MEMOIRE PROFESSIONNEL

MANAGER EN INGÉNERIE INFORMATIQUE Spécialité Développement

PERNON Etienne

DEVELOPPEUR

Entreprise : PICC DEVELOPPEMENT

Adresse : 3 rue des Cigognes 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 tuteur :

Résumé

Ce mémoire professionnel relate les trois années d'alternance que j'ai effectuées chez PICC Développement, start-up d'innovation, installée à Entzheim et spécialisée dans la gestion de connaissances industrielles et de l'intelligence collective. J'intègre l'équipe de développement, sous la tutelle de Simon Fulhaber, chargé de/du développement et CTO de l'entreprise.

Durant cette période, deux missions principales me sont confiées. La première fut une refonte partielle d'un module de gestion de procédures, mission stratégique menée pour Hutchinson, un éventuel client. Le but était d'aligner les processus internes du groupe en laissant la place à une adaptation locale et spécifique. J'ai donné la priorité à une méthodologie collaborative, en apportant la matrice RASCI, décloisonnant rôle et responsabilité, j'ai ensuite ajouté une gestion en temps réel via des sockets pour améliorer la traçabilité et la réactivité du système.

La seconde consistait à réaliser une importante migration technologique : la migration d'une application web AngularJS en Angular 18. En raison, de l'obsolescence d'AngularJS, ce passage était nécessaire afin d'assurer la pérennité, la sécurité et la maintenabilité de l'application. J'ai privilégié une approche progressive grâce à une architecture hybride par l'intermédiaire du module ngUpgrade, ce qui a permis d'assurer une continuité de service pendant toute la migration. J'ai également instauré des bonnes pratiques telles que la méthode « folder-by-feature » et l'implémentation de composants indépendants.

Ces deux missions m'ont apporté une expérience en gestion de projet, en expertise technique et en orientation stratégique d'entreprise. Elles ont de surcroit été un défi notamment technique puisque la réalisation d'une application hybride m'a contraint à acquérir de nouvelles aptitudes professionnelles concernant des technologies qui m'étaient inconnues. Ce fut par ailleurs un défi communicationnel afin d'atteindre un produit satisfaisant pour les utilisateurs finaux.

Table des matières

1. Glossaire

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

1.1 PICC Solution

Approche systémique : Méthode de réflexion scientifique basée sur la compréhension des objets comme étant un système composé d'éléments en interaction. Cette méthode permet une vision collaborative en intégrant des points de vue différents.

Intelligence collective : Méthode de gestion de la connaissance basée sur un réseau de connaissances et d'expertises. Elle encourage la synergie et l'auto-organisation. Elle permet chez PICC de modéliser les savoir-faire via la structure problème-solution.

Buisness Act Grand Est : Plan de relance et de transformation économique initié par la région Grand Est. Ce plan pousse à l'accélération de la transition des entreprises vers des modèles durables, résilients et innovants.

SRDEII : Pour schéma régional de développement économique, d'innovation et d'internationalisation. Document obligatoire fournie par toute les régions française depuis la loi NOTRe de 2015. Il définit la vision économique de la région.

KMAP : Pour *Knowledge MAP*, c'est une adaptation des cartes mentale intégrant les principes d'approche systémique et l'architecture problème solution.

Concept : Les concepts sont les problèmes et solutions qu'il est possible d'intégrer à un KMAP. Une fois posée, ils représentent un schéma de pensée ou la construction d'une problématique.

Entité : Elément positionnable sur un KMAP. Elle représente tous les éléments qui gravitent autour des concepts, cela peut être une machine, une personne de l'organisation ou des objets plus spécifiques comme un cadre de gestion de projet.

Domaine : Etiquette apposable sur n'importe quel concept ou entité, il permet d'associer un secteur d'activité, un service de l'organisation, une typologie de problème aux éléments d'un KMAP.

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.

1.2 Management d'entreprise

CEO : Acronyme anglophone désignant le Chef Exécutif Officer, c'est-à-dire la personne qui prend les décisions stratégiques dans l'entreprise.

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

Industrie manufacturière : Large sous ensemble du secteur secondaire, ce type d'industrie vise à la fois les sites de transformations de biens que les entreprises de réparation et d'installation d'équipements.

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

Lean management : Méthode de management tirée du système de production de Toyota. Il vise à maximiser la valeur pour le client tout en réduisant le gaspillage.

Matrice RASCI : Outil de gestion de projet qui permet de définir les responsabilités de chacun sur un objet donné. Chez PICC elle peut être ajouté à n'importe quelle entité permettant de savoir qui contacter pour obtenir des précisions.

1.3 Développement de logiciel

LLM : Pour Large Language Model, c'est une technologie d'intelligence artificielle basée sur le Deep Learning. Elle consiste à entrainer un modèle sur de grands volumes de textes pour comprendre et générer du texte.

RAG : Pour *Retrival Augmented Generation*, c'est une méthode combinant la recherche d'informations dans une base de données et la génération de texte via un LLM. Cette méthode permet de cadrer les réponses d'un modèle de langage qui apportera sa réponse en se basant sur la base documentaire de l'entreprise.

Agent IA : Programme capable d'interagir avec son environnement, récupérer des informations, raisonner, transmettre ces informations. Dans le cas de PICC, cette technologie est utilisée pour interagir en langage naturel via un LLM enrichi par un RAG.

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

Big data : Désigne un ensemble de données qui par son volume nécessite d'être traité par des moyens numériques.

MES : Pour Manufacturing Execution System, c'est un logiciel qui relie, surveille et contrôle une chaine de production en temps réel.

Plateforme IoT : Logiciel basé sur la technologie Internet of Things qui permet de collecter, analyser et exploiter les données issues des objets connectés. Chez PICC les données des capteurs IoT permettent de nourrir la base de connaissances du logiciel.

Framework : Pour cadre de développement en français, c'est une structure de base, des conventions et des outils intégrés qui permettent de simplifier le développement d'une application et évite de réinventer la roue.

Single Page Application (SPA) : Technologie Front End mise en avant par des Framework tel que Angular et React, elle permet le chargement d'une application Web depuis une page HTML qui sera mise à jour dynamiquement en JavaScript.

Frontend / client : Désigne la partie visible d'une application avec laquelle l'utilisateur va interagir.

Backend / server : Désigne la partie invisible d'une application qui se charge de créditer les authentifications, de faire persister les informations ou encore de connecter les utilisateurs entre eux.

Folder by feature structure : Méthode d'organisation du code qui consiste à regrouper les fichiers par fonctionnalités de métiers. A l'inverse on retrouve une organisation par type de fichier qui regroupe tous les composants d'un côté et tous les services d'un autre.

Rule Of One : Principe d'architecture qui incite à limiter chaque fichier à une seule responsabilité (class, composant, service) et permet un nommage plus explicite de chaque fichier.

TypeScript : Technologie permettant d'ajouter un typage fort dans le langage JavaScript qui en est usuellement dépourvu.

Module Javascript : Fichier de code isolé qui exporte une partie de sa logique via des classes, fonctions ou constantes et qui peut être important dans un fichier javascript.

Modules loader : Mécanisme permettant de charger dynamiquement des modules JavaScripts et de réduire la taille initiale d'une application.

Angular Materials : Ensemble d'éléments graphiques prêts à l'emploi permettant d'accélérer la construction d'une interface utilisateur.

Dette technique : Désigne l'ensemble des compromis techniques fait à court terme ou choix pris il y a longtemps qui génèrent des coûts futurs en matière de maintenance, performance et évolutivité.

CLI : Pour Commande Ligne Interface, c'est un logiciel auquel on accède via le terminal et qui permet d'exécuter des scriptes. Le CLI d'Angular sert à créer des composants, lancer la compilation de l'application ou démarrer un serveur.

Compilation : Opération de transformation du code initial écrit par un développeur dans un langage compréhensible par la machine. Dans le cas d'Angular avec TypeScript, la transformation se fait vers une version plus optimisée du code source.

Linting : Processus d'analyse du code permettant la détection d'erreurs de syntaxe, d'incohérences ou de problèmes potentiels. C'est une lecture du code par un logiciel nommé Linter.

Application monolithique : Application considérée comme un bloc indivisible où toutes les parties ont une forte interconnexion les unes avec les autres. L'opposé est une

infrastructure en micro-service où toutes les parties sont au maximum découplées les unes des autres et déployables individuellement.

Bootstrapping : Démarrer ou initialiser l'application, c'est le point d'entrée de l'application qui va régir la manière dont le reste du code sera appelé.

Minification : Mécanisme consistant à réduire la taille finale du code source en supprimant les espaces, commentaires et formatages, le rendant ainsi plus facilement lisible par un humain.

Obfuscation : Mécanisme consistant à rendre le code source d'une application le plus compliqué à lire possible sans en altérer le fonctionnement.

2 Introduction

En octobre 2022, j'ai intégré l'entreprise PICC Développement en tant que développeur. Simon Fulhabert mon maître d'apprentissage et actuel CTO étant pris par de nombreuses réunions, il avait besoins d'une aide humaine pour prendre en charge le développement d'application Frontend. Ayant une formation initiale en Java et en JavaScript, mon rôle a d'abord été de me former à l'infrastructure du logiciel puis par la suite de réaliser des projets de bout en bout. Dans un premier temps j'ai été chargé de petites actions pour lesquelles j'étais complètement guidé. Par exemple : l'historique des recherches, la signature du contrat de licence, la refonte des notifications envoyées par mail. Ces diverses expériences ont contribué à me familiariser avec l'architecture *front* et *back end* ce qui m'a permis de prendre en main des missions plus conséquentes avec une certaine d'autonomie.

Par exemple, j'ai pris en charge des tâches telles que l'instauration des entités customisée. Celle-ci, a demandé d'importants changements au niveau du serveur Java, l'introduction d'une nouvelle table dans la Base de données et la création d'un nouveau service Angular avec ses composants.

D'autre part, lorsque j'ai organisé la refonte du module de procédure, j'ai réalisé mon premier développement long. Ce développement m'a permis de m'impliquer dès la phase d'analyse du besoin, phase durant laquelle j'ai pu proposer de nouvelles structurations des procédures. Confronté à de nouvelles problématiques, j'ai pu acquérir des compétences tant sur le plan méthodologie de projet que sur le plan technique.

J'ai également planifié la transition de l'application *front* sous AngularJS vers le *Framework* Angular. Cette mission, techniquement exigeante, m'a apporté des savoir-faire dans les technologies modernes telles que : Angular CLI, TypeScript, Architecture logiciel. Au-delà de l'aspect technique cette mission m'a aussi amené à planifier les différentes phases de migration, à analyser la dette technique de l'entreprise et à organiser sa pérennité technique.

Dans ce mémoire, je présenterai PICC Solution, ses technologies et domaines d'activité ainsi que son organisation et ses équipes. J'aborderai mes principales missions, mon rôle, mes décisions importantes, mes apprentissages et je ferai mon autocritique. Enfin, je terminerai ce mémoire par des perspectives concernant mes contributions à l'entreprise et concernant mon avenir professionnel.

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 industrielle 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'idées vont être établis : le premier est que le moyen le plus efficace sera de travailler avec des personnes d'horizons et de cultures différentes ; le second est que les idées devront être sélectionnées via des procédés scientifiques et non biaisées par le charisme ou l'influence des personnes qui les ont portées.

Une équipe de 20 personnes composée de sociologues, psychologues et d'informaticiens est alors formée, dont Simon Fuhlhaber actuel CTO de PICC Solution, pour réfléchir à une approche systémique et non biaisée de l'innovation. Le résultat de cette recherche fut l'ébauche d'un logiciel qui sera racheté par les actuels dirigeants 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 pensés pour améliorer la compétitivité des entreprises.

3.1.2 Domaine d'activité

Aujourd'hui, l'entreprise PICC agit sur plusieurs domaines d'activités à commencer par la gestion de connaissances industrielles[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 : décloisonnées et accessibles dans un même système de gestion centralisée ; fiable car faisant partie d'une stratégie de management de la donnée impliquant des processus de nettoyage, de vérification et de mise à jour des données ; valorisées par un système de recherche efficace permettant le croissement et l'analyse, ainsi que transformables en indicateur, prévision ou alerte.

Face à la masse d'informations récoltées et à leur complexité, l'intelligence artificielle est devenue un outil incontournable pour traiter et analyser ces ensembles de

[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/

données. Cette technologie ne se contente pas seulement d'automatiser l'analyse de grands volumes de données mais propose également leur restructuration via l'approche problème-solution. Cette méthode permet de proposer des actions concrètes qui seront validées 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 génèrent le plus de valeurs ajoutées pour l'entreprise. Dans la pratique, c'est accéder aux savoirs et aux savoir-faire de l'entreprise, être guidé dans la résolution de problèmes complexes, recevoir une assistance à la prise de décisions, anticiper et résoudre les problèmes en proposant des solutions et enfin, valoriser et 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 dans le développement des outils informatiques au sein même de leurs chaines de productions. PICC s'inscrit comme un intermédiaire capable d'accélérer et d'optimiser la mise en place de solutions lourdes, telles que les MES, qui sont aujourd'hui indispensables pour leurs capacités à analyser en temps réel les données de production et à alerter 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 performances industrielles 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 collectivités territoriales[4] comme le Grand Est pour traiter de grandes quantités d'informations afin de prendre des décisions. Lors de l'initiative « Buisness Act Grand Est » ce sont 22 groupes de travail pour un total de 600 personnes qui ont proposé des milliers de pages de rapports portant sur l'innovation 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

[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/

- L'écologie
- La gouvernance

Ces analyses ainsi que le suivi des réunions de concertation et les SRDEII de toutes les régions françaises ont permis 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 permis de mettre en lumière des écarts de perception dans l'importance des effets potentiels que pouvait avoir chaque orientation stratégique.

3.2 Technologie de PICC Solution

3.2.1 Gestion et structuration de l'information

Les informations sont tout d'abord structurées 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 savoir-faire complexes. 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 leur tour 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 créer une multitude de problèmes connexes ; une solution ne peut pas être la source d'une autre solution, c'est une erreur de logique car il doit nécessairement y avoir un problème entre les deux solutions.

Au milieu de ces concepts abstraits, 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 serait 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 documents d'entreprise. La lecture de sources permet d'extraire des problèmes, solutions et entités, d'identifier des domaines pour catégoriser ces concepts et enfin de relier ces informations au sein d'un KMAP.

3.2.2 Technologie d'intelligence artificielle utilisée

Tout d'abord, PICC utilise les modèles LLM (Largue Langage Modèle) comme gemma 3 pour comprendre le langage naturel utilisé par les opérateurs, techniciens ou ingénieurs. Ces modèles permettent d'interpréter des questions ouvertes, des descriptions informelles de problèmes, ou des demandes complexes exprimées avec un vocabulaire spécifique à l'entreprise.

Combiné à la méthodologie RAG pour *(Retrival Augmented Generation)*, cette technologie permet la recherche d'informations 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 pannes, solutions et incidents passés mais aussi de fournir des réponses contextualisées, validées et documentées, 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 industrielles 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'expériences (intervention, baisse de qualité) permettent de proposer des diagnostics précis et des recommandations proactives.

3.3 Structure interne de l'entreprise

3.3.1 Organisation interne

PICC est une petite start-up comptant moins de 10 collaborateurs actifs 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-entrepreneurs ou des micro-entreprises.

Il est possible de distinguer 4 types de services au sein de PICC. La commercialisation, s'occupant de trouver de nouveaux acheteurs, partenaires ou opportunités pour l'entreprise. La gestion de projets qui organise le suivi des entreprises partenaires, comprenant la prise en main du logiciel, la réalisation de tâches complexes ou spécifiques sur le logiciel ou l'élaboration de nouveaux procédés 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 définit les axes sur lesquels la solution doit s'orienter en fonction de la demande de nos partenaires et de l'évolution du marché.

La solution PICC dispose de ses propres outils de gestion de projet, développés en internet et intégrés à notre logiciel. Utilisé à la fois par nos partenaires et par nos équipes, ils nous permettent l'organisation et le suivi de projets de taille conséquente. Basé sur des préceptes du Framework AGILE et le Lean management, PICC associe l'utilisation d'un Kanban avec un tableau de suivi ainsi qu'un module d'analyse statistique.
La Kanban nous permet d'organiser nos tâches quotidiennes en limitant le nombre de tâches simultanées. 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 à la tâche dans la catégorie suivante.

Le tableau de suivi permet d'observer l'avancement des tâches d'un projet de manière plus précise en apportant un niveau de détails supplémentaires sur les ressources affectées ou les problèmes liés.

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

3.3.2 Exemple de méthodologie de pilotage

Comme précisé précédemment PICC fonctionne, pour ce qui est du développement logiciel, avec des petites équipes indépendantes, celles-ci bénéficient 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 quotidiennes et d'identifier les blocages.

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

- **Revue hebdomadaire** le *vendredi de 30 minutes – 1 heure (3-6 personnes)*, pour rendre compte des tâches réalisées.

- **Session de pair programming** ponctuelle *1h – 1h30 (2 personnes)*, pour faire avancer les blocages identifiés en points quotidiens

4 Mission réalisées

4.1 Refonte du module de procédure

4.1.1 Définition de la mission

4.1.1.1 Contexte général

L'entreprise Hutchinson est un groupe d'industriels filiale de Total Energies et spécialisé dans la fabrication de solutions multi-matériaux. Leader mondial dans la transformation d'élastomères, ils fournissent des secteurs comme l'automobile, l'aéronautique ou l'énergie[5]. Leurs produits incluent des systèmes antivibratoires, des solutions d'étanchéité et des systèmes de gestion des fluides[6].

Dans la continuité de leur stratégie d'amélioration continue, Hutchinson cherche à harmoniser ses processus internes. Leur objectif est donc de créer un référentiel de procédures standardisées tout en permettant des adaptations spécifiques dans leurs sites de production.

PICC étant en discussion avec l'entreprise Hutchinson pour lui vendre notre solution, cette dernière souhaite développer certaines parties du logiciel pour répondre à ses besoins. Ainsi, le module de gestion des procédures sera remanié pour qu'il soit à la fois standardisé et adaptable. Cela permettra de faciliter la diffusion et l'application des meilleures pratiques au sein du groupe. Les enjeux de cette mission seront donc :

- D'Assurer la cohérence des procédures au niveau du groupe tout en respectant les particularités locales
- Permette une exécution collaborative des procédures avec une traçabilité et un suivi en temps réel
- Faciliter la mise à jour et l'évolution des procédures en fonction des retours d'expérience

[5] Wikipédia. (n.d.). *Hutchinson (entreprise)*. Wikipédia https://fr.wikipedia.org/wiki/Hutchinson_(entreprise)
[6] France Hydrogène. (n.d.). *Hutchinson – Annuaire des acteurs*. https://vighy.france-hydrogene.org/annuaire-des-acteurs/hutchinson

4.1.1.2 Objectifs de la mission

Après avoir analysé le besoin de notre futur client Hutchinson, nous avons défini une suite d'objectifs pour répondre à ces attentes. Bien que le module de procédures soit déjà fonctionnel, il incombe d'apporter les modifications suivantes :

- Permettre de créer un référentiel commun à l'ensemble des sites de production
- Créer un mécanisme d'héritage au sein des procédures permettant la modification des procédures locales et la mise à jour par rapport aux procédures globales
- Améliorer la gestion des procédures en offrant une vue d'ensemble de toutes celles exécutées dans un contexte donné
- Ajouter un mécanisme de synchronisation en temps réel
- Permettre de visualiser en un coup d'œil l'avancement, les blocages
- Ajouter un système pour définir le rôle et la responsabilité de chacun vis-à-vis de chaque étape d'une procédure
- Permettre l'exécution partagée des procédures
- Ajouter un mécanisme de validation contrôlant les moments critiques de l'exécution

Ces modifications nous permettrons d'apporter clarté et uniformisation dans l'exécution des procédures non seulement pour l'entreprise Hutchinson mais aussi pour tous nos clients qui bénéficieront de ces nouvelles fonctionnalités.

4.1.1.3 Organisation du projet

Pour réaliser ce projet nous avons privilégié le cadre Agile qui nous permettait de rester au plus proche de l'attente de nos futurs clients. Par le biais de Constant qui tenait le rôle de Product Owner, nous avons pu ajuster le développement des fonctionnalités au fur et à mesure du projet. La définition de nos rôles face au cadre Agile était :

- Constant **Product Owner** : donne la vision globale du projet ; est en contact direct avec le client ; réalise des tests de mise en situation réel ; propose des critiques sur le côté fonctionnel.

- Simon **Scrum Master et Manager Opérationnel** : Définit la liste des tâches, traduit les besoin en fonctionnalités, valide les choix techniques ; aide le développer en cas de blocage.

- Etienne **Développer et Manager Opérationnel** : Définit la liste des tâches, traduit les besoins en fonctionnalités, propose des solutions.

Les outils ainsi que la méthodologie de pilotage utilisée sont définis dans la partie Organisation Interne de la présentation de l'entreprise (3.2.1 & 3.2.2).

4.1.2 Choix techniques et justifications

4.1.2.1 Intégration de la matrice RASCI

Nous avons ajouté la matrice RASCI dans la définition de chaque étape d'une procédure pour affiner la gestion des rôles attribués. Son objectif est de :

- Clarifier la responsabilité au plus petit niveau
- Garantir qu'une personne soit désigné comme responsable de chaque étape
- Permettre une exécution partagée mais contrôlée des procédures

La matrice est appliquée à chaque étape de la procédure, une étape ne peut être complétée que par un utilisateur désigné comme responsable. Les autres rôles sont moins exploités mais pourront à l'avenir avoir des droits spécifiques.

Les rôles disponibles dans cette matrice sont :

- R – Responsible : Exécution de l'étape
- A – Accountable : validateur final ou garant du résultat
- S – Support : Fournit des informations ou ressource
- C – Consulted : expert à consulter avant l'action
- I – Informed : reçoit des notifications liées à l'étape

En intégrant un outil du Lean management au cœur de notre module nous nous attendons à réduire les ambiguïtés dans la gestion des connaissances pratiques et à aligner les standards de qualité et de conformité de chaque site. C'est aussi un moyen efficace d'encadrer un workflow qui peut être complexe une fois que l'exécution des procédures est autorisée.

4.1.2.2 Utilisation de socket

A l'origine, l'actualisation des états des procédures se faisait via une requête envoyée au serveur avec un intervalle régulier. Ce mécanisme nous a causé certains problèmes notamment en test avec les clients qui ne voyaient pas apparaître immédiatement une procédure à laquelle il devrait pourtant avoir accès. Avec un besoin croissant dans ce module d'interconnectivité nous avons envisagé d'actualiser l'exécution des procédures via le socket client. Son objectif est :

- D'émettre des évènements pour notifier les changements d'états des étapes de procédure
- De diffuser des mises à jour à tous les clients pour s'assurer que l'information affichée est cohérente

Cette technologie permet d'améliorer l'expérience utilisateur grâce à des mises à jour instantanée sans rafraîchir la page ou attendre la fin de l'intervalle entre deux requêtes. C'est aussi une réduction de la charge pour le serveur qui subissait des requêtes répétitives de la part de tous les utilisateurs, même lorsqu'aucun avancement n'avait été fait dans l'exécution d'une procédure. Enfin, cela facilite la collaboration entre utilisateurs puisque cette fonctionnalité nous a permis d'intégrer une vue dédiée à l'avancement de toutes les procédures en cours.

4.1.3 Méthodologie mise en œuvre

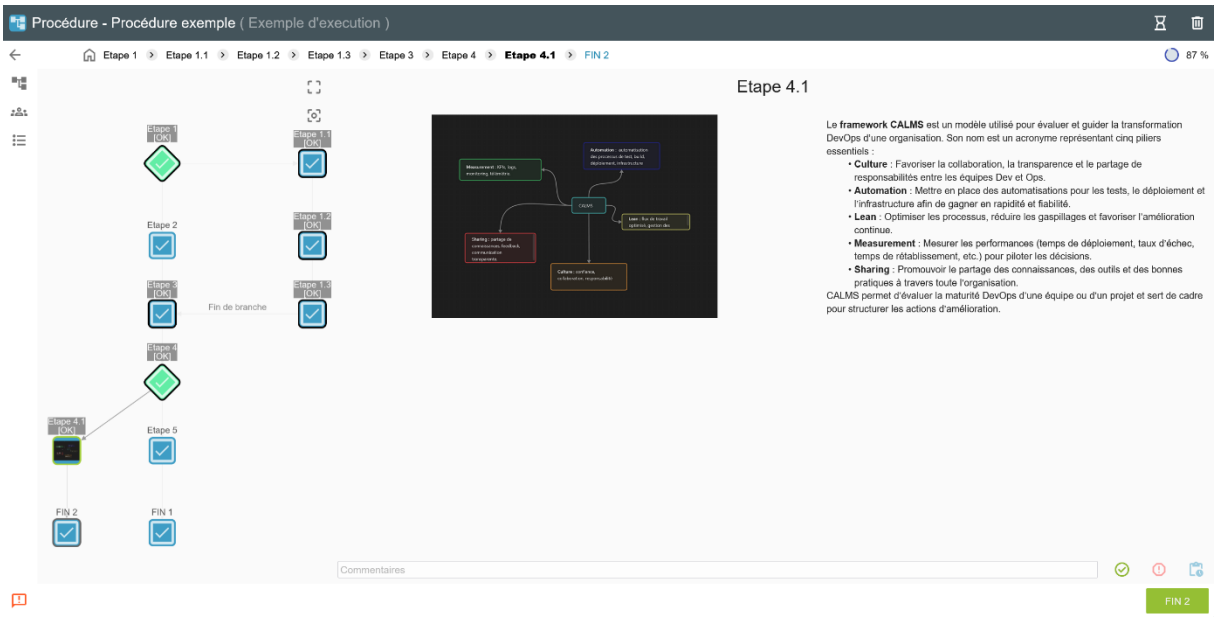
4.1.3.1 Procédure désordonnée et mécanisme de validation

Avant la refonte, les procédures s'exécutaient toujours de manière séquentielle. L'utilisateur devait compléter chaque étape dans l'ordre de la première à la dernière sans pouvoir en laisser une seule pour plus tard. L'idée était d'apporter plus de souplesse dans l'exécution des procédures en permettant de compléter chaque étape dans n'importe quel ordre.

La modélisation des procédures se fait déjà sous forme de graphiques orientés où chaque nœud est une étape et chaque arc la transition possible vers un autre nœud. Cette modélisation permet de générer des procédures complexes avec plusieurs embranchements et incluant des branches conditionnelles. La validation est alors simple puisqu'il suffit d'avoir un chemin reliant la première étape à la dernière. De plus cette validation n'a pas lieu d'être codée explicitement puisque les contraintes d'une procédure exécutée séquentiellement, c'est-à-dire ne pas pouvoir sauter d'une étapes à une autre et être obligé de valider une étape pour passer à la suivante, permettent à elles seules d'être sûr de la validité d'une procédure une fois arrivée sur l'étape de fin.

L'ajout des procédures désordonnées complique cette logique puisqu'il est possible de compléter l'étape de fin à n'importe quel moment de l'exécution. Il est aussi possible de valider une étape dans une branche conditionnelle avant d'avoir passé le nœud représentant la condition.

Nous avons donc utilisé l'algorithme de Parcours en Profondeur pour trouver tous les chemins possibles entre le point de départ et le point d'arrivée. Nous avons aussi modifié cet algorithme pour qu'il exclut d'office tous les chemins ne comprenant pas l'étape courante, ce qui nous permet de réduire nos recherches au fur et à mesure de l'avancement de l'exécution.



EXEMPLE D'EXECUTION D'UNE PROCEDURE

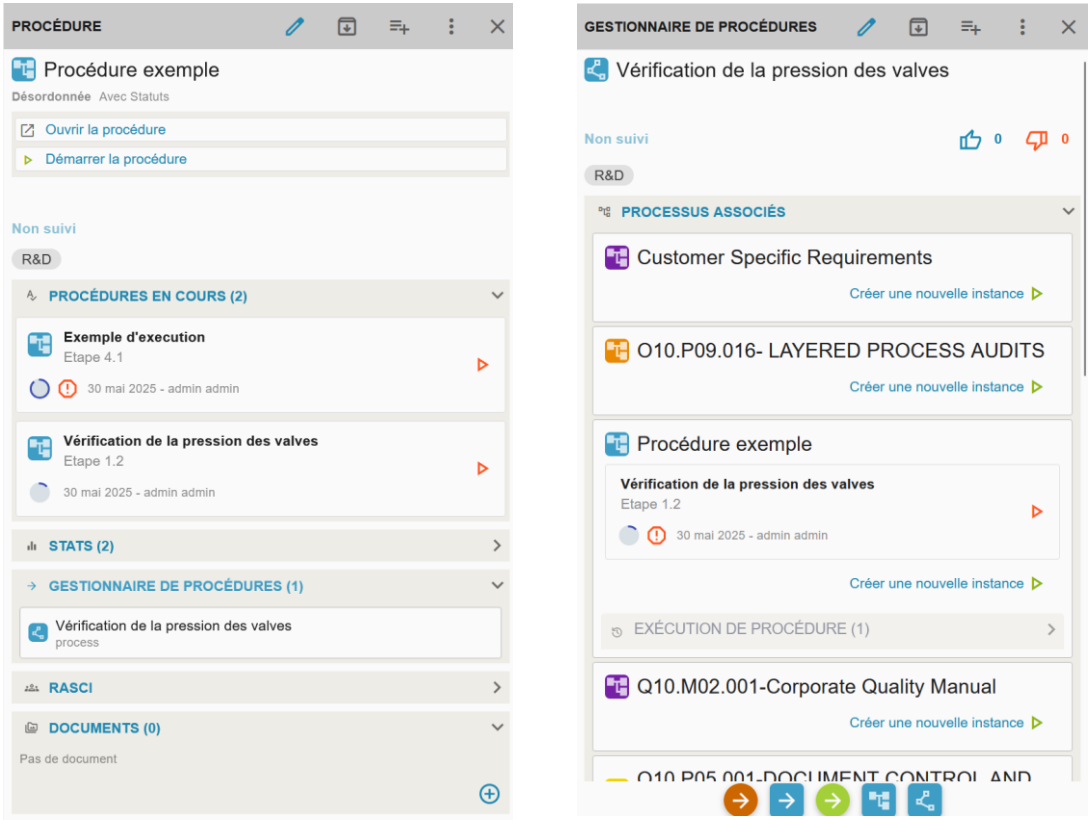
Une fois la liste de chemin possible trouvé il nous faut chercher le chemin le plus court comprenant un maximum d'étapes valide. En cherchant le chemin le plus court, nous nous assurons que le chemin prédéfinie se rapproche le plus de celui que l'utilisateur emprunte actuellement. Si l'étape courante n'est plus dans liste des chemins possible, c'est que l'utilisateur a changé d'embranchement, il faut alors recalculer les chemins possible.

Ce mécanisme permet à chaque étape de la procédure désordonnée de définir un chemin allant de la première à la dernière étape. Il est alors possible de valider une procédure une fois que toutes les étapes de ce chemin sont validées. Cela permet également d'ajouter de nouvelles fonctionnalités comme un fil d'Ariane, une liste des étapes et un pourcentage de complétion. Ces fonctionnalités sont fortement liées au chemin qu'emprunte l'utilisateur et sont amenées à être modifiées au fil de l'exécution de la procédure. En effet, en choisissant un chemin conditionnel le pourcentage de complétion diminue et la liste des étapes s'allonge.

4.1.3.2 Amélioration des outils de gestions des procédures

Avant la refonte, les procédures en cours d'exécution étaient affichées dans le panneau descriptif de l'entité procédure ainsi que dans la page d'accueil de l'utilisateur. Ce mécanisme ne permettait pas une gestion des exécutions par projet et pouvait même entraîner des surcharges d'éléments dans l'interface.

Pour pallier ces problèmes nous avons créé une nouvelle entité dédiée à la gestion des procédures. Celle-ci permet d'affilier une liste de procédure pour ainsi suivre l'exécution de toutes les procédures lancées depuis cette entité.



GESTION DES INSTANTES DE PROCEDURES AVEC ET SANS GESTIONNAIRE DE PROCEDURE

Le gestionnaire de procédures permet de segmenter les différentes instances de chaque procédures. En effet, dans l'onglet procédures en cours, toutes les instances de cette procédure qui sont lancées simultanément sont affichées. Bien que chaque instance dispose d'un nom, il devient vite compliqué de s'y retrouver. En revanche, dans les processus associés, seules les instances créées depuis cette interface apparaissent.

De plus, la mise à jour par socket permet de synchroniser l'état des procédures en temps réel assurant la cohérence des informations et une vue d'ensemble plus sécurisée.

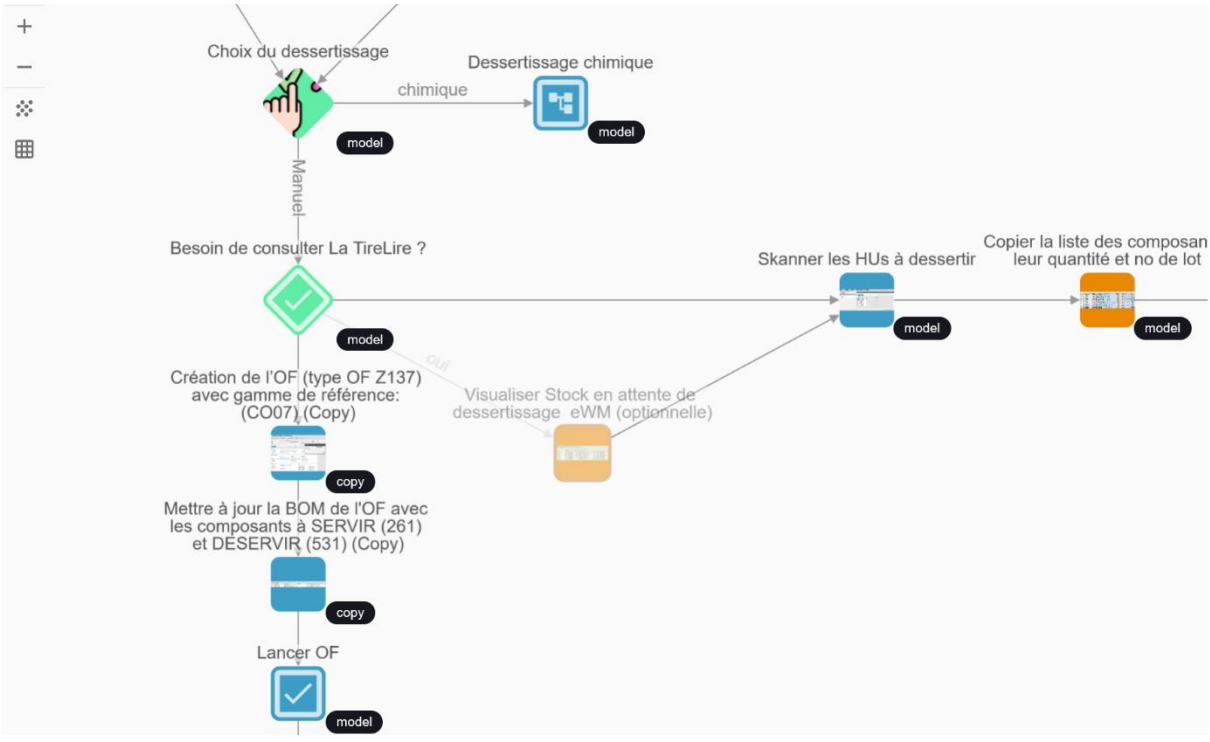
Enfin des statuts ont été ajoutés, ils permettent d'identifier en un coup d'œil les éventuels blocages sur une étape. Couplés à la barre de progression de la procédure, ils permettent aux managers d'être plus réactifs vis-à-vis de leurs équipes.

4.1.3.3 Intégration de l'héritage dans les procédures

La mise en place d'héritage dans les procédures permet leur standardisation et leur adaptation. L'entreprise peut définir des procédures modèles et autoriser leur adaptation dans un contexte local. C'est aussi un moyen de diminuer le nombre de doublons facilitant ainsi la maintenance de ces procédés standardisés.

En pratique une procédure modèle ne diffère en rien d'une procédure classique. L'utilisateur devra créer une suite d'étapes et de liens pour les relier. La relation parent/enfant se crée lorsque cette procédure est copiée à partir d'un modèle. L'entité qui sera créé à la suite de cet évènement sera légèrement différente puisqu'elle bénéficie :

- D'élément graphique rappelant la provenance de chaque étape, local ou modèle
- D'affichage fantomatique de la procédure modèle lorsque des étapes sont supprimé
- De bouton supplémentaire permettant le retour à la version d'origine



AFFICHAGE D'UNE PROCEDURE LOCALE AVEC HERITAGE

En ce qui concerne la propagation des modifications, elles se font sans action de l'utilisateur lorsqu'une étape n'est pas modifiée. De même lorsqu'une étape est ajoutée dans la procédure modèle, elle sera ajoutée automatiquement si la procédure locale n'a pas modifié les étapes qui l'entourent. En revanche, si ces étapes ont été modifiées, la

nouvelle étape apparaîtra de manière fantomatique dans la procédure locale. Enfin, lorsqu'une étape est modifiée dans la procédure locale et modèle, l'utilisateur peut choisir d'intégrer la nouvelle version de l'étape.

Cette gestion augmente la flexibilité dans la mesure où les sites de production peuvent apporter leurs modifications sans compromettre la standardisation globales. Le coup en maintenance est aussi moindre puisque les mises à jour sont, autant que possible, propagées automatiquement.

4.1.3.4 Problème rencontré

Dans la première phase de développement, nous nous sommes heurtés à des problèmes de complexité dans la gestion des types de procédures. En effet, nous utilisons 4 types de procédures ayant chacun des règles de navigation et de validation d'étapes différentes. A ce stade l'organisation du code ne permettait pas de gérer cette diversité, ce qui entrainait des incohérences dans l'exécution et la validation. Les 4 types de procédures utilisés sont :

- Ordonnée sans statut
- Ordonnée avec statut
- Désordonné sans statut
- Désordonné avec statut

Les étapes ont alors fait face à des problèmes de validation. Les mécanismes de navigation et de validation étaient tout d'abord partagés et s'adaptaient mal aux différents types de procédures. Les sauvegardes et les statuts ne s'appliquaient pas au même moment, d'autant plus qu'en fonction du type de procédure ces mécanismes sont parfois automatiques. Ainsi, chaque ajout demandait trop d'attention pour ne pas enrayer le mécanisme d'un autre type de procédure.

Le projet a donc souffert de l'apparition de nombreux bugs dûs à l'incohérence dans la gestion des différents types de procédures, ce qui a affecté la fiabilité du module. Ces problèmes ont causé des retards au regard de la complexité de développement et de la résolution de bugs. Ces problèmes ont aussi impacté la satisfaction et la productivité des utilisateurs finaux en phase de test. Pour répondre à ce problème, nous avons revu l'architecture de notre module en veillant à bien séparer le code liée à chaque type de procédure. Le mécanisme de validation d'une étape a lui aussi été revu pour être en harmonie avec la gestion des statuts. Enfin, nous nous sommes concentrés sur la formation des utilisateurs en interne qui devait interagir avec un système bien plus complexe et l'expliquer au client.

4.1.4 Analyse critique et réflexive

4.1.4.1 Points forts de la mission

Cette mission a un intérêt stratégique important puisqu'elle consiste à adapter un module clef du logiciel dans le cadre d'une négociation commerciale avec un client. Apport qui pourra s'étendre à l'ensemble de nos clients et créer un avantage concurrentiel durable. C'est aussi l'opportunité d'aligner notre application sur les standards industriels.

Du fait de l'introduction de concept avancé, avec la théorie des graphiques pour modéliser les procédures et la mise en place d'un algorithme de parcours en profondeur pour détecter les chemins valides en temps réel, ainsi nous avons apporté une détection dynamique des embranchements conditionnels améliorant la robustesse du système.

Nous avons aussi amélioré l'UX et rendu les exécutions plus traçables via des statuts d'étape, indicateur de progression et fil d'Arianne. Nous avons aussi facilité la gestion des exécutions permettant à un manager de suivre l'avancement de ses équipes en un coup d'œil. Avec cela la synchronisation en temps réel apporte un réel plus, indispensable pour une expérience utilisateur fluide. Enfin le système de visualisation fantôme fonctionne bien pour permettre de toujours visualiser l'héritage dans l'éditeur de procédure.

4.1.4.2 Axes d'amélioration et leçon apprises

Nous avons fait face à une complexité grandissante du module à cause des différents types de procédures et des logiques de validation historiquement partagées. Ces problèmes sont dûs à un manque d'anticipation vis-à-vis de l'architecture du module, celui-ci n'étant pas conçu pour gérer différents modes de navigation et de validation. Ainsi, la nécessité de restructurer en partie le module, à entraîner un retard non négligeable dans le développement.

A l'avenir, il nous faudra porter plus d'attention à la structure du code, notamment lorsque plusieurs workflow se croisent. Mais aussi isoler les responsabilités, c'est-à-dire définir clairement les parties de code qui seront utilisées par tous les types de procédures et celles qui seront spécifiques à un type.

D'un autre côté, les feedbacks réguliers via les tests de notre Product Owner nous ont montré l'importance des mises au point permis par la méthode Agile. Ceux-ci ont permis de faire des ajustements dans la manière dont on avait d'envisager la solution.

4.1.5 Liens avec la formation

Liens avec la formation M2I	
Référentiel	**Compétence éprouvé**
Bloc 1 – Analyse stratégique du SI	Identification des faiblesses structurelles et incohérences fonctionnelles
	Intégration des pratiques du Lean management
Bloc 2 – Conception et Déploiement	Compréhension du besoin du client
	Utilisation de la théorie des graphs
	Développement de fonctionnalité (héritage, synchronisation)
	Réorganisation structurelle du module
Bloc 3 – Pilotage de projets complexes	Collaboration avec plusieurs acteurs, PO et testeurs
	Projet Agile, interaction et adaptation des livrables
Bloc 4 – Intégration de technologies complexes	Utilisation des sockets pour la mise à jour en temps réel
	Utilisation d'algorithme de parcours de graph modifié
	Gestion de l'héritage des procédures
Bloc 5 – Veille, documentation, communication de projets	Accompagnement et formation des équipes aux nouvelles fonctionnalités

4.2 Transition vers angular18 : mise en place d'une application hybride

4.2.1 Définition de la mission

4.2.1.1 Contexte général

AngularJS est un framework javascript crée en 2010 par Misko Hevery qui est alors ingénieur chez Google[7]. Pendant les 6 années suivantes cet outil sera massivement utilisé par des millions de sites web au point qu'il deviendra un incontournable des Single Page Application[8]. Cependant pour résoudre des problèmes de structure qui freinent son intégration à des projets plus récents, le Framework est recréé de zéro avec Angular2.0 en 2016.

Pourtant le 31 décembre 2021 l'entreprise Google, qui avait montré un rôle actif dans le développement d'AngularJS, annonce officiellement la fin de son support. Cela implique[9] :

- **Des vulnérabilités de sécurité**, les nouvelles failles ne seront plus corrigé.
- **Des problèmes de compatibilité**, les nouvelles versions des navigateurs pourraient ne plus être compatibles avec AngularJS ce qui nécessiterait une mise à jour rapide.
- **Amoindrissement du support communautaire,** on peut s'attendre à une baisse significative du nombre de développeurs sur cette technologie diminuant ainsi l'entraide.
- **Absence de nouvelles fonctionnalités**, les nouvelles librairies ne seront pas compatibles avec AngularJS

Cette annonce est un problème stratégique majeur pour l'entreprise PICC qui base tout son Frontend sur la technologie AngularJS. Au total, ce sont 3 applications pour un peu plus d'une dizaine de modules qui seront affectés.

Le maintien d'AngularJS freine la modernisation de l'architecture logicielle telle que la Rule Of One ou folder by feature structure qui sont des concepts modernes facultatifs devenus obligatoires dans l'architecture Angular2+. C'est aussi l'absence de nouveaux outils tel que TypeScript ou l'utilisation de modules loaders ??? . Enfin, c'est un frein au développement des compétences des équipes qui travaillent sur une technologie qui a déjà périclitée.

---

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

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

[9] 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 dernières versions d'Angular Materials
- De faciliter la maintenabilité et l'évolutivité grâce à un code mieux structuré et l'intégration de tests unitaires
- De réduire notre dette technique[10].

4.2.1.2 Objectif de la mission

L'objectif principal 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'impact 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 serveur couramment utilisée et qui n'a pas pour but de changer.

4.2.1.3 Présentation de l'équipe

Nous sommes deux à être chargés de cette mission, Simon Fulhabert qui a un rôle de supervision et moi-même avec un rôle de chef de projet. Ensemble nous définissons les objectifs principaux de cette mission et validons chaque étape de sa réalisation.

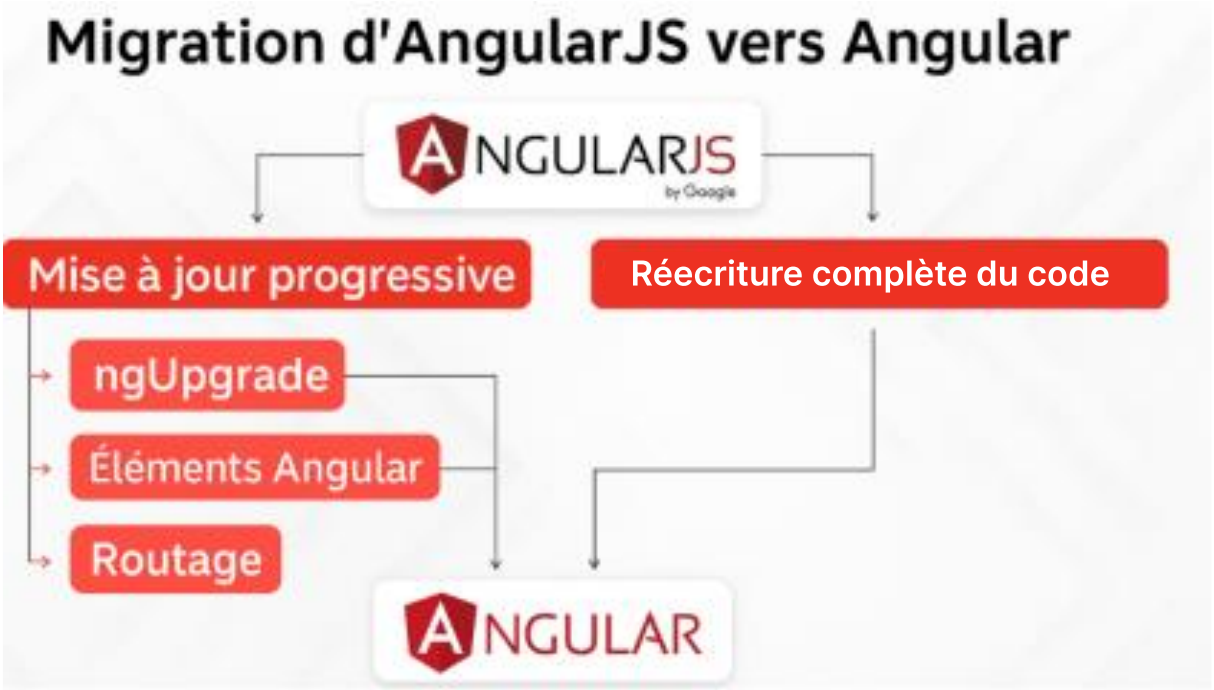
Les responsabilités qui m'incombent sont :

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

[10] 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

4.2.2 Choix technique et justification

4.2.2.1 Méthode de migration



SCHEMA DES DIFFERENTES METHODE DE MIGRATION VERS ANGULAR

Deux scénarios sont possibles 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 performances globales de l'application

Evidemment, cette méthode bien qu'elle permette de supprimer toutes dettes techniques présente de sérieux inconvénients puisque :

- Le coût en ressources humaines 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 utilisées peuvent être oubliées
- Deux versions de l'application sont à maintenir simultanément pendant une durée indéterminée
- La capacité de développement et de réactivité en cas de bug se retrouve / trouve divisée, ce qui n'est pas idéal dans les petites structures
- Les nouvelles fonctionnalités devront subir un double parcours de développement

Ainsi, cette stratégie n'est pas recommandée pour les applications de grande taille, notamment lorsqu'elles sont monolithiques. Cette méthode peut paraître brutale, cependant elle prend sens puisque l'effort humain cumulé sur toute la migration n'est pas plus important que pour la migration incrémentale 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 mutuelle des deux Framework. En somme, elle 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 bugs ou de comportements inattendus liés à la migration
- Réduire le risque technique puisque les équipes ont le temps de se former au nouveau Framework au fur et à mesure des nouveaux composants qui sont créés ou migrée vers 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 élevée. La mise en place d'un double bootstrap et la communication des deux Framework nécessite une compréhension rigoureuse des deux environnements, ce qui peut demander une montée en compétences. De plus, cette méthode nécessite que le code AngularJS se rapproche autant que possible du Style Guide Officiel de John Papa[11], si cela n'est pas le cas cette approche demandera une étape supplémentaire de mise à niveau des bonnes pratiques. Ainsi, la phase intermédiaire visant à rendre la solution hydride opérationnelle peut s'avérer assez longue.

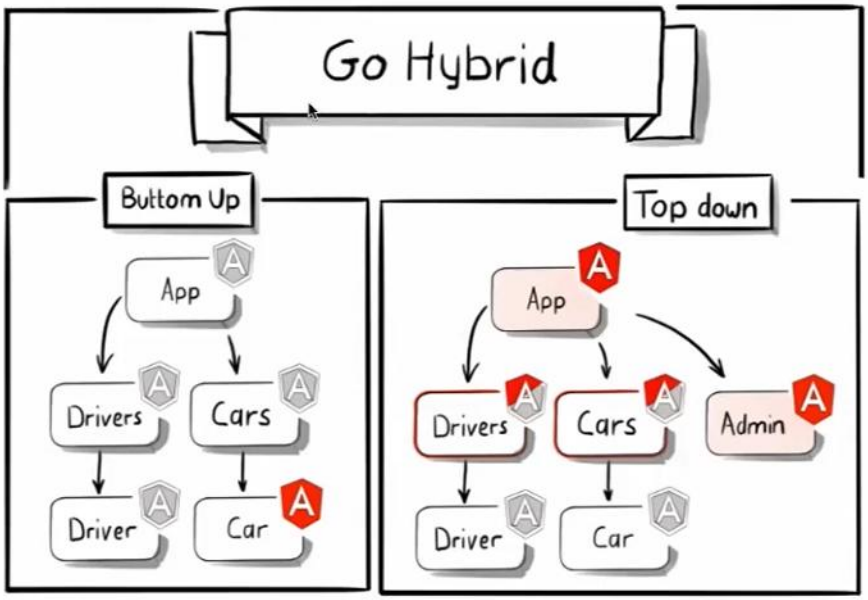
Enfin et bien que la fondation Angular ait fait de nombreux efforts d'accompagnement pour aider à la mise en place de cette solution, les projets open source ayant réalisés ces étapes de développement sont peu nombreux et l'on peut supposer que la plupart des entreprises ayant réalisées leur transition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

Notre choix portera donc sur la migration incrémentale 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 respecte 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.

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

4.2.2.2 Méthodologie d'hybridation

Il existe deux méthodes pour réaliser son hybridation comme l'illustre le graphique ci-dessous[12] :



SCHEMA DES DIFFERENTS METHODES D'HYBRIDATION

La première stratégie dit Bottom-Up est une migration progressive des composants AngularJS vers Angular2+ en partant des composants les plus bas dans l'arborescence vers les composants parents. Cette méthode permet de convertir en premier les composants les plus isolés, sans affecter l'ensemble de l'application. Elle présentera donc moins de complexités initiales car elle nécessitera moins de changements 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 nouveaux composants entièrement écrits avec Angurlar2+ tout en bénéficiant de toutes les nouvelles fonctionnalités apportées 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 importants en vue de notre architecture :

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

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

La différence entre les méthodes se résume à : « Quelle application encapsule l'autre ? ». Est-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 installer. En effet, les efforts à fournir pour uniquement préparer la mise en place de l'application hybride Top-down se compte en mois, durant lesquelles :

- Un nouveau projet devra être créé 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'ancien depuis le changement majeur
- Des librairies devront être mise à jour, impliquant potentiellement des changements dans leurs API.
- Des Tests manuels sur toutes les fonctionnalités de l'application

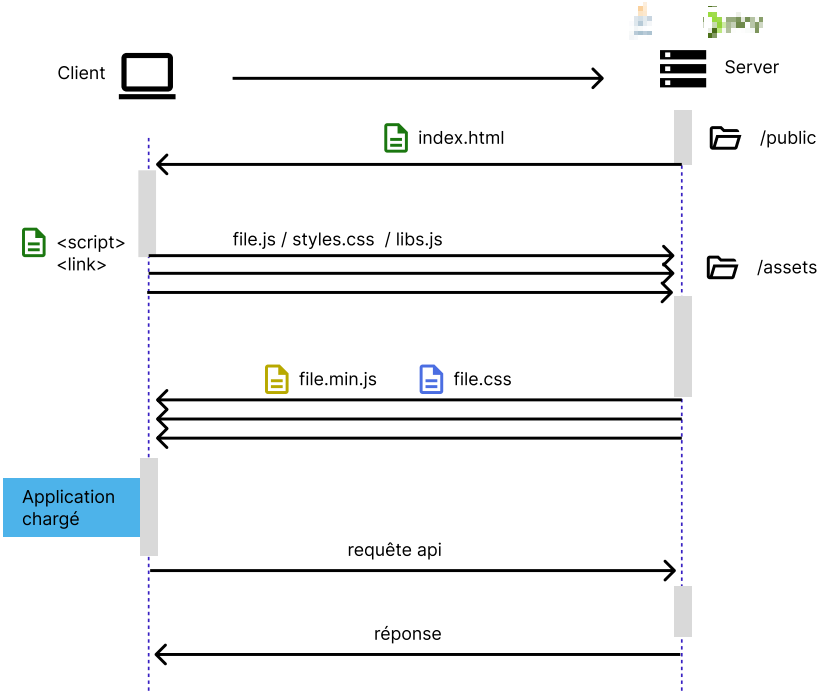
Cependant, la stratégie Bottom-Up n'est privilégiée que par 10% des entreprises. Pour cause, les efforts, somme toute assez conséquent conséquents, mis en œuvre ne valent pas les faibles gains apportés par un ponctuel ajout de quelques composants Angular2+ dans une application qui continue de se reposer 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 importants cités plus haut.

4.2.3 Méthodologie mise en œuvre

4.2.3.1 Phase de recherche et d'analyse de la nouvelle structure

Nous avons tout d'abord commencé nos recherches par une analyse des projets regroupant un Frontend Angular2+ et un serveur Java utilisant le Play Framework. L'enjeu était de comprendre la structure et les mécanismes d'intégration entre le Frontend et le Backend, ainsi que de les comparer avec la solution existante. Notre choix s'est résolu sur la angular-play-java-seed, un projet Github de 8 ans d'âge qu'il nous a fallu mettre à jour[13]. Les schémas suivants montrent comment l'application Web est distribuée par le serveur 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 séquences ci-dessus, la solution historique renvoyait à un fichier index.html contenant toutes les balises scripts pour tous les fichiers JavaScript de l'application. Ceux-ci étaient demandés au serveur de manière unitaire à la réception du fichier index.html, il en va de même pour le fichier CSS. Une fois

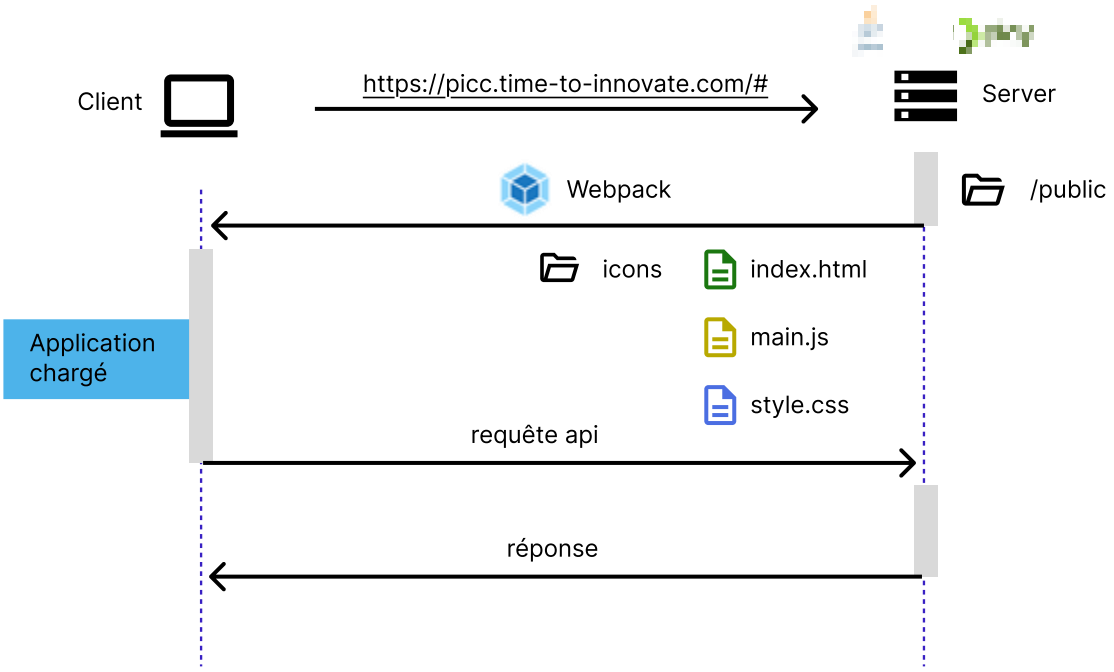
[13] playframework. (n.d.). *play-scala-angular-seed* [Code source]. GitHub. https://github.com/playframework/play-scala-angular-seed

cette transaction terminée et les premiers templates HTML reçus, l'application était chargée et disponible.

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

Le chargement du Framework AngularJS se fait via des balises Script qui chargent les librairies principales et optionnelles du Framework. Cette méthode repose sur le **Globale Name Space pattern**, qui permet une mise en place simple et une disponibilité dans tous les navigateurs 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, tous les fichiers javascripts sont chargés en bloc dans le fichiers index.html, seul une minification et une obfuscation sont effectuées lorsque le serveur est utilisé en production. Il en va tout autrement du beau[14].



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

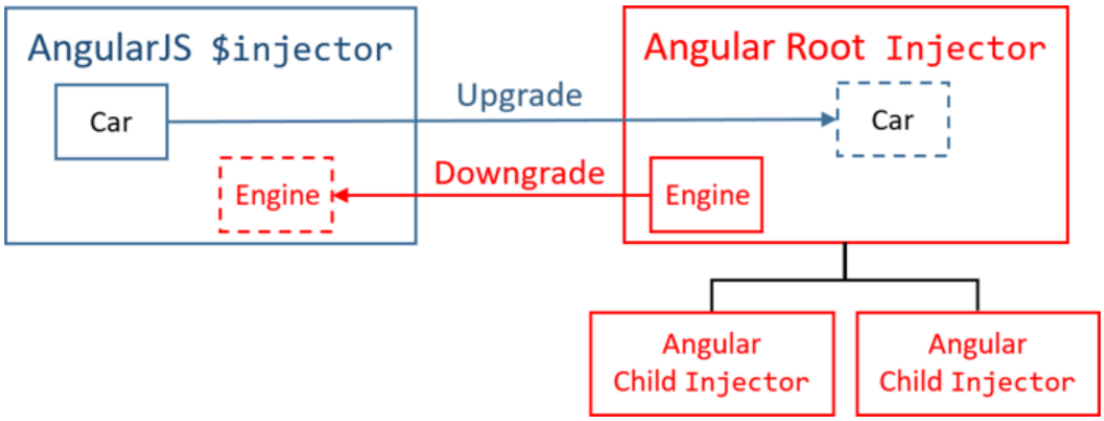
En effet, la nouvelle solution intègre des outils plus complexes impliquant des étapes de compilation, d'optimisation et de paquetage qui modernisent la solution. Ces changements commencent par la structure puisque le code de l'application Frontend est contenu dans le dossier */ui*, ce qui en fait une application à part entière intégrant une gestion des sources externes via NPM.

De plus, le dossier */public* est à présent temporaire, il est auto-généré au lancement de la commande STAGE (sbt shell). Cela implique des changements de structure dans la manière de ranger les fichiers statiques. Ce dossier */public* comprend les ressources de l'application Angular2+ comme les images et les *polices caractère* fonds, le point d'entrée index.html mais aussi l'application Angular2+ et ses dépendances, compilées, minifiées et paquetées en 3 fichiers javascripts : main, polyfills et runtime.

4.2.3.2 Test des différents mécanisme d'hybridation

Nous avons testé tous les différents mécanismes d'hybridation possibles ce qui nous a permis à terme de définir la méthodologie la plus adaptée à notre projet. Pour cela, il nous a fallu comprendre comment fonctionne la librairie UpgradeModule fournie par Angular[15].

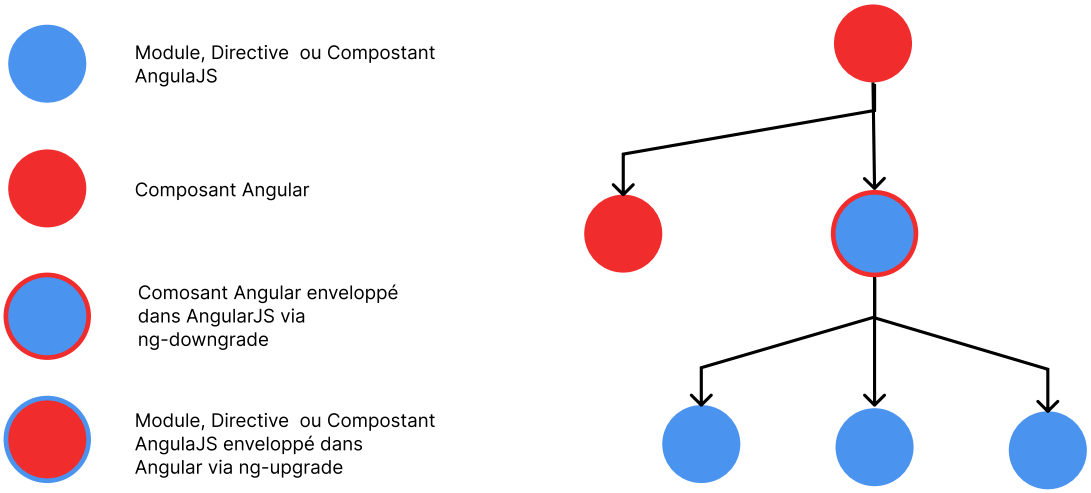
Lorsque nous utilisons ce module, les deux Framework cohabitent dans la même application. En effet, le code AngularJS tourne à l'intérieur de son propre Framework et vise vers ça pour le code Angular. Cela implique qu'il n'y a pas de perte de fonctionnalité puisque le Framework AngularJS n'est pas émulé,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



SCHEMA DE L'INJECTION DE DEPENDANCE DANS UNE APPLICATION HYBRIDE

[15] Angular. (n.d.). *Upgrading from AngularJS to Angular*. Angular Documentation. https://v17.angular.io/guide/upgrade

Comme vous pouvez le voir sur le graphique ci-dessus les composants et les services gérés par un Framework peuvent également interagir avec l'autre. Ce mécanisme n'est pas limitant, sa force est de permettre d'utiliser des nouveaux services dans les deux sens. Même si un composant n'est pas immédiatement migré il peut tout de même bénéficier des services nouvellement créés avec Angular[16]. De plus et dans le cas d'un Singleton la même instance de ce service sera partagée entre les deux Framework.

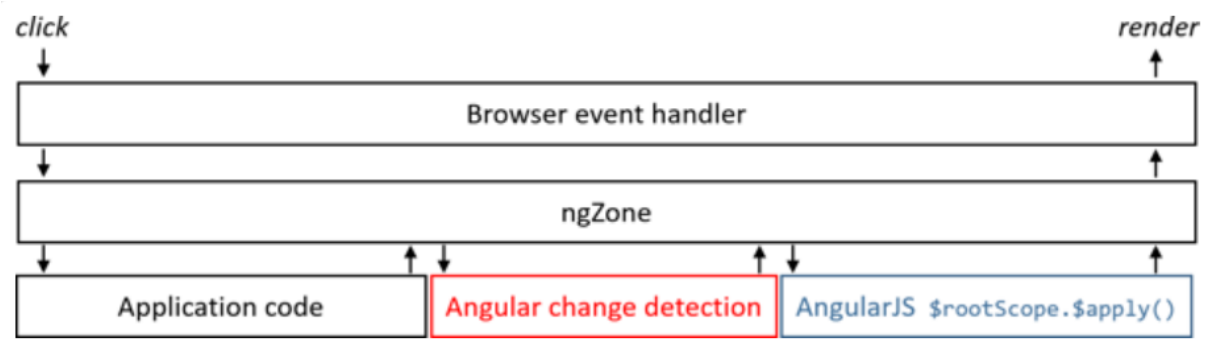


HIERARCHIE DES COMPOSANTS DANS UNE APPLICATION HYBRIDE

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 due à ces particularités :

- Les directives html applicable à un composant seront toujours celles du Framework lié aux composants parents ce qui demande une certaine 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.

[16] Codurance. (n.d.). *Migrating AngularJS to Angular*. https://www.codurance.com/publications/migrating-angularjs-to-angular



[17]

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 de leur code. L'Astuce qui rend ce mécanisme possible réside 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 changements. Par la suite l'UpgradeModule appelle par lui-même la détection des changements propres à AngularJS. Ainsi, c'est le Framework Angular qui capte tous les évènements provenant du navigateur et s'assure de les transmettre au workflow du Framework concerné. Cependant, cette méthode n'est pas sans défaut, cet aller-retour entre les deux Framework est une perte indéniable de temps dans le cycle de l'application. Nous nous retrouvons, à ce stade, contraints à avancer dans le projet tout en sachant que des tests devront être faits 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 organisationnels. Ceux-ci étaient encore incertains car à ce stade de développement du projet nous ne savions pas encore quelle méthode serait la meilleure. Cette cohabitation contrôlée offre une souplesse indispensable que ce soit au niveau du rythme de migration que pour la gestion des services partagés.

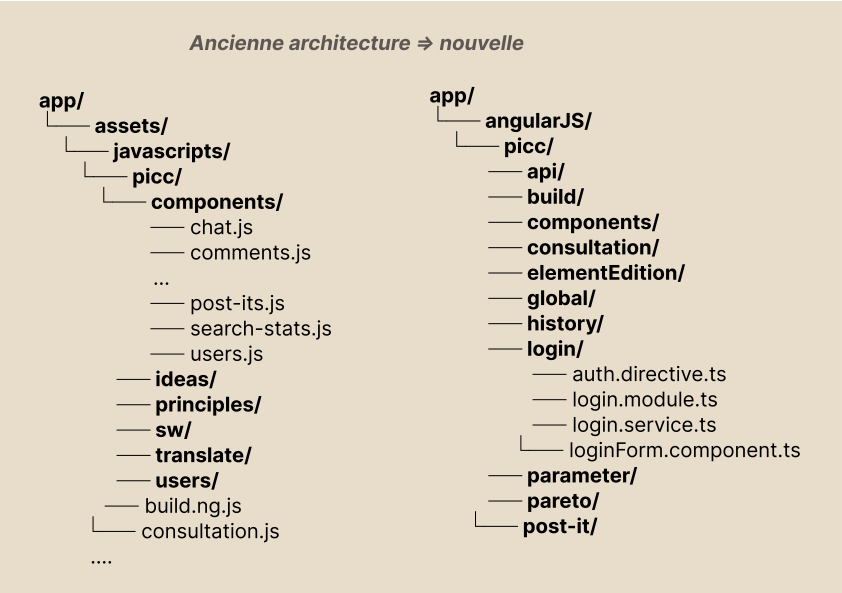
[17] Angular. (n.d.). *Upgrading from AngularJS to Angular*. Angular Documentation. https://v17.angular.io/guide/upgrade

4.2.3.3 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 restructurée selon les derniers principes d'Angular, cette étape est intermédiaire, elle prépare l'application pour une migration future en intégrant des principes de programmation qui harmonise l'écriture du code entre les deux Framework.

La programmation orientée objet, consiste à transformer les anciens contrôleurs ou services en classes ES6 avec constructeurs, propriétés et méthodes. Cela apporte une meilleure lisibilité et encapsule la logique du comportement dans chaque classe. Chaque élément de l'interface devient une classe structurée et isolée 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 implique la création de type explicite comme les interfaces, enums et typage de fonction permettant une détection plus rapide des erreurs à la compilation. Cette étape est définie comme une tâche de fond qui sera complétée au fur et à mesure de la transition vers Angular. En effet, le cout en ressource humaine et en temps pour recréer 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 sont définis les différents modules. Les fichiers dans le dossier composant contiennent généralement plusieurs composants et tous les composants de l'application ne sont pas contenus dans ce dossier. Il y a un mixte entre les méthodologies 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* contiennent 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 directive 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 composants comportant parfois plus de 7.000 lignes de code au total. Partant de ce remaniement de la structure générale, nous avons décidé d'y intégrer le principe de *feature folder structure*. Chaque fonctionnalité métier sera à présent isolée dans un dossier contenant ses propres 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 fini la restructuration. Cette phase de traversée du désert est critique car elle peut donner lieu à l'apparition de nouveaux bugs qui seront difficiles à repérer et réparer une fois l'application reconstruite. Nous devions donc trouver une méthode pour appliquer ces changements tout en modifiant au minimum le code source d'origine.

Lorsqu'il faut réécrire un code en programmation orientée 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 :

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

La quantité de modification que demande cette méthode est trop importante et peut entrainer des problèmes tels que l'oubli de certaines fonctions, propriétés ou arguments 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 importantes.

Pour réaliser ces modifications nous nous sommes orientés vers une solution qui impactait 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 vue moins propre nous permet :

- De conserver au maximum le corps des composants et des services
- De supprimer un niveau de profondeur en opérant l'extraction d'un composant vers un fichier sans toucher unitairement à chaque fonction
- De ne corriger que les erreurs TypeScript et les changements majeurs de librairies mises à jour
- De ne pas faire de re-réécriture massive du code pour chaque composant ou service en conservant les propriétés globales définies dans le constructeur

Cette phase de modification fut la plus longue du projet, chaque fichier d'origine a été divisé en parfois une dizaine de fichiers dans la nouvelle architecture. Le nouveau compilateur a révélé des erreurs dans le code qui n'avaient pas été repérées jusque à présent et qu'il a 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 derrière le code. Ainsi, chaque modification du code a été scrupuleusement référencée pour faciliter un potentiel retour en arrière.

D'un autre côté, les librairies qui était importées via le globale namespace paterne depuis un fichier statique référencé dans le fichier index.html se sont sûr vu être importées en tant que module via NPM. Ce changement se faisait le plus régulièrement sans problème à l'exception de certaines librairies qui ne maintenaient pas leurs versions globale namespace paterne à jour. Ainsi, en passant à la version module de la librairie nous avions plusieurs versions 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 hybridée. 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.

4.2.3.4 Difficultés rencontrées

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 modifications faites à leur template. Ce problème bien que 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 cherchait 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 dès lors impossible de stocker nos fichiers HTML directement dedans. Les fichiers HTML placés 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. Cependant et bien que le compilateur en mode **verbose** nous montre que le fichier modifié est bien pris en compte, cette méthode ne fonctionnait que pour la première modification apportée. En effet, une deuxième modification et relancement de l'application n'appliquait plus le 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 cet échec, nous avons repris le problème à sa base et envisagé de le contourner par un autre moyen. Le problème résidait dans une utilisation différente de l'instruction templateUrl par les deux Framework. Nous avons alors choisi d'utiliser un combo d'instructions pour contourner le problème (la difficulté) : template + require. Cette méthode permet d'importer un fichier HTML présent dans le même dossier que son composant. Elle permet également d'apposer le code HTML au côté 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 fichiers HTML via l'instruction *require*. Ainsi, les modifications des fichiers HTML sont désormais prises en compte 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 sources de l'application d'origine devenaient obsolètes 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 modifications. 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 lignes supprimées et déplacées ce qui nous a systématiquement amené à le perdre.

Une fois la restructuration terminée, 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** avancés. Ayant conservé les fichiers d'origine à leur emplacement initial, nous pouvions dans un éditeur GIT, montrer leurs différences avec une branche plus avancée. Chaque différence devait alors non pas être acceptée, mais copier/coller dans la nouvelle architecture. Cette étape fut réalisée après une longue phase de tests et de résolution de bugs dus à l'hybridation de l'application, elle présentait un grand risque pour le projet car nous pouvions sans nous en rendre compte ajouter de nouveaux bugs.

4.2.4 Évaluation et résultats obtenus

4.2.4.1 Indicateurs de performance et résultats quantitatifs

Performance

Avant l'hybridation l'application présentait des temps de chargement raisonnables qui n'impactaient en rien l'expérience utilisateur. Une légère augmentation du temps de chargement a été observée 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 légères baisses de réactivité au niveau de l'interface dues à la synchronisation des cycles des deux Framework. Cependant, ces baisses n'ont pas été constatées. Pourtant, certains modules comme le Build & Solve ont subi une forte augmentation de leurs temps de réponse. Cette 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 discrets a pu être constatée notamment grâce à l'introduction de TypeScript et à une meilleure structuration du code. Certains bugs n'avaient jamais été repérés via l'interface utilisateur mais ils ont été immédiatement vus puis corrigés grâce au compilateur de TypeScript. Cependant, des bugs spécifiques à la cohabitation entre AngularJS et Angular ont émergé, ceux-ci nécessitent une attention particulière.

4.2.5 Analyse critique et réflexive

4.2.5.1 Point forts de l'approche utilisé

La stratégie de migration progressive est maîtrisée et nous pouvons valider la migration incrémentale via l'UpgradeModule. Celle-ci, nous permet d'intégrer progressivement Angular sans réécriture complète du code source. Elle a aussi garanti 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 maîtrise de la hiérarchie des composants tout en rendant possible une migration par module, par service ou par composant. Cette méthode a su montrer qu'elle était non destructive et nous a permis de capitaliser sur les modules AngularJS déjà existants tout en nous offrant la possibilité d'intégrer de nouveaux composants Angular à l'intérieur d'anciens modules AngularJS.

C'est aussi une montée en compétences 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 stratégies de détection de changements. Connaissances qui ont données lieu à la création de référentiels internes sur la création de composants Angular2+ et un socle commun de connaissances sur la seed Angular18 et Play Framework.

4.2.5.2 Axes d'amélioration et leçons apprises

Au niveau de l'anticipation et du pilotage technique, le projet a subi un manque notamment sur certains impacts liés au scope partagé de deux FrameWork. Cette cohabitation bien qu'elle soit fonctionnelle, 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 critiques qui aurait permis de mieux cibler les zones à optimiser en amont de la migration.

La refactorisation a 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 ralentit le passage vers TypeScript. Une meilleure planification du typage fort, en ciblant en priorité les objets les plus conséquents de l'application, aurait permis une adoption plus efficace.

La gestion de projet a reposé sur un pilotage réactif, plus que stratégique. Le projet n'avait pas au départ de chemin clairement planifié, bien que des livrables intermédiaires fussent présents tout au long du processus. Certaines décisions importantes, comme la

réorganisation en Folder by feature ou l'implémentation d'un webpack personnalisé, ont été prises en réaction aux problèmes courants sans réelle planification au préalable. Pour 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 entiers à la montée en compétences qui s'est faite au fil des erreurs rencontrées.

4.2.6 Compétences acquises et liens avec la formation

Cette mission m'a permis d'approfondir mes connaissances techniques en développement frontend par /grâce à la mise en place d'une nouvelle architecture et à la configuration d'un projet sur une technologie nouvelle pour moi. Cela m'a donné la possibilité de monter en compétences dans les domaines suivants :

- Développement Angular18
- Utilisation avancée du CLI Angular pour la configuration et la compilation de projets
- Stratégies de distribution d'une application web et les webpack
- 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 techniques complexes, ainsi qu'à des enjeux réels face à la réalisation de ce projet pour la pérennité de l'entreprise ainsi que des enjeux de priorisation. Cette mission m'a alors apporté des compétences en gestion de projets tel que :

- L'élaboration et le suivi d'un plan de migration incrémentale sans rupture de service
- La Gestion du risque liée à la dette technique d'AngularJS
- L'adaptation agile des priorités face aux imprévus comme la compatibilité HTML
- La priorisation des tâches liées à la dette technique, tout en assurant le maintien et la correction du logiciel existant
- La rédaction de guides de migration et de documentations des composants
- La réalisation d'une étude comparative sur les stratégies de migration
- La sélection mise en place et l'amélioration d'un environnement Angular + Play Framework

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

5 Conclusion et perspectives

La réalisation des missions décrites dans ce mémoire a été déterminante pour la compagnie PICC ainsi que pour mon développement professionnel. En effet, la refonte du module de procédure et la transition technologique vers Angular18 correspondaient directement aux enjeux stratégiques déterminés en introduction, à savoir l'amélioration de l'efficacité opérationnelle et la pérennisation technique de l'application.

Pour l'entreprise la refonte des procédures à permis de s'aligner sur les exigences managériales de groupes industriels opérant à l'international. C'est aussi un gain pour nos équipes puisque ce module leur permettra d'améliorer la rédaction de documentations en interne. En ce qui concerne la transition vers Angular18, c'est avant tout une réduction de la dette technique en apportant des exigences actuelles en matière de performance et de sécurité des applications web. Ce qui permet également d'envisager des stratégies de développement plus modernes comme le développement piloté par le test *(TDD)* et le typage fort via TypeScript.

Ces missions m'ont permis de développer des compétences personnelles, telles qu'une meilleure autonomie, une capacité à prendre des initiatives et a renforcé mes compétences relationnelles. Tandis que, sur le plan professionnel j'ai eu la possibilité d'approfondir mes connaissances techniques avec angular18 et la mise en place d'applications hybrides. J'ai également bénéficié de l'acquisition de connaissances en gestion de projet via la planification, le suivi, la gestion des risques et des imprévus. Enfin, j'ai eu la chance de de pouvoir appliquer mes propres choix et d'expérimenter avant de trouver des solutions valides ce qui a fortement renforcé ma capacité à mettre en place une veille technologique et à proposer des solutions innovantes.

Cette alternance m'a ouvert la porte au pilotage de projets complexes et de longue durée, tout en consolidant mon expertise technique en développement et en ingénierie logiciel. Cette formation me permettra dans un future proche d'accompagner des entreprises dans leur transition technologique future.

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/

Préfecture de la région Grand Est. (n.d.). *Business Act Grand Est*. https://www.prefectures-regions.gouv.fr/grand-est/Actualites/Economie-et-emploi/Developpement-economique/Business-Act-Grand-Est

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

Wikipédia. (n.d.). *Hutchinson (entreprise)*. Wikipédia. https://fr.wikipedia.org/wiki/Hutchinson_(entreprise)

France Hydrogène. (n.d.). *Hutchinson – Annuaire des acteurs*. https://vighy.france-hydrogene.org/annuaire-des-acteurs/hutchinson

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

endoflife.date. (n.d.). *AngularJS*. https://endoflife.date/angularjs

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

Angular. (n.d.). *Upgrading from AngularJS to Angular*. Angular Documentation. https://v17.angular.io/guide/upgrade

umdjs. (n.d.). *umd* [Code source]. GitHub. https://github.com/umdjs/umd