MEMOIRE PROFESSIONNEL

MANAGER EN INGÉNERIE INFORMATIQUE Spécialité Développement

PERNON Etienne

DEVELOPPEUR

Entreprise : PICC DEVELOPPEMENT

Adresse : 3 rue des Cigogne Entzheim

Tuteur : Fulhaber Simon

Fonction du tuteur : CTO

Mémoire conforme : ☐ OUI ☐ Non

Mémoire relu et validé par le tuteur_____

Date, cachet de l'entreprise et signature du tueur :

Résumé

Table des matières

1. Glossaire

Ce dictionnaire liste tous les termes technique et acronyme présent dans ce mémoire professionnel. Découpé par domaines il vise à comprendre le vocabulaire spécifique due à L'entreprise PICC Solution mais aussi plus largement celui du monde du management de l'entreprise et de l'ingénierie de l'informatique.

1.2 PICC Solution

Approche systémique

Méthode de réflexion scientifique basé sur....

Intelligence collective

Méthode de gestion de la connaissance basé sur...

Buisness Act Grand Est

SRDEII

KMAP

Concept

Entités

Domain

know-how management

Traduction littérale gestion du savoir-faire. Ensemble des techniques permettant l'agencement, la structuration et la mise à disposition des connaissances métier de l'entreprise. Cela comprend les KMAP, l'architecture problème-solution, l'utilisation de matrice Lean comme le RASCI.

1.1 Management d'entreprise

CEO

Acronyme anglophone désignant le Chef Executif Officer, c'est-à-dire la personne qui prend les décisions stratégie dans l'entreprise.

CTO

Acronyme anglophone désignant le Chef technique Officer, c'est-à-dire la personne qui a la responsabilité des choix technique de l'entreprise.

Industrie manufacturière

Largue sous ensemble de du secteur secondaire, ce type d'industrie vise à la fois les sites de transformation de bien que les entreprises de réparation et d'installation d'équipement.

Industrie 4.0

4ème révolution du monde industriel après l'automatisation machine, l'industrie 4.0 consistent en interconnexion des machines à des systèmes d'informations complexe. Elle vise à optimiser les moyens de productions en utilisant l'internet des objets, la robotique, la réalité augmentée, l'intelligence artifice et les Saas afin d'exploiter les données du big data.

Lean management

Matrice RASCI

Framework AGILE

1.2 Intelligence Artificiel

Intelligence artificielle

LLM

RAG

Agent IA

1.3 Développement de logiciel

Saas

Pour Software As A Service ou Logiciel en tant que service c'est un type d'application qui n'est pas installé sur machine d'utilisateur mais sur un serveur distant.

Big data

Désigne un ensemble de données qui par son volume nécessite d'être traité par des moyens numérique.

MES

Plateforme IoT

Single Page Application

Frontend / client

Backend / server

folder by feature structure

Rule Of One

TypseScripte

modules loader

Angular Materials

Dette technique

CLI

compilation

Linting

Application monolithique

bootstrap

minification

2 Introduction

2.1 Contexte général et choix de l’entreprise

2.2 Présentation du cadre professionnel

2.3 Objectif de l’alternance

2.4 Présentation des missions réalisé

2.5 Structure du mémoire

3 Présentation de l'entreprise

3.1 Présentation globale de l'entreprise.

3.1.1 Historique et contexte général

PICC solution est une entreprise Suisse crée en 2020[1]. Son histoire débute 15 ans plutôt avec un consortium européen réunissant Arcelor Mittal, Alstom et le gouvernement français. Motivé par le constat que l'Asie n'est plus seulement le lieu de la production industriel mais aussi celui de la recherche, il devenait urgent d'aider les entreprises européennes à innover plus facilement. Pour ce faire deux principes pour la génération d'idée vont être établit : le premier est que le moyen le plus efficace serait de travailler avec des personnes d'horizon et de culture différentes ; le second est que les idées devraient être sélectionné via des procédés scientifique et non biaisé par le charisme ou l'influence des personnes qui les ont portés.

Une équipe de 20 personnes composé de sociologues, psychologues et d'informaticiens est alors formé, dont Simon Fuhlhaber actuel CTO de PICC Solution, pour réfléchir à une approche systémique et non biaisée de l'innovation. Le résultat de cette cherche fut l'ébauche d'un logiciel qui sera racheté par les actuelles dirigeant de PICC solutions, à savoir Contant Ondo CEO et Simon Fuhlhaber CTO. C'est ainsi que l'entreprise PICC pour Privet Innovation Compétence Center et plus tard sa branche française PICC développement basé dans le Grand Est furent penser pour améliorer la compétitivité des entreprises.

3.1.2 Domaine d'activité

Aujourd'hui, l'entreprise PICC agit sur plusieurs domaines d'activité à commencés par la gestion de connaissances industrielle[2]. Pierre angulaire de son activité la connaissance et sa gestion permettent d'anticiper et de prendre des décisions rapidement. Pour ce faire les données doivent être : décloisonner et accessible dans un même system de gestion centralisée ; fiable car faisant partie d'une stratégie de management de la donné impliquant des processus de nettoyage, de vérification et de mise à jour des données ; valorisé par un system de recherche efficace permettant le croissement et l'analyse, ainsi que transformable en indicateur, prévision ou alerte.

Face à la masse d'information récolté et à leurs complexité, l'intelligence artificiel est devenue un outil incontournable pour traiter et analyser ces ensembles de données. Cette technologie ne se contente pas seulement d'automatiser l'analyse de grand volume

[1] MSM. (2023, septembre 22). *L'intelligence collective au service des entreprises*. https://www.msm.ch/lintelligence-collective-au-service-des-entreprises-a-92c9b99b23c357114bd90d6e81d7086b/

[2] PICC Solution. (n.d.). *Gestion des données : un levier incontournable de la transformation digitale*. https://www.picc-solution.com/fr/gestion-donnees-transformation-digitale/

de données mais propose également leur restructuration via l'approche problème-solution. Cette méthode permet de proposer des actions concrètes qui seront validé par l'entreprise.

Enfin, PICC se concentre sur l'intelligence collective. L'objectif est simple, donner la possibilité aux collaborateurs de se concentrer sur les activités qui génère les plus de valeurs ajoutées pour l'entreprise. Dans la pratique c'est accéder aux savoir et aux savoir-faire de l'entreprise, être guidé dans la résolution de problème complexe, recevoir une assistance à la prise de décision, anticipé et résoudre les problèmes en proposant des solutions et enfin valoriser, capitaliser les retours d'expériences.

3.1.3 Positionnement sur le marché

Les entreprises manufacturières en pleine reconversion vers une industrie 4.0 sont le cœur du marché visé par PICC. Ces sociétés montrent un besoin croissant de dans le développement des outils informatiques au seins même de leurs chaines de productions. PICC s'inscrit comme un intermédiaire capable d'accélérer et d'optimiser la mise en place de solution lourde tel que les MES qui sont aujourd'hui indispensable pour leurs capacités à analyser en temps réel les données de production et a alerté en cas de problème[3]. Aujourd'hui des entreprises comme Berry plastiques ont pu accélérer leur virage vers l'industrie 4.0 en combinant gestion du savoir-faire technique, automatisation des processus, plateforme IoT et aide à l'innovation. En effet, la plateforme d'intelligence collective sert de complément aux services MES pour aider les responsables de performance industrielle à bâtir un plan d'action, notamment lorsqu'il se charge de plusieurs sites en même temps.

Dans une autre mesure PICC se positionne aussi au côté de collectivité territorial[4] comme le Grand Est pour traiter de grande quantité d'information afin prendre des décision. Lors de l'initiative « Buisness Act Grand Est » ce sont 22 groupes de travail pour un totale de 600 personnes qui ont proposé des milliers de pages de rapport portant sur l'innovation après d'après COVID-19. PICC a alors pu analyser l'ensemble des contributions pour trouver 4 000 problèmes, 5 000 solutions et 14 000 relations autour des quatre thématiques suivantes :

- Le numérique
- L'accompagnement
- L'écologie
- La gouvernance

[3] PICC Solution. (n.d.). *Industrie 4.0* https://www.picc-solution.com/fr/utiliser-les-donnees-des-capteurs-iot-dans-une-plateforme-dintelligence-collective/
[4] PICC Solution. (2023, juin 5). *Ils utilisent PICC : La Région Grand Est*. https://www.picc-solution.com/fr/ils-utilisent-picc-la-region-grand-est-sappuie-sur-lia-pour-elaborer-son-srdeii/

Ces analyses ainsi que le suivie des réunions de concertations et les SRDEII de toutes les régions françaises on permit de mettre en lumière 70 millions de liens au travers de cette base documentaire. Dans un premier temps l'analyse documentaire automatique a permis d'identifier les sujets importants à traiter. Puis dans un second temps, l'analyse qualitative a permit de mettre en lumière des écarts de perception dans l'importance des effets potentiel que pouvait avoir chaque orientation stratégique.

3.2 Technologie de PICC Solution

3.1.1 Gestion et structuration de l'information

Les informations sont tout d'abord structuré dans un grand ensemble appelé carte des connaissances pour Knowledge Map ou KMAP. Cette forme particulière de Mind Map permet aux utilisateurs de noter leurs idées sur un canevas en posant des concepts, c'est-à-dire des Problèmes, des Solutions ou des Entités.

L'architecture de ces KMAP incite l'utilisateur à structurer sa réflexion en problèmes et en solutions ce qui permet de synthétiser les savoirs faire complexe. La réflexion se porte souvent sur un problème central puis évolue vers des sous problèmes, les solutions représentent alors des indications, des actions ou des pistes de réflexions qui peuvent à leurs tours entrainer de nouveaux problèmes.

L'écriture de cet arbre de possibilité n'est pas libre et demande de suivre une certaine logique. En effet, si un problème majeur peut crée une multitude de problème connexe ; une solution ne peut pas être la source d'une autre solution, c'est une erreur de logique car il doit nécessairement avoir un problème entre les deux solutions.

Au milieu de ces concepts abstrait, les entités quant à elles permettent de représenter le monde réel. Allant d'une personne dans l'organisation à une machine de la chaine de productions, elles viennent connecter les différentes composantes d'une entreprise entre-elles. Un cas simple serrait d'associer à une solution l'utilisation d'une procédure internet de l'entreprise, l'appel d'un membre particulier du personnel ou la vérification d'un composant d'une machine.

L'avantage et la puissance de PICC résident également dans la génération automatique de KMAP à partir de document d'entreprise. La lecture de sources permet d'extraire des problèmes, solutions et entités, d'identifier des domaines pour catégoriser ces concepts et enfin relier ces informations au sein d'un KMAP.

3.1.2 Technologie d'intelligence artificiel utilisé

Tout d'abord PICC utilise les modèles LLM (Largue Langage Modèle) comme gemma 3 pour comprendre le langage naturel utilisé par les opérateurs, technicien ou

ingénieurs. Ces modèles permettent d'interpréter des questions ouvertes, des descriptions informelles de problèmes, ou des demandes complexes exprimé avec un vocabulaire spécifique à l'entreprise.

Combiné à la méthodologie RAG pour (Retrival Augmented Generation), cette technologie permet la recherche d'information dans une base de données et de générer des réponses à partir de ces informations via un LLM. Cela permet à PICC de retrouver des cas similaires en matière de panne, solutions et incidents passés mais aussi de fournir des réponses contextualisé, validée et documenté, ce qui limite grandement les hallucinations inhérentes aux modèles de génération de texte.

Ces moyens ont permis d'améliorer grandement la structuration en problème solution en facilitant l'interconnexion entre un nouveau problème et une solution déjà existante. En comparant les problèmes nouveaux avec ceux rencontrés dans le passé la technologie RAG permet de renforcer la fiabilité des recommandations et l'apprentissage collectif de l'entreprise.

Enfin, l'exploitation des données industriel et l'IoT permettent un enrichissement conséquent de la base de données proposée aux modèles LLM. Ces données (Température, vitesse, pression) corrélées aux évènements et aux retours d'expérience (intervention, baisse de qualité) permettent de proposer des diagnostiques précis et des recommandations proactives.

3.2 Structure interne de l'entreprise

3.2.1 Organisation interne

PICC est une petite start-up comptant moins de 10 collaborateur actif simultanément, ces chiffres variant selon les besoins. A l'exception des alternants et des dirigeants, tous les contrats sont des prestations de services impliquant des auto-entrepreneur ou des micro-entreprise.

Il est possible de distinguer 4 types services au sein de PICC. La commercialisation s'occupant de trouver de nouveau acheteur, partenaire ou opportunité pour l'entreprise. La gestion de projet qui organise le suivie des entreprises partenaire, comprenant la prise en main du logiciel, la réalisation de tâche complexe ou spécifique sur le logiciel ou l'élaboration de nouveau procédé pouvant advenir à la commande de nouvelles fonctionnalités. Le développement logiciel qui s'occupe de l'amélioration continue, de l'optimisation et du développement des nouvelles fonctionnalités. Enfin, la direction stratégique définie les axes sur lesquelles la solution doit s'orienté en fonction de la demande de nos partenaires et de l'évolutions du marché.

La solution PICC dispose de ses propres outils de gestion de projet développé en internet et intégré à notre logiciel. Utilisé à la fois par nos partenaires et par nos équipes, ils nous permettent l'organisation et le suivie de projet de taille conséquente. Basé sur des préceptes du Framework AGILE issu et le Lean management PICC associe l'utilisation d'un Kanban avec un tableau de suivie ainsi qu'un module d'analyse statistique.

La Kanban nous permet d'organiser nos tâches quotidiennes en limitant le nombre de tâche simultané. Il nous permet d'améliorer la visibilité des tâches et d'identifier rapidement les blocages. Il intègre aussi un mécanisme de validation entre chaque étape obligeant à compléter une procédure pour passer tâche dans la catégorie suivante.

Le tableau de suivie permet d'observé l'avancement des tâches d'un projet de manière plus précise en apportant un niveau de détaille supplémentaire sur les ressources affectées ou les problèmes liées.

Ces outils sont directement présents à l'intérieur des KMAP et interagissent directement avec tous les concepts qui y sont renseigné ce qui permet l'intégration de la logique problème solution au sein même de la gestion projet, ainsi qu'une visualisation graphique des projets.

3.2.2 Exemple de méthodologie de pilotage

Comme précisé précédemment PICC fonctionne, pour ce qui est du développement logiciel, avec des petites équipes indépendantes, celle-ci bénéficie alors d'une grande autonomie. Voici donc un exemple de pratique de pilotage au sein des équipes de développement :

- **Réunion quotidienne** *de 15 – 20 minutes (2 personnes)*, permettant de redéfinir les tâches quotidienne et d'identifier les blocages.

- **Revue hebdomadaires** le lundi *30 minutes – 1 heure (3-6 personnes)*, pour prévoir les tâches de la semaine.

- **Revue hebdomadaires** le *vendre 30 minutes – 1 heure (3-6 personnes)*, pour rendre compte tâches réalisé.

- **Session de pair programming** ponctuel *1h – 1h30 (2 personnes)*, pour faire avancer les blocages identifier en point quotidien

4 Mission réalisées

4.1 Mise en place des entités personnalisé

4.2 Refonte du module de procédure

La normalisation des procédure ?

4.3 Transition vers angular18 : mise en place d'un application hybride

4.3.1 Définition de la mission

4.3.1.1 Contexte général

AngularJS est un framework javascript crée en 2010 par Misko Hevery qui est alors ingénieur chez Google[5]. Pendant les 6 prochaine années cette outils sera massivement utilisé par des millions de sites web au point qu'il deviendra un incontournable des Single Page Application[6]. Cependant pour résoudre des problèmes de structure qui freinent sont intégration à des projets plus récent le framework est recrée de zéro avec Angular2.0 en 2016.

Depuis le 31 Démembre 2021 l'entreprise Google, qui avait montré un rôle actif dans le développement d'AngularJS, annonce officiellement la fin de son support. Cela implique[7] :

- **Des vulnérabilités de sécurité**, les nouvelles failles ne seront plus corrigé.
- **Des problèmes de compatibilité**, les nouvelles versions des navigateurs pourrait ne plus être compatibles avec AngularJS qui nécessiterait une mise à jour rapide.
- **Amoindrissement du support communautaire,** on peut s'attendre à une baisse significative du nombre de développeur sur cette technologie diminuant l'entre aide.
- **Absence de nouvelle fonctionnalité**, les nouvelles librairies ne serons pas compatible AngularJS

Cette annonce est un problème stratégique majeur pour l'entreprise PICC qui base tout sont Frontend sur la technologie AngularJS. Au totale c'est 3 application pour un peu plus d'une dizaine de module qui seront affecté.

Le maintien d'AngularJS freine la modernisation de l'architecture logicielle tel que la Rule Of One ou folder by feature structure qui sont des concepts moderne facultatif devenu obligatoire dans l'architecture Angular2+. C'est aussi l'absence de nouveau outils tel que TypeScript ou l'utilisation de modules loader. Enfin, c'est un frein au développement des compétences des équipes qui travail sur une technologie qui a déjà périclité.

---

[5] Amiltone. (n.d.). *Angular et le développement web*. https://www.amiltone.com/tech-place/angular-et-le-developpement-web

[6] Ambient IT. (n.d.). *Statistiques Angular*. https://www.ambient-it.net/statistiques-angular/

[7] Uzinakod. (2021, janvier 26). *Google abandonne AngularJS*. https://www.uzinakod.com/blogue/google-abandonne-angularjs

Nous avons donc conclu à la nécessité d'une migration pour garantir la pérennité des applications au long terme. En outre cela nous permettra de : moderniser l'interface utilisateur en utilisant les dernière version d'Angular Materials ; facilité la maintenabilité et l'évolutivité grâce à un code mieux structure et l'intégration de test unitaire ; réduire notre dette technique[8].

4.3.1.2 Objectif de la mission

L'objectif principale de la mission est de moderniser l'application existante en migrant le code source historiquement développé sous AngularJS vers une version Angular18. C'est aussi s'inscrire dans une stratégie de refonte progressive et non de réécriture totale pour limiter le risque et l'impacte sur la continuité du développement. En remplaçant progressivement les composants AngularJS par des composants Angular modernes incluant typescript et les nouvelles librairies nous profitons des nouvelles options de compilation et de Linting du CLI sans affecter notre code existant.

Le deuxième objectif est de garantir la continuité du service et de son développement pendant toute la migration. Durant toutes les étapes de la migration de nouvelles fonctionnalités doivent continuer à sortir, il serait impossible de figer le logiciel dans une version spécifique en attente d'une restructuration aussi globale. De plus la nouvelle version de cette architecture doit rester compatible avec l'architecture server couramment utilisé et qui n'a pas pour but de changer.

4.3.1.3 Présentation de l'équipe

Nous sommes deux à être chargé de cette mission, Simon Fulhabert qui un rôle du supervision et moi-même avec le rôle de chef de projet. Ensemble nous définie les objectif principaux de cette mission et validé chaque étapes de sa réalisation.

Les responsabilités qui m'incombent personnellement sont :

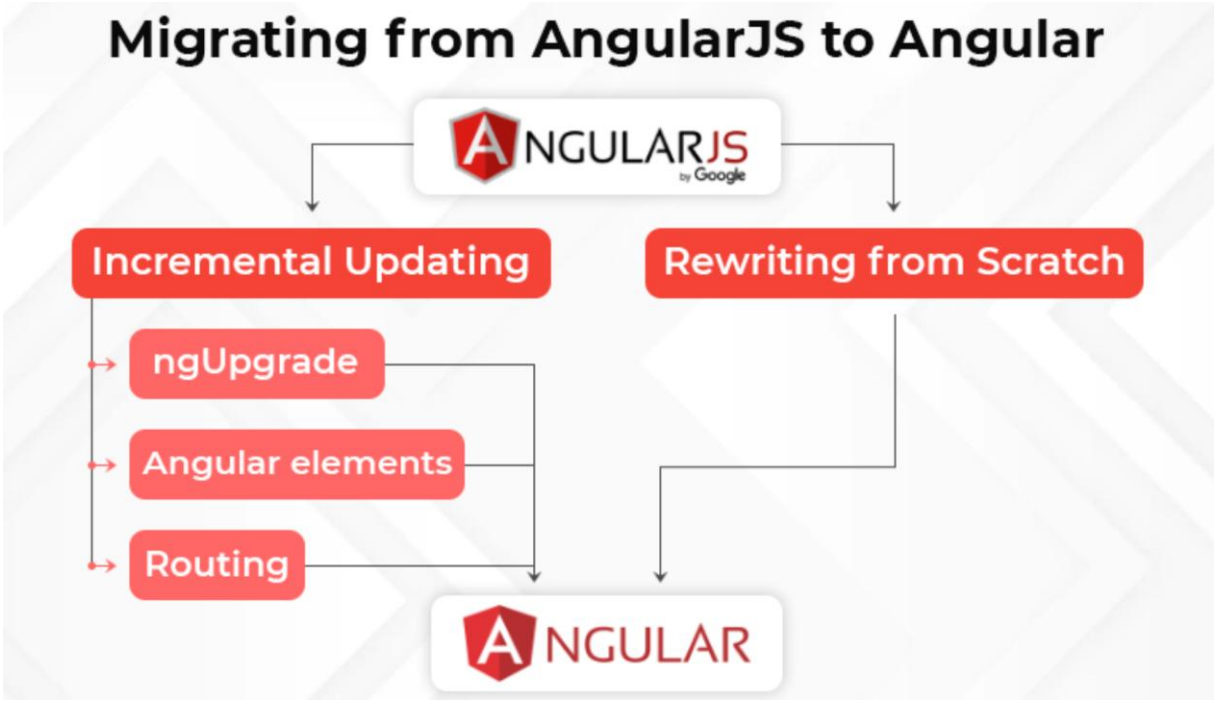
- La recherche sur les différentes méthodes pour mettre en place cette migration.
- La gestion des outils de suivie de projet
- La réalisation des stratégies migration
- La rédaction de documentation pour le suivie du développement sous Angular2+

[8] Pérez, J. (2021, janvier 11). *AngularJS end-of-life is here. Now what?* Medium. https://medium.com/@javperezp79/angularjs-end-of-life-is-here-now-what-bd7961eb19b4

4.3.2 Analyse de la mission

4.3.2.1 Choix technique et justification

Méthode de migration



SCHEMA DES DIFFERENTES METHODE DE MIGRATION VERS ANGULAR

Deux scénario sont possible lorsque l'on aborde une migration. La première, consiste en une réécriture complète de l'application, repartir à zéro avec Angular18 sans conserver aucun code AngularJS. Cette méthode a l'avantage :

- De fournir un code plus propre en supprimant le code obsolète
- De proposer un code moderne qui respect les standards Angular.
- D'utiliser les dernières librairies
- De repenser l'architecture de l'application
- D'améliorer les performance globale de l'application

Evidemment, cette méthode bien qu'elle permette de supprimer toute dettes technique présente de sérieux inconvénient puisque :

- Le cout en ressource humaine est très élevé et ne fait qu'augmenter avec la taille de l'application.
- Certaines fonctionnalités peuvent être difficiles à recoder dans le nouveau Framework
- Certaines fonctionnalités peu utilisé peuvent être oublié
- Deux versions de l'application sont à maintenir simultanément pendant une durée indéterminé
- La capacité de développement et de réactivité en cas de bug se retrouve divisé, ce qui n'est pas idéale dans les petites structures
- Les nouvelles fonctionnalités devront subir un double parcourt de développement

Ainsi, cette stratégie n'est pas recommandée pour les applications de grandes taille, notamment lorsqu'elles sont monolithiques. Cette méthode peut paraître brutale, cependant elle prend sens puisque l'effort humain cumulé sur tout la migration n'est pas plus important que pour la migration incrémental que nous verrons ci-dessous.

La deuxième approche est une stratégie de mise à jour incrémental. Elle consiste à faire coexister le Framework AngularJS et Angular2+ an sein de la même application. Cette approche est permise car la fondation Angular a développé un module d'upgrade et de downgrade permettant la cohabitation et la compréhension mutuel des deux Framework. En somme il permet à une application Angular2+ d'afficher le composant d'une application AngularJS et inversement. Cette méthode permet de :

- Conserver l'intégralité des fonctionnalités de l'application tout en s'assurant un minium de régression
- Limiter L'apparition de bug ou de comportement inattendu liée à la migration
- Réduire le risque technique puisque les équipes on le temps de se former au nouveau Framework au fur et à mesure des nouveau composant qui sont créé ou migré pour Angular2+

Cette méthode permet de conserver l'intégralité des fonctionnalités de l'application tout en s'assurant un minium de régression, de bug ou de comportement inattendu liée à la migration. C'est une réduction du risque technique, puisque les équipes on le temps de se former au nouveau Framework au fur et à mesure des nouveau composant qui sont crée ou migré pour Angular2+.

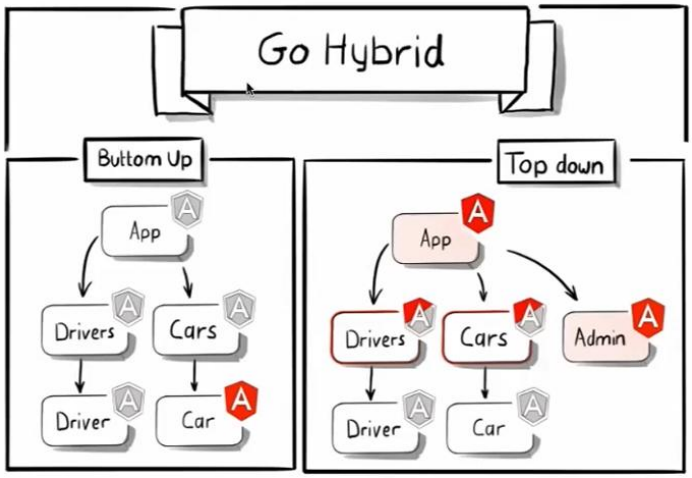
Par ce mécanisme d'hybridation la migration est plus fluide car elle ne nécessite pas dans un premier temps de réécriture du code. Cependant, cette stratégie s'accompagne d'une complexité plus élevé. La mise en place d'un double bootstrap et la communications des deux Framework nécessite une compréhension rigoureuse des deux environnements, ce qui peut demander une monté compétence. De plus, cette méthode nécessite que le code AngularJS se rapproche autant que possible du Style Guide Officiel de John Papa[9], si cela n'est pas le cas cette approche demandera une étapes supplémentaire de mise à niveau des bonnes pratique. Ainsi, la phase intermédiaire visant à rendre la solution hydride opérationnel peut s'avérer assez longue.

Enfin et bien que la fondation Angular ai fait de nombreux effort d'accompagnement pour aider à la mise en place de cette solution, les projet open source ayant réalisé ces étapes de développement sont peut nombreux et l'on peut supposer que la plupart des entreprises ayant réalisées leurs transition de cette manière gardent leurs code privé.

Notre choix portera donc sur la migration incrémental via les modules Angular Upgrade et Angular Downgrade. Cette stratégie est en alignement direct avec nos objectifs puisqu'elle nous permet de conserver une continuité dans le développement des fonctionnalités de PICC. Aussi, notre logiciel respect également certaines des recommandations faites dans Style Guide Officiel ce qui devrait alléger la phase intermédiaire de mise en place de la solution hybride. Enfin, l'application est un monolithe AngularJS de taille conséquente, la réécriture complète s'avèrerait bien trop longue doublé du risque de ne jamais arriver en bout de projet et de conserver un retard perpétuel sur l'application historique.

[9] Papa, J. (n.d.). *Angular style guide*. GitHub. https://github.com/johnpapa/angular-styleguide

Il existe deux méthodes pour réaliser son hybridation comme illustré sur le graphique ci-dessous[10] :



SCHEMA DES DIFFERENTS METHODES D'HYBRIDATION

La première stratégie dit Bottom-Up est une migration progressive des composants AngularJS vers Angular2+ en partant des compostant les plus bas dans l'arborescence vers les composants parent. Cette méthode permet de convertir en premier les composants les plus isolé, sans affecter l'ensemble de l'application. Elle présentera donc moins de complexité initial car elle nécessitera moins de changement de prime abord. Cependant, cette stratégie induit une dépendance accrue à AngularJS, il sera par conséquent plus difficile d'intégrer un system de routage uniforme entre les deux Framework.

La deuxième stratégie dit Top-Down consiste à commencer par le composant racine et d'encapsuler le reste de l'application AngularJS dans une application Angular2+. Cette méthode permet d'ajouter immédiatement de nouveau composant entièrement écrit avec Angurlar2+ tout en bénéficiant de toutes les nouvelles fonctionnalités apporté par ce Framework. Cela oblige la mise en place d'une architecture plus moderne et réduit les dépendances vers AngularJS. Cependant, cette stratégie nécessite des ajustements important en vue de notre architecture :

- **Rule of One** : découper l'application pour qu'un fichier soit liée à une seul et unique composant, service, module ou factory.
- **Class Component** : intégrer les principes de la POO *(Programmation Orienté Objet)*

[10] Pixel Perfect. (2023, octobre 4). *Migration d'AngularJS vers Angular – La méthode complète* [Vidéo]. YouTube. https://www.youtube.com/watch?v=lYOHB_yTEmo

- **Abandonner** le système d'importation Globale Namespace au profit du système de module introduit avec ECMAScript 2015 (ES6)

La différences entre les méthodes se résume à : qu'elle application encapsule l'autre ? Es ce que le point de départ et donc l'architecture qui en découle est une application Angular2+ ou AngularJS.

En considérant le projet initial, nous avons tout d'abord longuement penché pour la solution Bottom-up car bien plus simple et rapide à mettre en place. En effet, les efforts à mettre en place pour uniquement préparer la mise en place de l'application hybride Top-down se compte en mois durant lesquelles :

- Un nouveau projet devra être crée pour la nouvelle architecture
- Un changement majeur viendra interrompre la continuité entre les deux projets
- Le nouveau projet devra rattraper le retard accumulé sur l'anciens depuis le changement majeur.
- Des librairies devront être mise à jour impliquant potentiellement des changement dans leurs API.
- Des Tests manuel sur toutes les fonctionnalités de l'application

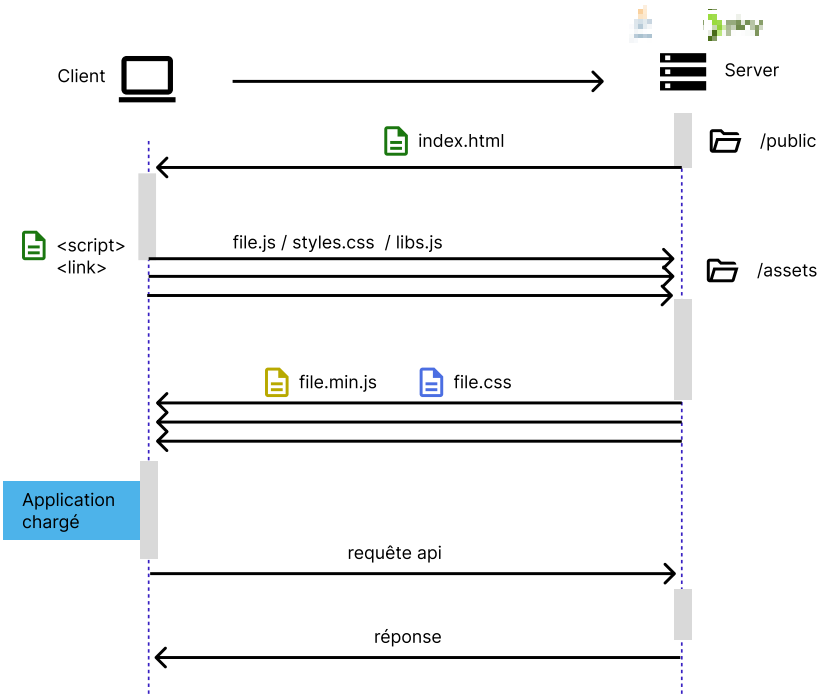
Cependant, la stratégie Bottom-Up n'est privilégier que par 10% des entreprises. Pour cause, les efforts tout de même conséquente mis en œuvre ne val pas les faibles gain apporté par un ponctuel ajout de quelque composant Angular2+ dans une application qui continuer de se reposer sur un FrameWork non maintenu.

Nous avons donc fait le choix de prendre le temps de repenser la structure de l'application malgré les efforts important cité plus haut.

4.3.2.2 Méthodologie mise en œuvre

Phase de recherche et analyse de la nouvelle structure

Nous avons tout d'abord commencé nos recherche par une analyse des projets regroupant un Frontend Angular2+ et un serveur Java utilisant le Play Framework. L'enjeux était de comprendre la structure et les mécanismes d'intégration entre le Frontend et le Backend, ainsi que les comparer avec la solutions existante. Notre choix s'est résolu sur la angular-play-java-seed, un projet Github de 8 ans d'âge qui nous a fallu mettre à jours[11]. Les schémas suivant montre comment l'application Web est distribué par le server dans la solution historique et dans la nouvelle structure.



VERSION HISTORIQUE : DIAGRAMME DE SEQUENCE MONTRANT COMMENT L'APPLICATION ANGULARJS EST SERVIE

Comme explicité dans le diagramme de séquence ci-dessus, la solution historique renvoyait un fichier index.html contenant toutes les balises scripts pour tous les fichiers JavaScript de l'application. Ceux-ci étaient demandé au serveur de manière unitaire à la réception du fichier index.html, il en va de même pour fichier CSS. Une fois

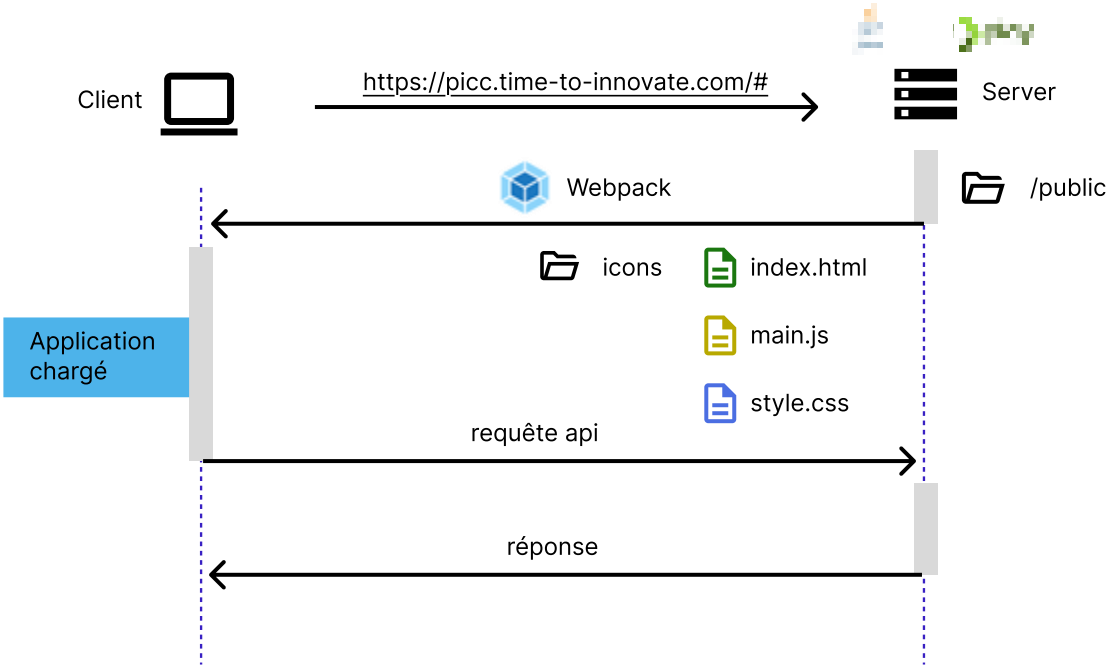
[11] playframework. (n.d.). *play-scala-angular-seed* [Code source]. GitHub. https://github.com/playframework/play-scala-angular-seed

cette transaction terminée et les premiers template HTML reçut, l'application était chargée et disponible.

Ce mécanisme implique d'avoir un dossier statique nommé /public dans lequel sont rangé toutes les ressources HTML de l'application. Les fichiers JavaScripts et CSS sont quant à eux ranger dans un dossier /app/asset au chevet des fichiers Java sans réel distinction entre les deux application Client et Server.

Le chargement du Framework AngularJS se fait via des balises Script qui chargent les librairies principales et optionnels du Framework. Cette méthode se repose sur le **Globale Name Space pattern**, qui permet une mise en place simple et une disponibilité dans tout les navigateur même les plus anciens mais induit une absence de gestion des dépendances et de l'isolation des modules.

Ainsi l'application AngularJS ne nécessite pas de compilation au préalable, tout les fichiers javascripts sont chargé en bloc dans le fichiers index.html, seul une minification est effectué lorsque le serveur est utilisé en production. Il en va tout autrement du beau[12].



En effet, la nouvelle solution intègre des outils plus complexes impliquant des étapes de compilation, d'optimisation et de paquetage qui modernise la solution. Ces

[12] Kant, I. (1790). *Critique du jugement* (paragraphe 7) [PDF]. https://www.ecolpsy-co.com/download/Kant_Critique_du_Jugement.pdf

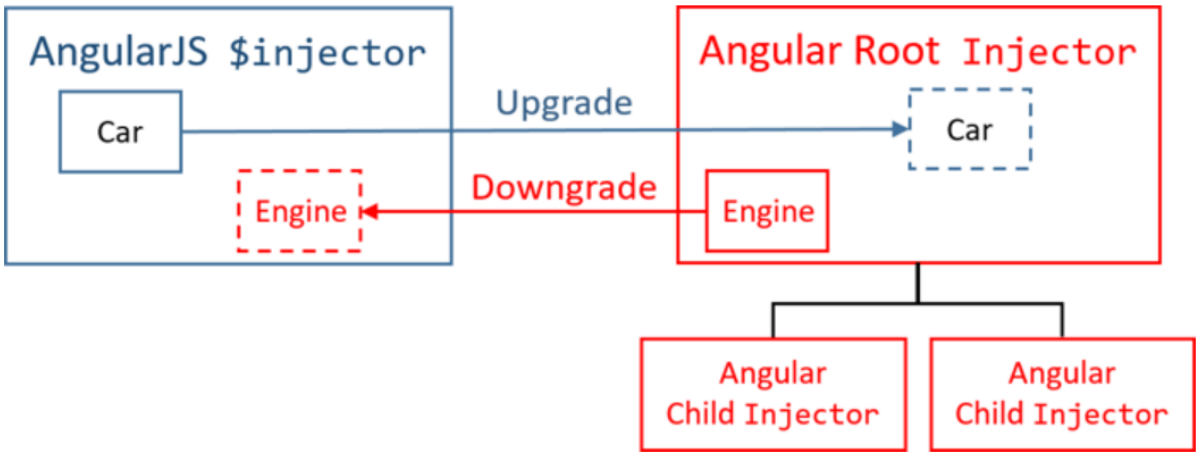
changement commence par la structure puisque le code de l'application Frontend est contenu dans le dossier /ui, ce qui en fait une application à part entière intégrant une gestion des sources externe via NPM.

De plus, le dossier /public est à présent temporaire, il est auto-générer au lancement de la commande STAGE (sbt shell). Cela implique des changements de structure dans la manières de ranger les fichiers statique. Ce dossier /public comprend les ressources de l'application Angular2+ comme les images et les fonts, le point d'entré index.html mais aussi l'application Angular2+ et ses dépendances, compilé, minifié et paqueter en 3 fichier javascripts : main, polyfills et runtime.

Test des différents mécanisme d'hybridation (stratégie shell)

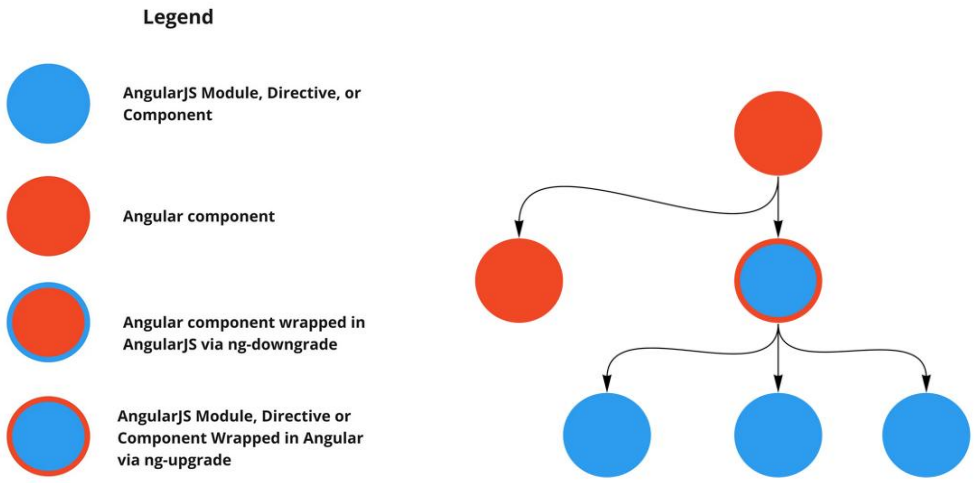
Nous avons testé tous les différents mécanisme d'hybridation possible ce qui nous a permis à terme de définir la méthodologie la plus adapté à notre projet. Pour cela il nous a fallu comprendre comment fonctionne la librairie UpgradeModule fournie par Angular.

Lorsque nous utilisons ce module, les deux Framework cohabite dans la même application. En effet, le code AngularJS tourne à l'intérieur de son propre Framework et vise vers ça pour le code Angular. Cela implique qu'il n'y a pas de perte de fonctionnalité puisque le Framework AngularJS n'est pas émulé, il est toujours présent, exécute son propre code et gère les modules qui lui sont assigné.



SCHEMA DE L'INJECTION DE DEPENDANCE DANS UNE APPLICATION HYBRIDE

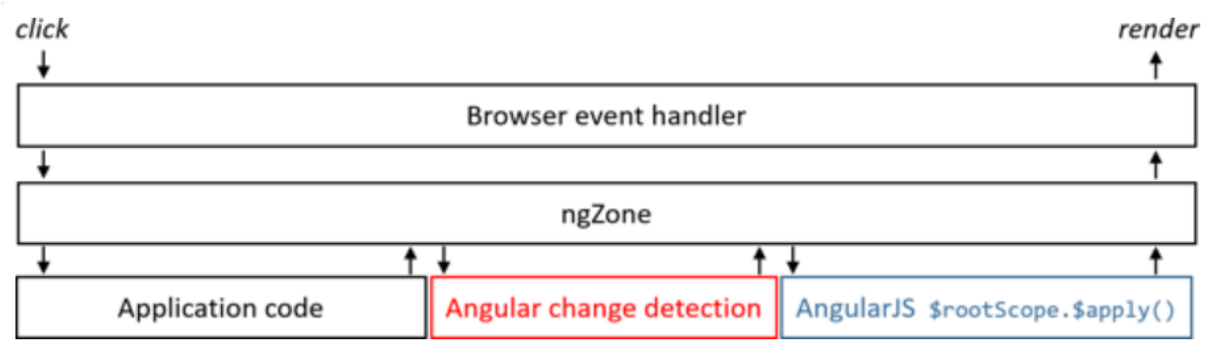
Comme vous pouvez les voir sur le graphique ci-dessus les composants et les services gérer par un Framework peuvent également interagir l'autre. Ce mécanisme n'est pas limitant, sa force est de permettre l'utiliser des nouveaux service dans les deux sens. Même si un composant n'est pas immédiatement migré il peut tout de même bénéficier des services nouvellement crée avec Angular. De plus et dans le cas d'un Singleton la même instance de ce service sera partagée entre les deux Framework.



HIERARCHIE DES COMPOSANTS DANS UNE APPLICATION HYBRIDE

Cette interpolation se retrouve aussi dans la gestion du **DOM** où chaque template html est géré par un Framewrok tandis que l'autre l'ignore complètement. Par exemple, si un composant Angular est inséré dans un template AngularJS, ce dernier contrôle l'élément hôte, tandis que Angular gère le contenu interne. Cela implique que les directives ou fonctionnalités spécifiques à un Framework ne s'appliquent qu'aux éléments possédés. L'utilisation de ce mécanisme demande une certaine attention du à ces particularités :

- Les directives html applicable à un composant seront toujours celle du Framework liée au composant parent ce qui demande une certaines adaptabilité dans l'écriture du code.
- Les libraires d'UI comme Materials design sont spécifiques à un Framework ce qui amène nécessairement des disparités dans l'affichage de l'application.



MECANISME DE DETECTION DES CHANGEMENTS DANS UNE APPLICATION HYBRIDE

Pour compléter l'interaction entre les deux Framework une application hybride a également besoin de détecter les changements provenant des deux parties et de synchroniser l'exécution leurs code. L'Astuce qui rend ce mécanisme possible résident dans le fait que tout le code est exécuté de la zone Angular qui s'assure de la cohabitation. Après chaque évènement Angular déclenche sa propre détection de changement, par la suite l'UpgradeModule appel par lui-même la détection des changements propre à AngularJS. Ainsi, c'est le Framework Angular qui capte tous les évènements provenant du navigateur et s'assure de transmettre a u workflow du Framework concerné. Cependant, cette méthode ne va pas sans un défaut majeur, cette aller et retour entre les deux Framework est une perte indéniable de temps dans le cycle de l'application. Nous nous retrouvons, à ce stade, contraint à avancer dans le projet tout en sachant que des tests devrons être fait pour s'assurer que les performances ne seront pas impactées par ces changements.

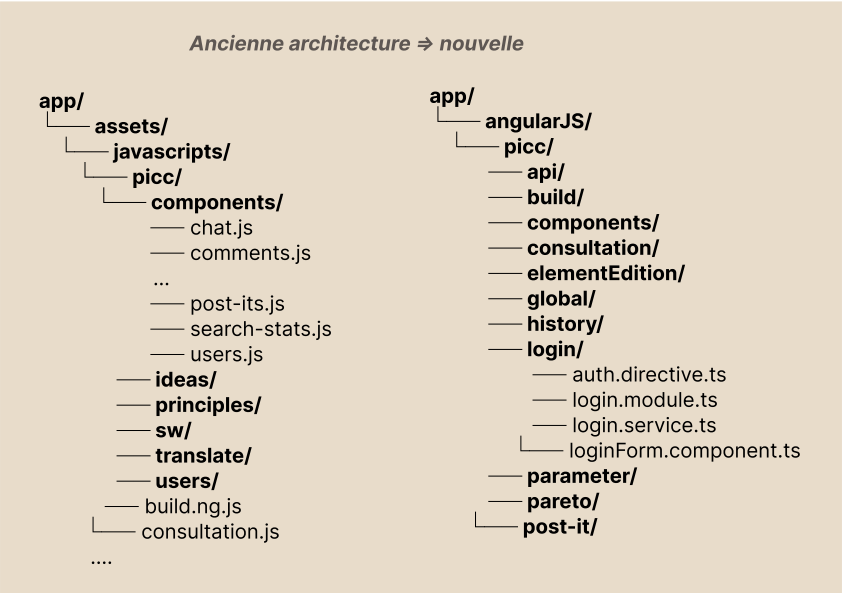
Ainsi, grâce à ces expérimentations et recherches nous avons pu formaliser nos choix techniques et organisationnel. Ceux-ci étaient encore incertain car à ce stade de développement du projet nous ne savions pas encore quelle méthode serait la meilleurs. Cette cohabitation contrôlée nous a offre une souplesse indispensable que ce soit pour le rythme de migration que pour la gestion des services partagé.

Intégration de la nouvelle architecture

Une fois les phases de test terminées, nous avions tous les éléments pour décider de la marche à suivre. L'application historique allait être restructuré selon les derniers principes d'Angular, cette étape est intermédiaire, elle préparer l'application pour une migration future en intégrant des principes de programmation qui harmonise l'écriture du code entre les deux Framework.

La programmation orienté objet, consiste à transformer les anciens contrôleurs ou services en classes ES6 avec constructeur, propriété et méthodes. Cela apporte une meilleur lisibilité et encapsule la logique du comportement dans chaque classes. Chaque élément de l'interface devient une classes structurés et isolé ce qui permettra par la suite d'intégrer le mécanisme d'Upgrade nécessaire à l'hybridation de l'application.

L'introduction de TypeScript, langage officiel de Angular, dans tous les fichiers de l'application. Cela signifie la création de type explicite comme les interfaces, enums et typage de fonction permettant une détection plus rapides des erreurs à la compilation. Cette étape est définie comme une tâche de fond qui sera complété au fur et à mesure de la transition vers Angular. En effet, le cout en ressource humaine et en temps pour recrée un typage fort à partir du code existant est bien trop important.



SCHEMA DE L'EVOLUTION DU LA STRUCTURE DE APPLICATION

Nous pouvons remarquer sur le schéma ci-dessus, qu'il est assez difficile de repérer là où son définie les différents modules. Les fichiers dans le dossier composant contiennent généralement plusieurs composants et tous les composants de l'application ne sont pas contenu dans ce dossier. Il y a un mixte entre les méthodologie où parfois les fichiers sont arrangés en module comme pour *principles*, *ideas* ou *users*, tandis que parfois les composants sont classés dans le dossier composant. Enfin la plupart des fichiers comme *consulatations.js* contienne à la fois un service et plusieurs composants.

L'application de la Rule Of One, consiste à réorganiser le code selon le principe un fichier une responsabilité : Un seul composant ou service ou directe par fichier. Ce principe s'accompagne d'une modification de la nomenclature des fichiers intégrant leur rôle dans le nom comme *user.component.ts*. Cette réorganisation permet de supprimer les fichiers fourre-tout qui réunissent un service et plusieurs composant compotant parfois plus de 7.000 lignes de code au totale. Partant de ce remaniement de la structure général, nous avons décider d'y intégrer le principe de *feature folder structure*. Chaque fonctionnalités métier sera à présent isolé dans un dossier contenant ses propre composants, services, tests et styles. Cette structure est encouragée par Angular pour profiter au maximum de la segmentation de l'application par module.

Lors de l'application de ces changements nous avons dû faire face à un problème majeur puisque nous ne pouvions pas tester l'intégration de la nouvelle structure avant d'avoir finit la restructuration. Cette phase de traversé du dessert est critique car elle peut donner lieu à l'apparition de nouveau bug qui seront difficile à repérer et réparer une fois l'application reconstruite. Nous devions donc trouver une méthode pour appliquer ces changement tout en modifiant au minimum le code source d'origine.

Lorsqu'il faut réécrire un code en programmation orienté objet la méthode commune consiste à copier le corps d'une fonction dans le corps d'une méthode de la classe cible. Cependant, cette méthode demande de :

- Passer de manière unitaire sur chaque fonction de l'application
- Ajouter le mot clef *this* devant toutes les variables globales devenu des propriété de la classe

La quantité de modification que demande cette méthode est trop important et peut entrainer des problèmes tel que l'oublie de certaines fonction, propriété ou argument modifiant le comportement de l'application. Ne souhaitant pas négliger le caractère faillible du facteur humain nous avons jugé cette méthode trop dangereuse et le cout en temps pour corriger les erreurs trop important.

Pour réaliser ces modifications nous nous somme orienter vers une solution qui impactais au minimum le code source d'origine. C'est pourquoi nous avons choisi d'intégrer le corps d'un composant ou d'un service à l'intérieur du constructeur de sa classe. Cette méthode à première vu moins propre nous permet de :

- Conserver au maximum le corps des composants et des services
- Supprimer un niveau de profondeur en opérant l'extraction d'un composant vers un fichier sans toucher unitairement à chaque fonction
- Ne corriger que les erreurs TypeScript et changement majeur de librairie mise à jour
- Ne pas faire de re-réécriture massive du code pour chaque composant ou service en conservant les propriétés globales définie dans le constructeur

Cette phase de modification fut la plus longue du projet, chaque fichier d'origine a été diviser en parfois une dizaine de fichier dans la nouvelle architecture. Le nouveau compilateur a révélé des erreurs dans le code qui n'avais pas été repérer jusque à présent et qu'il été nécessaire de régler à l'aveugle afin de pouvoir continuer la restructuration. A ce point, bien que le compilateur nous assure qu'il n'y ait pas d'erreur de syntaxe, le risque est de changer la logique dernière le code. Ainsi, chaque modification du code à été scrupuleusement référencer pour facilité un potentiel retour en arrière.

D'un autre côté, les librairies qui était importé via le globale namespace paterne depuis un fichier statique référencé dans le fichier index.html se sont sûr vu être importé en tant que module via NPM. Ce changement se faisait le plus régulièrement sans problème à l'exception de certaines librairie qui ne maintenaient pas leurs version globale namespace paterne à jours. Ainsi, en passent à la version module de la librairie nous avions plusieurs version qu'il nous a fallu rattraper, encore une fois à l'aveugle, sans pouvoir tester l'exécution du code.

Une fois l'intégralité de ces modifications appliquées l'application était enfin prête à être hybridé. Nous avons pu Upgrade l'ensemble de l'application AngularJS à l'intérieur de la nouvelle application Angular. Cela s'est fait en créant un nouveau composant qui encapsule l'entièreté de l'application AngularJS et qui fait office de point d'entrée pour celle-ci.

Difficultés rencontrés

Hot reload des fichiers html

Durant toute l'intégration de la nouvelle architecture un problème non bloquant persistait. Les fichiers HTML demandait un redémarrage de l'application pour appliquer les modification fait à leurs template. Ce problème bien non bloquant rendait inutilisable l'application en mode de développement.

Conscient de ce problème et désirant respecter au mieux l'architecture *feature by folder* nous avions pendant nos phases de tests essayé l'instruction templateUrl permettant, dans le Framework Angular, de chercher un fichier HTML à partir du dossier courant. Cependant, dans le Framework AngularJS cette instruction chercher les fichiers HTML à l'intérieur du dossier /public via une requête http envoyé au server. Nous avions donc, dans un premier temps, configuré **l'architecte** de l'application pour que les fichiers HTML soient copier/coller dans le dossier /public à la compilation de l'application. Pour rappel le dossier /public dans le Framework Angular est à présent auto-générer et donc supprimable à tout moment. Il était des lors impossible de stocker nos fichier HTML directement dedans. Les fichiers HTML placé dans un dossier à part ils n'étaient pas copier/coller à chaque sauvegarde de l'application. Seul une suppression du dossier /public permettait leurs mises à jour.

Pour palier à ce problème nous avons cherché à modifier les paramètres de mises en cache des fichiers dans le but de forcer le compilateur à actualiser toutes les sources. De plus, nous avons ajouter notre dossier /HTML au dossier surveillé par le module **Watcher** ce qui a permis de relancer l'application à chaque sauvegarde d'un fichier HTML. Cepen dant et bien que le compilateur en mode **verbose** nous montre que le fichier modifier est bien pris en compte. Cette méthode ne fonctionnait que pour la première modification apporté. En effet, une deuxième modification et relancement de l'application n'appliquait plus changement apporté au template HTML. Il semblerait qu'un mécanisme interne à Angular empêche l'utilisation du module Watcher de cette manière.

À la suite de cette échec, nous avons repris le problème à sa base et envisagé de le contourné par un autre moyen. Le problème résidait dans une utilisation différentes de l'instruction templateUrl par les deux Framework. Nous avons alors choisi d'utiliser un combo d'instruction pour contourner le problème : template + require. Cette méthode permet d'importer un fichier HTML présent dans le même dossier que sont composant. Elle permet également d'apposer le code HTML au coté de notre composant comme si le template avait été écrit à l'intérieur du fichier TypeScript. Pour réaliser cette solution nous avons dû mettre en place un fichier webpack.config.js personnalisé en modifiant l'architecte d'Angular18. Ce fichier utilise la librairie html-loader pour permettre le

chargement des fichier HTML via l'instruction require. Ainsi, les modifications des fichiers HTML sont désormais prises en comptes instantanément sans redémarrage de l'application ou suppression du dossier / public. De plus, notre architecture respecte à présent entièrement la structure feature by folder.

Obsolescence des fichiers sources

Pendant toute la durée de la restructuration nous avons accumulé un problème grandissant avec les semaines de développement. Les fichiers source de l'application d'origine devenaient obsolète au fur et à mesure que les autres équipes de développement continuaient à développer de nouvelles fonctionnalités.

Chaque fichier de l'ancienne architecture avait été divisé en plusieurs services, composants ou modules. Cette manipulation nous avait fait perdre l'historique GIT en recréant un nouveau fichier. Nous avons essayé de converser cette historique GIT en utilisant un scripte permettant de dupliquer un fichier ainsi que l'historique de ces modification. Cela nous a permis de faire suivre le déplacement du fichier d'un dossier à l'autre de l'application et son changement de nom. Cependant, l'historique ligne par ligne ne supportait pas les milliers de ligne supprimer & déplacé ce qui nous a systématiquement amener à le perdre.

Une fois la restructurations terminé nous nous sommes donc retrouvés avec une version en retard par rapport à l'application mise en production. Ce rattrapage a dû se faire minutieusement à la main à l'aide d'outils de **versionning** avancé. Ayant conservé les fichiers d'origine à leurs emplacement initiale, nous pouvions dans une éditeur GIT montrer leurs différences avec une branche plus avancé. Chaque différences devaient alors non pas être acceptées, mais copier/coller dans la nouvelle architecture. Cette étape fut réalisée après une longue phase de test et de résolution de bug dû à l'hybridation de l'application, elle présentait un grand risque pour le projet car nous pouvions sans nous en rendre compte ajouter de nouveau bug.

4.3.3 Évaluation et résultats obtenus

4.3.3.1 Indicateurs de performance et résultats quantitatifs

Performance

Avant l'hybridation l'application présentait des temps de chargement raisonnable qui n'impacter en rien l'expérience utilisateur. Une légère augmentation du temps de chargement a été observé qui peut être dû à la cohabitation entre les deux Framework, la duplication de certaines ressources mais aussi la compilation du code JavaScript.

Tableau comparatif des performances de l'application		
Application	AngularJS	Hybride
Temps de démarrage	21s	27.36
Temps de relance	1.59s	2.34s
Nb. Requête http au démarrage	175	59
Chargement d'un projet Kmap	2.761s	2.34s
Chargement d'un projet Build & Solve	>1s	7.329s

Après la migration nous avions prévu de légère baisse de réactivité au niveau de l'interface dû à la synchronisation des cycles des deux Framework. Cependant, ces baisse n'ont pas été constaté. Cependant, certains modules comme le Build & Solve ont subi une forte augmentation de leurs temps de réponse. Ce augmentation significative peut être liée à la cohabitation entre les deux Framework ou la mauvaise gestion d'une librairie mise à jour.

Maintenabilité

Une diminution des bugs discret a pu être constaté notamment grâce à l'introduction de TypeScript et à une meilleurs structuration du code. Certains bug n'avais jamais été repérer via l'interface utilisateur mais ils ont été immédiatement vu puis corrigé grâce au compilateur de TypeScript. Cependant, des bugs spécifiques à la cohabitation entre AngularJS et Angular ont émergé, ceux-ci nécessitent une attention particulière.

4.3.4 Analyse critique et réflexive

4.3.4.1 Point forts de l'approche utilisé

La stratégie de migration progressive est maîtrise et nous pouvons valider la migration incrémental via l'UpgradeModule. Celle-ci nous permet d'intégrer progressivement Angular sans réécriture complète du code source. Elle a aussi garantie la continuité du service tout au long du projet en permettant aux autres équipes de continuer à développer des fonctionnalités. De plus, la méthodologie Top-Down qui initialise l'application via Angular puis encapsule le code AngularJS existant offre une bonne maître de la hiérarchie des composants tout en rendant possible une migration par module, par service ou par composant. Cette méthode à su montrer qu'elle était non destructive et nous a permis de capitaliser sur les modules AngularJS déjà existant tout en nous offrant la possibilité d'intégrer de nouveau composant Angular à l'intérieur d'ancien module AngularJS.

C'est aussi une monté en compétence de l'équipe avec une formation continue sur Angular2+ et les dernières pratiques de ce Framework. Le gain en culture de l'architecture logicielle avec des concepts comme l'injection de dépendances, la modularisation par domaine métier et les stratégie de détection de changement. Connaissance qui on donner lieu à la création de référentiels internes sur la création de composants Angular2+ et un socle commun de connaissance sur la seed Angular18 et Play Framework.

4.3.4.2 Axes d'amélioration et leçon apprise

Au niveau de l'anticipation et du pilotage technique, le projet a subi un manque notamment sur certains impacts liée au scope partagé les deux FrameWork. Cette cohabitation bien quelle soit fonctionnel, génère une complexité accrue dans la gestion des performances, avec des ralentissements dans certains modules. Pour pallier ce problème nous aurions dû avoir une cartographie des composants critique qui aurait permis de mieux cibler les zones à optimiser en amont de la migration.

La refactorisation à révélé une dette technique plus profonde que prévue, notamment dans la structure du code source historique. De plus, l'absence d'une stratégie de typage progressive documentée ralenti le passage vers TypeScript. Une meilleurs planification du typage fort en ciblant en priorité les objets les plus conséquents de l'application aurait permit une adoption plus efficace.

La gestion de projet à reposé sur un pilotage réactif plus que stratégique. Le projet n'avait pas au départ de chemin clairement planifié, bien que des livrables intermédiaire était présent tout au long du processus. Certaines décision importante comme la

réorganisation en Folder by featurer ou l'implémentation d'un webpack personnalisé ont été prises en réaction au problème courant sans réel planification au préalable. Nous mieux faire, dans le cadre d'un projet de cette taille, nous aurions dû renforcer le cadre de la méthodologie agile en intégrant des sprints entier à la monter en compétence qui s'est faite au file des erreurs rencontré.

4.3.5 Compétence acquise et liens avec la formation

Cette mission m'a permit d'approfondir mes connaissances technique en développement frontend par la mise en place d'une nouvelle architecture et de la configuration d'un projet sur une technologie nouvelle pour moi. Cela m'a permis de monter en compétence dans les domaines suivant :

- Développement Angular18
- Utilisation avancé du CLI Angular pour la configuration et la compilation de projet
- Les stratégies de distribution d'une application web et les webpack
- L'intégration et l'utilisation de TypeScript
- Module spécifique à l'hybridation d'application
- Cycle de vie partagé, la gestion des cycles et la détection de changement
- Architecture moderne via la Rule of One et le Folder by Feature

La conduite de cette mission quant à elle, m'a confronté à des problèmes technique complexe, des enjeux réel face à la réalisation de ce projet pour la pérennité de l'entreprise ainsi que des enjeux de priorisation. Ce projet m'a alors apporté des compétences en gestion de projet tel que :

- L'élaboration et le suivi d'un plan de migration incrémental sans rupture de service
- Gestion du risque liée à la dette technique d'AngularJS
- Adaptation agile des priorités face aux imprévu comme la compatibilité HTML
- La priorisation des tâches liées à la dettes techniques, tout assurant la maintient et la correction du logiciel existant
- Rédaction de guide de migration et documentations des composants
- Réalisation d'une étude comparative sur les stratégies de migration
- Sélection mise en place et amélioration d'un environnement Angular + Play Framework

Liens avec la formation M2I	
Référentielle	**Compétence éprouvé**
Bloc 1 – Analyse stratégique du SI	Diagnostique de l'obsolescence d'AngularJS
	Choix raisonné de la stratégie Top-Down & méthode d'hybridation
Bloc 2 – Conception et Déploiement	Construction d'un architecture hybride modulaire
	Mise en place de la Dockerisation (Angular, Play Framework, MongoDB)
Bloc 3 – Pilotage de projet complexe	Conduire un projet technique sur plusieurs mois
	Adaptation continue du plan d'action en fonction des contraintes du projet
Bloc 4 – Intégration de technologie complexe	Mise en place d'Angular18
	Configuration de Webpack personnalisé
	Configuration d'environnement multi-framework entre SBT et NPM
Bloc 5 – Veille, documentation, communication de projet	Documentation du projet et problème potentiel
	Veille sur les différentes stratégie de migration

5 Conclusion et perspectives

Références

MSM. (2023, septembre 22). *L'intelligence collective au service des entreprises*. https://www.msm.ch/lintelligence-collective-au-service-des-entreprises-a-92c9b99b23c357114bd90d6e81d7086b/

PICC Solution. (n.d.). *Gestion des données : un levier incontournable de la transformation digitale*. https://www.picc-solution.com/fr/gestion-donnees-transformation-digitale/

PICC Solution. (2023, juin 5). *Ils utilisent PICC : La Région Grand Est*. https://www.picc-solution.com/fr/ils-utilisent-picc-la-region-grand-est-sappuie-sur-lia-pour-elaborer-son-srdeii/

PICC Solution. (n.d.). *Industrie 4.0* https://www.picc-solution.com/fr/utiliser-les-donnees-des-capteurs-iot-dans-une-plateforme-dintelligence-collective/

Amiltone. (n.d.). *Angular et le développement web*. https://www.amiltone.com/tech-place/angular-et-le-developpement-web

Ambient IT. (n.d.). *Statistiques Angular*. https://www.ambient-it.net/statistiques-angular/

Uzinakod. (2021, janvier 26). *Google abandonne AngularJS*. https://www.uzinakod.com/blogue/google-abandonne-angularjs

Pérez, J. (2021, janvier 11). *AngularJS end-of-life is here. Now what?* Medium. https://medium.com/@javperezp79/angularjs-end-of-life-is-here-now-what-bd7961eb19b4

Papa, J. (n.d.). *Angular style guide*. GitHub. https://github.com/johnpapa/angular-styleguide

Kant, I. (1790). *Critique du jugement* (paragraphe 7) [PDF]. https://www.ecolpsy-co.com/download/Kant_Critique_du_Jugement.pdf

This is Angular. (n.d.). *Guides for decision makers*. https://this-is-angular.github.io/angular-guides/docs/category/decision-makers

Pixel Perfect. (2023, octobre 4). *Migration d'AngularJS vers Angular – La méthode complète* [Vidéo]. YouTube. https://www.youtube.com/watch?v=lYOHB_yTEmo

playframework. (n.d.). *play-scala-angular-seed* [Code source]. GitHub. https://github.com/playframework/play-scala-angular-seed

Walker, N. (n.d.). *angular-seed-advanced* [Code source]. GitHub. https://github.com/NathanWalker/angular-seed-advanced