



Nom et prénoms : Bourrée François

Classe : HEI52

Domaine : ITI

Type d'expérience : Ingénieur

Sujet du stage : Développement d'applications au sein du D.Lab

Dates du stage : 01-04-2019 au 30-09-2019

Nom de l'entreprise : Deloitte General Services

Nom du tuteur en entreprise : Daniel Brunner

Fonction du tuteur : Technology Innovation Lab Leader

**Rapport Confidentiel : Indiquer
OUI ou NON selon les consignes
données par l'entreprise. En
indiquant OUI, sachez que le
rapport sera renvoyé à l'autorité**

Sommaire

Résumé	3
Remerciements	4
Introduction	5
Première partie : rapport d'activité	7
I) Environnement de travail	7
II) Le suivi de version	8
III) La gestion de projet	10
IV) Contexte des projets	12
Deuxième partie : rapport technique	15
I) Application mobile multi-plaformes de réalité augmentée	15
A) Choix techniques et initialisation du projet	15
B) Structure du projet	19
C) Logique d'affichage de l'application	20
1) Le rendu	20
2) Contrôler l'ordre de rendu	22
D) Les différentes scènes	24
1) Arbook	24
2) Les projectShowcases	25
3) Les projectShowcaseGallery	26
E) Les cibles	28
F) Le « tracking » d'image	30
G) Les données et leur récupération	33
II) Chatbot utilisant l'Intelligence Artificielle	35
A) Choix techniques et initialisation du projet	35
B) Structure du projet	36
C) Les dialogues	37
D) Les ressources	42
1) Les constantes	43
2) Les tutoriels	47
E) Les méthodes	48
F) Interface de LUIS	50
Conclusion	54
Les Annexes	55
Webographie	57

Résumé

Le monde financier connaît une importante transformation numérique dont la réussite sera cruciale pour les entreprises du secteur. C'est dans ce contexte que j'ai effectué mon stage de fin d'études d'ingénieur généraliste au sein de Deloitte Luxembourg. Cette entreprise de conseil est un acteur financier majeur et accueille un département dédié au développement de projets informatiques innovants. C'est donc la double compétence sur le domaine informatique et financier que je cherche pour mon projet professionnel et personnel.

C'est au sein de cette équipe, le D.Lab que j'ai travaillé pendant six mois dans le but d'explorer les possibilités offertes par les nouvelles technologies. C'est ainsi que j'ai pu mener à bien deux projets qui ont pour objectif commun de sensibiliser les membres du monde financier à l'importance de l'informatique pour le cœur de métier de l'entreprise.

Le premier projet permet de mettre en lumière les projets du D.Lab. C'est une application mobile qui a pour particularité d'utiliser la Réalité Augmentée pour immerger le spectateur dans les présentations. Nous avons aussi utilisé cette application pour animer des affiches ou des cartes de visite. Ce projet a été réalisé en React Native pour la partie mobile et avec la librairie Viro pour la partie Réalité Augmentée. Ce projet a été essayé lors de l'inauguration des nouveaux locaux de Deloitte Luxembourg.

Le second projet a pour objectif de soutenir la gestion des tickets du service informatique de l'entreprise. C'est la solution d'un chatbot dirigeant l'utilisateur vers le bon ticket selon sa requête voire vers un tutoriel permettant de la résoudre qui a été retenue. Le chatbot se base sur le Bot Framework de Microsoft pour sa mise en place rapide, couplé à l'Intelligence Artificielle LUIS pour sa capacité à comprendre le langage humain.

Je vais au cours de ce rapport de stage présenter plus en détail l'entreprise, le département que j'ai intégré et comment cela s'inscrit dans mon projet professionnel et personnel. Nous verrons ensuite la méthodologie utilisée pour le développement informatique. Enfin, j'approfondirai les contextes de ces projets, le choix des technologies utilisées ainsi que les fonctionnalités que j'ai développées de manière illustrée.

Remerciements

J'aimerais remercier dans un premier temps Samuel Denys qui m'a permis d'intégrer Deloitte dans les meilleures conditions possibles, que ce soit dans son processus de recrutement, ainsi que pour préparer mon intégration dans l'équipe et ce nouveau pays.

Je tiens ensuite à remercier Daniel Brunner pour m'avoir accueilli au sein du D.Lab et pour m'avoir consacré du temps à me superviser et me former. Je tiens par ailleurs à remercier toute l'équipe du D.Lab pour leur accueil chaleureux et leur aide inconditionnelle lors de mes questionnements. C'est-à-dire : Yang Zhang, Kağan Onbaşıl et Emmanuel Tran.

Enfin, j'adresse mes sincères remerciements à tous ceux qui ont été présents au quotidien, notamment le pôle innovation avec qui nous avons partagé l'open-space soit : Peupedje Mendy, Justine Guyot, Nadia Andersen et Marc Sniukas.

Introduction

J'ai effectué un stage de fin d'études, soit 6 mois, au sein de Deloitte Luxembourg en tant que développeur informatique dans l'équipe du D.Lab. Deloitte Touche Tohmatsu Limited ou Deloitte sous son nom le plus commun est une des « Big Four », les quatre plus importants cabinets d'audit et de conseil au monde.

Deloitte existe depuis 1845, ce qui en fait le plus ancien et le plus important des Big Four avec un effectif mondial de 286 000 employés. Au Luxembourg la société est créée en 1950 en portant le nom de « l'Agence » et ne rejoint Deloitte Touch Tohmatsu via l'absorption de son groupe Ross Luxembourg qu'en 2008. Deloitte délivre des services d'audit, de taxe, de risque d'entreprise et de conseil financier. En 2018, l'entreprise a engendré un record de 43,2 milliards de dollars de chiffre d'affaires.

J'ai choisi d'intégrer cette entreprise et plus particulièrement ce département pour plusieurs raisons. La première est que je souhaitais effectuer mon stage de fin d'études au sein d'une entreprise du monde financier. J'ai en effet développé un intérêt pour le monde de la finance sur mon temps personnel. Ayant hésité à changer de cursus d'ITI à Banque, Finance et Assurances au sein de mon école, j'ai décidé de prolonger mon cursus en informatique et de me former à côté au domaine financier. J'ai pour projet professionnel de travailler dans l'informatique financière. Ce stage représente une occasion de m'introduire dans le monde de la finance et de me familiariser avec ses acteurs.

La seconde raison qui m'a poussé à intégrer Deloitte est que je cherchais à diversifier mes missions dans l'optique de progresser sur des domaines variés. C'est donc vers une entreprise de conseil que je me suis tourné pour m'assurer du renouvellement des missions et des technologies qui rythmeront mon stage. C'est un moyen pertinent lorsque l'on débute de ne pas s'enfermer dans une technologie ou un domaine particulier. Il est bon de savoir que le monde financier opère une transformation numérique importante et en profondeur.

De cette manière, il n'est pas impossible d'effectuer un stage dont l'objectif est simplement de maintenir un programme obsolète. Ainsi, il est important de pratiquer un apprentissage continu, notamment en informatique, pour rester à jour sur les langages informatiques porteurs. C'est la raison pour laquelle j'ai intégré un département axé sur l'innovation technologique.

Le dernier point est j'avais pour axe d'amélioration la maîtrise technique dans le développement informatique. Étant donné que j'ai suivi une formation d'école généraliste, je considérais devoir m'améliorer pour acquérir les compétences techniques que les étudiants sortants d'écoles plus spécialisées. J'ai pu ainsi travailler exclusivement sur le développement informatique durant ce stage parmi d'autres développeurs. Je pensais apprendre de la manière la plus effective en sortant de ma zone de confort.

Le stage a été composé de deux projets majeurs : le premier, qui dura trois mois est une application mobile tandis que le second est un chatbot interne. Dans un premier temps j'ai construit la base de l'application mobile, puis j'ai construit des jeux de données pour un projet annexe. J'ai par la suite poursuivi et achevé

le développement de l'application mobile. Une fois ce projet achevé, j'ai travaillé sur le second projet de chatbot, pendant environ deux mois, jusqu'à la fin de mon stage.

Je vais au cours de ce rapport, présenter les différents projets en les mettant dans leurs contextes. Nous verrons comment j'ai essayé de développer une application qui a pour objectif de rendre accessible les projets informatiques dans une entreprise dont ce n'est pas le cœur d'activité. Enfin, nous nous intéresserons à la solution informatique qu'est le chatbot et son fonctionnement dans le but de faciliter la communication interne.

Première partie : rapport d'activité

I) Environnement de travail

L'insertion dans une grande structure demande généralement un peu de temps. En effet, il est nécessaire de s'approprier la culture de l'entreprise et ses règles intrinsèques. C'est pourquoi, il a été organisé une demi-journée d'accueil pour intégrer les nouveaux arrivants. Par la suite, on nous assigne un D.Buddy, une personne qui nous épauler pour le début de notre intégration. Il existe un certain nombre de certificats à valider dans un temps imparti afin de respecter la politique qualité de l'entreprise. Cela peut comprendre des formations sur le blanchiment d'argent, une sensibilisation au phishing ou un rappel de l'importance de l'éthique. L'ensemble des formations à valider et des certificats à obtenir correspond environ à la première semaine au sein de l'entreprise.

Dans un monde où la technologie est omniprésente et l'innovation la clé du succès, Deloitte Luxembourg a mis en place le D.Lab. C'est une équipe d'innovation avec comme mission d'explorer des technologies innovantes et de développer des proof-of-concepts (PoC) pour mettre en lumière des idées disruptives à l'aide de techniques de développement logiciels de pointe.

L'équipe du D.Lab est composée de développeurs, de designers et d'experts, collaborant étroitement avec d'autres initiatives à toutes les échelles, de local à global. Au cours de mon stage, l'équipe du D.Lab se composait de 5 personnes 2 développeurs en stage dont moi, 2 développeurs en CDI, un junior, un senior et manager d'équipe. Son objectif est d'explorer des technologies innovantes comme la blockchain, l'automatisation de processus robotique, l'internet des objets, l'intelligence artificielle et de les appliquer à différentes industries. Le D.Lab implémente aussi et teste de nouvelles méthodes de développement logiciel comme l'Agile, le DevOps ou le développement piloté par les tests (TDD).

Le D.Lab permet aussi bien l'application rapide et itérative de nouvelles technologies que l'exploration de leurs impacts dans les affaires et les modèles d'exploitations. Il apporte un processus qui a fait ses preuves et une capacité à expérimenter dans de courts délais aux moyens de démonstrations.

Le but du D.Lab est de créer des PoC (Proofs of Concept), des projets rapides dont le but est de mettre en avant une fonctionnalité d'une technologie de pointe. Les PoC sont caractérisés par des temps de développement courts, des sprints rapides (méthodes agile) et une priorisation du résultat. On peut décider par la suite de continuer leur développement en utilisant notre portefeuille de kits de développement préconçus.

Il est intéressant de travailler sur ce type de projets car on travaille nécessairement sur de nombreux projets différents, ce qui permet d'explorer beaucoup de technologies différentes. De plus, étant donné que l'on doit obtenir rapidement des résultats à présenter (j'avais par exemple une démonstration chaque semaine de mon avancement), les projets sont concrets et rapidement gratifiants. Un autre point intéressant à

souligner est que les projets sont généralement gérés par petites équipes. J'ai pour ma part travaillé seul la majorité du temps, ce qui oblige à trouver des solutions par soi-même. J'ai pu cependant m'appuyer sur les connaissances de mes collègues quand je ne parvenais pas à trouver une solution par mes propres moyens.

Ces projets sont présentés aux responsables et peuvent alors être validés en véritables projets avec des ressources accrues en temps, argent et effectifs (Figure 1). Les projets peuvent avoir soit des clients internes soit des clients externes. Dans la très vaste majorité des cas, les projets sont d'abord internes et parfois s'externalisent. Dans le cas échéant, les projets sont alors mis de côté et sauvegardés pour servir de support au cas où des projets techniquement proches seraient mis en développement.

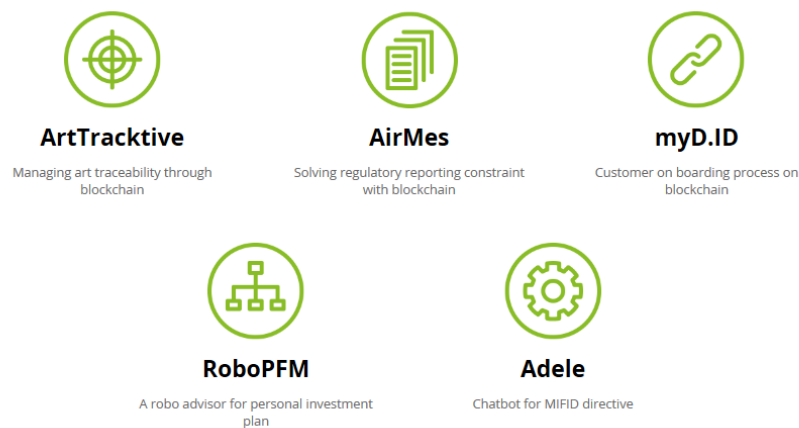



Figure 1 : Les projets réalisés à mon arrivée par le D.Lab ayant dépassés le stade de PoC

II) Le suivi de version

Nous avons effectué de l'intégration continue en utilisant un logiciel de version nommé GitLab, logiciel libre basé sur git. Il permet de suivre l'évolution d'un fichier texte ligne de code par ligne de code. Ce logiciel permet à la fois de suivre l'évolution du code source mais aussi de travailler à plusieurs. Il est ainsi presque indispensable d'utiliser un logiciel de gestion de versions. L'ensemble des sauvegardes d'un projet est appelé « dépôt » ou repository (Figure 2).

D.Lab > AR Project Showcase > Details



AR Project Showcase

Project ID: 51 | [Leave project](#)

Star

0

Fork

0

Clone

[Add license](#)
[24 Commits](#)
[1 Branch](#)
[0 Tags](#)
[1.1 GB Files](#)

Figure 2 : Repository d'un projet suivi sur GitLab

Lors de la modification d'un fichier de code puis de la sauvegarde de cette modification, une nouvelle version du projet est créée. L'envoi de cette nouvelle version au repository s'appelle un « commit » (Figure 3).

changed reload button

parent `bb1fcb7a` `master`

No related merge requests found

Changes **4**

Showing **4 changed files** with **40 additions** and **18 deletions**

Hide whitespace changes
Inline
Side-by-side

Figure 3 : Commit décrivant la modification du code

On peut voir ici les ajouts en vert et les suppressions en rouge lors d'une modification d'un bouton (Figure 4). Ce suivi des modifications trouve une importance croissante avec la taille d'un projet. Cela permet de connaître qui a modifié le code, quand et pour quelles raisons.

```

48 -         onPress={ () => this.resetAR()}
49 -         underlayColor={'#00000000'} >
50 -         <Image style={styles.buttonImage} source={require("../ressources/icons/refresh.png")} />
51 -         </TouchableHighlight>
52 -     </View>
45 +
46 +         {this.displayViroAR()}
47 +         {this.displayUI()}
53 48     </View>
54 49     )
55 50     }
...   @@ -69,6 +64,21 @@ class ReactNativeARKit extends Component {
69 64     }
70 65     }
71 66
67 +     displayUI = () => {
68 +         return (
69 +             <View style={styles.buttonView}>
70 +                 <TouchableHighlight
71 +                     onPress={ () => this.resetAR()}

```

Figure 4 : Ajouts et suppressions de code lors d'un commit

Il m'est arrivé lors de mon stage de consulter les commits effectués par des collègues, voire de projets sur internet, pour comprendre comment ils ont réussi à résoudre un problème. Il est donc

essentiel de le faire régulièrement et de les documenter convenablement pour un suivi efficace des projets au travers de l'historique des commits effectués (Figure 5).

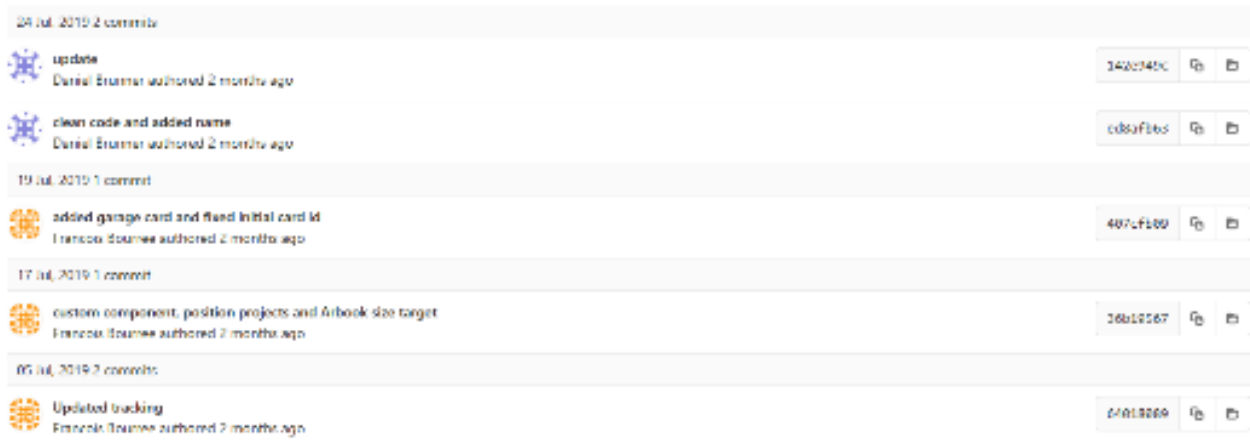


Figure 5 : Historique des commits effectués

III) La gestion de projet

En parallèle pour la gestion de projet, nous avons utilisé un système développé par Atlassian : Jira. Nous nous sommes basés sur un schéma Scrum en termes d'organisation de développement. L'infrastructure de développement s'appuie sur le découpage d'un projet en boîtes de temps en « sprints ». Ces derniers correspondent à des périodes de temps dédiées au développement de fonctionnalités. On commence par une estimation puis une planification des tâches à effectuer. Un sprint s'achevant par une démonstration.

Parallèlement, nous avons découpé le projet en « récit d'utilisateur » ou « user story », des descriptions simples d'un besoin. Voici un exemple pour le projet de réalité augmentée (Figure 6).

DLAB-25 / DLAB-25
As a user I am able to use my IOS device to display AR

Edit Comment Assign More To Do In Progress Workflow

Details
Type: Story Status: IN PROGRESS
Priority: Medium (View Workflow)
Labels: None Resolution: Unresolved
Epic Link: Showcase app

Description
Click to add description

Attachments
Drop files to attach, or browse.

Sub-Tasks

Order	Task	Status	Assignee
1.	As a developer, I can fix a 3D object on a surface/image	TO DO	François Bourrée
2.	As a developer, I create another project using react-native-arkit library	REVIEW	François Bourrée
3.	As a user I am able to use my camera with the app	REVIEW	François Bourrée
4.	As a user I able to see a 3D object in AR	REVIEW	François Bourrée
5.	As a user I am able to trigger the app with a image	REVIEW	François Bourrée
6.	As a developer I can change the font of the 3D text to make it easily readable	REVIEW	François Bourrée
7.	As a developer I can display a 3D graph	REVIEW	François Bourrée
8.	As a developer, I display a video on the surface of a 3D object/surface	TO DO	François Bourrée
9.	As a developer I create a custom component "Presentation"	TO DO	François Bourrée
10.	As a developer I import the "Presentation " custom component from another file using the state	REVIEW	François Bourrée

Figure 6 : Récit d'utilisateur, lui-même découpé en sous-tâches

On peut voir qu'une tâche peut être décomposée en sous-tâche. Étant donné que le but de l'équipe est de délivrer des PoC, les sprints sont en général assez courts. Les sprints ou « itérations » sont des intervalles de temps courts (d'un mois maximum) pendant lequel les développeurs conçoivent, réalisent et testent les nouvelles fonctionnalités (Figure 7). La durée d'un sprint lors de mon stage pour mes projets était d'une semaine. Les objectifs sont fixés le lundi lors de la réunion matinale et les avancements sont présentés en fin de semaine, avec un échange presque journalier avec le manager (cela est permis grâce à la taille restreinte de l'effectif).

DLAB Sprint 1 5 issues ACTIVE
18/Mar/19 2:14 PM • 22/Mar/19 3:00 PM

Task	Assignee	Estimate
DLAB-1 Install an Active Directory On AWS	DevOps platform	40
DLAB-22 As a user I am able to see project list mocked in my app	Showcase app	4
DLAB-23 As a user I am able to log in to mobile app	Showcase app	4
DLAB-24 As a developer I defined app skeleton and navigation	Showcase app	4
DLAB-8 Add HTTPS on Nginx	DevOps platform	20

Figure 7 : Sprint de la semaine 1

Ici le premier sprint d'une semaine voit la réalisation de plusieurs récits d'utilisateurs. Pour qu'une tâche soit terminée, elle doit respecter un « workflow » ou « flux de travaux ». C'est une suite d'étapes à effectuer pour accomplir une tâche précise (Figure 8). Une fois défini, le flux de travail de l'équipe de développement

est facilité par son auto-organisation. De cette manière l'indépendance des développeurs et mise en avant tout en reculant le rôle du gestionnaire de projet.

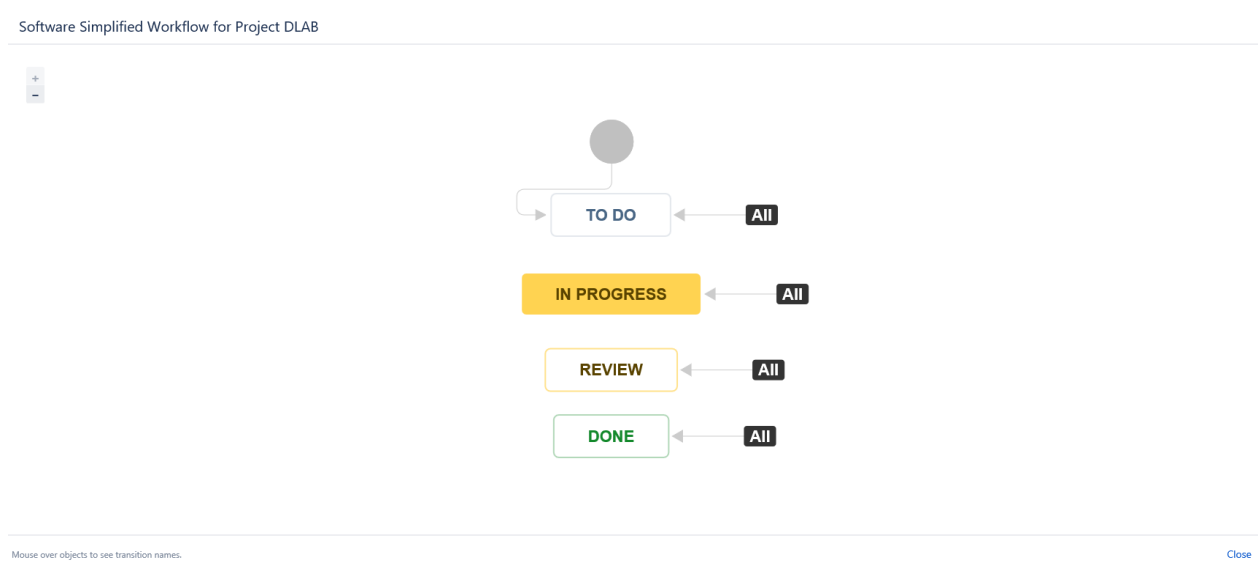


Figure 8 : Workflow utilisé pour ce projet

On définit d'abord les tâches à effectuer, en effectuant au préalable un travail de priorisation. Une fois le développement commencé, on peut modifier le statut de la tâche sur « en progrès ». Une fois que l'on estime la tâche terminée, il convient de le faire valider par un responsable avant de l'indiquer comme fini. Voici un exemple de tâche en attente de validation (Figure 9). On peut y voir qui est la personne développant la fonctionnalité et qui est responsable de sa vérification.

Figure 9 : Tâche en attente de validation

IV) Contexte des projets

Au moment de mon arrivée les locaux étaient situés rue de Neudorf à Luxembourg. Au début du mois de juin, nous avons déménagé à Cloche d'or, dans un bâtiment en cours de finition. Cela est dû à l'accroissement des effectifs et est l'aboutissement d'un projet de nombreuses années. Ce déménagement nous a également permis de disposer d'espaces supplémentaires pour présenter les innovations de l'entreprise à ses clients. Sensibiliser les acteurs du monde financier à l'importance des processus informatiques performants a en effet été une partie non négligeable du stage. Beaucoup de salariés ne se

sentent pas réellement concernés par la transformation de l'entreprise et non pas forcément conscience à la fois de l'impact mais aussi de l'intérêt pour leur métier. C'est pourquoi se faire connaître, même en étant un département relativement réduit constitue une tâche inhérente aux membres du D.Lab. On a donc saisi l'occasion de mettre en avant le travail effectué par le groupe à la fois aux membres internes de l'entreprise mais aussi pour des intervenants externes qui pourraient être intéressés par les travaux.

C'est dans cette optique qu'il a été décidé de créer une application permettant de mettre en lumière les projets du département. Le choix s'est porté sur une application mobile, pour son universalité et pratique. Afin de rendre la présentation des projets plus attractive et impactante, l'application utilise la Réalité Augmentée (ou AR pour Augmented Reality), qui consiste à afficher au travers de la caméra de son téléphone des éléments 3D en plus de la vue réelle de l'objectif. L'un des exemples les plus connus de l'utilisation de cette technologie est le jeu Pokémon go.

Dans un premier temps il a été question de faire un benchmark des start-ups luxembourgeoises capables de répondre à cette demande. À la vue du marché existant proposant peu de solutions adaptées et devant le souhait de développer l'application en interne, il a finalement été décidé de m'en confier le développement.

Cette partie sur l'AR nécessitant des ordinateurs Apple « Mac », il a fallu attendre que le matériel commandé arrive et qu'il nous soit alloué. En effet, ce matériel ne nous était pas directement destiné. C'est pourquoi je suis allé renforcer l'équipe en charge de la présentation de l'algorithme de Machine Learning (ML).

J'ai par la suite été chargé de la création de datasets pour entrainer un algorithme de Machine Learning chargé de faire une démonstration à l'ICTSpring du 21 Mai, ensemble de conférences sur la technologie auxquelles Deloitte participe.

J'ai ensuite repris le développement de l'application mais cette fois-ci la partie réalité augmentée. J'ai créé un projet à part afin de tester les fonctionnalités et de pouvoir les implémenter par la suite au projet. Étant donné que l'ICTSpring nécessitait la présence de tous les macs, je n'ai pas pu continuer à travailler sur mon projet pendant les deux jours de l'évènement. Il faut ajouter à cela que, étant donné que les ordinateurs n'appartiennent pas à l'équipe, ils ne sont disponibles que sur une plage horaire prédéfinie dans la journée, de 9 h à 17 h 30. C'est pourquoi il est particulièrement important d'optimiser son temps de travail disponible et de s'organiser pour faire les tâches annexes en dehors de ces horaires. Ces tâches peuvent être nombreuses et diverses. Comme consulter ses mails, organiser ses sprints sur JIRA, rédiger le readme du projet, rechercher les solutions pour les erreurs dans la programmation ou les solutions envisageables.

Il est en général assez difficile d'estimer la durée nécessaire pour le développement d'une fonctionnalité, surtout quand on est junior ou que l'on découvre une technologie. On a en général tendance à sous-estimer la durée nécessaire étant donné qu'il est possible de suivre des projets existants lors des commencements.

Mais une fois que des fonctionnalités spécifiques à notre projet doivent être développées, il faut souvent chercher par soi-même des solutions et le temps nécessaire associé accroit.

N'étant pas familier à la technologie employée pour l'application mobile il m'est permis de solliciter l'assistance d'un de mes collègues ayant déjà travaillé avec cette technologie. Cependant, dû aux sprints de courtes durées, j'essaye de prendre soin de ne pas déranger quelqu'un lors de périodes critiques.

Le dernier projet sur lequel j'ai travaillé est un chatbot interne. Le but de ce dernier est de désengorger le service informatique de l'entreprise en permettant d'assister les gens directement et si besoin en redirigeant les collaborateurs vers le ticket/service adapté à leur requête.

Un chatbot ou « agent conversationnel » est un logiciel qui peut interagir ou « chatter » avec un utilisateur humain dans un langage naturel tel que l'anglais. Les chatbots ont déjà fait leurs preuves comme agents de service client pour le support technique, à la fois pour l'entreprise qui le déploie que pour les utilisateurs, notamment internes. C'est une opportunité de réduire à la fois les coûts mais aussi d'optimiser voire d'épauler un service technique saturé.

La transformation numérique est une étape par laquelle toutes les entreprises doivent passer au cours de leur évolution et un service informatique performant permet d'alléger les lourds processus propres aux structures importantes dont Deloitte fait partie.

Historiquement, le premier chatbot a été créé en 1966 par le MIT. À l'origine, le chatbot fonctionne en s'appuyant sur une base de données de questions-réponses en repérant des mots clés dans la conversation. Mais, portés par les progrès de l'Intelligence Artificielle et plus précisément du Machine Learning (Apprentissage Automatique ou Apprentissage Machine), les chatbots sont désormais bien plus évolués car dotés d'un système d'analyse du langage naturel et s'améliorent au cours de leurs utilisations.

Le langage naturel est une langue « normale » parlée par un être humain, comme le français ou l'anglais, s'opposant au langage formel tel que le langage informatique. La difficulté qui réside dans ce type de langage est qu'un même mot peut avoir plusieurs sens comme « terrible » ou « mortel ». Ce sont toutes ces subtilités et ambiguïtés que l'IA (Intelligence Artificielle) tente de cerner.

Conçu pour identifier les informations importantes au sein des conversations, LUIS (Language Understanding Intelligent Service) interprète les objectifs des utilisateurs (intentions) et extrait les informations utiles des phrases (entités) afin de créer un modèle de langage nuancé et de qualité. LUIS s'intègre parfaitement à Azure Bot Service, un ensemble d'outils pour créer, tester et déployer facilement un bot sophistiqué.

Deuxième partie : rapport technique

I) Application mobile multi-plaformes de réalité augmentée

A) Choix techniques et initialisation du projet



Figure 10 : Projet final utilisant la réalité augmentée pour animer des présentations

Étant donné que l'application devait être le plus accessible possible on a choisi de développer l'application en React Native, afin de permettre de délivrer un produit cross-platform (IOS + Android) (Webographie 1).

On m'a confié seul le développement de ce projet (Figure 10). Dans un premier temps, il a fallu que je construise la base de l'application mobile et ainsi me familiariser avec cette technologie. On a choisi de développer l'application en cross-platform (IOS et Android) car le but est de présenter un produit donc le logiciel doit être accessible au plus grand nombre. La technologie choisie a été le React Native pour 2 raisons. La première est qu'il permet le développement cross platform et que la technologie est utilisée par un grand nombre de développeurs. Cela assure que cette dernière est bien maintenue (mise à jour régulière) et qu'elle ne sera pas obsolète sous peu.

On trouve 2 types d'applications mobiles : les applications natives et les applications cross-platforms. Les applications natives sont des applications créées spécifiquement pour un OS (système d'exploitation). Les OS pour mobiles les plus connus sont iOS et Android. Pour développer une application native fonctionnant sur les deux, il faudra développer deux applications complètement différentes.

La première pour iOS sera en Swift ou Objective-C pour iOS et la deuxième en Kotlin ou Java. Ainsi, pour une solution native, chaque support aura son propre langage de programmation. Étant donné les délais assez courts avec lequel le département travaille, il n'était pas envisageable d'apprendre des langages supplémentaires et des architectures différentes. De plus si jamais un projet est validé, il faut recommencer à zéro et développer l'autre projet avant de pouvoir continuer le développement. On peut donc affirmer que cette proposition ne s'inscrit pas dans les objectifs du D.Lab.

Au contraire des applications natives, les applications cross-plaforms ne demandent à être développées qu'une seule fois pour être compatibles sur iOS et Android. On passe pour cela par des frameworks (ensembles de composants logiciels).

Réputées pour être moins performantes et moins fluides que les applications natives, il est tout de même nécessaire de faire des ajustements pour chaque plateforme. Pour ma part j'ai travaillé pour iOS tandis qu'un collègue a continué d'adapter le projet pour Android. Cependant, étant donné que l'objectif n'est pas de développer l'application la plus fluide et la plus performante mais bien de créer une maquette rapidement, il semble que cette solution soit la plus adaptée.

Un des points forts qui nous a poussé à choisir React Native est que ce framework utilise des composants mobiles natifs, c'est-à-dire que lorsque l'on crée un composant avec React Native, comme un bouton, du texte ou un chargement, React Native convertit cet élément en son équivalent iOS ou Android. De cette manière l'application gagne en performance, en fluidité et ressemblance à une application mobile.

Les autres points forts de React Native est que c'est gratuit. Ne sachant pas si le projet serait poursuivi, ce point tient donc une assez grande importance. De plus, ce framework est Open Source, en d'autres termes, il nous est permis de le modifier et de le redistribuer. Cette accessibilité a permis en outre la création d'une communauté et assure son accroissement. Il est à noter que le framework est porté par Facebook, l'application mobile est d'ailleurs développée en React Native, ce qui assure son suivi.

Enfin, React Native permet de tester son application immédiatement, point très important si l'on garde en tête l'objectif de développement de PoCs rapides. L'un des problèmes majeurs dans le développement d'application mobile native est nécessaire de « build » (compiler en français) pour recharger donc voir ses modifications. La compilation est le processus qui transforme le code dans le langage que comprend la machine (ici le téléphone). Or ce temps de compilation peut être important et doit se voir le plus réduit possible dans la création de PoCs. La dernière force de React Native est d'être écrit en Javascript, langage qui n'a pas besoin d'être compilé pour être exécuté. Etant capable d'afficher le rendu immédiatement, on effectue un gain de temps important tout au long du projet.

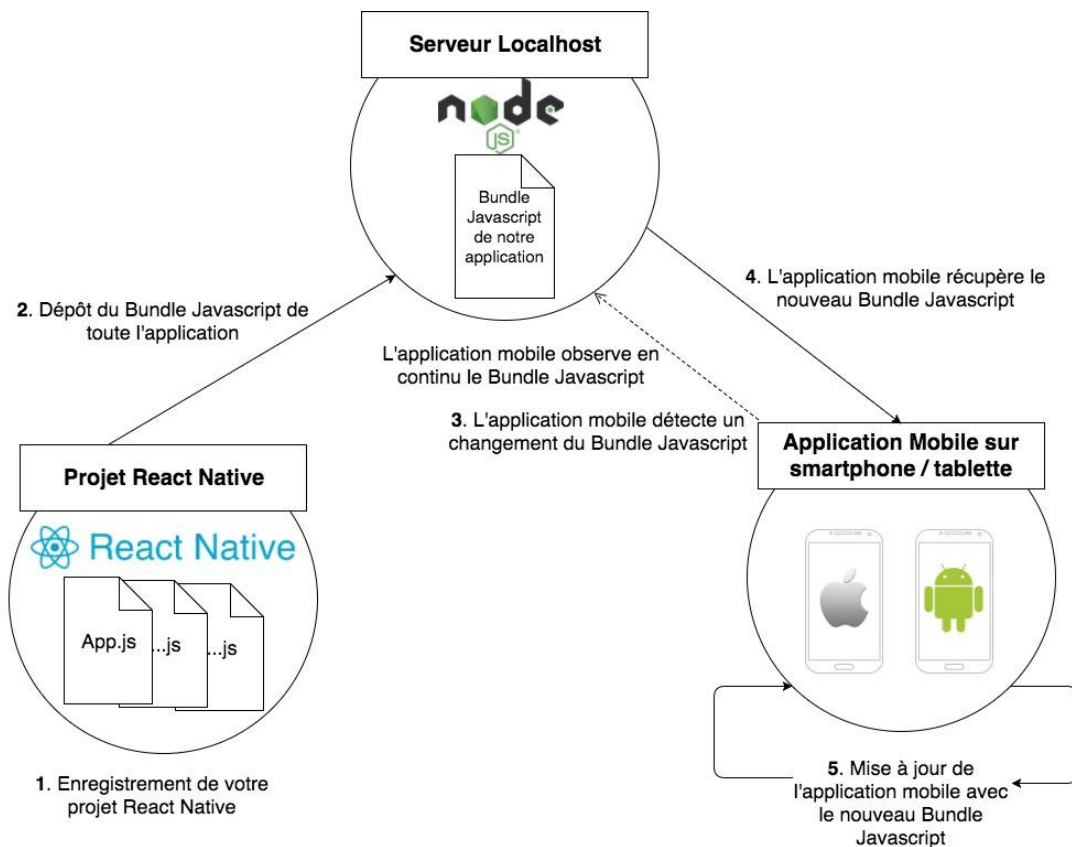


Figure 11 : Schéma du rechargement d'une application React Native

Lors de l'enregistrement du projet, React Native génère un fichier Javascript de notre application (un bundle) et l'envoie au serveur Node.JS hébergé localement (Webographie 2). L'application sur le smartphone observe ce bundle et se met à jour de manière automatique lorsqu'il change. L'exécution du code Javascript est presque instantanée (Figure 11).

La base de l'application était donc composée d'une page d'accueil (une page ou un écran correspond à une « vue »). On trouvait ensuite une liste des projets existants que le département avait antérieurement développés. Il a fallu mettre en place une logique de navigation entre les vues. J'ai utilisé pour se faire une librairie : React Navigation. J'ai utilisé le type de navigation « *StackNavigator* ». C'est la navigation la plus basique où l'on pousse une vue sur iOS et qui permet de présenter une vue sur Android. Le *StackNavigator* gère une pile de vues qui augmente lorsque vous naviguez vers une nouvelle vue et diminue lorsque vous revenez en arrière.

Il existe actuellement 2 librairies performantes pour la réalité augmentée : « *Arkit* » (Apple) et « *ARCore* » (Google). Cependant React Native ne supporte pour le moment que la solution d'Apple, c'est-à-dire *Arkit*.

Dans un premier temps, j'ai créé une application exempte de tout AR. J'ai choisi comme solution une *Create-React-Native-App* ou une (CNRA). Une CNRA est une solution créée par la communauté React Native pour se lancer rapidement dans les développements ; elle nécessite peu de configuration. Il faut en

effet garder en tête que la durée des sprints est très courte et que l'on développe des POC. J'ai donc utilisé Node.js. Grâce à Node.js j'ai pu installer Expo CLI, l'interface de commande d'expo, qui génère un environnement de développement en local sur la machine (Webographie 3). Dans un souci de gain de temps j'ai donc préféré la solution Expo, outil de développement permettant de créer des applications React Native à la création d'un projet avec du code natif. La seconde raison fut l'absence de mac nécessaire au développement iOS. Il est en effet nécessaire d'utiliser XCode pour build le projet.

Dans ce projet, j'ai construit les différents screens et les customs components, ainsi que la navigation de l'app. Une fois le squelette de base de l'application créée, j'ai initié un autre projet indépendant, afin de pouvoir y faire des tests d'AR. J'ai décidé, malgré le fait que cette fois-ci je pouvais développer sur un mac de conserver un développement Expo pour conserver une compatibilité entre les deux projets et facilement intégrer l'AR au projet initial. De plus, le « hot reload », c'est-à-dire la régénération du build lors de la sauvegarde de modifications permet d'optimiser le temps de développement. Cependant, alors même qu'Expo est très bien suivi par la communauté, la librairie AR d'Expo ne bénéficie pas d'un tel suivi et la documentation s'en trouve relativement assez limitée (Webographie 4).

C'est pourquoi, après avoir réussi à configurer la 3D de l'application, en m'appuyant sur la librairie créatrice de contenu 3D expo-THREE, et en utilisant la base de l'AR, j'ai éprouvé plus de difficulté à développer des fonctionnalités annexes plus avancées. Après avoir essayé de figurer par moi-même la solution à adopter et demandé à un collègue plus expérimenté de m'aider, j'ai décidé d'explorer d'autres librairies, suite de ces tentatives non concluantes.

Je me suis donc tourné vers une librairie plus populaire et mieux documentée, React-Native-Arkit (Webographie 5). Cette dernière m'a aussi permis de voir des premiers résultats satisfaisants mais je rencontrais des problèmes de compatibilité avec la structure d'expo cli, notamment dans l'utilisation du debugger. Etant donné qu'un premier PoC avait pu être conçu pour la démonstration, je disposais d'un temps accru pour me permettre de tester un nouveau projet cette fois-ci avec du code natif, sachant que c'était une possibilité grâce à la disposition d'un mac.

Cependant, les outils développés par la communauté de React-Native-Arkit ne suffisaient pas à développer des fonctionnalités avancées. J'ai du une nouvelle fois chercher une nouvelle librairie offrant plus de possibilités. Je me suis donc tourné vers Viro (Webographie 6). Ce dernier, moins suivi, est cependant compatible avec Apple et Google. De plus, Viro possède une librairie bien documentée, complète tout en restant intuitive.

Pour ce projet, j'ai installé Homebrew, un manager de package pour mac, ainsi que watchman, un outil développé par Facebook permettant de suivre les changements dans le filesystem, dont l'utilisation est recommandée pour améliorer les performances lors du développement. Cette fois-ci on installe l'interface de ligne de commande de React Native 'React-Native-cli'.

Il a fallu par la suite configurer XCode pour permettre le build du projet. Il faut en effet un compte développeur chez Apple et l'ajout de la librairie React-Native-Arkit doit se faire manuellement du fait de problèmes récents.

B) Structure du projet

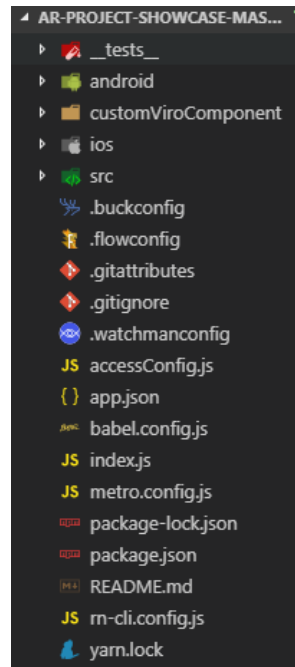


Figure 12 : Structure du projet de l'application mobile

Voici la structure générale du projet (Figure 12). Tout le code propre au projet est regroupé dans le dossier « src » signifiant « Source Code File Computing », c'est-à-dire le code source de l'application.

Android et ios correspondent aux fichiers relatifs à la configuration d'Android Studio et XCode.

XCode est l'IDE (Integrated Development Environment) développé par Apple qui va permettre de build le projet.

Les autres fichiers sont des fichiers de configuration, mis-à-part « README.md » qui est un fichier texte servant à présenter le projet et guider les utilisateurs dans sa mise en place. En voici un court extrait (Annexe 1) puis son rendu (Annexe 2).

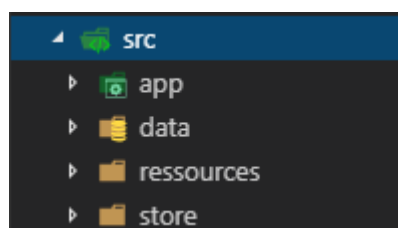


Figure 13 : Structure du dossier src

Je ne vais approfondir que la partie « src » (Figure 13) car c'est dans ce dossier que réside le code de l'application :

- App : logique de l'application (majeure partie du code)
- Data : ensemble des fichiers contenant de la donnée
- Ressources ou assets : dossier stockant les images/vidéos...
- Store : récupération des informations sur AWS

C) Logique d'affichage de l'application

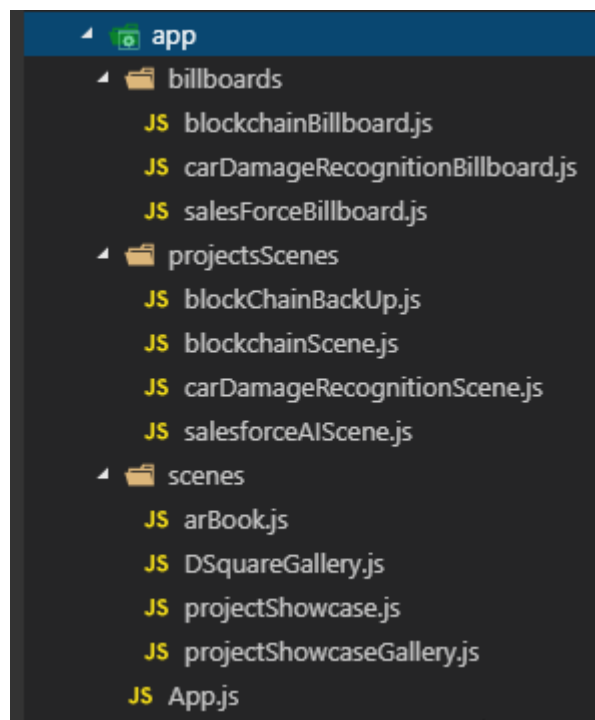


Figure 14 : Structure du dossier app

1) Le rendu

Le fichier App.js est la porte d'entrée de l'application (Figure 14). C'est le premier fichier appelé lors du démarrage.

App.js

```
render() {
  return (
    <View style={styles.mainContainer}>
      {this.displayViroAR()}
      {this.displayUI()}
      {this.displayFocusWarning()}
      {this.displayNetworkErrorMessage()}
      {this.displayToast()}
    </View>
  );
}

displayViroAR = () => {
  if (this.state.loaded) {
    return (
      <ViroARSceneNavigator
        key={this.state.key}
        numberOfTrackedImages={1}
        autofocus={true}
        initialScene={{
          scene: require('./scenes/arBook') // projectShowcase arBook projectShowcaseGallery
        }}
        apiKey={config.Viro.apiKey}
      />
    );
  }
}
```

`render()` est la méthode la plus utilisée pour tout composant alimenté par React Native qui retourne un JSX. Lorsque le fichier composant est appelé, il appelle par défaut la méthode `render()`.

On peut voir que la méthode `render` retourne 5 méthodes qui ont pour objectif d'afficher des composants aux tâches propres.

- **displayViroAR()** : la méthode principale qui retourne l'AR
- **displayUI()** : affiche des éléments 2D sur l'écran facilitant l'usage (bouton reload)
- **displayFocusWarning()** : affiche un warning si on vise plusieurs images censées déclencher l'apparition d'AR

- **displayNetworkErrorMessage()** : affiche un message d'erreur en cas de problème pour récupérer les données d'AWS
- **displayToast()** : permet d'afficher du texte complémentaire

On peut voir que le « loaded » initialisé dans le state permet de conditionner l'affichage de l'AR. Dans ces conditions, on n'affiche les ressources uniquement lorsque ces dernières sont effectivement chargées.

La propriété « numberOfTrackedImages » correspond au nombre d'images censées déclencher l'AR que l'application doit tracker simultanément. Plus le nombre est élevé plus l'application consomme de ressources. Un des problèmes que j'ai rencontrés au cours du développement est que le nombre d'images suivies était trop élevé ce qui entraîne en une baisse importante des images par secondes affichées. Cela donne un effet saccadé a l'application.

La prop *initialScene* permet de définir quelle scène sera affichée en premier par le composant *ViroARSceneNavigator*, ici en l'occurrence *arBook.js*

2) Contrôler l'ordre de rendu

App.js

```
class ReactNativeARKit extends Component {

  constructor(props) {
    super(props);
    this.state = {
      reset: false,
      key: 0,
      loaded: false,
      displayNetworkError: false,
      displayToast: false
    }
  }
}
```

Le constructeur de la classe est une méthode spéciale qui permet de créer et d'initialiser les objets créés avec une classe. On initialise ici aussi le state.

Il existe 2 types de données que contrôle le composant : les « props » et le « state ». Les props sont fixés par le composant parent et sont fixes durant la vie du composant. Pour les données qui sont amenées à changer, nous utilisons le state. Ce dernier s'initialise dans le constructeur et se verra être changé en appelant un « setState ».

Ici on peut voir que l'on initialise une valeur booléenne « loaded (chargé) » a « faux ».

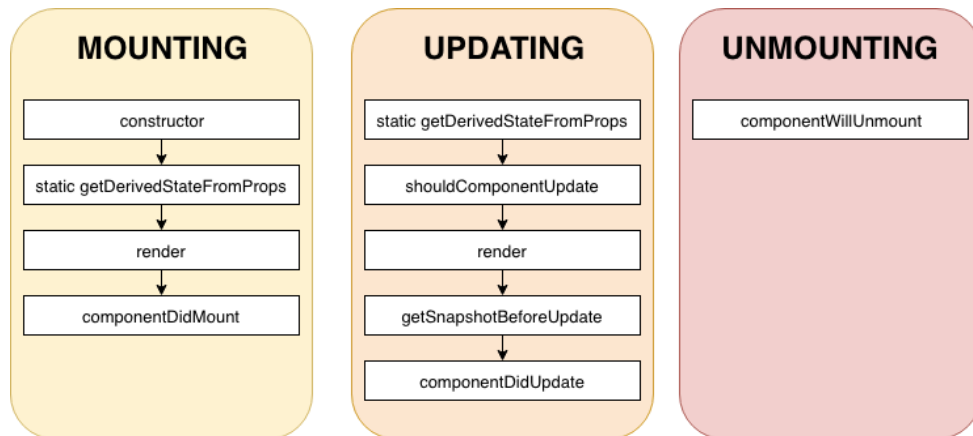


Figure 15 : Cycle de vie d'un composant

- Le **mounting** d'un composant (monter un composant en français) correspond à l'affichage d'un composant à l'écran. Ce cycle de vie englobe l'initialisation du composant jusqu'à son affichage à l'écran.
- **Unmounting** (démonter), par déduction, est la suppression d'un composant de l'écran.
- **Updating** est le cycle de vie qui correspond à la mise à jour du composant. C'est lorsque l'on met à jour ses données et que celui-ci est re-rendu.

Si on s'intéresse au cycle de vie des composants, on peut voir que lorsqu'on lance l'application pour la première fois « mount », ensuite le constructeur est appelé la méthode « render() » (Figure 15).

Comme on a pu le voir précédemment dans le cycle de vie d'application, la méthode `componentDidMount()` n'est appelée que lorsque la méthode `render()` est terminée.

App.js

```
// we wait to load all the data from AWS in the arStore
```

```
componentDidMount(){
  this.loadData();
}

loadData = () => {
  arStore.loadData().then(() => {
    this.setState({
      loaded: true,
      displayNetworkError: false
    });
  });
})
```

}

Il est nécessaire d'attendre que les appels vers AWS soient exécutés pour récupérer les ressources nécessaires (images, vidéos, objets...) pour le bon fonctionnement de l'application. Une fois la récupération des données effectuée, on peut passer le booléen « loaded » à « vrai » pour signifier que le chargement est terminé. De la même manière, on peut utiliser une autre variable pour récupérer la présence d'erreur en cas de problème.

D) Les différentes scènes

Les scènes sont des composants customs (personnalisés) composés d'images, vidéos, d'objets 3D (texte notamment crée sur blender). Une application se basant sur ViroReact est composée de scènes *ViroARScene*. Les scènes sont les équivalents 3D des vues comme expliqué précédemment pour les applications 2D. Elles contiennent ainsi tous les composants que ViroReact rend.

Etant donné que ce projet comporte plusieurs sous-projets indépendants, on a créé 3 scènes distinctes que l'on appelle selon les besoins des démonstrations dans App.js en changeant l'*initialScene*. Il faut garder en tête que ce genre de structure peu pérenne s'inscrit dans le gain de temps cherché pour le développement de PoC. On affiche ainsi soit arBook.js, soit projectShowcase.js, soit une des galeries.

1) Arbook



Figure 16 : Illustration du résultat final de l'application

Le premier projet a pour but de superposer une vidéo à une image (Figure 16). Que ce soit pour un book comme prévu initialement ou une affiche, l'objectif est de donner l'impression qu'une image s'anime.

arBook.js

```
render() {
  return(
    <ViroARScene onCameraARHitTest={this.updateCamera}>
      {arStore.data.map(data =>
        <ViroARImageMarkerCustom
          target={data.id.toString()}
        />
      )}
    </ViroARScene>
  )
}
```



```

        key={data.id}
        onAnchorUpdated={({anchor}) => this.updateCurrentImageId(anchor, data.id)}
      >
        {this.displayVideo(data)}
        {this.displayButtons(data)}
        {this.displayLoading(data)}
      </ViroARImageMarkerCustom>
    }
  </ViroARScene>
);
}

```

On affiche ainsi la vidéo, un chargement le temps que la vidéo charge, voire des boutons supplémentaires dans le cas de cartes de visites. On peut voir au travers du `data.map` que l'on crée un `ViroARImageMarkerCustom` pour chaque image et on affiche ainsi une vidéo par image. `ViroARImageMarkerCustom` est un composant qui s'affiche uniquement si la cible a été détectée. La scène d'ARbook consiste à afficher une vidéo au-dessus de chaque image pré-enregistrée. Ainsi je crée un `ViroARImageMarkerCustom` pour chaque image qui affichera la vidéo lui correspondant.

2) Les projectShowcases

C'est le second composant à afficher comme l'ARbook. L'objectif ici est non pas d'afficher une simple vidéo au-dessus d'une image mais d'afficher des présentations de projets, comportant vidéo, image, texte et autres objets 3D.

projectShowcase.js

```

render() {
  return(
    <ViroARScene onCameraARHitTest={this.updateCamera}>
      {arStore.dataProjects.map(data =>
        <ViroNode key={data.id}>
          <ViroARImageMarker
            target={data.id.toString()}
            key={data.id}
            onAnchorUpdated={ (anchor) => this.onAnchorUpdated(anchor, data)}>
          </ViroARImageMarker>
          {this.displayScene(data)}
        </ViroNode>
      )}
    </ViroARScene>
  )
}

```

```
)  
}
```

Cette fois-ci on utilise une fonction qui regroupe les éléments à afficher car plus nombreux *displayScene()*. On peut aussi remarquer que la fonction est en dehors des balises *ViroARImageMarker*. La raison est que les balises affichent le résultat en suivant la cible alors que nous souhaitons disposer fixement la présentation une fois l'image détectée, même si cette dernière venait à bouger. Il suffit alors de n'afficher la scène que quand les conditions requises pour l'afficher sont réunies, c'est-à-dire quand *ViroARImageMarker* détecte une image via *onAnchorUpdated()*. Je reviendrai plus tard sur cette fonction car elle est centrale dans ce projet.

projectShowcase.js

```
displayScene = (data) => {  
  if (this.state.currentScene == data.id) {  
    let componentArray = [];  
    data.render.components.map(component => {  
      componentArray.push(this.displaySceneComponent(component))  
    });  
    return (  
      <ViroNode  
        rotation={[90, 0, 0]}  
        scale={[0.1, 0.1, 0.1]}  
      >  
        <ViroAmbientLight color={"#ffffff"} intensity={3000} />  
        <ViroNode  
          position={data.render.position}  
          rotation={data.render.rotation}  
          scale={data.render.scale}>  
          {componentArray}  
        </ViroNode>  
      </ViroNode>  
    );  
  }  
}
```

3) Les projectShowcaseGallery

projectShowcaseGallery.js

```
export default class projectShowcaseGallery extends Component {
```

```
render() {
  return(
    <ViroARScene onCameraARHitTest={this.updateCamera}>
      {projectData.map(data =>
        <ViroNode key={data.id}>
          <ViroARImageMarkerCustom target={data.id.toString()} key={data.id} onAnchorUpdated={
(anchor) => this.onAnchorUpdated(anchor, data)}>
            {this.displayLoading(data)}
          </ViroARImageMarkerCustom>
          {this.displayScene(data)}
        </ViroNode>
      )}
    </ViroARScene>
  )
}

displayScene = (data) => {
  if (this.state.currentScene == data.name) {
    return (
      <DSquareGallery/>
    )
  }
}
```

projectShowcaseGallery est le dernier sous-projet. Il a pour but d'afficher une « Gallery » constituée d'un ensemble de *billboards*, c'est-à-dire des *projectShowcase* sous la forme de panneaux d'affichage qui forment une galerie dans laquelle se promener une fois réunis (Figure 17). On peut voir dans le code que l'on affiche une *Gallery* sous forme d'un composant, ce qui en permet une utilisation simplifiée.



Figure 17 : Exemple d'un Billboard présentant le D.Lab

E) Les cibles

projectShowcaseGallery.js

```
projectData.forEach(data => {
  let trackingObject = {};
  trackingObject[data.id] = {
    source : data.image,
    orientation : "Up",
    physicalWidth : 0.07
  }
  ViroARTrackingTargets.createTargets(trackingObject);
})
```

Pour chaque donnée d'une variable d'un fichier JSON, on crée une target (cible) ou anchor (ancree) via la fonction `Viro createTargets()`. Cette dernière enregistre les images que l'application doit détecter pour afficher l'AR.

- **source** : désigne l'image
- **orientation** : l'orientation de la cible dans le but d'afficher l'objet AR en conséquence
- **physicalWidth** : la largeur réelle de l'image à détecter. En effet, l'application utilise cette donnée pour suivre efficacement la cible si elle est en mouvement.

Voici quelques exemples de logos utilisés comme cible pour déclencher l'apparition de la réalité augmentée (Figure 18). Ce sont des images qui ont pour but d'être visibles, d'attirer l'œil, tout en permettant à l'utilisateur d'identifier quel projet il s'apprête à afficher le contenu.

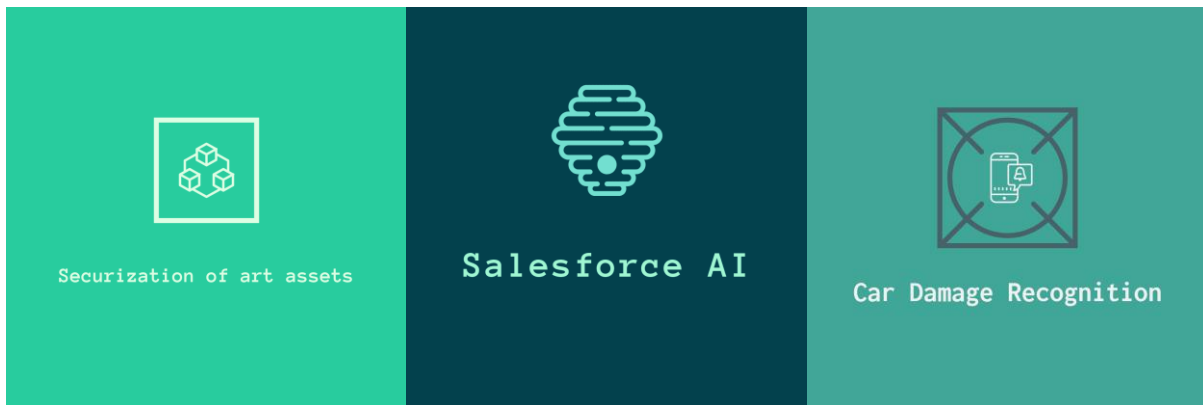


Figure 18 : Première version des logos utilisés comme targets

Par la suite on a changé les images à reconnaître pour améliorer les performances. Il est en effet nécessaire que l'image comporte un nombre important de détails. Voici par exemple une image que l'on a utilisée pour afficher une vidéo par-dessus. D'un côté l'image est bien hétérogène ce qui en facilite la détection et d'un autre côté l'image correspond à la première image de la vidéo, ce qui permet de créer une transition entre l'image et la vidéo en réalité augmentée (Figure 19).



Figure 19 : Exemple de cible, première image de la vidéo

Un autre aspect à prendre en compte est la surface de l'image sur son support. Par exemple, dans le cas d'une carte de visite il faut que l'image couvre l'intégralité de la carte pour que l'effet de réalité augmentée soit vraiment prenant.

On avait ainsi commencé par cette première carte mais étant donné que l'on superpose la vidéo à l'image, la vidéo ne couvrait alors pas entièrement la carte. On a alors exploré d'autres solutions. La première consistait à prendre comme image l'entièreté de la carte mais les zones noires rendaient la reconnaissance et le suivi de l'image difficile. La deuxième solution a été d'afficher la vidéo avec un ratio de taille de telle manière qu'elle couvre l'ensemble de son support. Cependant, nous ne voulions gérer tous ces cas particuliers. Nous avons donc décidé de standardiser les supports en leur faisant suivre ces quelques règles. La deuxième carte est celle qui a été finalement retenue (Figure 20).



Figure 20 : Cible peu propice au tracking (gauche) et idéale au tracking (droite)

F) Le « tracking » d'image

La gestion du suivi (tracking) des cibles a été une part importante du développement. On souhaite n'afficher qu'un seul contenu d'AR à la fois pour que les présentations restent claires et bien distinctes. De plus, pour certains projets comme l'ARbook, il est indispensable de suivre correctement la cible si l'on souhaite y superposer une vidéo.

Une des problématiques du projet fut d'optimiser la reconnaissance et le suivi de l'image cible. En effet, une détection trop sensible a tendance à changer continuellement de cible et ne permet pas d'afficher une vidéo entièrement sans détecter une autre image. Une sensibilité trop faible aura tendance à avoir des difficultés à détecter initialement la cible et par la suite de la suivre si cette dernière se déplace dans l'espace. On veut aussi que le changement de cible soit fluide et ne requiert pas de manipulations complexes de la part de l'utilisateur.

ArBook.js

```
updateCurrentImageId = (anchor, id) => {

  if(id !== this.state.currentImageId) { // If we detect a different image
    const imageDistanceTested = this.closer(anchor)
    if (imageDistanceTested.testPassed) { // If it's closer than the previous one, we change the current image
      this.setState({
```

```

        currentImageId: id,
        currentImagePosition: imageDistanceTested.position,
        count: this.state.count + 1
    });
}
} else { // If it's the current image, we update its position
    this.setState({
        currentImagePosition: anchor.position
    });
}
}
}

```

onAnchorUpdated() est une méthode appelée à chaque fois que l'image est à nouveau détectée. C'est une méthode centrale car c'est elle qui nous permettra de suivre la détection ou non des cibles, mais aussi d'en connaître leur identifiant ou leur position. Cette méthode va appeler *updateCurrentImageId()* pour l'affichage des galeries ou *onAnchorUpdated()* pour l'ARbook, des fonctions bien distinctes mais aux comportements proches.

Cette fonction va suivre quelle image est détectée. Si c'est la même image que précédemment on actualise simplement sa position. Cependant si l'image détectée est différente on vérifie si cette nouvelle image détectée est plus proche du téléphone.

Une grande part du projet a été de stabiliser l'affichage de la vidéo affichée car on ne veut afficher qu'une seule vidéo et c'est celle qui est la plus proche de nous. « *closer()* » (Annexe 3) permet de calculer la différence de distance entre deux cibles et le téléphone pour en définir la plus proche. On souhaite en effet animer l'image la plus proche au cas où deux cibles nous font face. Pour cela on suit en temps réel la position de la caméra et la position de l'image dans un repère. La fonction « *closer()* » renvoie un booléen si le test est validé, c'est-à-dire que la nouvelle image est plus proche.

projectShowcase.js

```

onAnchorUpdated(anchor, data) {
    if (this.state.currentScene !== data.id) { // If we detect a different image

        const imageDistanceTested = this.closer(anchor)

        if (imageDistanceTested.testPassed) { // If it's closer than the previous one, we change the current
image
    }
}
}

```

```

    let projectPosition = [this.cameraPosition[0] + this.cameraForward[0] * 1, this.cameraPosition[1]
+ 0.2, this.cameraPosition[2] + this.cameraForward[2] * 1]
    storeProps.detectedProjectCameraPosition = projectPosition

    let projectRotation = [90, this.cameraRotation[1], 0] // rotation on the Y axis need to be fixed
    storeProps.detectedProjectCameraRotation = projectRotation
    this.setState({
      currentScene: data.id,
    })
  }
}

```

Ici aussi on vérifie la distance de la nouvelle image et on ne sélectionne que la plus proche. Il faut cependant aussi suivre le déplacement du téléphone. En effet, on ne suit plus une cible en lui superposant une vidéo par-dessus, il faut afficher le projet à une distance fixe devant le téléphone. Or l'origine du repère de l'AR est à l'emplacement du téléphone lors du lancement de l'application. Il faut donc savoir où est le téléphone au moment où on repère une image mais aussi son orientation, de manière à être sûr que le projet s'affiche bien en face de l'utilisateur.

De plus, à chaque fois qu'une cible plus proche est détectée, on incrémente un compteur qui se réinitialise au bout d'un temps.

arBook.js

```

// reset the list of detected images
resetImagesDetected = () => {
  arStore.setDisplayFocusWarning(this.state.imagesDetected.length >= 3);
  this.setState({
    imagesDetected: [],
  });
}

```

Si ce nombre est trop élevé, *resetImagesDetected()* affiche un message à l'utilisateur demandant de se concentrer sur une seule image. Au bout de l'intervalle de temps, *updateCurrentImage()* reset ce nombre à 0.

arBook.js

```

// If no image has been detected in a intervalDelayBeforeRemovingVideo, we reset the current image id
to 0 and the position far
updateCurrentImage = () => {

```



```
if (this.state.count == 0) {
  this.setState({
    currentImageId: -1,
    currentImagePosition: [1000, 1000, 1000]
  });
}
this.setState({
  count: 0
});
}
```

Si aucune image a été détectée, cela signifie que l'utilisateur ne vise plus une image avec son téléphone donc on arrête d'afficher l'AR. On définit aussi la longueur à comparer via *closer()* très loin pour être certain que la prochaine image détectée sera plus proche.

projectShowcase.js

```
class SceneMarker extends Component {

  constructor(props) {
    super(props);
    this.state = {
      currentScene: -1,
      currentImagePosition: [1000, 1000, 1000],
      cameraPosition: []
    };
  }
}
```

De la même manière que dans l'ARbook, on initialise dans le state la scène actuelle et une position très loin.

G) Les données et leur récupération

Data/projectData.js

```
const projectData = [
  {
    id: 1,
    name: "blockchainScene",
```

```
scene: <BlockchainScene/>,
image: require("../ressources/ProjectShowcase/Blockchain/image/logo.png"),

},
{
  id: 2,
  name: "carDamageRecognitionScene",
  scene: <CarDamageRecognitionScene/>,
  image: require("../ressources/ProjectShowcase/CarDamageRecognition/image/logo.png")
}
];
```

ProjectData référence les scènes utilisables pour la présentation de projet. Tous les assets sont stockés sur Amazon S3, qui est un site d'hébergement de fichiers proposé par Amazon Web Services (AWS).

Ressources comprend toutes les images, vidéos et objets utilisés à la fois dans l'ARbook ou dans les projets. J'ai peu à peu transféré le stockage d'objet vers « Amazon S3 » car le stockage y est plus sécurisé et la centralisation des ressources améliore son accessibilité. Amazon s3 (Amazon Simple Storage Service) est un service de stockage d'objets sur Internet fourni par Amazon. La partie ressources représentant un stockage local des ressources tend ainsi à disparaître.

La particularité est que j'ai créé tous les objets sur blender, logiciel libre et gratuit de modélisation, d'animation par ordinateur et de rendu en 3D. Les objets créés sur blender sont exportés en .fbx. La particularité est que Viro ne supporte que son format propre : le .vrx.

Il faut donc convertir chaque objet pour l'utiliser dans les projets, ce qui rend le développement moins rapide.

II) Chatbot utilisant l'Intelligence Artificielle

A) Choix techniques et initialisation du projet

Le projet se compose de 2 parties. La première est l'interface de communication avec l'utilisateur via un chat. On a pour cela utilisé « bot framework » de Microsoft (Webographie 7) étant donné que le but premier du chatbot est de comprendre l'intention de l'utilisateur, le chatbot seul ne suffit pas. On ne peut pas en effet rediriger l'utilisateur selon l'emploi de phrases préétablies.

Pour comprendre le besoin de l'utilisateur on a alors utilisé une IA (Intelligence Artificielle) comprenant le langage naturel. L'IA utilisée est celle de Microsoft appelé LUIS. C'est un modèle pré-entraîné qui propose une interface où ajouter des phrases d'entraînement (utterances) associées à une intention.

- Un exemple d'intention pourrait être « Réserver un vol ».
- Le chatbot va alors transmettre le message reçu à LUIS qui va alors lui associer une série d'intentions scorées. Plus le score est haut plus LUIS a associé le message à l'intention avec certitude.
- On dégage alors une intention dominante (top intent).
- On pourra alors traiter cette intention dans le chatbot par une réponse personnalisée.

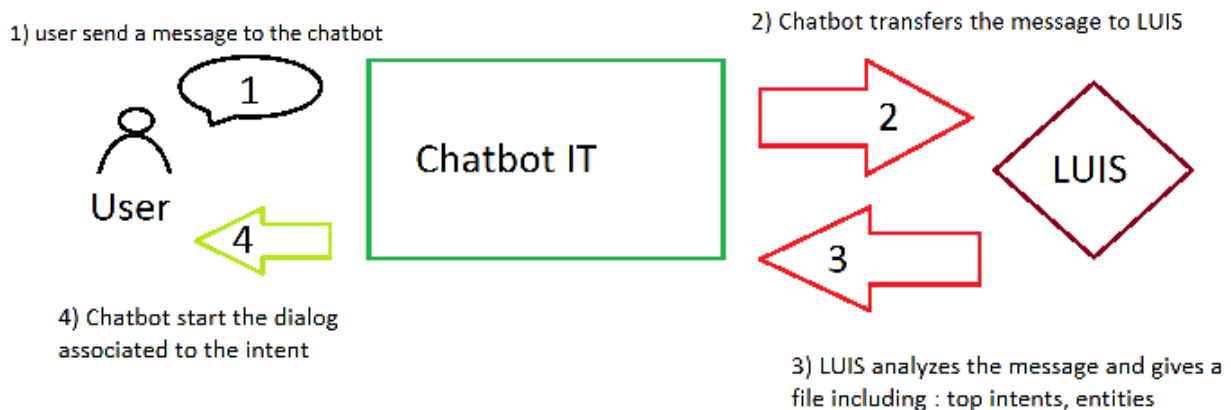


Figure 21 : Schéma récapitulatif du fonctionnement du projet

Dans un premier temps l'utilisateur pose une question ou sa requête sous forme de message dans le chatbot (Figure 21). Le chatbot reçoit alors le message. Il faut associer une activité, généralement un message de réponse, à ce message entrant (Webographie 8). Cependant, au vu du nombre de possibilités de message, il est impossible d'associer une réponse à chaque message pouvant exister.

La compréhension du langage naturel est considérée comme un problème relativement difficile. C'est la nature même du langage humain qui le rend difficile. Les règles qui dictent la transmission d'information est difficilement compréhensible pour les ordinateurs. Certaines règles peuvent avoir un haut niveau d'abstraction comme le sarcasme, d'autres peuvent être très concrètes, par exemple l'emploi du « s » pour signifier la pluralité d'un objet. Comprendre le langage humain nécessite de maîtriser à la fois les mots mais aussi comment les concepts sont reliés entre eux pour délivrer le message voulu. Alors que des humains peuvent maîtriser avec facilité un langage, l'ambiguïté et les caractéristiques du langage naturel sont ce qui rend difficile son utilisation avec des ordinateurs. C'est pour cette raison que l'on transmet ce message à LUIS. LUIS est un modèle pré-enregistré pour comprendre le langage naturel (Webographie 9).

Ce dernier associe la phrase avec des intents classés avec un système de score. Le score indique le niveau de confiance que porte LUIS sur son association phrase – intent. Plus le score est élevé plus le niveau de confiance est élevé. Par exemple une intention avec un score de 0.90 indique que LUIS est sûr à 90 pourcents de l'intention de l'utilisateur après analyse de son message.

Le chatbot reçoit la réponse de LUIS sous forme de JSON avec les intents classés. On peut alors créer une réponse pour chaque intent, dont le nombre est limité. Dans le projet, on a associé chaque intent à un dialogue (expliquer le fonctionnement des dialogues) pour plus de modularité.

B) Structure du projet

Le projet se compose de deux parties. La première est hébergée en local et comporte tous les fichiers nécessaires au fonctionnement du bot tandis que la deuxième partie accessible par navigateur est une interface relative à l'entraînement et le déploiement de LUIS (Figure 22).

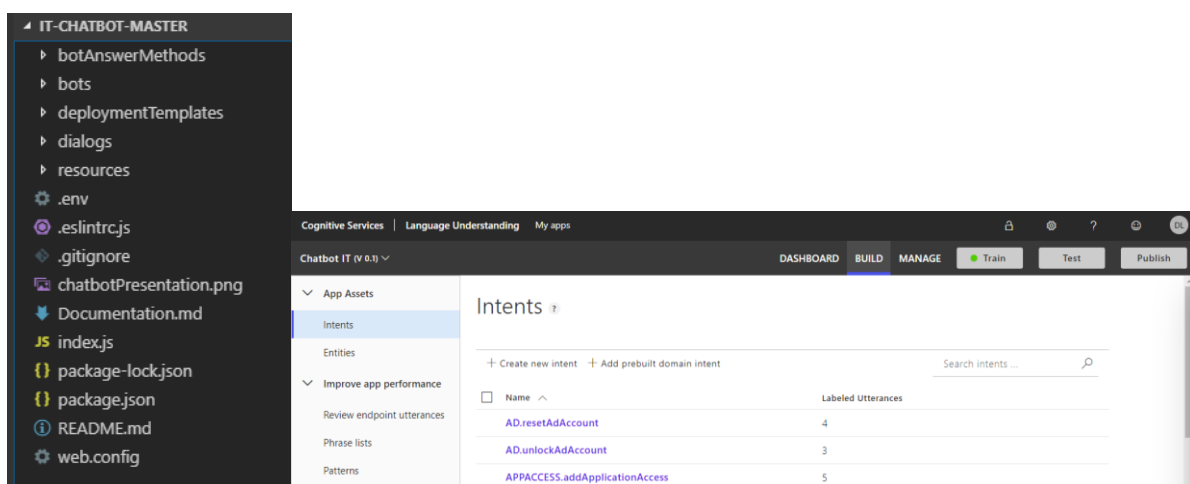


Figure 22 : Structure du bot (gauche) et interface de LUIS (droite)

Ici encore nous ne nous intéresserons qu'aux différents dossiers relatifs aux fonctionnalités du projet et mettrons de côté la partie configuration du projet. Le fichier index.js est le premier fichier appelé par le projet et a globalement pour rôle de déclarer les bots et dialogues utilisés par l'application.

C) Les dialogues

Les dialogues constituent un concept central. Ils offrent un moyen utile de gérer une conversation avec l'utilisateur. Les dialogues sont des structures dans le bot qui agissent comme des fonctions dans le programme du bot ; chaque dialogue est conçu pour effectuer une tâche spécifique, dans un ordre spécifique. On peut spécifier l'ordre des dialogues individuels pour guider la conversation, puis les appeler de différentes façons : parfois, en réponse à un utilisateur ou à partir d'autres dialogues.

La bibliothèque de dialogues propose quelques fonctionnalités intégrées, comme les *invites* et les *dialogues en cascade* pour rendre la conversation du bot plus facile à gérer. Les invites sont utilisées pour demander différents types d'informations, comme du texte, un nombre ou une date. Les dialogues en cascade peuvent combiner plusieurs étapes en une séquence, ce qui permet de suivre facilement cette séquence prédéfinie pour transmettre des informations à l'étape suivante (Webographie 10).

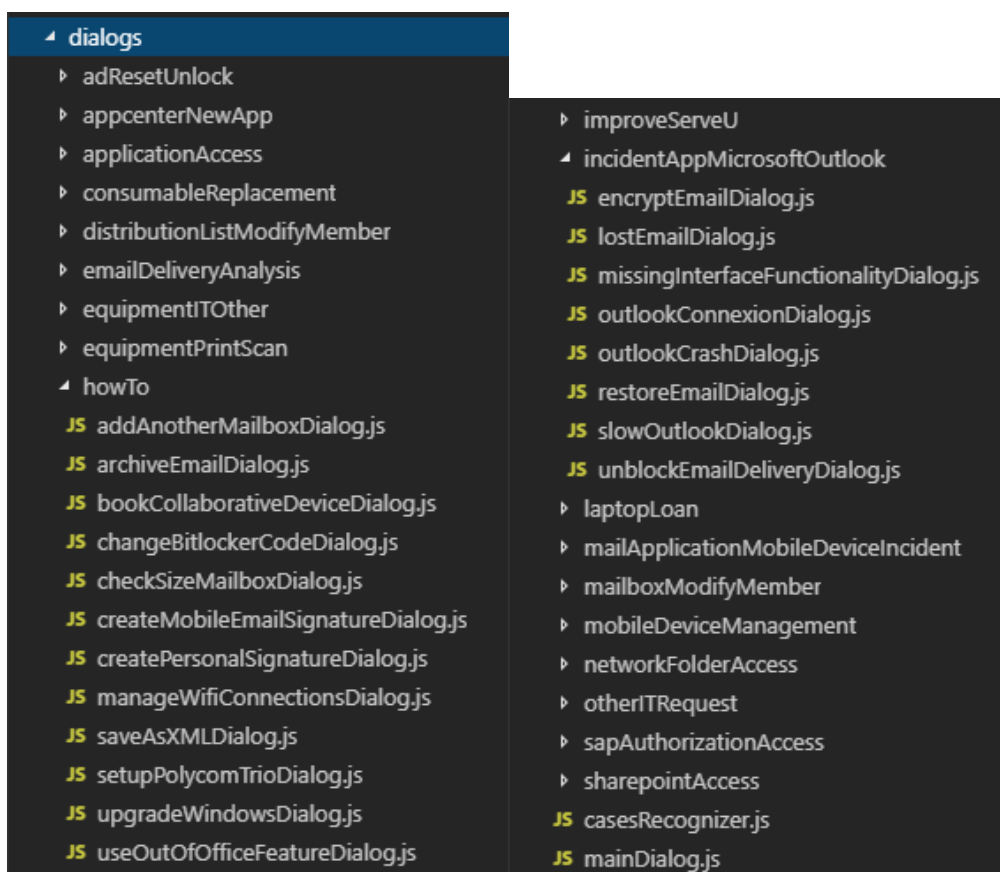


Figure 23 : Liste de tous les dialogues

Chaque intent est associé avec un dialogue (Figure 23). Cela permet de personnaliser les réponses du chat pour une intention donnée. Il faut garder à l'esprit que les clients et utilisateurs du projet ne seront pas des développeurs mais l'équipe IT de l'entreprise. C'est pourquoi il faut rendre accessible et modifiable chaque partie du projet.

C'est le premier dialogue appelé sur l'application (défini dans index.js). L'objectif de ce dialogue est de rediriger l'utilisateur vers un dialogue propre à l'intention identifiée.

mainDialog.js

```
// adResetUnlock
const { ResetAdAccountDialog } = require('./adResetUnlock/resetAdAccountDialog');
const { UnlockAdAccountDialog } = require('./adResetUnlock/unlockAdAccountDialog');
// appCenterNewApp
const { RequestNewAppDialog } = require('./appcenterNewApp/requestNewAppDialog');
```

J'ai supprimé ici la plupart des imports pour le rapport mais le nombre d'imports nécessite de trouver une autre méthode.

mainDialog.js

```
class MainDialog extends ComponentDialog {
  constructor(luisRecognizer) {
    super('MainDialog');

    // Define the main dialog and its related components.
    this.addDialog(new TextPrompt('TextPrompt'))

    // adResetUnlock
    .addDialog(new ResetAdAccountDialog(dialogsIdConstants.RESET_AD_ACCOUNT_DIALOG))
    .addDialog(new UnlockAdAccountDialog(dialogsIdConstants.UNLOCK_AD_ACCOUNT_DIALOG))
```

De la même manière l'ajout des dialogues prend une place importante mais cette solution moins viable à long terme est la plus rapide pour le développement d'un PoC. Si le projet est continué, ces points constituent des axes d'amélioration importants.

mainDialog.js

```
// sharepointAccess
.addDialog(new
GrantSharepointAccessDialog(dialogsIdConstants.GRANT_SHAREPOINT_ACCESS_DIALOG))
.addDialog(new WaterfallDialog(MAIN_WATERFALL_DIALOG, [
  this.introStep.bind(this),
  this.mainStep.bind(this),
  this.finalStep.bind(this)
]));

this.initialDialogId = MAIN_WATERFALL_DIALOG;
```

Le *contexte du dialogue* contient des informations sur le dialogue. Il sert à interagir avec un jeu de dialogues depuis le gestionnaire de tour. Le contexte du dialogue inclut le contexte du tour actuel, le dialogue parent et l'état du dialogue, ce qui offre une méthode permettant de conserver les informations au sein du dialogue. Le contexte du dialogue permet de démarrer un dialogue avec son ID de chaîne, ou de continuer le dialogue actuel (par exemple, un dialogue en cascade qui comporte plusieurs étapes).

Quand un dialogue se termine, il peut retourner un *résultat du dialogue* avec certaines informations obtenues à partir du dialogue. Ce résultat est retourné pour permettre à la méthode d'appel de voir ce qui s'est passé au cours du dialogue et d'enregistrer les informations.

La méthode *run()* gère l'activité du bot sous la forme d'un *TurnContext* et le transmet à travers le système de dialogue. Si aucun dialogue n'est actif, la méthode *run()* démarrera le dialogue par défaut, ici le *mainDialog*.

mainDialog.js

```
async run(turnContext, accessor) {
  const dialogSet = new DialogSet(accessor);
  dialogSet.add(this);

  const dialogContext = await dialogSet.createContext(turnContext);
  const results = await dialogContext.continueDialog();
  if (results.status === DialogTurnStatus.empty) {
    await dialogContext.beginDialog(this.id);
  }
}
```

La première étape dans le dialogue est de vérifier la configuration de LUIS et d'accueillir l'utilisateur.

mainDialog.js

```
async introStep(stepContext) {
  if (!this.luisRecognizer.isConfigured) {
    const messageText = 'NOTE: LUIS is not configured. To enable all capabilities, add `LuisAppId`, `LuisAPIKey` and `LuisAPIHostName` to the .env file.';
    await stepContext.context.sendActivity(messageText, null, InputHints.IgnoringInput);
    return await stepContext.next();
  }

  const messageText = stepContext.options.restartMsg ? stepContext.options.restartMsg : 'How can I help you today?';
```

```
await stepContext.context.sendActivity(messageText, null, InputHints.IgnoringInput);
return await stepContext.prompt('TextPrompt');
}
```

L'étape principale récupère la réponse de LUIS and commence le dialogue lui correspondant.

Si LUIS n'a pas réussi à définir l'intention de l'utilisateur, on le redirige vers la première étape avec un message lui signifiant que le message n'a pas été compris.

mainDialog.js

```
// Retrieves the LUIS result and begins the corresponding dialog
async mainStep(stepContext) {
  const luisResult = await this.luisRecognizer.executeLuisQuery(stepContext.context);
  if (LuisRecognizer.topIntent(luisResult) == 'None'){
    return stepContext.replaceDialog(this.initialDialogId, { restartMsg: 'I didn\'t understood your request, please note that I am still in development. You can try another sentence' });
  }
  return stepContext.beginDialog(dictionary.intentDataSource[LuisRecognizer.topIntent(luisResult)]);
}
```

Lorsque l'on revient de *l'intentDialog*, on accède à la dernière étape. Cette dernière reboucle simplement sur la première étape avec un message spécifique.

mainDialog.js

```
// Restarts the main dialog
async finalStep(stepContext) {
  return await stepContext.replaceDialog(this.initialDialogId, { restartMsg: 'What else can I do for you?' });
};
}
```

C'est un *intentDialog*. Chaque intent a son dialogue associé. Ce dernier ne possède qu'une seule étape qui envoie en message sous forme de lien l'url du ticket à ouvrir sur l'intranet.

OutlookConnexionDialog.js

```
class OutlookConnexionDialog extends ComponentDialog {
  constructor() {
    super(dialogsIdConstants.OUTLOOK_CONNEXION_DIALOG);
  }
}
```



```

this.addDialog(new WaterfallDialog(WATERFALL_DIALOG, [
    this.mainStep.bind(this)
]));
}

async mainStep(stepContext) {

    const mainStepMessage = dialogsWording.OUTLOOK_CONNEXION.mainMessage;
    await stepContext.context.sendActivity(mainStepMessage);

    return await stepContext.endDialog();
}
}

```

Voici l'exemple d'un dialogue un peu plus complexe. Il se compose de 3 étapes :

- *introStep* : introduit le tutoriel et fournit le lien du tutoriel sous format PDF présent sur l'intranet.
- *tutorialStep* : affiche le tutoriel en envoyant un message par chaîne de caractères présentes dans le tableau. On appelle la méthode *confirmStep()* demandant si le lien a été utile.
- *finalStep* : récupère le résultat de la question. Si la réponse est négative, on envoie le lien du ticket. On reboucle ensuite sur le mainDialog.

AddAnotherMailboxDialog.js

```

class AddAnotherMailboxDialog extends ComponentDialog {
    constructor() {
        super(dialogsIdConstants.ADD_ANOTHER_MAILBOX_DIALOG);

        this.addDialog(new ConfirmPrompt(CONFIRM_PROMPT))
            .addDialog(new WaterfallDialog(WATERFALL_DIALOG, [
                this.introStep.bind(this),
                this.tutorialStep.bind(this),
                this.finalStep.bind(this)
            ]));

        this.initialDialogId = WATERFALL_DIALOG;
    }

    // introduce the tutorial and gives a link to the pdf format from the intranet
    async introStep(stepContext) {
        const introStepMessage = dialogsWording.ADD_ANOTHER_MAILBOX.introMessage;
    }
}

```

```

    await stepContext.context.sendActivity(introStepMessage);
    return await stepContext.next();
}

// display the tutorial stores in "resources/tutorial" with a confirm step at the end
async tutorialStep(stepContext) {
    await sendTutorial.sendTutorial(stepContext,
addAnotherMailboxSentences.addAnotherMailboxSentences);
    return await confirmStep.confirmStep(stepContext);
}

// use the confirm step response to know if the tutorial was useful, send the ticket link on negative
answer
async finalStep(stepContext) {
    if (stepContext.result != true) {
        const finalErrorMessageSolution1 =
dialogsWording.ADD_ANOTHER_MAILBOX.finalErrorMessageSolution1
        await stepContext.context.sendActivity(finalErrorMessageSolution1, null, InputHints.IgnoringInput);
        const finalErrorMessageSolution2 =
dialogsWording.ADD_ANOTHER_MAILBOX.finalErrorMessageSolution2
        await stepContext.context.sendActivity(finalErrorMessageSolution2, null, InputHints.IgnoringInput);
    } else {
        await stepContext.context.sendActivity(dialogsWording.ADD_ANOTHER_MAILBOX.finalMessage,
null, InputHints.IgnoringInput);
    }
    return await stepContext.endDialog();
}
}

```

D) Les ressources

Les ressources (ou *resources* en anglais) correspondent à tous les éléments nécessaires à l'application pour fonctionner mais n'appartenant pas à la logique de l'application (Figure 24).

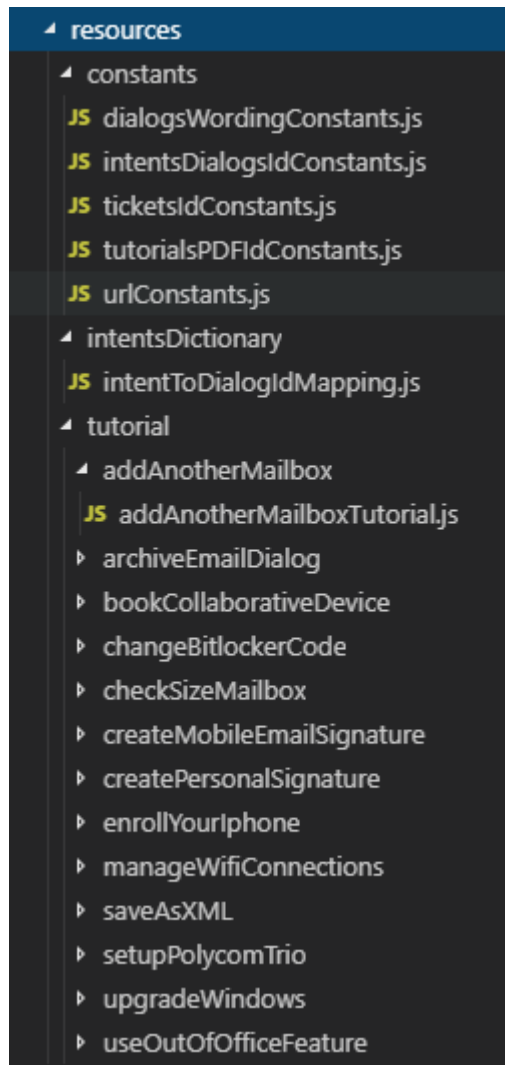


Figure 24 : Ensemble des ressources en local

1) Les constantes

Le *Wording* est un fichier regroupant tous les textes affichés à l'utilisateur. Cette structure permet une implémentation et un suivi plus simple de la part des techniciens et développeurs.

Le fichier se présente sous la forme d'un dictionnaire et chaque dialogue possède ses différents messages. Il existe pour le moment deux types de structures dans le Wording :

- Réponse par un simple lien vers le ticket dédié à sa demande

dialogsWordingConstants.js

```
// equipmentPrintScan
DEBUG_PRINT_SCAN : {
  mainMessage : "To debug the printer or scanner, open this " +
createLink.createLink(urlConstants.SERVEU_URL + ticketsId.EQUIPMENT_PRINT_SCAN, "ticket")
},
```

```
PAPER_JAMMED : {
  mainMessage : "To report paper jammed incident for printer or scanner, open this " +
createLink.createLink(urlConstants.SERVEU_URL + ticketsId.EQUIPMENT_PRINT_SCAN, "ticket")
},
```

- Présentation d'un tutoriel avec un message d'introduction et 2 messages finaux selon l'utilité du tutoriel (positive ou négative) (Figure 25).

dialogsWordingConstants.js

```
// howTo
ADD_ANOTHER_MAILBOX : {
  introMessage : INTRO_TUTORIAL_MESSAGE1 + "add another mailbox. " +
INTRO_FIND_TUTORIAL_MESSAGE + createLink.createLink(urlConstants.HOW_TO_URL +
tutorialsPDFIdConstants.ADD_ANOTHER_MAILBOX, "here") + "<br><br>" +
INTRO_TUTORIAL_MESSAGE2,
  finalErrorMessageSolution1 : "if you added the mailbox but got this message « cannot display folder
», you can open this " + createLink.createLink(urlConstants.SERVEU_URL +
ticketsId.MAILBOX_MODIFY_MEMBER, "ticket"),
  finalErrorMessageSolution2 : "Else, open this " + createLink.createLink(urlConstants.SERVEU_URL
+ ticketsId.INCIDENT_APP_MICROSOFT_OUTLOOK, "ticket"),
  finalMessage : FINAL_MESSAGE
},
ARCHIVE_EMAIL : {
  introMessage : INTRO_TUTORIAL_MESSAGE1 + "manage archived emails." +
INTRO_FIND_TUTORIAL_MESSAGE + createLink.createLink(urlConstants.HOW_TO_URL +
tutorialsPDFIdConstants.ARCHIVE_EMAIL, "here") + "<br><br>" + INTRO_TUTORIAL_MESSAGE2,
  finalErrorMessageSolution : FINAL_ERROR_MESSAGE_SOLUTION +
createLink.createLink(urlConstants.SERVEU_URL + ticketsId.INCIDENT_APP_MICROSOFT_OUTLOOK,
"ticket"),
  finalMessage : FINAL_MESSAGE
}
}
```

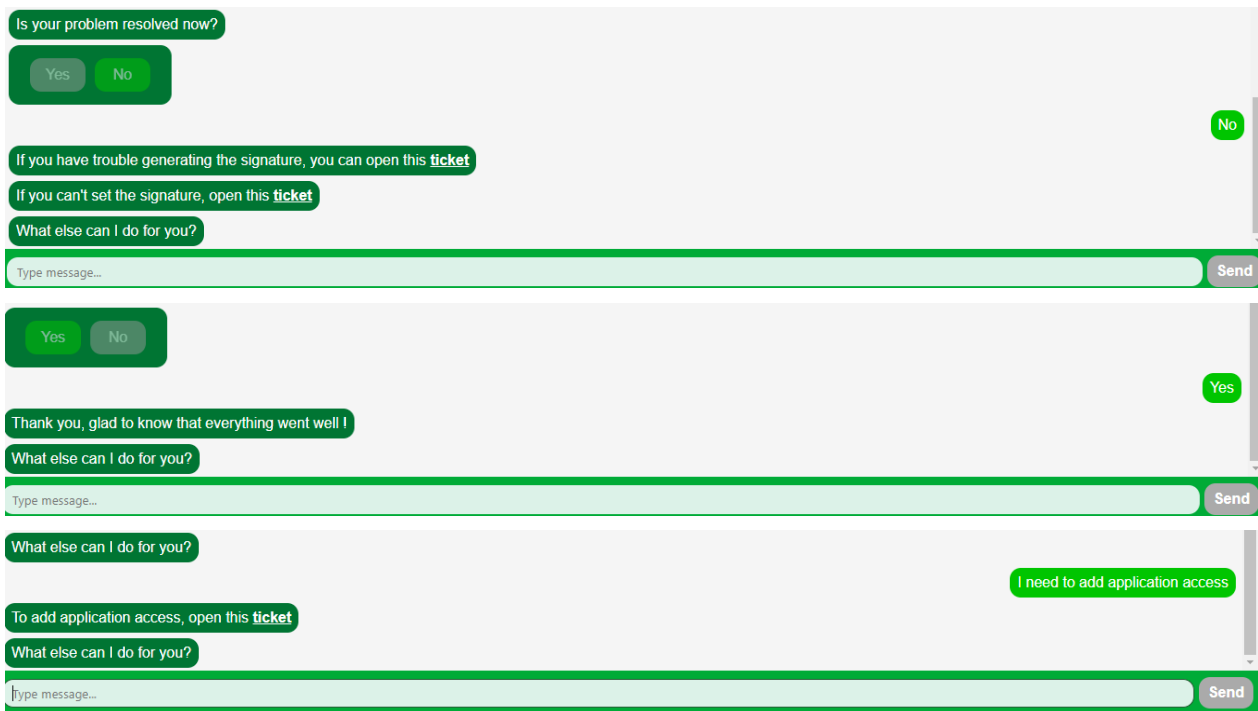


Figure 25 : Différents dialogues selon la réponse à la question

Chaque dialogue est identifié par un identifiant unique et ces derniers sont regroupés dans un fichier unique. Cette démarche s'inscrit toujours dans la volonté de simplifier la prise en main et la maintenance du projet.

intentsDialogsIdConstants.js

```
// This file stores all the intent dialog ids. Each intent has his own intent dialog

// adResetUnlock
const RESET_AD_ACCOUNT_DIALOG = 'resetAdAccountDialog';
const UNLOCK_AD_ACCOUNT_DIALOG = 'unlockAdAccountDialog';

// appCenterNewApp
const REQUEST_NEW_APP_DIALOG = 'requestNewAppDialog';

// applicationAccess
const ADD_APPLICATION_ACCESS_DIALOG = 'addApplicationAccessDialog';
const REMOVE_APPLICATION_ACCESS_DIALOG = 'removeApplicationAccessDialog';
```

Voici un échantillon du grand nombre de tickets existant sur l'intranet de l'entreprise. Tous les id des tickets interviennent dans l'url utilisée pour atteindre la page du ticket. Etant donné que l'url et l'id du ticket risquent de subir des modifications lors du passage de stage de développement à celui de production, il a été choisi de les déclarer sous forme de constantes et de les regrouper afin de les modifier aisément à l'avenir.

ticketsIDConstants.js

```
// This file stores all the id of the ticket in ServeU
```

```
const APPCENTER_NEWAPP = 17;
const EMAIL_DELIVERY_ANALYSIS = 30;
```

D'une manière similaire, on déclare sous forme de constantes les parties d'url dirigeant l'utilisateur vers le PDF des tutoriels car ces dernières sont susceptibles d'être changées ou d'en voir de nouvelles ajoutées. Etant donné que ces parties d'url sont uniques, on peut les identifier comme des identifiants.

tutorialsPDFIdConstants.js

```
// This file stores all the ids (or urls) the tutorial of the "How to" section in the intranet
```

```
const ADD_ANOTHER_MAILBOX = 'How%20to%20add%20another%20mailbox%20001.2018.pdf';
const ARCHIVE_EMAIL = '/IT%20How%20To/How%20to%20archive%20your%20emails.pdf';
```

Pour le moment, ce fichier ne contient que 2 urls différentes :

- SERVEU_URL : correspond à l'url ServeU contenant les différents tickets ouvrables
- HOW_TO_URL : correspond à l'url HowTo de l'intranet regroupant les tutoriels sous forme de PDF.

urlConstants.js

```
// this files stores the urls
```

```
const SERVEU_URL = 'http://serveu/Pages/ServiceForm/CreateRequest.aspx?ServiceId=';
const HOW_TO_URL = 'ludeloitteintranetv3/practice_support/information_technologies/HowTo/';
```

intentToDialogIdMapping.js

```
// this file makes the connection between the luis intent and the intent dialog associated
```

```
const intentDataSource = {

  // NOTE : for the intents, the dot "." is replaced by an underscore "_" in the JSON response of LUIS
  // AD (LUIS service name keyword) adResetUnlock (service name)
  "AD_resetAdAccount" : dialogIdConstants.RESET_AD_ACCOUNT_DIALOG,
  "AD_unlockAdAccount" : dialogIdConstants.UNLOCK_AD_ACCOUNT_DIALOG,

  // appCenterNewApp APPCENTER
```

```
"APPCENTER_requestNewApp" : dialogIdConstants.REQUEST_NEW_APP_DIALOG,
```

Ce fichier fait le lien entre l'intention enregistrée sur LUIS et le dialogue associé à l'intention.

2) Les tutoriels

Voici l'exemple d'un tutoriel que l'on affiche sous forme de message. Le tutoriel est enregistré sous forme d'un tableau de chaîne de caractères et chaque chaîne sera envoyée sous forme de message par le bot (Figure 26).

On peut remarquer la présence de balise HTML et CSS dans le texte. En effet, notre interface personnalisée de messagerie permet l'utilisation de ces 2 technologies.

Cela ne permet pas cependant de créer des feuilles de style externes, ce qui alourdit sensiblement le code.

De plus, l'émulateur utilisé pour les tests ne reconnaît pas ces balises ce qui ne permet pas de faire des tests dans des conditions réelles.

archiveEmailTutorial.js

```
`Note that you can only view archived emails if you are connected to the Deloitte network by LAN, wireless
or VPN`,
,
<h1 style="text-decoration: underline;"> How to view/find an archived mail: </h1> <br>

1) Open the Archive folder <br>
2) Run an Outlook search, but remember it will only run on the 1000 first characters <br>
3) For a more in-depth search, use the "Capax Archive Solutions" button on your Outlook bar
,
,
<h1 style="text-decoration: underline;"> To run a deep search through all mail contents </h1> <br>

1) Select Capax Archive Solutions on the Outlook menu <br>
2) Select Search icon <br>
3) Fill in the different boxes: <br>
<p style="margin-left: 50px; font-size: 0.875em;">
  a) <span style="font-weight: bold;"> Contents: </span> The key word (in this case GSM) is present
in the email <br>
  b) <span style="font-weight: bold;"> Subject</span>: The subject of the email <br>
  c) <span style="font-weight: bold;"> To / From</span>: To whom and from the email was sent <br>
  d) <span style="font-weight: bold;"> Attachment</span>: Enter key word of attachment, if known
<br>
  e) <span style="font-weight: bold;"> Folder</span>: In which folder the email was stored <br>
```

f) Date: The period in which the email was sent

</p>
4) Define in the "Users" box (upper right corner) which mailbox to search

5) Click on "Search"

<h1 style="text-decoration: underline;"> How to view the complete content of an archived email : </h1>

1) Select the email in your inbox

2) Double-click on the selected email, the original email with the original length and content will be displayed

How to view/find an archived mail

- 1) Open the Archive folder
- 2) Run an Outlook search, but remember it will only run on the 1000 first characters
- 3) For a more in-depth search, use the "Capax Archive Solutions" button on your Outlook bar

To run a deep search through all mail contents

- 1) Select Capax Archive Solutions on the Outlook menu
- 2) Select Search icon
- 3) Fill in the different boxes:
 - a) **Contents**: The key word (in this case GSM) is present in the email
 - b) **Subject**: The subject of the email
 - c) **To / From**: To whom and from the email was sent
 - d) **Attachment**: Enter key word of attachment, if known
 - e) **Folder**: In which folder the email was stored
 - f) **Date**: The period in which the email was sent
- 4) Define in the "Users" box (upper right corner) which mailbox to search
- 5) Click on "Search"

How to view the complete content of an archived email :

How to view the complete content of an archived email :

- 1) Select the email in your inbox
- 2) Double-click on the selected email, the original email with the original length and content will be displayed

Is your problem resolved now?

Yes

No

Figure 26 : Rendu du tutoriel sur le front custom

E) Les méthodes

botAnswerMethods regroupe toutes les methodes utilisées par le bot dans sa communication avec l'utilisateur via les dialogues (Figure 27).

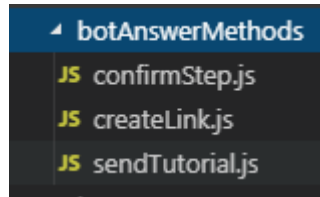


Figure 27 : Les trois méthodes développées

confirmStep() est une methode qui permet de proposer un message (sous forme de question) attendant une réponse de l'utilisateur (prompt). Etant donné que le prompt attend 2 reponses possibles, on associe la question avec l'envoi de 2 boutons «oui » et « non » dans un message. La méthode *sendActivity()* correspond à l'envoi de message.

confirmStep.js

```
async function confirmStep(stepContext) {

    const buttonData = [
        {
            text: "Yes"
        },
        {
            text: "No"
        }
    ]
    const promptData = {
        type: "buttons",
        message: "",
        data: buttonData
    }
    await stepContext.context.sendActivity("Is your problem resolved now?");
    return await stepContext.prompt(CONFIRM_PROMPT, promptOptions);
}
```

Comme le front personnalisé que l'on utilise comprend les balises HTML et CSS, on peut les inclure dans les messages que l'on envoie. Cela alourdit le code et pour cette raison, j'ai centralisé la création de lien .

confirmStep.js

```
function createLink(url, message) {
```

```
return "<a style='font-weight: bold; text-decoration: underline; color: white;' href='" + url + "'> " +  
message + " </a>"  
};
```

F) Interface de LUIS

Sur le site dédié à la gestion de LUIS, on peut créer des intentions, que l'on a regroupées sous un service (Figure 28). Le mot-clé comme « APPACCESS » regroupe plusieurs intentions gérées par le même service. Par exemple l'ajout ou la suppression d'accès à une application.

Intents ?

+ Create new intent + Add prebuilt domain intent

<input type="checkbox"/> Name ^	Labeled Utterances
AD.resetAdAccount	4
AD.unlockAdAccount	3
APPACCESS.addApplicationAccess	5
APPACCESS.removeApplicationAccess	6
APPCENTER.requestNewApp	4
CONSUMABLE.tonerReplacement	11
DISTRIBUTIONLIST.addDistributionListMember	14
DISTRIBUTIONLIST.removeDistributionListMember	3

Figure 28 : Liste des intentions enregistrées

Le chiffre dans la colonne « Labeled Utterances » correspond au nombre de phrases enregistrées pour entraîner la reconnaissance de cette intention. Autrement dit, ce sont les phrases susceptibles d'être entrées par l'utilisateur (Figure 29). À terme, il faut au moins 30 utterances pour reconnaître efficacement les intentions de l'utilisateur, même pour des phrases complexes.

FOLDERACCESS.addNetworkFolderAccess

Labelled entities: None








 Edit	 Reassign intent 	 Add as pattern	 Delete	
<input type="checkbox"/> Example utterance	Score 			
<input type="text" value="Enter an example of what a user might say and hit Enter."/>				
could you give access to personName to this folder	0.94			
could you provide full access to personName to the following path	0.94			
give full access read and write also on the subfolders	0.94			
could you give acces to franco to h folder	0.88			
provide both read and write access to the folder	0.96			
the network takes place in the shared network and the folder that i ' m looking is missing	0.93			

Figure 29 : Phrases enregistrées servant à détecter une intention particulière

« Train » signifiant « entraîner » permet à l'IA d'assimiler ces nouvelles intentions et de les reconnaître au moyen des phrases d'entraînement utilisées.

On peut ensuite immédiatement tester des phrases pour vérifier que l'IA reconnaît bien les phrases que l'on souhaite comprendre.

Par exemple, LUIS associe ici la phrase « I want to add another mailbox » (Je veux ajouter une autre boîte mail) à `addAnotherMailbox`, qui est effectivement l'intention de l'utilisateur (Figure 30). On peut voir un chiffre entre parenthèses (0.886) qui correspond au taux de sûreté de l'association. LUIS est sûr à 88.6 % que cette phrase correspond à cette intention.

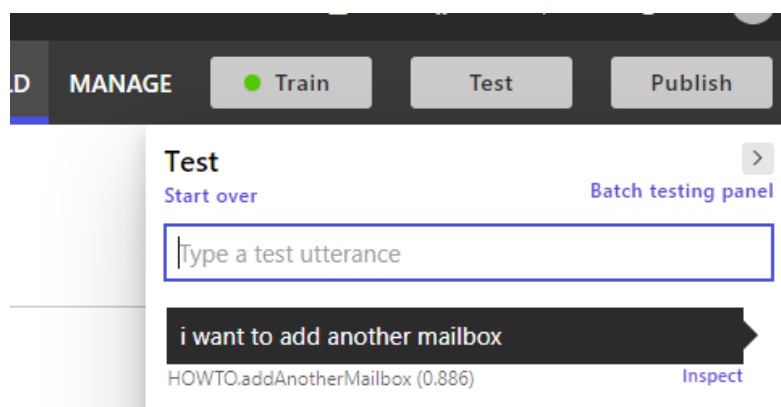


Figure 30 : Possibilité de tester des phrases et de voir le score des intentions détectées

On peut aussi voir le score des autres intentions pour vérifier si d'autres intentions sont proches ou voir le score de l'intention normalement attendue en cas d'erreur d'association (Figure 31).

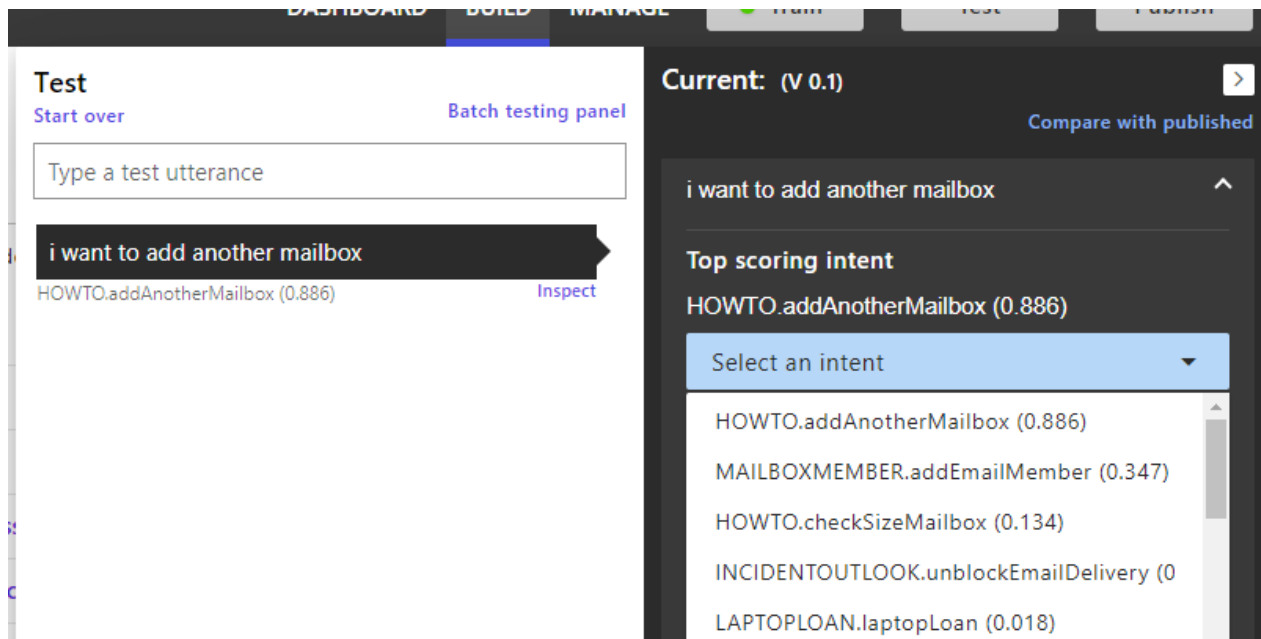


Figure 31 : Liste complète des intentions classées par ordre de score

LUIS propose aussi un certain nombre de phrases dont il souhaite connaître l'association. Il crée donc une liste de phrases en attente de vérification dans le but de s'améliorer lors de la reconnaissance d'intention. Il propose aussi préalablement l'intention qui lui semble la plus vraisemblable dans l'optique d'un gain de temps pour le technicien (Figure 32).

Review endpoint utterances ?

Filter:	AD.resetAdAccount	<input checked="" type="checkbox"/> Entities view
Utterance	Aligned intent ?	Add/Delete
add new mailbox	HOWTO.add... ▼	✓ ✕
i don ' t receive email	EMAILANAL... ▼	✓ ✕
i need to add another mailbox	HOWTO.add... ▼	✓ ✕
add another mailbox	HOWTO.add... ▼	✓ ✕

Figure 32 : Assignment de phrases à des intentions en vue d'affiner la reconnaissance

Il est possible de créer des listes de mots ayant la même signification. Par exemple faire une liste de noms de villes. De cette manière, une phrase d'entraînement traitera de la même manière les différentes villes.

On a utilisé cette fonctionnalité pour créer des listes de mots importants avec des fautes d'orthographe. Il faut en effet ajouter dans les phrases d'entraînement les erreurs de frappe commises par les utilisateurs. Or pour des mots importants comme Outlook, si un utilisateur fait une faute et que LUIS n'y a jamais été confronté, il ne sera pas capable de reconnaître ce mot (Figure 33). Et il est difficilement envisageable de réécrire chaque phrase avec les fautes de frappe pour ce mot important. De cette manière, on écrit une seule fois un mot avec ses erreurs d'écriture communes et on permet à l'IA de reconnaître un plus large rayon de phrases.

Edit Phrase list

Name (Required)

Outlook

Value (Required)

Type comma separated values and press enter ...

Phrase list values

outlook ×	outlook ×	outlookk ×	outlouk ×
outlok ×	outlook ×	ootlook ×	outloo ×
outlook ×	outlok ×	utlook ×	oulook ×
outlokk ×			

Related Values

Add all [Recommend](#)

No suggestions found. Add more values and click 'Recommend' to try again.

Done Cancel

Figure 33 : Création d'une "Phrase list" pour palier au problème des fautes de frappe

Conclusion

L'objectif de ce stage était de me sortir le plus possible de ma zone de confort. J'ai dans un premier temps décidé d'effectuer ce stage au Luxembourg, pays qui se veut être un véritable carrefour de langues et de culture.

Le choix de la mission aussi se voulait être challengeant. HEI étant une école généraliste, je savais que mon axe d'amélioration principal était le côté technique vis-à-vis d'autres étudiants d'écoles plus spécialisées. C'est pour cette raison que j'ai voulu intégrer un département qui privilégie la technicité. De cette manière, j'ai pu apprendre et m'améliorer tout au long de mon stage.

Grâce à des projets et des problématiques majeures, j'ai pu inscrire mon travail dans l'important processus de transformation numérique des entreprises liées à la finance. Au travail informatique se lie toute une sensibilisation des acteurs du milieu à l'importance de cette transformation numérique.

Ce sont donc les compétences que je cherchais à acquérir : renforcer ma maîtrise technique pour être capable de la transmettre. J'espère ainsi pouvoir lier fonctionnel et technique dans mon projet professionnel, ce qui s'inscrit bien dans un diplôme d'ingénieur généraliste.

Les Annexes

Project Showcase

This React Native app will be used to showcase the D.Lab projects, using AR.

Prerequisites

To build the project on IOS, you need to install first :

* [Xcode](https://developer.apple.com/xcode/) - Apple IDE used to build the project

* [React Native](https://facebook.github.io/react-native/)

Deployment

For IOS

To run the project on your physical device, run the command :

```
...  
react-native run-ios --device  
...
```

Make sure your mobile phone and your computer are on the same Wi-Fi network.

The project support Live Reloading. To activate it, shake your phone and select ``Enable Live Reload``.

Documentation and Useful Links

- [Viro Github](https://github.com/viromedia/viro)
- [Viro Documentation](https://docs.viromedia.com/docs)

Annexe 1 : extrait du ReadMe.md

Project Showcase

This React Native app will be used to showcase the D.Lab projects, using AR.

Prerequisites

To build the project on IOS, you need to install first :

- [Xcode](#) - Apple IDE used to build the project
- [React Native](#)

Deployment

For IOS

To run the project on your physical device, run the command :

```
1. react-native run-ios --device
```

Make sure your mobile phone and your computer are on the same Wi-Fi network.

Annexe 2 : Rendu de l'extrait du ReadMe.md

```
closer = (anchor) => {
  const xAxis = anchor.position[0];
  const yAxis = anchor.position[1];
  const zAxis = anchor.position[2];
  const newDistance = Math.sqrt(
    (xAxis - this.cameraPosition[0]) * (xAxis - this.cameraPosition[0]) +
    (yAxis - this.cameraPosition[1]) * (yAxis - this.cameraPosition[1]) +
    (zAxis - this.cameraPosition[2]) * (zAxis - this.cameraPosition[2])
  );
  const oldDistance = Math.sqrt(
    (this.state.currentImagePosition[0] - this.cameraPosition[0]) * (this.state.currentImagePosition[0] -
this.cameraPosition[0]) +
    (this.state.currentImagePosition[1] - this.cameraPosition[1]) * (this.state.currentImagePosition[1] -
this.cameraPosition[1]) +
    (this.state.currentImagePosition[2] - this.cameraPosition[2]) * (this.state.currentImagePosition[2] -
this.cameraPosition[2])
  );
  if (newDistance + 0.01 < oldDistance) {
    return {
      testPassed : true,
      position: [xAxis, yAxis, zAxis]
    }
  } else {
    return{
      testPassed : false,
```



```
position: [xAxis, yAxis, zAxis]
    }
  }
}
```

Annexe 3 : Fonction closer()

Webographie

- 1 - **React Native** : <https://facebook.github.io/react-native/>
- 2 - **Node JS** : <https://nodejs.org/en/docs/>
- 3 - **Expo CLI** : <https://docs.expo.io/versions/latest/workflow/expo-cli/>
- 4 - **Expo AR** : <https://docs.expo.io/versions/latest/sdk/AR/>
- 5 - **React-Native-Arkit** : <https://github.com/react-native-ar/react-native-arkit>
- 6 - **Viro** : <https://docs.viromedia.com/docs> et <https://github.com/viromedia/viro>
- 7 - **Bot Framework Documentation** : <https://docs.botframework.com>
- 8 - **Bot** : <https://docs.microsoft.com/azure/bot-service/bot-builder-basics?view=azure-bot-service-4.0>
- 9 - **LUIS** : <https://docs.microsoft.com/en-us/azure/cognitive-services/luis/>
- 10 - **Dialogues** : <https://docs.microsoft.com/en-us/azure/bot-service/bot-builder-concept-dialog?view=azure-bot-service-4.0>