BOREL Maxime
GI 3e semestre

UNIVERSITE DE TECHNOLOGIE DE COMPIEGNE

RAPPORT DE STAGE TN09

Développement d'applications web et mobile. Participation à la conception, au développement et à la mise en conformité d'applications web et mobile.

ENTREPRISE : AVANT-GOUT STUDIO

ADRESSE : 3 CHE DES CHATAIGNIERS, SAINT-ETIENNE, 42580 FRANCE

SUIVEUR UTC : MLLE DOMITILE LOURDEAUX

SUIVEUR ENTREPRISE : M. ADRIEN PIFFARETTI

03/09/17 ➜ 15/02/18

Remerciement

Fiche technique

SOMMAIRE

01

PRÉSENTATION AVANT-GOÛT ET DE L'ÉQUIPE D'ACCUEIL

Présentation d'Avant-Goût

Présentation du cadre général de l'entreprise

Avant-Goût Studio est une entreprise créée en 2003, le gérant est Adrien Piffaretti. Elle réalise des solutions numériques interactives pour mobile mais aussi pour le web. Voici une liste non exhaustive du genre d'applications qu'elle peut proposer : *visites interactives, réalité virtuelle, lecture augmentée, jeux vidéo*. Son but est la réalisation de projets interactifs innovants. Elle est située, depuis peu, au Village By CA à la cité du Design, à Saint-Etienne.

Son effectif varie en fonction de la taille des projets, et en fonction des différents projets en parallèle.Pendant la durée de mon stage, nous étions 8 en comptant les alternants.

L'endroit était donc idéal pour mon stage, car il favorisait les échanges avec les autres start-up permettant de découvrir d'autres projets innovants.

Le village by CA

C'est une pépinière de startups, avec des bureaux modernes en open-space, et tout le confort nécessaire, permettant la coopération entre start-ups. 22 villages existent actuellement. Leur but est de favoriser le développement d'entreprise avec des projets innovants afin d'avoir de grosses entreprises dans le futur.

Réalisation de l'entreprise et concurrence

Avant-Goût Studio est une entreprise qui propose donc de nombreux projets innovants et pendant la durée de mon stage, l'entreprise était en pleine face de commercialisation d'un livre en réalité augmenté .

C'est grâce à ESCA'PAD à Madagascar le premier livre augmenté, qu'elle souhaite, par la suite, continuer avec plusieurs volets. Le principe est simple : une tablette avec une application installée dessus, le livre, des cartes, et il suffit simplement de tourner les pages du livre, de placer des cartes devant la tablette, et des interactions surviennent à l'écran.

C'est une technologie innovante encore jamais proposée sur le marché.

ESCA'PAD à Madagascar

Esca'Pad à Madagascar est un album jeunesse immersif unique au monde, une aventure scientifique qui se lit, se regarde et se joue.
Avec le livre augmenté « Esca'pad à Madagascar », les enfants de 6 à 9 ans découvrent la grande île, en suivant les aventures drôles, insolites et interactives des reporters Rosie et Joe.

Plus d'informations sur :
https://www.escapadthebook.com/home/

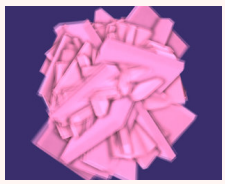
D'autres projets ont vu le jour par le passé comme par exemple :

Saint-Étienne 360°:

Application de visite virtuelle interactive des lieux emblématiques de la ville de Saint-Étienne avec votre smartphone et un casque de réalité virtuelle, on peut visiter la ville.

Memory Wall

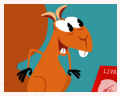
Réalisation d'une installation interactive tactile de 6 mètres de long mêlant jeu vidéo, mémoire et activité physique.



insitu.tech

Solution complète de visite guidée interactive, INSITU permet d'offrir aux visiteurs d'expositions et d'évènements, des contenus numériques qui enrichissent la visite.

et bien d'autre encore.



La liste complète de l'ensemble de ces réalisations est disponible à l'adresse web : https://www.avant-gout.com

Ces multiples créations lui ont permis d'être récompensé par notamment un Max awards, 3 FWA, un Adobe Max Award. Elle a obtenu d'autres distinctions comme par exemple : Adobe Mobile Challenge: « The BIG 4 special prizes » Design, Us Parents choice awards.

Le gros point positif d'obtenir ce genre de distinctions est d'avoir un avantage face à la concurrence dans le cas où un client souhaite avoir un prestataire. Néanmoins, étant donné que la plupart des projets que l'entreprise proposent sont des innovations, la concurrence n'est pas autant rude que dans certains domaines. En effet, il se peut que, pour certaines réalisations, ce soit un nouveau marché (exemple avec Esca'pad à Madagascar) . Mais dans le cas où elle développe des projets plus communs ou sollicités par un client, il faut savoir se démarquer, on peut citer par exemple le projet avec la ville de Saint-Etienne où il faut proposer un certain service de qualité, mais comme dit précédemment, les distinctions obtenues peuvent "rassurer" les clients.

De plus, l'entreprise est bien positionnée car elle propose aussi des créations en VR (réalité virtuel), qui est une technologie en forte croissance actuellement. Elle avance bien avec cette technologie qui va fortement se développer dans les années à venir.

De même avec la technologie utilisée par Esca'Pad à Madagascar, qui a été brevetée.

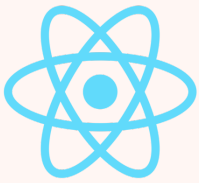
Conclusion

Pour résumer, le lieu était idéal car il permettait de rencontrer d'autres entreprises avec des projets innovants. De plus, l'entreprise où j'effectuais le stage a fait par le passé des projets très intéressants et elle en commercialisaient encore des nouveaux pendant

Présentation de l'environnement technologique

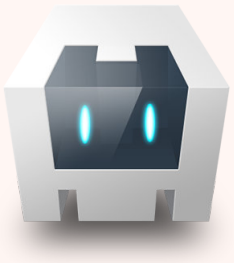
Dans cette partie, nous allons aborder les différentes technologies utilisées pendant mon stage mais aussi celle utilisées dans l'entreprise, sans trop rentrer dans les détails car je vais développer dans la seconde partie les outils, les technologies que j'ai utilisé.

L'ensemble des projets avaient des dépôts sur Github afin de faciliter la coopération, notamment pour la première partie du stage où j'ai eu l'occasion de développer une application mobile avec un employé.



J'ai développé sous plusieurs langages. Pendant la première partie du stage, j'ai développé, Dollycast (que nous présenterons par la suite), une application mobile en React Native[1].

J'ai aussi développé en php avec un framework Symfony pour la seconde partie de mon stage.



Au niveau de l'entreprise, beaucoup de langages étaient utilisées. Par exemple, des alternants développaient en Unity afin de faire une application en réalité virtuelle.

L'application Esca'pad à Madagascar est développé en Cordova avec un plugin Wikitude.

Il est assez ennuyant de faire la liste de l'ensemble des technologies utilisés en général par l'entreprise, mais il faut surtout comprendre que de nombreuses sont utilisées. Cependant, c'était surtout des langages pour développer des applications mobiles qui étaient utilisées pendant la durée de mon stage.

1 Explication dans la suite du rapport

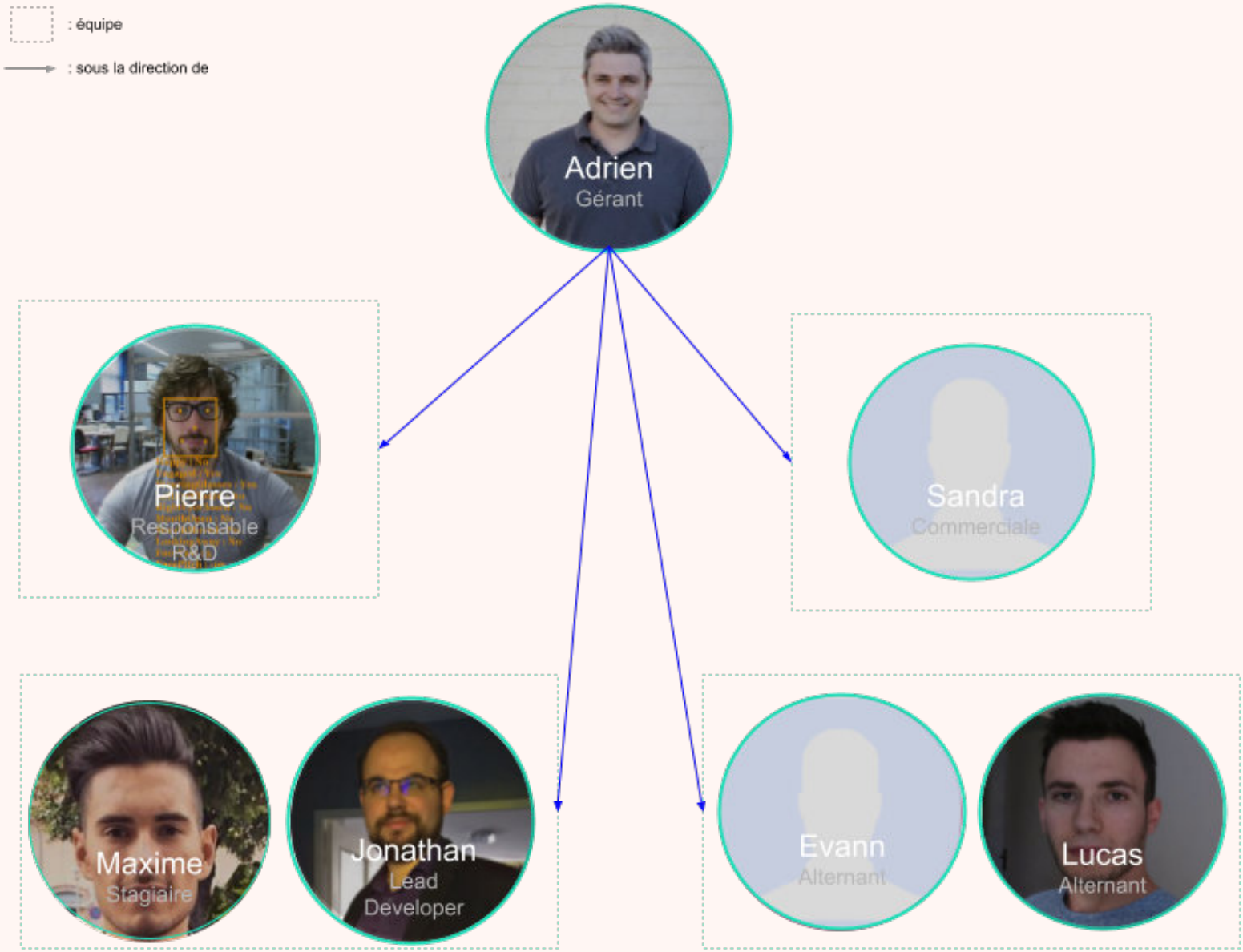
Présentation de l'équipe

Présentation générale

Etant donné que l'entreprise est une petite entreprise, il ne sera pas difficile de faire la liste de l'ensemble des membres de l'équipe.

Afin de faciliter la lecture et pour que ce soit plus simple à visualiser, voici un organigramme fait par mes soins présentant l'organisation au sein de l'entreprise :

Nous étions 7, répartis en petites équipes afin de réaliser des projets différents, comme j'ai déjà pu l'évoquer précédemment. J'ai travaillé en grande partie avec Jonathan, avec le développement de l'application mobile Dollycast les ¾ du stage et enfin j'ai bénéficié de ses conseils pour le développement d'une application web en php Symfony.



Présentation sur l'organisation des équipes

Communication orale

Que ce soit pour l'équipe dans laquelle j'étais ou pour les autres, nous avions pour obligation de faire un débriefing, le lundi matin, pour discuter de ce que nous avions fait la semaine précédente et ce que nous comptions faire la semaine. Cette pratique permettait de tenir au courant les autres membres de l'entreprise de ce que nous faisions, mais aussi de pouvoir interagir tous ensemble en cas de blocage sur un problème.

Nous avions donc une discussion commune le lundi matin pour mettre au courant les autres membres de l'entreprise, puis de nombreuses discussions par jour au sein de l'équipe dans laquelle j'étais afin de se tenir au courant.

Communication technologique

Le site web RedBooth, qui est un outil Web de collaboration et une plateforme de communication sur le lieu de travail, était utilisé très fréquemment.

C'est un système similaire à Trello avec des tâches à effectuer, un système de tableau, d'attribution de tâches, etc..

Ainsi, Adrien Piffaretti avait un œil sur ce que nous faisions, et au sein du groupe, nous pouvions savoir ce que faisait l'autre, et nous pouvions savoir ce qu'il restait à faire afin de finir la première version de l'application.

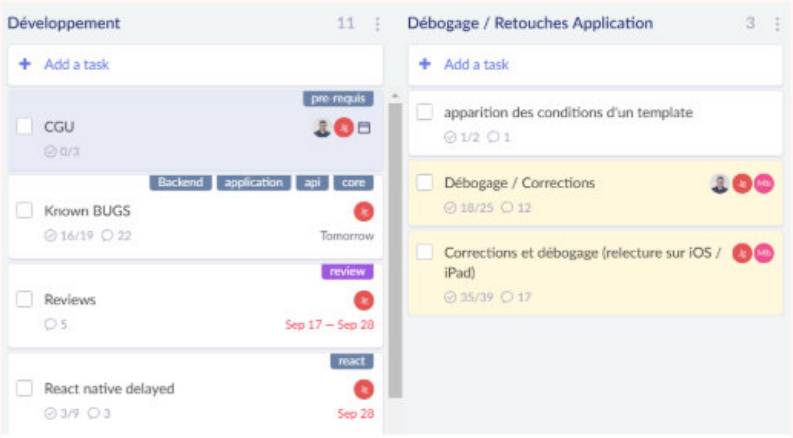
Adrien Piffaretti nous faisait des tickets concernant attentes.

Attentes sur les projets

Adrien Piffaretti donnait les directives. Par exemple, dans le cas de mon équipe, il fallait que l'application soit fini pour une date donnée, mais il donnait aussi des ordres sur ce qu'il fallait faire en priorité, ce qui n'allait pas dans l'application etc..

C'est lui qui dirige l'ensemble des groupes de travail. Étant donné que l'ensemble des projets étaient des projets internes (sans client avec une date que ce dernier souhaitait), c'est lui qui fixait ses attentes, donnait les ordres, et validait selon ses préférences. (Cf communication technologique et les tickets)

Logo Redbooth



Tâches dans Redbooth

➡ La taille réduite de l'entreprise permettait de communiquer facilement sans perdre d'information, et surtout de savoir ce que les autres faisaient et de leur venir en aide en cas de difficultés majeurs.

Présentation de mon équipe

Première partie du stage : développement de Dollycast

Pendant la première partie de mon stage, j'étais avec Jonathan un développeur très expérimenté (en php notamment mais il débutait lui aussi en React Native), qui a su fortement m'aider au début de mon stage. J'étais en grande partie avec lui, et c'est lui qui m'a suivi le long de mon stage afin de savoir si j'avais des difficultés par exemple. Nous avions pour mission d'achever l'application Dollycast que Jonathan avait déjà commencé.

De lourdes responsabilités pesaient sur nos épaules, car il fallait pouvoir livrer une première version avant une date donnée.

L'équipe étant petite, l'organisation n'était pas extrêmement compliqué. On essayait de se fixer des objectifs à réaliser sans perdre trop de temps sur des détails futiles. Il fallait que "le gros" de l'application soit fini pour la première version.

Afin de réussir à s'organiser le plus correctement possible, nous avons essayé de se fixer des heures dans la journée afin de discuter de ce que nous avions fait et sur quoi nous bloquions. C'était, je pense, primordial, même en équipe réduite, de communiquer afin de ne pas se perdre.

Deuxième partie du stage :

Conclusion

L'avantage de ce genre de structure est qu'il est plus simple de créer un esprit d'équipe, car tout le monde se connaît et tout le monde tente de s'entraider. C'est un gros point positif de mon stage.

Pas de réel méthode était utilisé a par une communication omniprésente pour ne pas s'égarer, un debriefing le lundi matin, et l'utilisation de RedBooth pour tenir au courant de ce que l'on fait..

02

MISSION DE L'ASSISSTANT-INGENIEUR DÉVELOPPEUR D'APPLICATION WEB ET MOBILE

Sujet du stage

Description de la mission

Avant-Goût Studios est une agence de développements interactifs et un acteur de l'innovation numérique en région Auvergne-Rhône-Alpes.

Dans le cadre de ses activités portées sur la lecture augmentée (http://www.thecovar.com),

Avant-Goût Studios recrute en stage : Un ingénieur développement Web - H/F

Rattaché auprès de la direction, vous participerez au lancement d'un produit innovant de lecture augmentée. Vous aurez en charge la réalisation d'une plateforme de production de contenus numériques à destination des grands

groupes d'édition.

Votre mission :

- Participer à la modélisation et à l'architecture du projet,

- Développer des modules de la plateforme

- Concevoir une architecture serveur,

- Tester les montées en charge,

- Faire le suivi et la mise en production de vos travaux.

Profil du candidat recherché

En école d'ingénieur informatique, vous maitrisez l'algorithmique, la

programmation orientée objet et les design pattern, et vous avez de bonnes connaissances web, Front et Back.

Idéalement, vous avez déjà travaillé sur des projets d'applications web, basés sur un framework PHP : Symfony, Laravel, Zend...

Vous vous intéressez au développement pour mobile et à l'apprentissage de ReactJS et React Native.

Cependant, le sujet a un peu changé, car quand je suis arrivé dans l'entreprise, l'application en réalité augmentée était déjà en phase de commercialisation (Esca'pad à Madagascar). Cette dernière a été fini plus tôt que ce qui avait été prévu.

C'est pour cette raison que j'ai été mis avec Jonathan pour développer une autre application et en React Native.

Ainsi, le projet sur lequel je devais travailler à changer mais pas les technologies que j'allais utiliser.

En effet, j'ai utilisé en priorité du React Native et du php Symfony pour la seconde partie de mon stage.

Première partie du stage : développement de Dollycast

Comme dit précédemment, je suis arrivé en pleine phase de conception de l'application Dollycast (que je présenterai par la suite).

Un cahier des charges ainsi qu'un cahier des charges technique avaient déjà été réalisé avant que j'arrive. La conception avait déjà débuté.

A mon arrivée, l'objectif était clair : maîtriser React Native pour ensuite aider Jonathan à développer pour finaliser une première version de l'application.

Puis par la suite développer d'autres fonctionnalités.

L'application à la fin de mon stage était donc largement utilisable, car nous avons pu livrer l'application finale.

Deuxième partie du stage

Présentation des projets

Dollycast

Dollycast est une application mobile permettant la création de films à partir de templates. Le fonctionnement est simple : grâce à des templates créés au préalable sur une API, l'utilisateur va choisir son template puis va simplement suivre différentes étapes en remplissant certains champs et en filmant des scènes en respectant certaines indications. Une fois, toutes les étapes réalisées, l'utilisateur peut envoyer le tout au serveur avec un simple bouton. La suite se passe au niveau du back end, l'utilisateur ne fait donc plus rien. Une fois que le serveur a reçu l'ensemble des vidéos filmées, il va réaliser un montage vidéo à la volée sur After Effects en fonction du template, des informations remplies, et des scènes filmées. Ce dernier est piloté grâce à un script javascript.

Une fois, le montage fini, l'utilisateur reçoit sa vidéo.

Il faut noter que l'utilisateur pourra envoyer ses vidéos pour le montage seulement s'il paye avec une monnaie virtuel, le Dolly.

> **Le business model variera légèrement en fonction du mode de vente, et pourra présenter quelques contournements de fonctionnement. Ainsi un consommateur final pourra payer la compilation à l'unité, alors que les grandes société auront accès à diverses offres d'achats de masse, avec des économies d'échelles, et des outils pour suivre l'attribution des crédits. L'offre Dollycast est donc centrée sur la vente de templates, aux entreprises et revendeurs d'abord pour ensuite voir si une offre grand public est viable**
>
> Cahier des charges client

Concernant l'API (backoffice), on peut aisément créer des templates, car Jonathan a développé lui même le back end. Un utilisateur (administrateur) crée un template en choisissant un groupe. Il va définir les différentes scènes à tourner, les différentes indications à écrire, etc.. grâce à un formulaire.

(METTRE EXEMPLE BACKOFFICE AVEC PLEINS EXEMPLE FORMULAIRE)

Nous avons fait un exemple de ce qu'on peut faire avec Dollycast et il est disponible à l'adresse : https://drive.google.com/open?id=1XVfu681gysCGu-H5sWxf_6o2FlSVpdf3

Et voici la vidéo modèle que l'application imite:
https://drive.google.com/open?id=1WvcNtNYdGIRKuTsFlaRaq3kWKqaDvJo2

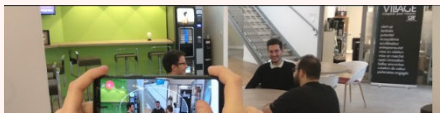
Ainsi l'utilisateur va lancer le template, va regarder la vidéo modèle pour se rendre compte du rendu qu'il va avoir. Il suit les indications à faire pour chaque scène. Par exemple: renseigner le nom de l'acteur 1, filmer l'acteur dans tel position. En lançant l'enregistrement de la vidéo, une image va se superposer pour indiquer la position dans laquelle l'acteur doit se mettre. Afin de mieux comprendre, voici un exemple d'utilisation de l'application sur la page suivante :

Dollycast

1.

À partir d'un film modèle
réalisé sur mesure
selon vos besoins

Dollycast



2.

Laissez-vous guider
dans la personnalisation
des plans et textes

Dollycast

3.

Recevez votre film personnalisé
quelques minutes plus tard
prêt à être partagé !

Dollycast

Planning

Dollycast

L'application devait être livré le 15 octobre. Cependant, suite à divers problèmes rencontrés sur le serveur, ou bien sur les retards pris sur les fonctionnalités, nous avons dû repousser la sortie. Le 22 octobre était donc attendu une application stable. Malheureusement, encore une fois la date a été repoussé afin de sortir une application sans bug.

Finalement, une application stable a vu le jour le 12 novembre. Cependant, j'ai appris plus tard que la vraie date de livraison était prévue pour le 15 novembre. Nous avions donc une marge de 1 mois par rapport à la date annoncée.

Après le 12 novembre, la première version était presque disponible. En effet, on nous a demandé de continuer de corriger les quelques bugs restants, d'ajouter d'autres fonctionnalités, et de faire quelques retouches graphiques jusqu'au 30 novembre. Des améliorations pour une deuxième version, comme la duplication de films par exemple, étaient encore à faire.

En effet, des fonctionnalités demandées n'étaient pas prioritaire pour la première version. Le peu de fonctionnalités restantes pour la prochaine version ne nécessitait pas deux développeurs, c'est pour cette raison que j'ai été écarté du projet.

Cependant, nous avons dû repousser maintes et maintes fois la sortie de la première version, à cause de bugs qui survenaient, mais aussi à cause de nouvelles fonctionnalités à implémenter demandées qui n'étaient pas prévues pour la V1.

Aucun test unitaire n'ont été fait, et qui est par conséquent un problème concernant l'évolutivité, car l'architecture de l'application est devenue importante, et si un bug survient, il faut plus de temps pour le résoudre.

fefe

grgr

Contributions

Dollycast

Le projet était déjà commencé, il a fallu que je m'adapte au code existant, qui n'était pas une épreuve facile. Le squelette de l'application était déjà fait, ainsi que de nombreuses fonctionnalités.

J'ai dû par la suite faire des tâches assez simples pour me familiariser sur ce qui avait déjà été développé.

J'ai commencé à m'entraîner sur le design pour que l'application soit au plus proche de la maquette.

Par la suite, j'ai pu commencé sérieusement avec du fonctionnel, qui n'a pas toujours été très simple. J'ai, par chance, pu bénéficier de beaucoup d'aide de Jonathan tout au long du projet en cas de blocage ou de problème.

Juste avant la date de rendu, il a fallu faire de nombreuses corrections de bugs mais aussi des corrections graphiques.

A la fin, notre application était finie.

Pour résumer, j'ai développé des fonctionnalités utiles à l'application selon moi, j'ai fait du design afin d'avoir une application proche de la maquette, j'ai corrigé de nombreux bugs, et j'ai ajouté des fonctionnalités.

Mon rôle en tant que stagiaire était donc d'assister Jonathan en l'aidant à finaliser le tout pour avoir quelque chose de propre.

fefe

Outils et technologies

J'ai déjà commencé à aborder dans la première partie les technologies utilisés, je vais donc rentrer un peu plus dans les détails.

Technologies utilisées pour les applications

Comme dit précédemment pour l'application Dollycast a été développé en React Native qui est un framework créée par Facebook, pour construire des applications Android et iOS. C'est une solution hybride afin de se passer du développement natif pour chaque OS.

On utilise du Javascript avec une syntaxe ES6[1].

Elle est rapide à mettre en place, il n'y pas besoin de télécharger énormément de chose, a par avoir un appareil Android ou iOS, qui fait 2d'elle une technologie accessible, et il n'y pas besoin de compiler le code après chaque utilisation.

Elle est soutenue par de grosses structures comme notamment Facebook et Airbnb.

Redux a aussi été utilisé pour l'application. C'est une librairie Javascript. Je ne vais pas rentrer dans les détails techniques, mais c'est un système de centralisation des données et des actions. Pour simplifier, c'est similaire à une base de données locale.

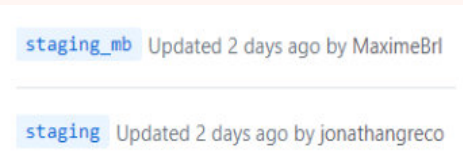
L'API qui retournait les résultats de la base de données MySQL a été codé en Symfony PHP par Jonathan.

Outils utilisées

Editeur de texte et gestionnaire de versions

A par une tablette, et un éditeur de texte (j'ai utilisé SublimeText), j'ai aussi utilisé des outils comme React Native Debugger pour le débogage de l'application React Native.

De plus, j'utilisais Git pour synchroniser le travail avec celui de Jonathan. Un dépôt sur Github avait été créé pour ça. Nous avions nos branches respectives afin de pouvoir faire nos tâches. J'étais dans le groupe de l'entreprise qui contenait d'autres dépots Git comme par exemple celui de l'API pour créer les templates. Nous avions nos branches respectives : staging et staging_mb pour moi. Comme je l'ai dit précédemment, nous communiquions énormément, c'est grâce à ça que nous avions réussi à avoir toujours les modifications faites par l'un et par l'autre sur nos branches respectives.



Développement

Android Studio a aussi été utilisé pour construire le projet android.

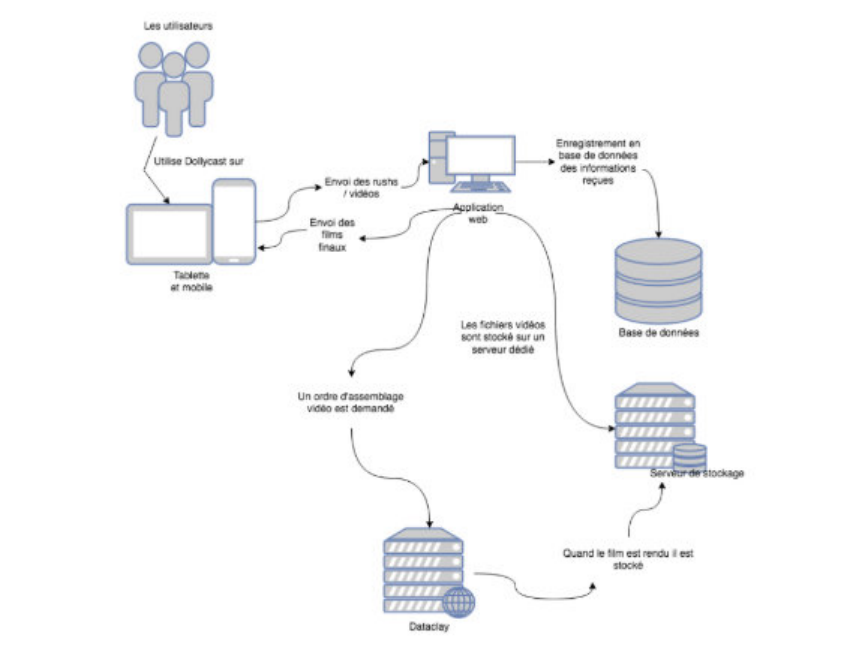
Je n'ai malheureusement pas pu tester l'application sur un appareil iOS pendant que je développais, car il fallait un Mac avec xCode. Par chance, Jonathan en avait un.

1 ECMAScript : c'est un standard de langage de programmation qui définit: syntaxe, types de variable, etc..

Serveurs

L'API Dollycast était sur un serveur distant.

Pour ce qui est du rendu des vidéos, un serveur hébergé sur Dataclay était utilisé qui est un serveur de traitement vidéo permettant d'affecter les particularités du template, de mixer les rushs ensemble et de fournir le film, comme je l'ai évoqué précédemment.



Un développement surveillé

Adrien Piffaretti avait accès au dépôt Git et pouvait surveiller ce qu'on faisait, en regardant notamment les commit et les modifications effectuées.

De temps à autre, il venait vérifier ce qu'on faisait, et testait l'application afin de nous dire ce qui n'allait pas.

Conclusion


Environnement Virtuel
PHPStorm
Développement Serveur php
Sublime Text 3
Atom
WebStorm
Développement Mobile
Outil de déploiement
PHPUnit
PHPLOC
PHPCPD
Intégration continue
Développement
GitHub
GitLab
Bitbucket
Versioning
Stockage Partage de Code
PC ou Mac
SmartGit
TortoiseGit
Source Tree
Git Kraken
Client Git et Mercurial
sonarqube
Code Review
Data
Expo
Test sur appareils
Jira Software
RedBooth
Gestion de tâches et d'équipe
Parc mobile
Google Drive
Partage documents Collaboration
MySQL
ORACLE
SGBD
redis
Système de cache

Prise de recul

Après réflexion, je suis assez content du travail effectué pour Dollycast, car malgré les difficultés que j'ai rencontré au début, j'ai su m'adapter pour ensuite aider au mieux Jonathan. Les fonctionnalités que j'ai pu développé et les modifications graphiques vont réellement servir, car malgré quelques bugs mineurs ou ajouts supplémentaires que l'entreprise va choisir concernant l'application, le plus dur a été fait.

L'application a été créé dans le but de faire du profit, donc de prospérer. Par conséquent, mon travail effectué servira pour l'entreprise dans le futur.

Concernant les vidéos enregistrés sur le serveur Dataclay, après y avoir réfléchi et après avoir demandé à Jonathan, nous sommes arrivés à la même conclusion, l'utilisateur final va devoir accepter certaines conditions au lancement de l'application, car il risque d'y avoir dans le futur de l'analyse de vidéos. En effet, il vaut mieux éviter d'avoir des utilisateurs qui profitent de Dollycast pour tourner des scènes illégales qui pourrait donner une mauvaise image à l'application. Les données seront donc protégées du public mais probablement visionnées par une personne au début, l'utilisateur devra faire attention à ce qu'il envoie au serveur. Dans le futur, on peut imaginer un traitement de vidéos automatisées

Ces conditions ne sont pas écrites et donc pas encore développées. Dans une future version, l'utilisateur devra les accepter s'il veut avoir accès aux fonctionnalités l'application

Malheureusement, après avoir réfléchi sur le problème et après en avoir discuter en entreprise, il va sûrement falloir que l'entreprise recode à nouveau l'application en natif. En effet, React Native est une technologie peu mature et assez prometteuse, sauf qu'actuellement, de nombreux problèmes/bugs non résolus existent, et par conséquent cela empêche le développement de certaines fonctionnalités, ou bien les ralentis.

Pour mieux visualiser, je peux citer un bug assez gênant avec l'upload de vidéo qui était trop long, nous avons dû créer une issue sur le dépôt de la librairie pour qu'ils corrigent le bug. C'est "essuyer les plâtres". Et nous en avons eu d'autre.

Et comme on ne sait pas si React Native va peser dans le futur et va continuer à avoir des bugs, l'entreprise va peut-être envisager de recoder en Java pour Android et en Swift pour iOS pour avoir quelque chose d'encore plus stable.

Nous avons finalement réussi à réaliser la première version, avec certaines difficultés concernant les limites de React Native. Cependant, peut-être qu'en natif, nous n'aurions pas eu ses problèmes.

De plus, nous avons réalisé plus que ce qui a été demandé initialement pour la première version

(METTRE AUTRE APPLICATION ET RELIRE EN FCT CE QUI A ETE FAIT)

02

Réalisations durant le stage

Mes réalisations

Dollycast

Mes réalisations ont déjà été expliqué précédemment, je vais donc rentrer un peu plus dans les détails. On peut les séparer en quatres parties. Tout d'abord, je parlerai de ma formation à React Native, pour vous parler ensuite des modifications graphiques que j'ai effectué. J'ai aussi fait du fonctionnel en ajoutant de nouvelles fonctionnalités. Enfin, sur la dernière ligne droite avant d'avoir une application stable, j'ai corrigé beaucoup de bugs.

Cependant, je vais tenter d'expliquer en détail les grosses fonctionnalités que j'ai fait, et laisser de côté certaines corrections de bugs, de créations de pages etc.. qui m'ont pris du temps mais qui serait trop difficile et non intéressants à expliquer. Ainsi, certaines choses seront seulement écrites mais pas expliquées en détails, alors que d'autres seront mises totalement de côté. Je vais donc vous parler de certaines fonctionnalités que j'ai développé mais pas toutes.

J'ai fait énormément de retouches graphiques. Cependant, la plupart d'entre elles ont été des tâches rapides à réaliser. Au contraire, j'ai fait peu de tâches fonctionnelles mais elles étaient très longues (surtout pour quelqu'un qui découvrait React Native et un code déjà existant). C'est sur le développement du fonctionnel que j'ai le plus eu de mal, car en effet, **Redux** est assez dur à appréhender. Etant donné que l'application reposait en partie sur **Redux** et qu'elle était déjà bien avancée, il fallait que j'utilise ce qui existait et par conséquent, il fallait que je comprenne en détail ce qui avait été fait.

Cependant, je n'ai pas fait que des modifications graphiques puis ensuite des modifications fonctionnels comme le plan pourrait laisser croire. J'ai d'abord fait des retouches graphiques pour ensuite faire du fonctionnel. Mais par la suite les deux se sont entrecroisés. En effet, si on fait une modification fonctionnel, il faut toucher au design.

(METTRE ANNEXE AVEC CHAQUE ECRAN ET LEUR NOM)

(METTRE ANNEXE AVEC TOUTES LES FONCTIONNALITES FAITES)

(METTRE ANNEXE EXEMPLE CODE REACT)

Afin de mieux comprendre, il est essentiel de se référer à l'annexe où je montre les différents écrans et leur nom.

Apprentissage de React Native

Durant les deux premières semaines de mon stage, j'ai dû m'autoformer à React Native. J'ai aussi dû comprendre le code existant pour me lancer dans le vif du sujet, et apprendre à utiliser les différents outils comme React Native Debugger ou bien Postman.

En plus de prendre connaissance du code existant, j'ai dû me familiariser avec le site web de création de templates, car il fallait que je développe certains composants sur l'application mobile comme des champs date par exemple, et il fallait comprendre comment l'API Web retournait les données JSON à l'application mobile. De plus, il était nécessaire par moment que je crée certains templates.

J'ai préféré commencer rapidement à développer plutôt que de faire des cours sur le web, car j'apprends mieux en commençant sur du concret. ("on apprend de ses erreurs").

C'est pour cette raison que j'ai décidé d'attaquer sur quelque chose de pas très compliquée comme de la correction graphique.

Corrections graphiques / Création de pages

L'application a pour vocation d'être simple d'utilisation. A mon arrivée, une maquette faite par un designer existait déjà. Etant donné qu'il fallait que je comprenne comment React Native fonctionnait, j'ai préféré me lancer sur quelque chose de moins compliqué. C'est pour cette raison que j'ai décidé de me lancer sur des corrections graphiques afin d'avoir une application le plus proche de la maquette.

J'ai eu énormément de mal au début, car React Native utilise Flex[1]. C'est un très bon outil mais assez dur à maitriser quand on débute.

J'ai commencé à faire quelques retouches graphiques tout en m'efforçant de comprendre le fonctionnement de React et de me familiariser avec le code. C'était assez difficile au début. Cependant, après quelques semaines, j'ai entièrement compris son fonctionnement, et je faisais des retouches graphiques très rapidement. J'ai donc perdu énormément de temps au début du stage.

J'ai à peu près modifié toutes les pages existantes.

Au tout départ, j'ai tenté d'éviter totalement l'aspect fonctionnel de l'application car je manquais de compétences techniques et je ne connaissais pas assez l'application.

Pour commencer, j'ai fait des tâches simples, comme faire coller à la maquette la liste du **ProjectScreen** en ajoutant une image, en gérant l'opacité, en ajoutant une fenêtre modale en tapant sur …. J'ai fait de même avec le **WorkflowScreen**.

Je ne vais pas faire la liste de l'ensemble des pages modifiées, mais je souhaite insister sur le fait que j'ai pu faire des modifications simple comme créer une modale, à d'autres beaucoup plus compliquées, nécessitant de modifier le fonctionnement global de la page.

J'ai par la suite eu la chance de créer des pages comme notamment le **ForgottenPasswordScreen**. Cela m'a permis de manipuler la librairie React Navigation, qui est une librairie permettant de naviguer d'écran en écran avec notamment un système de pile.

Pour choisir les pages que j'allais modifié j'ai commencé par le plus simple, pour ensuite continuer sur du plus compliqué. Je ne vais pas parler des petits changements que j'ai fait comme par exemple supprimer le positionnement en Flex pour placer en absolute ou autre.

Par moment, nous faisions des modifications graphiques pour ensuite y revenir dessus longtemps après. En effet, l'ajout de nouvelles fonctionnalités modifiait par moment la disposition de la page, ou bien on nous demandait de changer le design de la page car finalement ce n'était pas aussi bien que ce qu'on imaginait.

Le plus gros problème dont nous avons fait face, est le placement des écrans suivant l'orientation de l'appareil. En effet, certaines fois, nous avons quelque chose de convenable au format paysage, mais quand on tournait l'appareil, on se rendait compte que c'était trop chargé.

Il fallait gérer donc plusieurs cas pour afficher convenablement. Par exemple, sur **ProjectScreen** afficher "+ Nouveau Film" en paysage et "+" en portrait.

De plus, il fallait aussi gérer le cas de l'affichage en fonction de la résolution. En effet, sur certains écrans, tout était bon mais quand on passait sur un iPhone, l'affichage était médiocre.

L'un des composants qui s'affichait le plus mal était l'image.

Nous avons donc investiguer énormément sur la manière de faire. Nous avons notamment penser à utiliser une librairie pour pouvoir redimensionner automatiquement les images.

Finalement, nous avons découvert l'existence de la méthode OnLayout permettant de connaître la dimension en pixel de l'appareil ou bien d'un composant. Nous l'avons utilisé au besoin et dans les cas où ça ne suffisait pas, nous avons mis des conditions pour gérer l'affichage en fonction de la taille de l'écran.

```
onLayout = (e) => {
  const width = e.nativeEvent.layout.width;
  const height = e.nativeEvent.layout.height;
  this.setState({height: height, width: width});
};
```

1 Un composant peut spécifier la disposition de ses enfants à l'aide de l'algorithme flexbox. Flexbox est conçu pour offrir une mise en page cohérente sur différentes tailles d'écran - facebook.github.io

Cependant, l'une de mes plus grosses modifications graphiques développée a été le header du **WorkflowScreen**, que je vais volontairement développé dans la partie sur les fonctionnalités, car selon moi c'est un ajout graphique mais il y a beaucoup plus de travail fonctionnel.

Les deux dernières grosses modifications que l'on m'a proposé, a été de refaire entièrement la page de login en entier car elle n'était pas très belle sur certaines taille d'écran. On m'a donc conseillé de gérer l'affichage différement. Il a donc fallu faire une fonction calculant dynamiquement la hauteur du logo. Par exemple, on met une hauteur du logo à 45% et le formulaire à 55%, sauf que si la hauteur dépasse 360 pixels, dans ce cas, on fixe la hauteur à 360, et la hauteur du formulaire à : taille de l'écran - 360px.

```
getDimensions = () => {
  this.bannerHeight = this.state.orientationHeight*0.41;
  this.formHeight = this.state.orientationHeight*0.50;
  this.formWidth = this.state.orientationWidth*0.85;
  if (this.bannerHeight > 355) {
    this.bannerHeight = 355;
    this.formHeight = this.state.orientationHeight-370;
  }

  if(this.formWidth > 420) {
    this.formWidth = 420;
  }
}
```

Il fallait aussi avoir la largeur du formulaire à 80% de la largeur de l'écran, s'il dépassait 420, on prenait 420px. Vous pouvez regarder l'annexe avec l'ensemble des écrans pour visualiser le travail que j'ai effectué.

design@PriseDeRecul

design@PriseDeRecul **:~$** Apprendre à placer des éléments en React Native est assez similaire que placer en Web. C'est en s'efforçant de comprendre le fonctionnement global qu'on réussi à gérer la disposition rapidement. Cependant, il existe toujours des moments, que pour des raisons obscures, que l'affichage soit différent de ce qu'on imaginait, il faut donc être constamment vigilant.

J'aurai pu simplement me lancer plus brusquement dans le vif du sujet mais je pense que j'aurai été découragé trop rapidement. Chaque fois que je passais sur une nouvelle page à modifier ou bien à créer, s'il y avait des choses nouvelles, comme une nouvelle librairie, ou bien du code nouveau pour moi, j'allais lire documentation officielle React Native pour prendre connaissance du composant, de ses méthodes. Dans le cas où c'était une librairie externe, je me dirigeais vers son dépôt Github afin de prendre connaissance de cette dernière et des méthodes proposées.

Enfin, dans le cas où je devais comprendre le code existant, je le parcourais afin de comprendre son fonctionnement, si je ne comprenais pas, le lead developer m'aidait à comprendre.

Je faisais de même quand je devais faire du fonctionnel.

Ajout de fonctionnalités

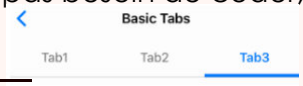
Comme cité précédemment, le header du Workflow a été l'une des mes plus grosses fonctionnalités développée.

Header du Workflow

J'ai du investiguer sur comment développer cette fonctionnalité, car il fallait pouvoir scroller horizontalement, et il fallait que le header affiche les données dynamiquement en fonction du nombre de scènes qui existait (l'API web retourne un JSON contenant des objets pour chaque template. Chaque objet contient les informations nécessaires pour chaque scène) , mais aussi il fallait par dessus tout pouvoir naviguer rapidement entre les pages. En cliquant sur un onglet, il fallait, qu'en fonction de la scène choisi, que la page soit générée dynamiquement.

(METTRE PHOTO ANNEXE)

Lors de notre développement, nous utilisions une librairie nommée Native Base[1] implémentant certains composants comme par exemple des boutons personnalisées, des modales pour choisir une date, etc.. Avant de commencer à développer quoi que ce soit, j'ai beaucoup réfléchi sur la manière de le faire. Après diverses recherches, j'avais pensé qu'utiliser un Tab de Native Base était la meilleure solution, car ce composant avait déjà des méthodes assez utile que je n'avais pas besoin de coder, comme par exemple le scroll



1 https://docs.nativebase.io/Components.html#tabs-def-headref

J'ai fait face à un gros problème, l'affichage de nos scènes étant dynamique (par exemple, nous avons 4 scènes, quand on clique sur la 1 ou la 3, on affiche la même vue mais avec des paramètres différents), on les générait à chaque fois qu'on tapait sur la page demandée, alors que les tabs demandaient du code "en brut" pour pouvoir afficher le contenu. J'avais trouvé une parade au problème. Cependant, après avoir essayé pendant longtemps, je me suis rendu compte que ce n'était pas une bonne solution, car en changeant de tab, certains bugs graphiques survenaient et aussi que ça n'était pas très adapté à notre code. C'était mon premier gros échec, et il fallait absolument que je trouve une autre solution.

C'est alors que j'ai opté pour ma roue de secours qui fût au final une bonne idée. J'ai décidé d'utiliser une FlatList[1]. Je ne l'avais pas choisi en première solution car elle me demandait de coder plus de fonctionnalités et j'avais préféré choisir la facilité. Par exemple, il fallait que je code le changement de couleurs quand on sélectionne un onglet. Il y avait plus de détails techniques à prendre en compte.

Il fallait aussi que j'adapte le changement de navigation avec ce qui avait été fait. Tout d'abord, j'ai commencé par prendre un template dont je connaissais à l'avance le nombre de scènes, afin de rentrer en "dur" le nombre de scènes. Une FlatList prend au minimum en paramètre un JSON d'objet (**data**) et il faut avoir une clé unique par objet (**keyExtractor**). Ensuite, on peut appeler afficher les items un par un (**renderItem**). Par chance, il existe des méthodes pour scroller horizontalement ou verticalement ou bien pour scroller à un item, ou des méthodes pour séparer les composants etc.. (Mais qui reste plus complexe qu'utiliser un tab qui va scroller automatiquement)

```
<FlatList
  data={jsonArr}
  showsHorizontalScrollIndicator={false}
  ref={(ref) => { this.flatListRef = ref; }}
  horizontal={true}
  getItemLayout={this.getItemLayout}
  renderItem={this.props.continue === false ? this.renderBeginList : this.renderList}
  keyExtractor={(item) => "" +  item.key}
  style={{...styles.flatList, backgroundColor: this.props.continue === false ? '#cccccc' : '#ff4459'} }
  extraData={this.state}
  ItemSeparatorComponent={this.renderSeparator}
/>
```

Etant donné que nous avons 4 écrans qui affichent le même header : **WorkflowScreen** (c'est celui qui affiche à une seule vue et qui affiche les pages dynamiquement), **ValidationScreen**, **RenderedScreen**, **BeginTemplateScreen**, il fallait appeler le composant HeaderTemplateWorkflow dans ces 4 vues. J'ai donc géré les paramètres à donner en props (en React Native, les props sont des propriétés que l'on peut passer à un composant par exemple), et j'ai géré comment afficher le header en fonction de ce qui est sélectionné ou pas.

Par la suite, j'ai dû trouvé un moyen d'obtenir le nombre de scènes du template afin de générer le nombre d'onglets suivant le nombre de scènes.

De nombreuses difficultés sont survenues pendant 1 mois, car nous utilisions React Navigation et son système de pile. Cependant, comme l'affichage était dynamique, il fallait le gérer différemment. De plus, nous avions au final un header pour tous mais concrètement comme nous avions 4 vues, nous n'avions pas le même header (même si graphiquement c'était le même, quand on affiche une vue elle va appeler le composant header, donc ce n'est pas le même composant on le génère une nouvelle fois). Ainsi, la navigation fonctionnait bien mais pendant 1 mois même après être passé à autre chose, nous avons fait face à des bugs récurrents.

(ANNEXE PROPS HEADER SUIVANT DIFFERENT ECRAN)

Enfin, il fallait aussi gérer la navigation différemment suivant nos 4 vues. (les scènes sont affichées dynamiquement, comme expliqué précédemment). J'ai dû utiliser les callback de Javascript pour avoir un header avec les mêmes props mais recevant la props changePage, faisant référence à une méthode différentes pour les 4 vues.

(METTRE ANNEXE CODE HEADER)

1 "Une interface performante pour rendre des listes simples et plates, prenant en charge les fonctions les plus pratiques" - https://facebook.github.io/react-native/docs/flatlist

Quand je réfléchis aujourd'hui, je me rends compte qu'il est plus judicieux d'utiliser ce qu'on peut coder soit même plutôt que de prendre une librairie. En effet, avec une librairie, nous pensons que ça va être le chemin de la facilité. Cependant, il faut savoir l'adapter à notre code, tandis que quand on le fait soit même on peut aisément créer ce qu'on veut en l'intégrant facilement.

Néanmoins, dans certains cas, la librairie est inévitable, car elle peut être un gain de temps énorme. Je peux citer par exemple, les barres de progression de notre application qui sont issues d'une librairie et qui auraient été trop longues à développer, ou encore les libraires d'envoi ou de réception de fichier qui sont tout simplement quasiment impossible à développer soit même.

Création de composants

Etant donné que les scènes sont générées dynamiquement, il faut pouvoir générer des composants (date, input, text area etc) dynamiquement. En effet, le serveur envoie un JSON (comme dit précedemment) avec les informations par scène. Par exemple, on peut avoir la scène 1 avec différents champs. Il faut pouvoir, en fonction de ce qu'on reçoit générer la page. A mon arrivée, il y avait déjà une classe javascript **ElementWrapper** qui à l'aide d'un switch permet de choisir le bon composant. Ma mission était de créer des composants en définissant une sorte de norme pour pouvoir créer d'autres composants rapidement dans le futur.

J'ai donc créé : un composant date, un composant input/area (s'il y a plus de 40 caractères, on crée un text area, sinon un input simple), un composant radio, un composant switch, un composant checkbox, un composant slider . Grâce à ces ajouts, on pouvait par la suite, sur l'API Web créer des templates avec ce type de champs. Avant ce n'était pas possible, car l'application ne "connaissait" pas ce genre de champs. Une fonctionnalité avait été codé sur le backoffice de l'API permettant de pouvoir rajouter directement un nouveau type de champ, car l'application mobile reçoit un fichier JSON comme expliqué précédemment, et elle reçoit par exemple pour la scène une, une liste d'objets correspondant à chaque champ contenant le titre du champ, la taille max de caractère etc.. Il faut donc que le backoffice envoie le bon type de champ à afficher (donc le bon JSON), c'est pour cette raison que l'on est obligé de déclarer un nouveau champ dans le backoffice. Par la suite, après la création du champ sur l'API, l'utilisateur qui veut créer un template, va remplir le formulaire et il aura la possibilité de choisir ce dernier.

(METTRE ELEMENTWRAPPER DEUX ENDROITS ET EXEMPLE TEXT INPUT + JSON TEXTE INPUT + SCREEN DIFFERENT EXEMPLE SWITCH CHECK ETC)

Ce genre de fonctionnalités a permis de me montrer qu'en évitant la redondance et en créant une sorte de wrapper, on perd du temps au début, mais par la suite, le gain de temps est notable. C'est le genre de bonne pratique à prendre.

Après avoir créer les champs, il fallait que je gère certaines contraintes de validation concernant les champs.

Contraintes de validation

En créant un template sur l'API Web, on crée les scènes, les descriptions, et on définit aussi les champs que l'on veut dans la scène. Par exemple, on peut créer une scène avec deux champs dates, un input, et un slider. On doit donc renseigner la taille maximale de l'input, les valeurs minimales et maximales du slider. Au niveau de notre application, même si l'utilisateur rentrait plus de caractères que ce qu'il devait y avoir, il pouvait envoyer la vidéo au serveur.

J'ai donc dû ajouter dans Redux une propriété **isValid** qui était à true quand l'utilisateur respectait les contraintes, false sinon. Ainsi, je l'ai géré de tel manière que si l'utilisateur lance une scène, certains champs renvoie **isValid** à true automatiquement car ils n'ont pas de contraintes (ex champ date). S'il n'y a aucune contrainte, alors elle est valide par défaut, exemple, le slider qui est initialisé avec la valeur minimale et qui peut le rester.

On a donc un JSON avec un objet par scène contenant lui aussi un objet par champ. l'**ElementWrapper** vu précédemment va générer les différents champs. Pour chaque champ, on va avoir un **isValid** à true ou false. Pour savoir si une scène est valide, on doit parcourir l'ensemble de ses champs, s'il y a au moins un champ à false, alors la scène n'est pas valide et on lui attribut à **isValid** à false, et true dans le cas contraire.Enfin, dans le **ValidationScreen** on ne propose pas la possibilité d'envoyer le film selon 3 cas, l'utilisateur n'a pas assez de dollys, l'utilisateur n'a pas filmé les scènes, l'utilisateur a oublié de remplir un champ obligatoire.

(METTRE EXEMPLE DE CODE VALIDATION avec le truc redux ET METTRE VALIDATION SCREEN NON VALIDE)

Refactoring et changement du système d'upload et de download

Notre application a besoin d'upload des fichiers une fois les scènes filmées afin de faire le rendu. Nous avions utilisé la libraire React Native File System pour le faire. Cependant, l'écran restait bloqué pendant l'upload, il était impossible de cliquer quelque part. Il fallait attendre la fin de ce dernier pour pouvoir faire quelque. Le souhait de base était de pouvoir laisser tourner l'upload en fond afin de pouvoir faire autre chose. Après avoir cherché d'où venait le problème, j'ai soupçonné que l'upload se lançait sur le même thread que l'interface utilisateur (UI). Malgré mes recherches, je n'ai pas trouvé comment le lancer sur un autre thread. J'ai décidé, sous les conseils du lead developer, de prendre une nouvelle librairie. J'ai pris son concurrent RN Fetch blob. Après avoir lu sa documentation, il est spécifié que ce dernier ne se lançait pas sur le même thread que l'UI.

J'en ai profité donc pour externaliser le code de l'upload qui était placé dans le même code de l'UI afin d'avoir quelque chose de plus propre.

Ainsi, j'ai dû prendre connaissance du fonctionnement d'un système d'upload de fichier. J'ai réussi à le faire et l'upload pouvait donc fonctionner en fond, par conséquent, on pouvait lancer l'upload et changer d'écran sans avoir un processus bloquant.

Etant donné que j'avais réussi à le faire, je me suis occupé du téléchargement des templates de NewFilmScreen en utilisant la même librairie. Il a fallu que je passe par une autre période d'apprentissage des systèmes de téléchargement en lisant la documentation, puis j'ai pu enfin faire ce que j'avais prévu tout en prenant le soin d'externaliser le code dans un autre fichier javascript.

J'ai ainsi appris à envoyer des données et à en recevoir. J'ai aussi appris les bonnes pratiques de React Native en externalisant le code afin d'avoir beaucoup de fichiers mais avec peu de code dedans. Cela permet d'être mieux organiser, car un fichier avec 300 lignes de code est illisible, alors qu'un fichier avec peu de ligne et des références avec des bons noms vers d'autres fichiers est parfois beaucoup plus lisible.

(METTRE PHOTO UPLOAD ET CODE UPLOAD)