainsi, lorsqu'une date n'existe pas encore, nous avons eu l'idée d'en enregistrer une bien antérieure, qui est le 1er janvier 1900. Cette fonction nous sert donc à déterminer si la date que nous recherchons est réelle. Et si elle est égale à cette fausse date, elle nous renverra une string vide.

```
<div className='row ml-2 no-space'>
    <h6 className='offset-1 col-4'>Date de début :</h6>
    <p className='col-7'>{DateAPI.formatDate(project.dateDebut)}</p>
</div>
```

Et voici enfin un exemple d'utilisation d'une des fonction du fichier, comme ici dans la page du détail d'un projet.

8 : La gestion des statuts des projets

Nous allons voir maintenant un autre petit système que nous avons mis en place dans notre application, qui est très intéressant, et qui intervient dans la suite directe de la gestion des dates.

En effet, pour les statuts, nous avons créé deux petits objets en JSON, constants, avec la même structure et les même propriétés.

```
const STATUS_CLASSES = {
    no_start: "info",
    in_progress: "warning",
    finished: "success",
    archived: "primary"
};

const STATUS_LABEL = {
    no_start: "Pas démarré",
    in_progress: "En cours",
    finished: "Fini",
    archived: "Archivé"
};
```

L'intérêt de ceci, c'est que nous allons pouvoir affecter plusieurs valeurs différentes pour un même statut, et que nous pourrons utiliser dans des optiques différentes.

Le premier affectera donc des classes css à chaque statut, et le deuxième un texte .

```
<div className='row ml-2 mt-5'>
    <h6 className='offset-1 col-4'>Statut :</h6>
    <p className={"col-2 badge badge-" + STATUS_CLASSES[DateAPI.determineStatus(project.dateDebut, DateAPI.verifyDateExist(project.dateFinReelle))]}>
        {STATUS_LABEL[DateAPI.determineStatus(project.dateDebut, DateAPI.verifyDateExist(project.dateFinReelle))]}</p>
</div>
```

Ainsi, en associant ce système avec les fonctions de gestion de date, nous pouvons donc automatiser l'affichage des statuts des projets. Et en concaténant les classes bootstrap, pouvoir faire de jolis badges avec des couleurs différentes suivant les différents statuts.

Il faudra néanmoins à terme refactoriser ces objets dans un fichier à part, car pour le moment, il est nécessaire de les déclarer dans chaque Component où ils sont utilisés. L'intérêt en sera augmenté du fait que si nous avons besoin d'une modification de l'un des objet, nous ne devrons le modifier qu'une seule fois pour qu'il s'applique partout.

9 : La gestion des requêtes

9.1 : Axios

Axios est un client http basé sur les Promises (promesses) très populaire dans la communauté React de part sa simplicité d'utilisation.

Il permet d'effectuer des requêtes vers les API de toute sorte, et ce de façon très simple.

9.2 : Les fonctions asynchrones

Pour effectuer les requêtes, il est plus que recommandé d'utiliser des fonctions asynchrones.

Une fonction asynchrone, comme son nom l'indique, s'exécute de façon asynchrone… En pratique, elle va attendre la réponse du serveur avant d'exécuter le reste du code. Et elle aura une promesse (Promise) en valeur de retour.

```
33  //----------------------------------------Récupération d'un projet----
34      const fetchProject = async id => {
35          console.log(id)
36          try {
37              const data = await ProjectsAPI.find(id);
38              setProject(data);
39
40          } catch (error) {
41              console.log(error.response);
42          }
43      }
```

Nous pouvons les reconnaitre grâce aux mots clé async, qui déclare la fonction comme asynchrone, et await qui est la ligne de commande dont nous devons attendre la fin avant de continuer.

9.3 : Les requêtes vers notre API

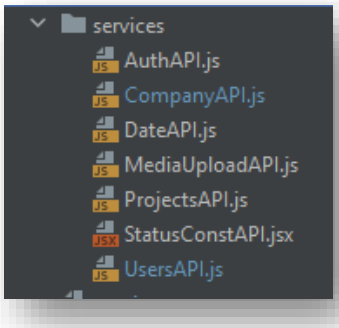
Encore et toujours, par soucis de factorisation et de réutilisation du code, nous avons dispatché les liens d'une part, et les requêtes d'autre part dans différents fichiers externes.

api_links_config.js

```
export const API_URL = "https://localhost:8000/api/";

export const LOGIN_API = API_URL + "login_check";
export const USERS_API = API_URL + "users";
export const PROJECTS_API = API_URL + "projects";
export const UPLOAD_API = API_URL + "media_objects";
export const COMPANY_API = API_URL + "companies";
```

Dans ce fichier, nous avons donc configuré tous les liens nécessaires au bon fonctionnement des requêtes de notre application.

Et pour les requêtes elles-mêmes, nous les avons dispatché dans différents fichiers suivant les objets qu'elle va atteindre.



Ainsi, celles pour l'authentification seront par exemple dans AuthAPI.js, alors que toutes celles concernant les utilisateurs dans UsersAPI.js .

UsersAPI.js

```
import axios from "axios";
import {USERS_API} from "../components/configs/api_links_config";

function findAll(){
    return axios.get(USERS_API).then(response => response.data['hydra:member']);
}

function find(id) {

    return axios.get( url: USERS_API + "/" + id).then(
        response => response.data
    );
}

function update(id, user){
    return axios.put( url: USERS_API + "/" + id, user);
}

function create(user){
    return axios.post(USERS_API, user);
}
```

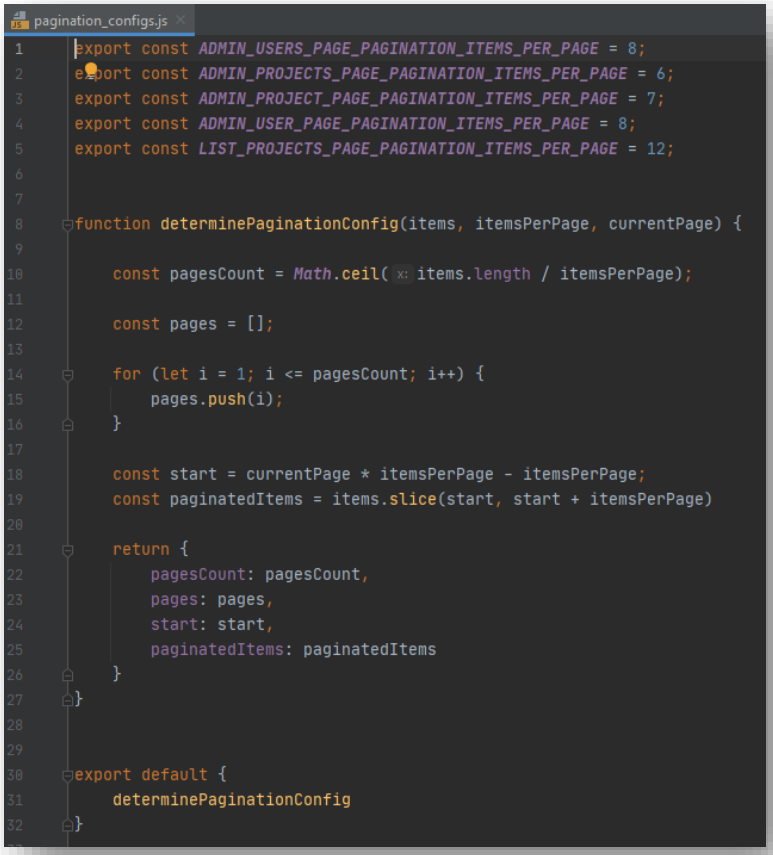
C'est justement ce dernier que j'ai choisi comme exemple.

Ici nous avons donc un exemple de requête GET pour récupérer tous les utilisateurs, une paramétrée afin en GET toujours pour en récupérer un seul, une en POST pour la création d'un utilisateur, et enfin une en PUT pour remplacer un utilisateur.

Il restera à aborder le cas particulier des requêtes pour l'upload des images, mais que j'aborderai dans la partie Back End…

10 : La pagination

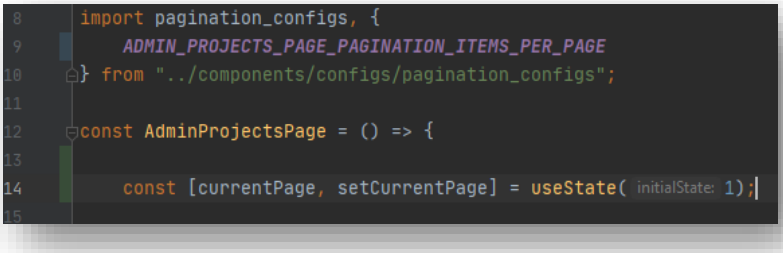
Nous avons également mis en place un système factorisé de pagination pour éviter d'avoir trop d'éléments à la fois affiché sur une seule page.



Le plus gros de la logique métier de la pagination a donc été externalisé dans un fichier à part.

On y trouve des constantes qui déterminent combien d'éléments doivent être affichés sur les différentes pages. Ce système centralisé permet de ne pas avoir à rechercher partout en cas de modification.

Et ensuite il y a une fonction qui va prendre les paramètres nécessaires, les traiter, et enfin renvoyer un objet avec toutes les informations pour le bon affichage de la pagination.



Ainsi, dans les Components où l'on veut l'implémenter, il ne reste pas grand-chose à faire. Nous avons besoin d'importer la constante bien sûr, et de créer un state CurrentPage qui représentera le numéro de la page courante.

```
// ---------------------------- Mise en place de la pagination ----------------------------------------

    const handleChangePage = (event, page) => {
        setCurrentPage(page);
    }

    const paginationConfig = pagination_configs.determinePaginationConfig(projects, ADMIN_PROJECTS_PAGE_PAGINATION_ITEMS_PER_PAGE, currentPage);
```

Nous avons également besoin d'une fonction qui gérera le changement de page. Et de récupérer les informations de configuration de la pagination avec la fonction que nous avons vu précédemment.

```
{paginationConfig.paginatedItems.map(project =>
    <tr key={project.id}>
        <td className="text-center">{project.id}</td>
        <td>Florent</td>
        <td className="text-center"><span className={"pl-2 pr-2 pt-1 pb-1 badge badge-" +
        STATUS_CLASSES[DateAPI.determineStatus(project.dateDebut, DateAPI.verifyDateExist(project.dateFinReelle))]}>
            {STATUS_LABEL[DateAPI.determineStatus(project.dateDebut, DateAPI.verifyDateExist(project.dateFinReelle))]}
        </span>
```

Dans le template, au lieu de faire un map sur l'ensemble des projets dans cet exemple, nous le faisons sur les projets paginés, qui font partie de l'objet que la fonction précédente nous a renvoyé.

```
        <div className="mt-2 d-flex justify-content-center">
            <Pagination count={paginationConfig.pagesCount}
                        color="primary"
                        page={currentPage}
                        onChange={handleChangePage}
            />
        </div>
```

Et enfin, nous affichons les puces à la fin du template.

Pour ceci nous avons utilisé une bibliothèque de pagination React Paginate. Ce qui nous donne un aspect moderne et adaptable à l'affichage des liens de pagination.

11 : Les notifications

Dans le but d'améliorer l'expérience utilisateur, nous avons mis en place un système de notification avec la librairie React Toastify.

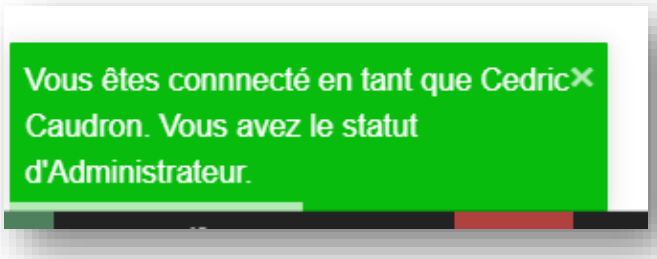
Ceci nous permet d'informer l'utilisateur quand une action à fonctionné ou non. Et pour pas qu'il reste dans le doute.

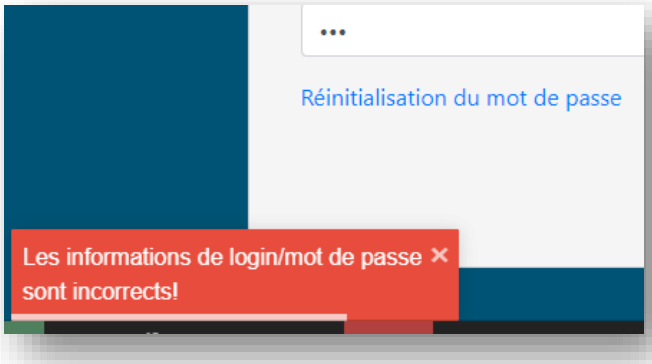
```
                </Switch>

            </HashRouter>

            < ToastContainer
                position={toast.POSITION.BOTTOM_LEFT}
            />

        </AuthContext.Provider>

    );
```

Après installation, son implémentation est vraiment des plus simple. Il suffit de le configurer dans le fichier app.js, pour que les notifications soient accessibles de partout, avec la position dans laquelle nous voulons les voir apparaitre, ici en bas à gauche.

```
if (AuthAPI.getUserActiveStatus())
{
    setIsAuthenticated(true);
    toast.success( content: "Vous êtes connnecté en tant que " + AuthAPI.getUserFirstNameLastName() + ". Vous avez le statut d'" + AuthAPI.isRole() + ".");
    history.replace("/projects");
}
else
{
    setIsAuthenticated(false);
    toast.error("Votre compte a été désactivé. Veuillez contacter un administrateur!");
    AuthAPI.logout();
}
```

Pour faire apparaitre les notifications, rien de plus simple. On importe toast, et on fait appel à ses différentes fonctions. Par exemple ici, pour l'authentification, success et error, qui donneront une couleur différente suivant le cas.





12 : L'icône de chargement

Toujours pour améliorer l'expérience utilisateur, nous avons mis en place une icone animée en CSS pour notifier le chargement des informations.

loading-icon.css

```css
#loading-icon
{
    width: 30px;
    height: 30px;
    border: 8px solid #005375;
    border-right-color: transparent;
    border-radius: 50%;
    box-shadow: 0 0 25px 2px #005375;
    animation: rotate 1s linear infinite;
    margin-right: auto;
    margin-left: auto;
    vertical-align: 50%;
}


@keyframes rotate
{
    from { transform: rotate(0deg);   opacity: 0.2; }
    50%  { transform: rotate(180deg); opacity: 1.0; }
    to   { transform: rotate(360deg); opacity: 0.2; }
}
```

Nous utilisons donc les fonctions CSS transform et rotate afin d'animer et de changer l'opacité de l'icône.

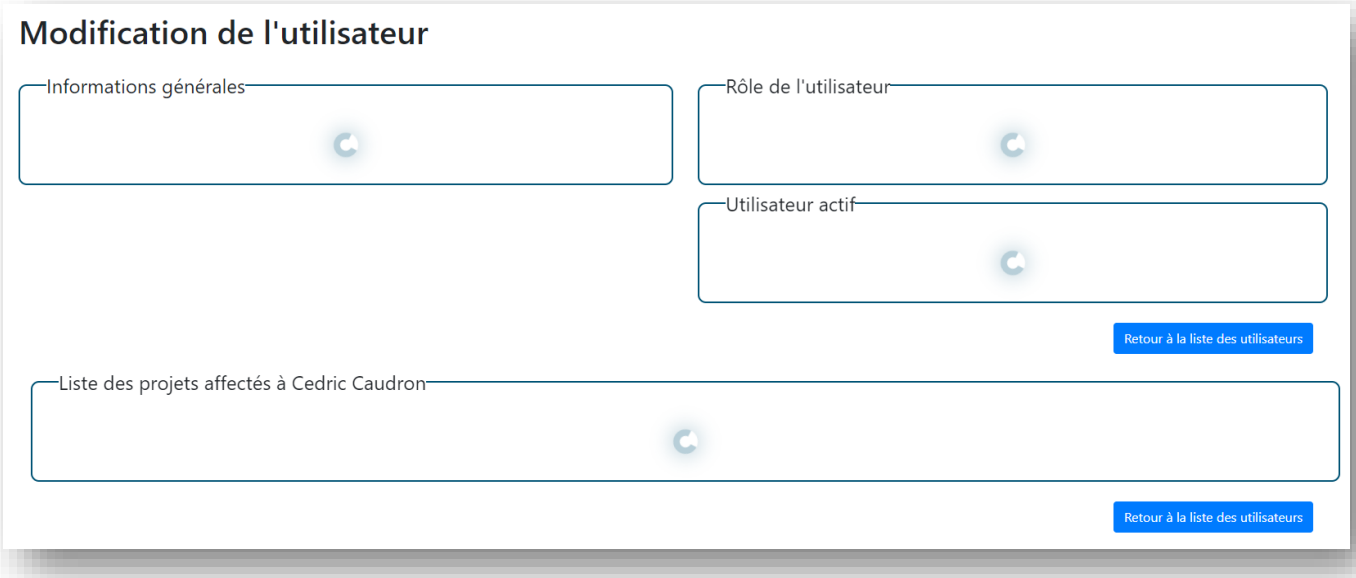
```
const [loading, setLoading] = useState( initialState: true);
```

Pour l'utiliser, il faut définir un nouveau State sur les Components concernés, que nous avons appelé loading, et que nous mettons à jour lorsque le chargement se termine.

```
const fetchUsers = async () => {
    try {
        const data = await UserApi.findAll();
        setUsers(data);
        setLoading( value: false);
```

```
{loading &&
<div id="loading-icon" className="mt-5 mb-5"/>
}
```

Enfin, dans le template, une simple condition suivant l'état du state permet d'appeler une div avec l'id loading-icon permet de l'afficher.



Et voici donc le résultat en image, par exemple ici sur la page de modification d'un utilisateur.

13 : Les fenêtres modales

Nous l'avons succinctement abordé dans une partie précédente, mais nous avons également mis en place des fenêtres modales pour certaines actions importantes et pour éviter que l'utilisateur ne clique par inadvertance sans avoir de confirmation de son action.

```
<Modal {...props}
       size="lg"
       aria-labelledby="contained-modal-title-vcenter"
       centered
       show={showModal} onHide={handleCloseModal}>
    <Modal.Header closeButton>
        <Modal.Title>Attention!!!</Modal.Title>
    </Modal.Header>
    <Modal.Body>
        Vous êtes sur le point {userToModifyActive.active ? <>de désactiver </> : <>d'activer </>}
        l'utilisateur {userToModifyActive.firstName} {userToModifyActive.lastName}! <br/>
        {userToModifyActive.active ? <>Tous les projets auquel il est affecté seront supprimé et vous devrez
            les réattribuer plus tard si
            vous réactivez le compte. <br/> </> : <>Il vous faudra réassigner manuellement les projets
            auxquels l'utilisateur pourra avoir accès. <br/></>}

        Êtes vous sûr de vouloir continuer?
    </Modal.Body>
    <Modal.Footer>
        <Button variant="primary" onClick={handleCloseModal}>
            Fermer
        </Button>
        <Button variant="danger" onClick={() => handleActiveUser(userToModifyActive)}>
            Confirmer
        </Button>
    </Modal.Footer>
</Modal>
```

Pour cela, nous avons utilisé la bibliothèque React Modal.

Il suffit donc de rajouter dans le template les Components nécéssaires à la structuration de la fenêtre et de faire son contenu en html.

Il est également nécessaire de créer les fonctions qui permettent de déclencher l'ouverture et la fermeture de la fenêtre, ainsi qu'un State pour déterminer si elle est ouverte ou non.

```
const handleShowModal = userToModifyActive => {
    setShowModal( value: true);
    setUserToModifyActive(userToModifyActive);
}

const handleCloseModal = () => setShowModal( value: false);
```

Nous pouvons donc ajouter autant de fenêtre modale que nous le voulons par ce biais et de façon très simple.

14 : La génération des rapports pdf

Pour terminer avec la partie Front de l'application, il nous reste à parler de la génération automatique du pdf. Comme d'habitude, nous avons utilisée une librairie spécialisée dans cette tâche : React Pdf.

Nous avons donc créé un nouveau Component qui ne s'occupera que de créer le rapport en pdf.

```
const styles = StyleSheet.create({
    page: {
        backgroundColor: 'white'
    },
    section: {
        margin: 10,
        padding: 10,

    },
    image: {
        width: 50,
        marginLeft: 'auto',
        marginRight: 'auto',
        marginTop: '15',
    },
    title: {
        textAlign: 'center',
        fontWeight: 'bold',
        marginBottom: 5,
        fontSize: '25pt'
    },
    sectionBorder: {
        border: '1pt solid #005274',
        marginLeft: 10,
        marginRight: 10,
        padding: 7,
        backgroundColor: '#F5F5F5',
        borderTopLeftRadius: '5',
        borderTopRightRadius: '5',
        borderBottomRightRadius: '5',
        borderBottomLeftRadius: '5',
        fontSize: '15pt'
    },
    text: {
        margin: '5pt',
    },
    textEffectifsNomEntreprise: {
        margin: '2pt',
        fontWeight: 'bold'
    },
    textEffectifsInfos: {
        marginLeft: '40pt',
    },
    listEffectifs: {
        marginTop: '15pt'
    }

});
```

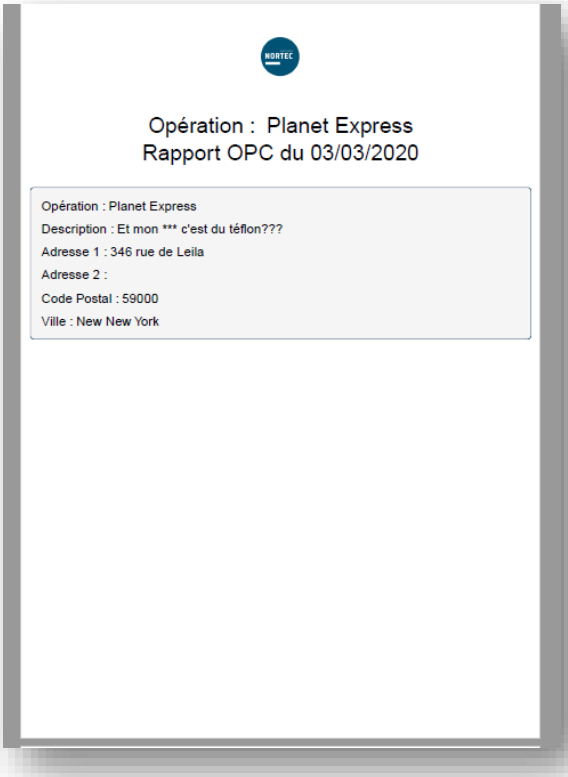
Pour le bon fonctionnement de la librairie, il est tout d'abord nécessaire de créer une liste de style qui permettra la mise en page du document pdf final.

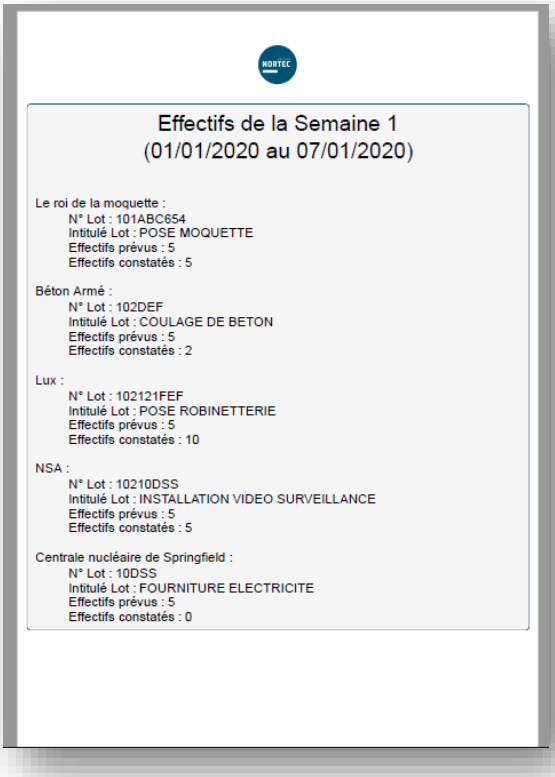
Pour cela nous appelons la fonction create contenue dans Stylesheet, et nous lui passons en paramètre un objet JSON contenant tous les styles dont nous aurons besoin.

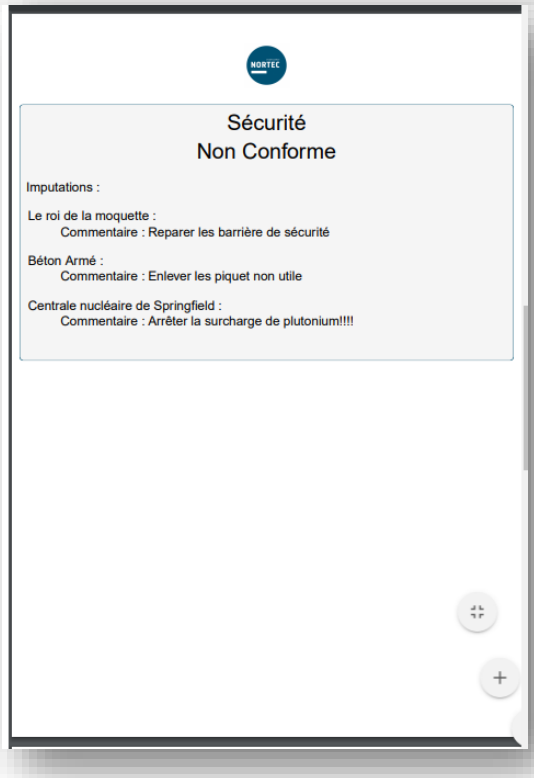
```
const ReportPdfComponent = ({report}) => {
    return (
        <Document title={report.project.name} subject={'Rapport du ' + DateAPI.formatDate(report.dateRedaction)}>
            <Page size="A4" style={styles.page}>
                <View style={styles.section}>
                    <Image source='../img/logo-company/logo-nortec.png' style={styles.image}/>
                </View>
                <View style={styles.section}>
                    <Text style={styles.title}>{'Opération :  ' + report.project.name}</Text>
                    <Text style={styles.title}>{'Rapport OPC du ' + DateAPI.formatDate(report.dateRedaction)}</Text>
                </View>
                <View style={styles.sectionBorder}>
                    <Text style={styles.text}>{'Opération : ' + report.project.name}</Text>
                    <Text style={styles.text}>{'Description : ' + report.project.description}</Text>
                    <Text style={styles.text}>{'Adresse 1 : ' + report.project.adresse1}</Text>
                    <Text style={styles.text}>{'Adresse 2 : ' + report.project.adresse2}</Text>
                    <Text style={styles.text}>{'Code Postal : ' + report.project.codePostal}</Text>
                    <Text style={styles.text}>{'Ville : ' + report.project.ville}</Text>
                </View>

            </Page>

            <Page size="A4" style={styles.page}>
                <View style={styles.section}>
                    <Image source='../img/logo-company/logo-nortec.png' style={styles.image}/>
                </View>

                <View style={styles.sectionBorder}>
                    <Text style={styles.title}>Effectifs de la Semaine 1</Text>
                    <Text style={styles.title}>(01/01/2020 au 07/01/2020)</Text>

                    <View style={styles.listEffectifs}>
                        {report.lots.map(lot =>
                            <View style={styles.listEffectifs}>
                                <Text style={styles.textEffectifsNomEntreprise}>{lot.entreprise.nom + ' :'}</Text>
                                <Text style={styles.textEffectifsInfos}>{'N° Lot : ' + lot.numeroLot}</Text>
                                <Text style={styles.textEffectifsInfos}>{'Intitulé Lot : ' + lot.libelleLot}</Text>
                                <Text
                                    style={styles.textEffectifsInfos}>{'Effectifs prévus : ' + lot.effectifPrevu}</Text>
                                <Text
                                    style={styles.textEffectifsInfos}>{'Effectifs constatés : ' + lot.effectifConstate}</Text>
                            </View>
                        )}
                    </View>
                </View>

            </Page>
```

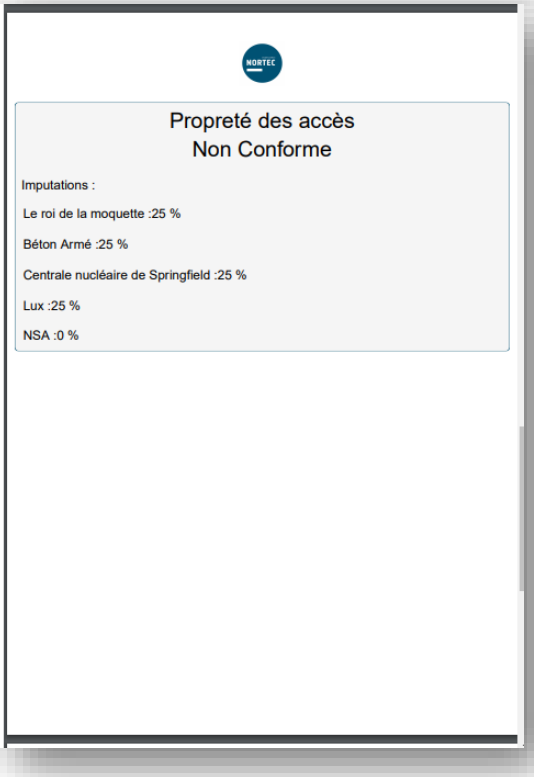
S'ensuit donc la création du document en tant que tel. Ci-dessus, le code pour la génération des deux premières pages du pdf.

La documentation de la bibliothèque nous donne les informations nécessaires quant aux différents Components à utiliser.









Le résultat est très largement perfectible, il manque en outre des informations, et de plus, il n'y a pas encore de système pour l'enregistrer sur le serveur à ce stade du développement.

16 : Conclusion

Pour conclure cette très longue partie sur le développement React, je dirais qu'il a été vraiment très formateur sur cette nouvelle technologie pour nous qu'est ce Framework.

La quasi-totalité du Front a ainsi été réalisé. Tout n'est pas parfait, loin de là ! Il reste beaucoup de travail à effectuer avant d'avoir une application totalement fonctionnelle et sans bug, mais la base est là.

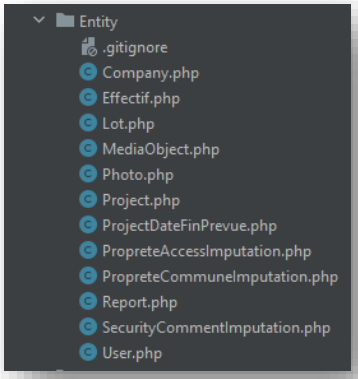
Il restera beaucoup de travail de débogage donc, d'amélioration de l'interface utilisateur, de la rendre plus homogène également, de refactorisation surtout… Bref, rien n'est encore terminé.

Nous allons maintenant passer au développement du Back end, qui devrait être moins indigeste que le Front. Vu qu'il y a eu finalement beaucoup moins de choses à faire niveau back que front, et que surtout, une bonne partie a déjà été expliquée en amont.

VIII : DEVELOPPEMENT BACK END

1 : La création des entités

La première chose à faire dans le développement du Back de l'application est la création des différentes entités dans Symfony, avec les différentes relations entre elles, et ce qui pour finir mettra à jour la base de données.



Doctrine nous permet la génération simple des entités via la ligne de commande dans Symfony.

Ayant détaillé la génération de l'entité User dans la partie Authentification, je ne rentrerai pas trop dans les détails dans cette partie.

Nous nous sommes basés sur la structure du Mock que nous avons établi, avec les modifications qu'il a subit au fil du temps suivant les différentes demandes du client, pour nous établir les relations entre les différentes entités.

```
    /**
     * @ORM\Column(type="string", length=255)
     * @Groups({"project"})
     * @Assert\NotBlank(message="Le nom de la ville est obligatoire !")
     */
    private $ville;

    /**
     * @ORM\OneToMany(targetEntity="App\Entity\ProjectDateFinPrevue", mappedBy="Project", orphanRemoval=true)
     */
    private $dateFinPrevues;

    /**
     * @ORM\OneToMany(targetEntity="App\Entity\Report", mappedBy="Project", orphanRemoval=true)
     */
    private $reports;

    /**
     * @ORM\ManyToMany(targetEntity="App\Entity\User", mappedBy="project")
     * @Groups({"project"})
     */
    private $users;

    /**
     * @ORM\OneToMany(targetEntity="App\Entity\Lot", mappedBy="project")
     * @Groups({"project"})
     */
    private $lots;

    /**
     * @ORM\ManyToMany(targetEntity="App\Entity\Company", inversedBy="projects")
     * @Groups({"project"})
     */
    private $companies;
```

Ci-contre un exemple d'entité (Project) ainsi générée avec des exemples de relations OneToMany et ManyToMany.

2 : Implémentation dans ApiPlatform et groupes de normalisation

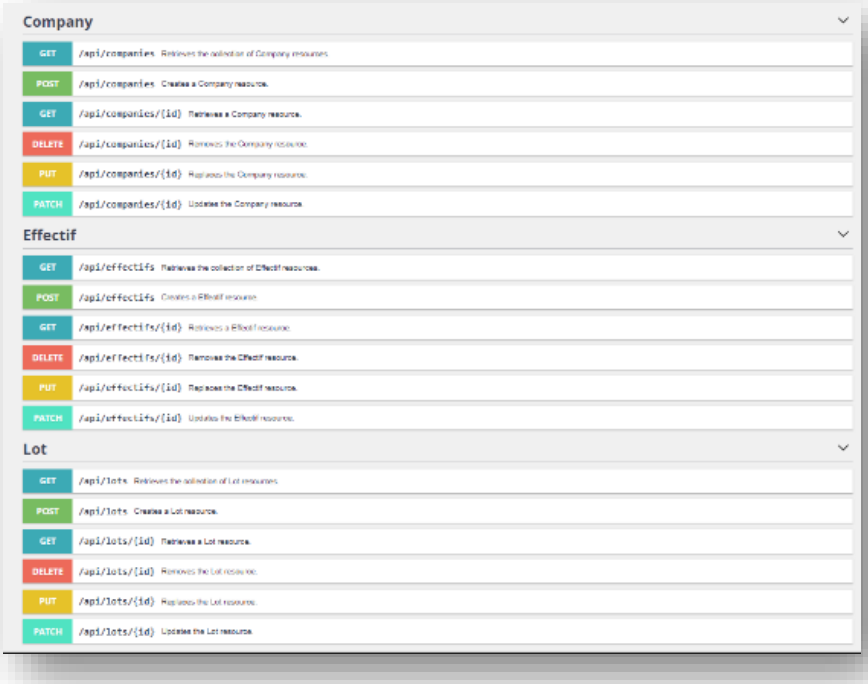
```php
/**
 * @ORM\Entity(repositoryClass="App\Repository\UserRepository")
 * @ApiResource(
 *     normalizationContext={"groups"={"users_read"}}
 * )
 */
class User implements UserInterface
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     * @Groups({"users_read", "project"})

     */
    private $id;
```

Comme nous l'avons également abordé avec l'entité User, il faut maintenant faire reconnaitre les différentes entités par ApiPlatform.

Ceci se fait via l'annotation ApiResource(), qui rend disponible l'entité dans l'API.

De plus, cette annotation, qui représente en arrière-plan une méthode de ApiPlatform, prend des paramètres.

La liste des entités et des actions est désormais disponible dans l'interface de ApiPlatform.



Une chose importante maintenant est donc de définir des groupes grâce à l'annotation normalizationContext, qui est un paramètre de ApiResource.

Les groupes permettent de définir la profondeur qu'aura le JSON retourné, donc avoir les informations de l'entité datesFinPrevues dans l'entité Project par exemple.

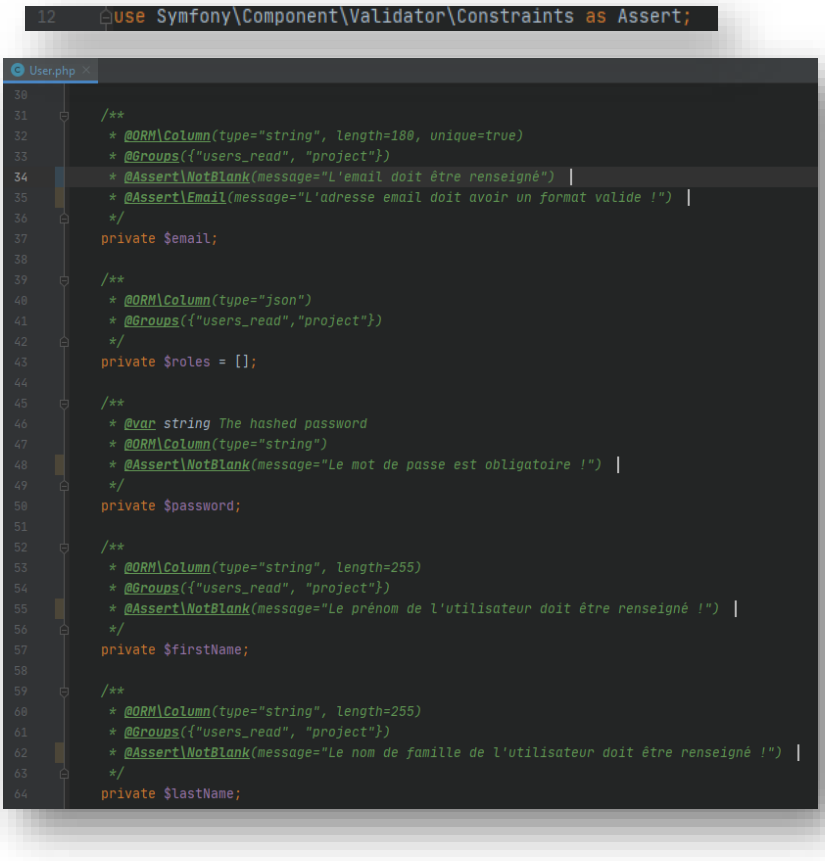
La mise en place de ces groupes doit être fait avec grande précaution. En effet, nous pouvons facilement nous retrouver avec une boucle infinie si nous n'y faisons pas attention.

Par exemple, nous définissons dans l'entité User un groupe « user_read », qui permettra de lire les Users à partir d'une autre entité. Dans cette même entité User, nous appliquons le groupe sur la propriété projects qui a une relation ManyToMany avec User. De là, nous aurons accès à la liste des Projects dans le JSON des User. Maintenant, si nous utilisons le même groupe pour avoir la liste des Users à partir du Project, nous aurons droit à une boucle infinie. En effet, lorsqu'il générera le JSON des Users, il ira chercher la liste des Projects, qui contiendra la liste des Users, qui contiendra la liste des Projects, qui contiendra la liste des Users etc…

Et voici le résultat obtenu lors d'une requête GET sur les projets. L'API nous renvoie une réponse en JSON avec la liste des projets, contenant toutes les informations dont nous avons besoin pour le traitement Front. Avec la structure se rapprochant de celle établie dans le Mock, nous permettant une adaptation facile sans modification profonde du traitement Front.

3 : La validation des données

Nous pouvons maintenant mettre en place un système de validation des données, afin de vérifier que l'utilisateur rentre bien dans les futurs champs du formulaire de login des informations qui correspondent à ce qui est attendu.

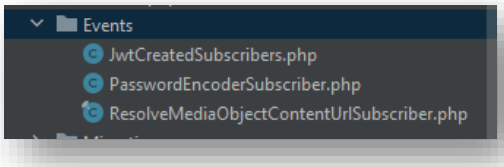


En effet, nous allons vérifier que lors de la validation, les champs obligatoires ne sont pas vides, que les adresse email correspondent au bon format etc…. Nous aurions pu utiliser des REGEX pour faire cette vérification, mais Symfony dispose d'un système automatisé et très simple afin de faire la même chose grâce au Validator.

Dans cet exemple, nous vérifions donc que les champs ne soient pas vides, et également que l'email est au bon format.

Dans le cas où l'utilisateur essaierait de valider un formulaire non valide, la réponse du serveur contiendra un objet Violations avec les messages définis dans les annotations.

4 : Intervenir au niveau des évènements du Kernel



Il nous reste maintenant à gérer des évènements lors de l'enregistrement des données.

En effet, en l'état actuel des choses, lors de la création d'un utilisateur, le mot de passe s'enregistre en clair dans la base de données. Ce qui évidemment ne peut nous convenir.

Nous pouvons gérer ce problème en créant un évènement au niveau du Kernel de Symfony. Nous avons déjà vu un système similaire lorsque nous avons voulu enrichir le token avec différentes données. Mais il utilisait les évènements du bundle Lexik JWT. Ici, ce sera géré par ceux de Symfony.

```php
class PasswordEncoderSubscriber implements EventSubscriberInterface {

    /** @var UserPasswordEncoderInterface*/
    private $encoder;

    public function __construct(UserPasswordEncoderInterface $encoder)
    {
        $this->encoder = $encoder;
    }

    public static function getSubscribedEvents()
    {
        return [
            KernelEvents::VIEW => ['encodePassword', EventPriorities::PRE_WRITE]
        ];
    }

    public function encodePassword(ViewEvent $event){
        $result = $event->getControllerResult();
        $method = $event->getRequest()->getMethod();

        if ($result instanceof User && $method === "POST") {

            $hash = $this->encoder->encodePassword($result, $result->getPassword());
            $result->setPassword($hash);
            $result->setRoles(["ROLE_USER"]);

        }

        if ($result instanceof User && $method === "PUT") {

            $hash = $this->encoder->encodePassword($result, $result->getPassword());
            $result->setPassword($hash);

        }

    }
}
```

La particularité ici, est qu'il faut que faire écouter par Symfony les événements. Pour cela nous passons dans la méthode l'événement de type ViewEvent par injection de dépendance.

Dans le constructeur, nous importons également l'interface UserPasswordEncoderInterface afin d'accéder à l'encoder qui permettra de hasher le mot de passe.

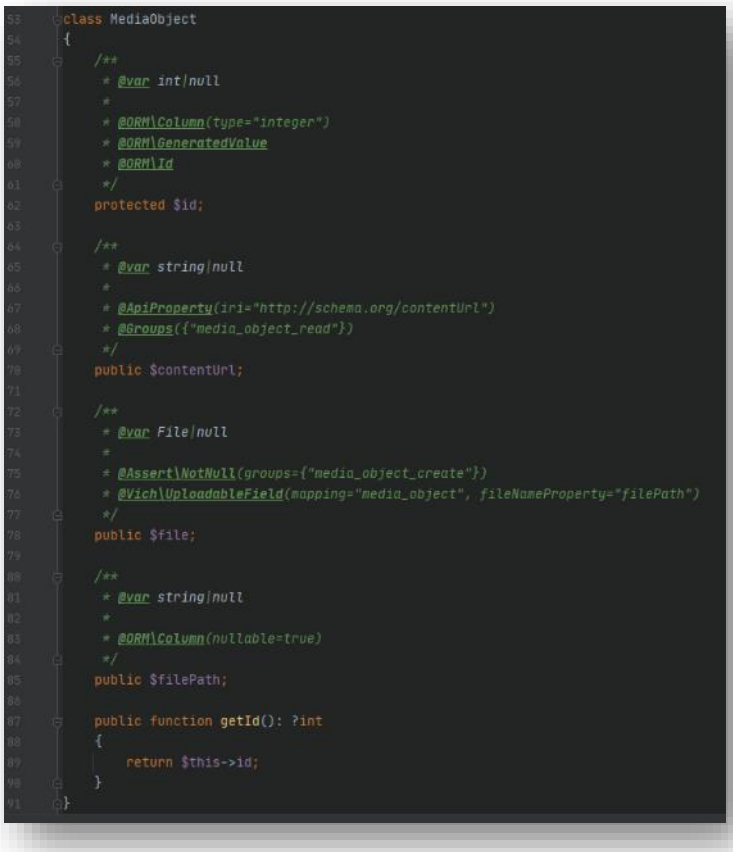
La dernière chose est de définir à quel moment de l'événement devra s'exécuter la méthode. Ici, avant l'écriture en base de données.

Enfin, nous hashons le mot de passe, et dans le cas de la création d'un utilisateur, nous lui définissons un rôle User par défaut.

5 : L'upload d'images

Pour terminer la partie Back End, il nous reste à aborder la gestion de l'upload de fichier, ici appliqué aux images.

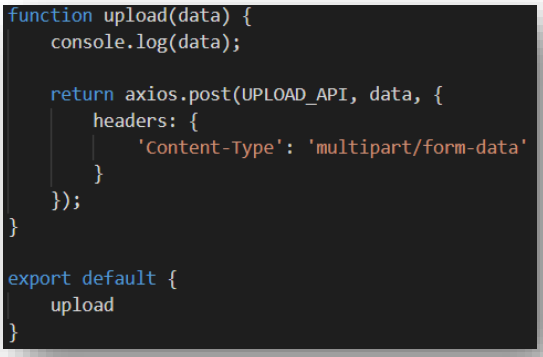
Pour ceci, j'ai suivi exactement la documentation de ApiPlatform car il fait intervenir beaucoup de notions.



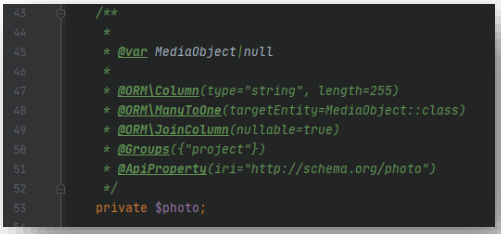
```php
class MediaObject
{
    /**
     * @var int|null
     *
     * @ORM\Column(type="integer")
     * @ORM\GeneratedValue
     * @ORM\Id
     */
    protected $id;

    /**
     * @var string|null
     *
     * @ApiProperty(iri="http://schema.org/contentUrl")
     * @Groups({"media_object_read"})
     */
    public $contentUrl;

    /**
     * @var File|null
     *
     * @Assert\NotNull(groups={"media_object_create"})
     * @Vich\UploadableField(mapping="media_object", fileNameProperty="filePath")
     */
    public $file;

    /**
     * @var string|null
     *
     * @ORM\Column(nullable=true)
     */
    public $filePath;

    public function getId(): ?int
    {
        return $this->id;
    }
}
```

La première chose à faire est de créer une nouvelle entité MediaObject qui gérera les fichiers.

Du coté API, il faudra envoyer la requête vers le lien de cette entité. Il faudra en outre modifier le header du JSON envoyé pour lui faire prendre en compte le multipart/form-data.

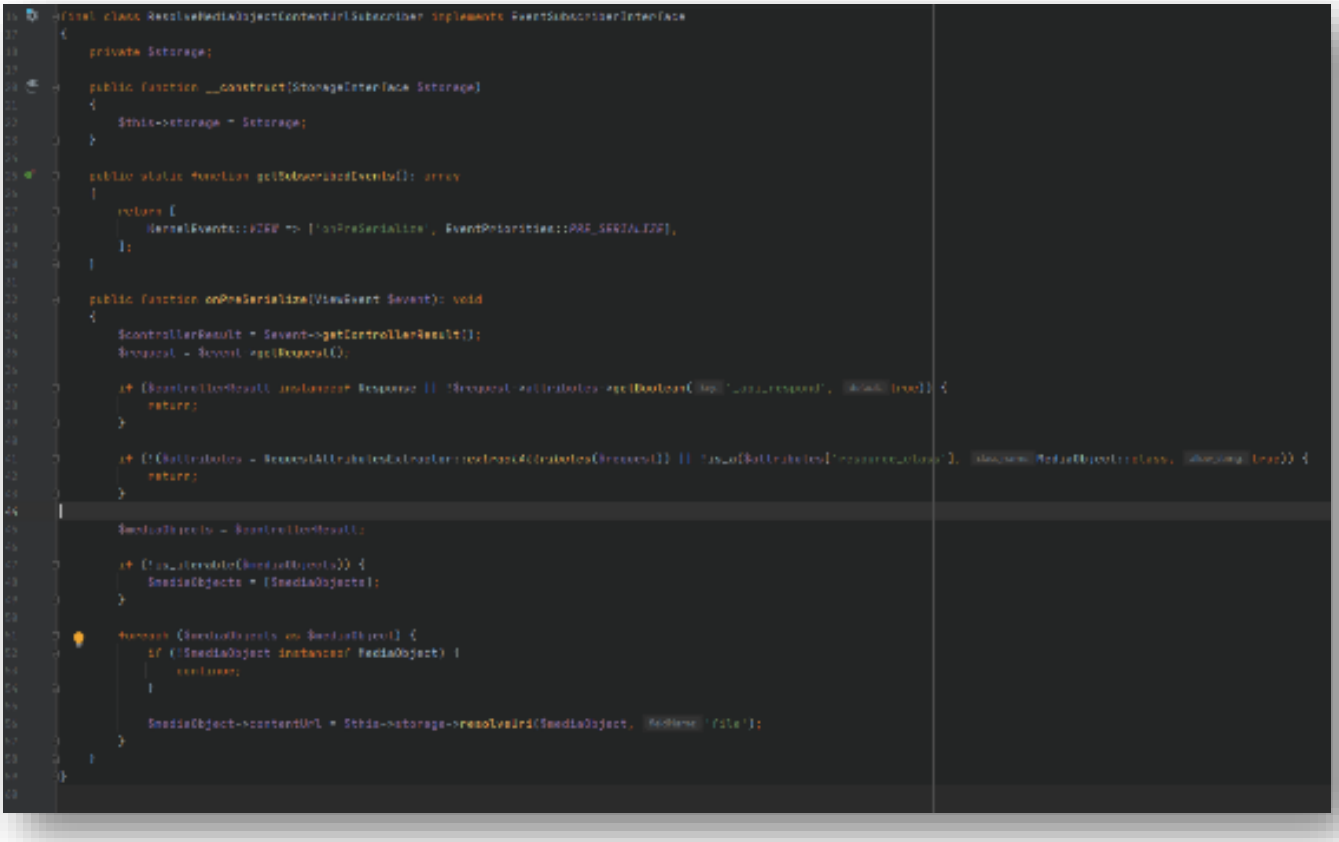


```javascript
function upload(data) {
    console.log(data);

    return axios.post(UPLOAD_API, data, {
        headers: {
            'Content-Type': 'multipart/form-data'
        }
    });
}

export default {
    upload
}
```



```php
    /**
     *
     * @var MediaObject|null
     *
     * @ORM\Column(type="string", length=255)
     * @ORM\ManyToOne(targetEntity=MediaObject::class)
     * @ORM\JoinColumn(nullable=true)
     * @Groups({"project"})
     * @ApiProperty(iri="http://schema.org/photo")
     */
    private $photo;
```

Il faut en outre rajouter le lien vers le MediaObject dans les entités qui en auront besoin. Comme ici dans l'entité Project.



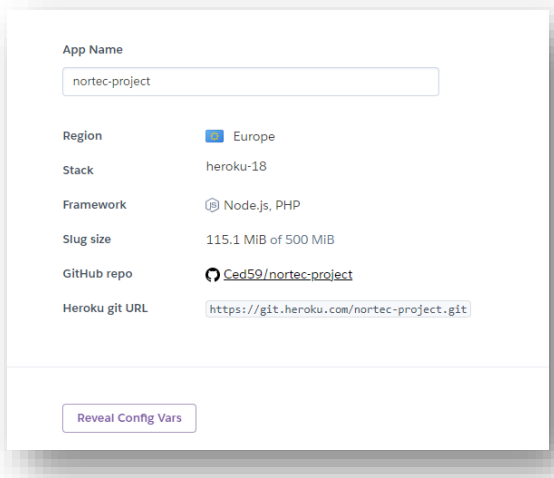
Enfin, pour terminer, nous devons créer un nouvel événement au niveau du Kernel de Symfony qui gérera l'upload, l'enregistrement du fichier sur le serveur, et la création et l'enregistrement du lien du fichier dans la base de données.

IX : MISE EN PRODUCTION

Nous arrivons enfin à la mise en production d'une version que je qualifierai de pré-alpha. Nous avons tenu à l'effectuer afin de voir la procédure de mise en ligne d'une application aussi complexe, et afin de donner une première expérience au client, qui, à ce jour, ne nous a toujours pas fait de retour…

Ma première idée était de mettre l'application en production sur mon hébergement OVH. Malheureusement, j'avais besoin d'une offre « pro » (pour avoir un accès SSH) et je n'ai pas pu l'upgrade à la suite d'un souci chez eux.

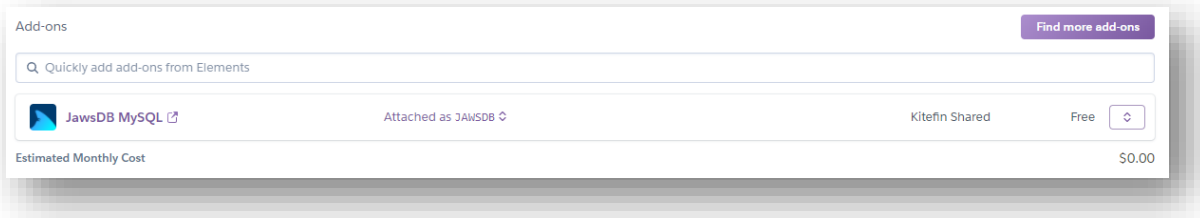
J'ai donc cherché une autre possibilité, gratuite cette fois, et mon choix s'est porté sur Heroku.



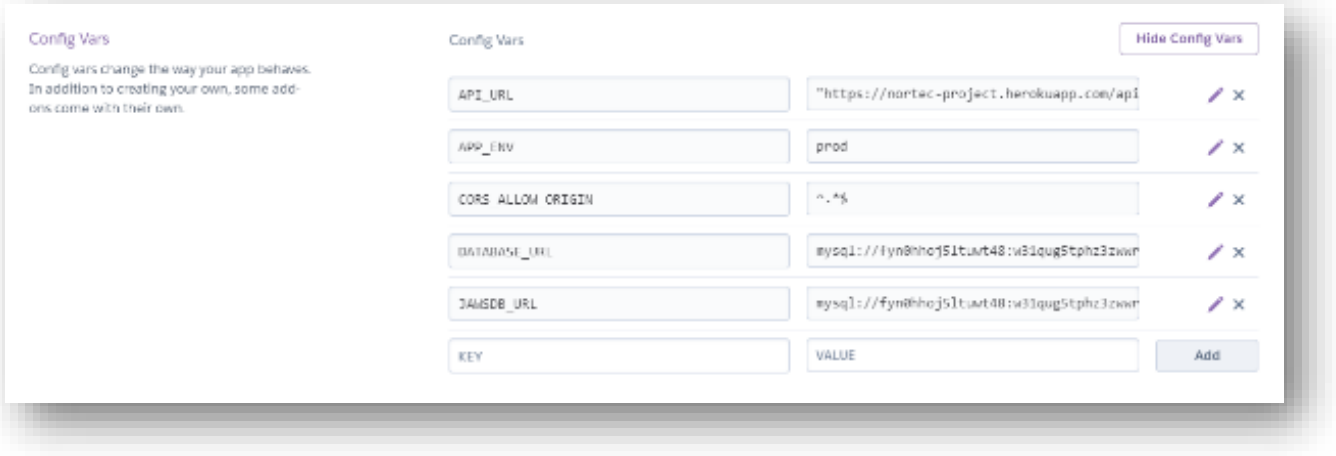
La première chose à faire, après s'être enregistré et avoir créé le projet, est d'installer les technologies nécessaires au projet. Ainsi dans notre cas nous installons Node.js et PHP.

Puis installer un addon nous permettant l'accès à une base de données MySQL (Heroku n'en fournit pas et fait appel à des prestataires externes).

Ensuite, et c'est l'avantage de cet hébergeur, nous pouvons y lier notre compte GitHub. La branche master dans notre cas. Et à chaque fois que nous ferons un push sur cette branche, il mettra à jour automatiquement la version en production.



Du coté de Symfony, nous devons créer un fichier .htaccess afin que le serveur Apache sache quoi faire des requêtes. Ceci se fait tout simplement en installant via Composer le apache-pack.

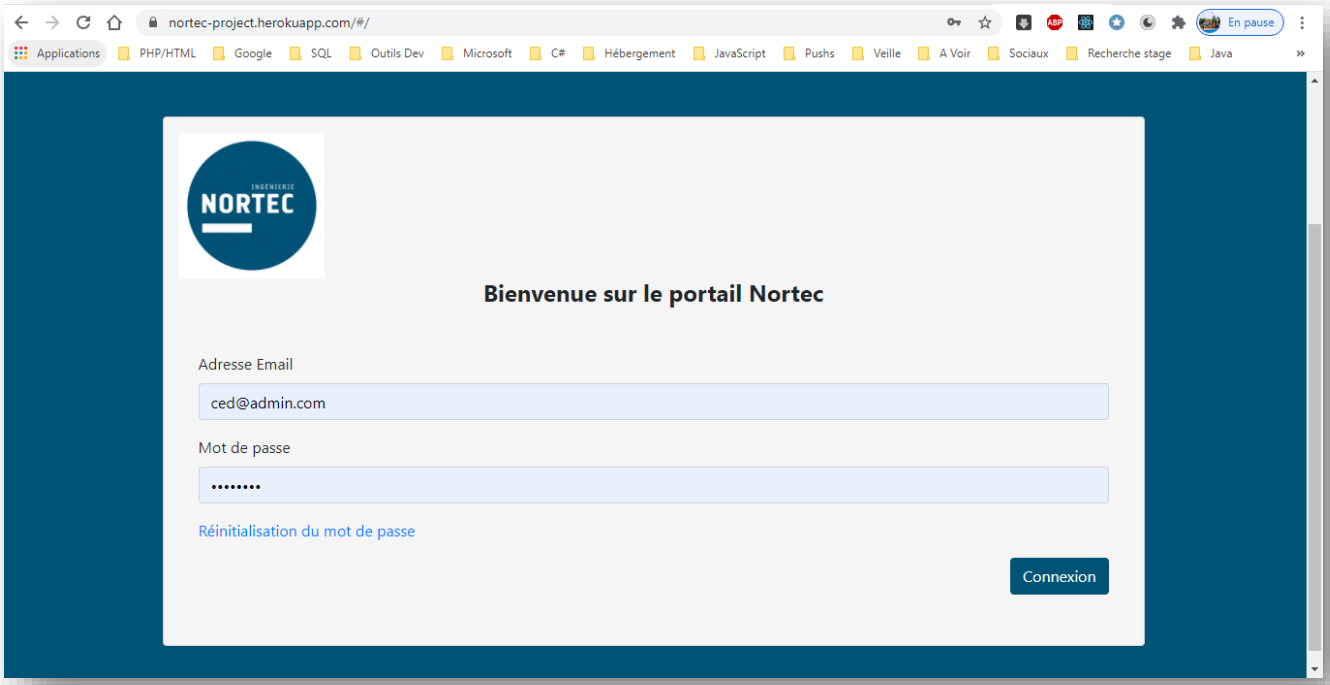


Nous devons ensuite surcharger les variables d'environnement. Heroku fait ça simplement via cette configuration. Il ira chercher dans le fichier .env de Symfony et remplacera les données en local par les nouvelles valeurs.

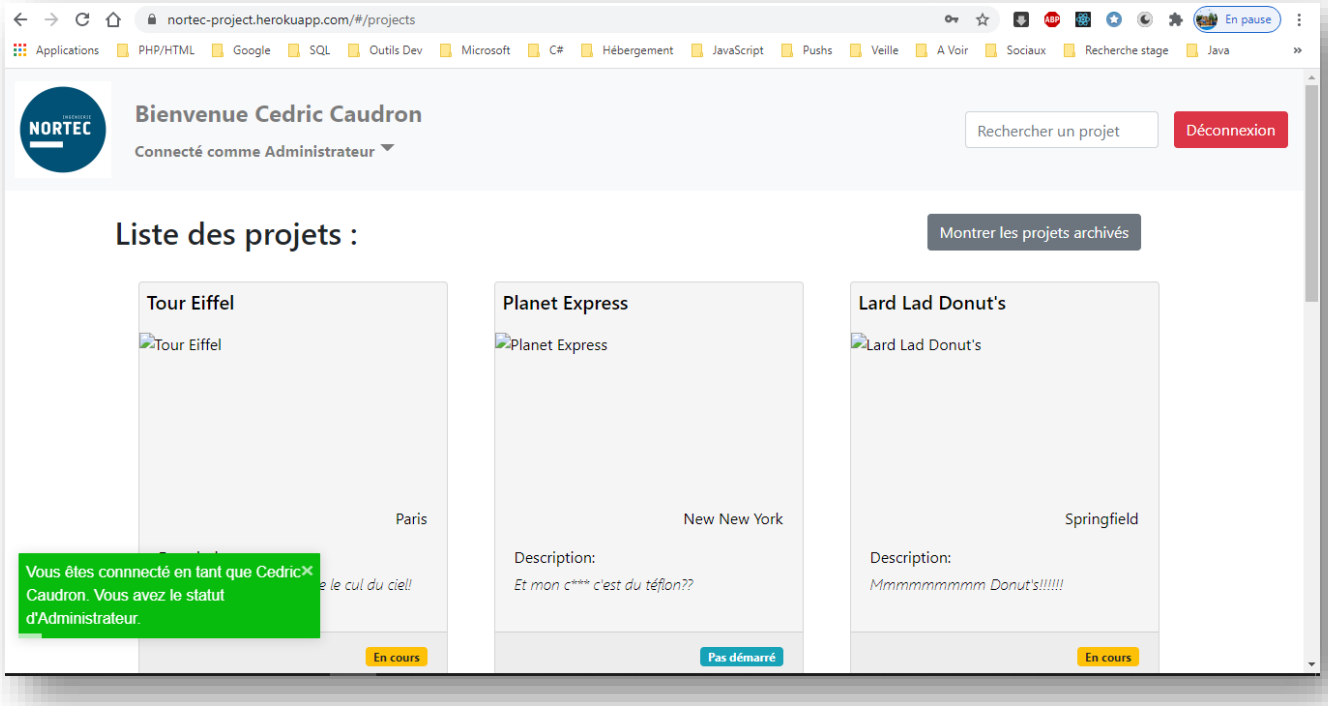
C'est ici que l'on configure également le lien vers la base de données externe.

Nous lançons donc la compilation sur le serveur, et nous voyons le résultat. Et, je ne sais pas si c'est de la chance, mais tout a quasiment fonctionné du premier coup !

Il subsiste tout de même un bug au niveau de l'affichage des images. Le problème vient du fait que les liens sont changés à cause apparemment de certains caractères spéciaux. Elles sont reconnues juste après l'upload mais pas lorsque l'on redémarre l'application. Ceci sera un des bugs à corriger prochainement. Il suffira de changer la méthode dont est généré le lien et son enregistrement en base de données.



L’application est enfin en ligne !!!



X : ET LA SUITE ?

1 : Ce qu'il reste à faire avant la V1

L'application n'est pas encore terminée. Il reste encore beaucoup à faire pour devenir un véritable livrable.

La première chose à terminer, sera l'implémentation des requêtes Axios concernant les rapports.

La correction également des divers bugs qui émaillent encore certaines parties de l'application.

Implémenter un système qui permettra d'enregistrer sur le serveur les fichiers pdf des rapports générés. Ce qui passera par une nouvelle entité de type MediaObject.

Finir l'implémentation des pages d'échéances et des effectifs, avec la gestion bien plus poussée des dates.

Améliorer le responsive design.

Implémenter le système d'envoi automatique des mails, avec le rapport pdf en pièce jointe. Pour ceci, deux solutions s'offrent à nous :

- L'utilisation d'un outil de mailing externe genre MailChimp, qui s'utilise comme une API d'ailleurs.
- L'implémentation directement dans Symfony en y adjoignant un serveur POP/IMAP

2 : Et à l'avenir ?

Plus tard, plusieurs axes d'améliorations pourraient être envisagés :

- L'application mobile dont il a déjà été question
- La mise en place d'un système de cache pour améliorer la réactivité
- L'optimisation des requêtes en utilisant par exemple la pagination intégrée à ApiPlatform
- Des fonctionnalités supplémentaires suivant les besoins

XI : CONCLUSION GENERALE

Nous arrivons enfin à la fin de ce dossier de présentation de l'application que j'ai développée avec Vincent.

Ce fut une grande aventure extrêmement enrichissante sur le plan de l'apprentissage mais aussi humainement.

Elle nous aura valu un très grand engagement que ce soit personnel ou même financier (en payant par exemple une licence de JMerise ou l'hébergement).

Nous aurons ainsi appris à gérer du début à la fin un projet, en faisant des erreurs, découvrant le domaine.

Nous aurons également appris à utiliser plusieurs technologies que nous n'avions jamais abordé auparavant, en commençant de zéro et en enrichissant notre maitrise au fur et à mesure.

Nous aurons vu à quel point un projet sérieux en développement demande du temps, beaucoup de réflexion et d'investissement.

Le seul regret que je pourrai avoir arrivé au point où l'on en est, c'est finalement le manque de suivi du projet par l'entreprise qui l'a demandé… La situation sanitaire je pense n'aura pas aidé… Mais malheureusement je pense que cela signera un coup d'arrêt au développement de l'application, même si j'aimerais continuer à la faire grandir et évoluer par la suite.

Je suis enfin très heureux d'avoir pu participer et mener ce projet, et remercie les personnes qui m'ont fait confiance, en premier lieu Vincent, avec qui le travail en équipe a été très prolifique et agréable.