Parler du fait qu’on réutilise le starter kit existant qui n’est pas fait pour le micro-service et qu’on l’adapte en micro-service.

Couverture

Parler du fait que grâce à Amilapp, l’entreprise va enfin créer ses propres packages sur npmjs, évite la redondance de code dans les projets, surtout dans un projet micro-service où chaque service est une API.

Permet de se préparer au Cloud, Amiltone ne l’utilise pas pour le moment.

Micro-services adaptés à la méthode agile, faire des livraisons à chaque fin de sprints. Grâce aux micro-services, il est même possible de livrer et de mettre en route directement l’application sans que le tout soit prêt.

Explications sur les micro-services (11 juin 2020) : [https://www.technologies-ebusiness.com/enjeux-et-tendances/architectures-micro-services-objectifs-benefices-defis-partie-1](https://www.technologies-ebusiness.com/enjeux-et-tendances/architectures-micro-services-objectifs-benefices-defis-partie-1%20(11) <https://www.technologies-ebusiness.com/enjeux-et-tendances/architectures-micro-services-objectifs-benefices-defis-partie-2>

Parler du développement de Babyltone V2 quand j’étais en 3ème année, montrer l’importance d’un chef de projet, d’un lead dev, des MR, etc… Montrer qu’Amiltone à appris de ses erreurs et à amélioré le déroulement des projets interne.

Parler du fait que le lead Dev est parti en mission (probablement à cause du confinement)

Parler du possible passage sur ESlint au lieu de tsLint qui est déprécié : permettra une meilleure réutilisation avec une meilleure visualisation des erreurs dans le code.

Présentation de l’entreprise

Question à poser à Alex (idéalement) :

* **Qui sont les concurrents d’Amiltone** ? Niji / SQLI / Smile / Agixis
* **Quels partenaires** ? Sword / Corus / Sopra / Aprr
* **Quelle politique de développement** ? En croissance, on recrute en permanence (une dizaine de personne par mois mai un peu compliqué en ce moment  
  à cœur de faire une croissance organique, augmentation naturelle sans rachat ou recrutement de masse, rencontre systématique avec les CEO pour approuver la personne, recrutement tourné vers l’humain, pas juste un profil. D’un côté, gros recrutement avec une dizaine de nouveau par mois, mais de l’autre côté reste humain  
  Chaque agence peut avoir ses spécificités dans les technos utilisée, technos adaptées à la région, pas à Paris, trop de concurrence mais basé à Niort qui s’en rapproche en termes de localisation.
* **Organigramme de la société**
* **Comment se passe un projet client, du début à la fin (va me permettre de comparer avec le projet interne) (peut-être à demander aux chefs de projet plutôt)**
* **Quels ont été les plus gros clients au cours de l’histoire d’Amiltone (avec rapide explication du projet)** ? Sopra Steria : filiale édition de logiciel ERP en gestion immobilière et les Hospices Civiles de Lyon sur le système d’information dossier patient médicalisé
* **Demander plus de description sur la branche ingénierie industrielle d’Amiltone**
* **Quelques chiffres (chiffre d’affaire sur 2019)** CA 2019 : 16M € - 240 collaborateurs
* **Pourquoi s’arrêter à 300 salariés ?**
* **Parler de la branche design, pourquoi avoir fondé studio Bluck ?** Afin de compléter l’offre en matière de transformation digitale et être en mesure d’accompagner les clients sur toute la démarche de transformation digitale
* **Pareil pour DNR ?** Afin d’accompagner Aprr dans le développement d’une nouvelle offre de service conjointe

**Pourquoi sommes-nous partis sur les micro-services pour le backend d’Amilapp ?** Pour des questions de modularité et être en mesure de pouvoir déployer la solution éventuellement pour d'autres clients dans l'avenir en mode édition logicielle Saas. Et les microservices permettent de pouvoir piloter finement la charge sur les serveurs AWS

* **Comment a été gérée la crise du coronavirus ?**

Présenter les différentes certifications qu’à acquis Amiltone.

Principaux éléments relatifs à l’entreprise notamment ceux qui ont un rapport avec le sujet du mémoire :  
Entreprise assez jeune dans son état d’esprit et qui a à cœur d’innover dans les nouvelles technos, ce qui m’a permis d’apprendre énormément (parler du panel de technologie d’Amiltone,

Malgré un grand nombre de collaborateur, pas l’impression d’être dans une usine,  
Quelles opportunités : travailler sur des technologies de pointes et récentes telles qu’Angular, NodeJS

Analyse du fonctionnement de l’équipe de travail :

Organigramme :

Directeurs  
commerciaux – Chefs de projets  
lead dev  
dev

Fonctionnement :  
Suivi de la méthode agile du mieux possible sachant que je travaille sur un projet interne et donc moins « important » qu’un projet client. Daily meeting tous les matins avec toutes l’équipe, le chef de projet, ainsi que le lead dev de temps en temps, et designer qui s’occupe des maquettes du projet s’il y en a des nouvelles. Sprint de 2 semaines pour permettre une meilleure réactivité aux possibles évolutions et améliorations.   
Lorsqu’il y a des nouvelles grosses évolutions avec des nouvelles page sur le site web, grosse réunion avec chef de projet, designer et les développeurs. On parle des maquettes, de ce qui est faisable ou non, plus ergonomique, chacun peut donner son avis, tout le monde est écouté. Nouvelle réunion avec les développeurs et le chef de projet qui a prévu un tableau avec toutes les nouveautés, estimations de toutes les tâches. Etant sur le projet depuis le début, je m’occupe de créer les différentes tâches sur le tableau Jira. A chaque fonctionnalité développée, une merge request doit être faite avant de merger sur la branche principale.

Communication :  
utilisation d’outils professionnel pour la communication comme Teams. Communication très facile avec tous les développeurs puisque nous sommes tous dans le même bureau. Possibilité de demander de l’aide à tout le monde. Si quelque chose ne va pas, que ce soit au sein de l’équipe ou autre part, possibilité d’en parler au chef de projet qui essaiera de changer les choses. L’entreprise demande des retours à intervalles réguliers à ses développeurs ce qui rentre bien dans la méthode agile.

Condition de travail :   
dans un openspace avec environ 8 personnes. Climatisé.

Relations :  
De nouvelles têtes arrivent sur le projet de temps en temps. Besoin qu’ils se forment sur les nouvelles technos, plus simple de se former sur un projet interne qu’un projet client. L’équipe change donc souvent. De mon point de vue, cela peut faire perdre pas mal de temps au projet car je dois expliquer le projet aux nouveaux arrivants, les former, etc. Mais, point positif, cela me permet de former ces personnes justement, j’apprends de nouvelles choses aussi et finalement, j’aime bien le faire.

Spécificités :   
Projet interne donc à priori pas de deadline mais plutôt des objectifs.

Répartition des tâches :  
chacun peut choisir la tâche qu’il préfère accomplir tout en respectant l’ordre logique de réalisation. Il faut d’abord créer l’architecture du projet avant de commencer à créer une page de connexion par exemple.

Place du stagiaire.

Sur le projet Amilapp depuis 2 ans maintenant. Un des piliers du projet.  
Lead dev sur le projet uniquement. Je m’occupe de corriger les merge request faites par les autres développeurs de l’équipe. Quant à moi, j’envoie les merge request au lead dev de la web factory.

**Analyse du contexte**

Présentation générale de l’environnement :  
Amilapp vient de sortir en version 1.0. C’est une application qui permet une communication de l’entreprise envers ses collaborateurs plus simple et plus efficaces que les habituels courriels. Elle permet par exemple la création de sondages ou d’événements, habituellement communiqués via Teams ou mail. Tous les collaborateurs pourront y répondre facilement grâce à une application iOS ou Android. Amilapp dispose donc d’une application mobile, d’un backoffice administrateur et d’un frontoffice. Je me suis occupé des parties front et backoffice. Elles ont été codées en Angular 8. Amilapp ne dispose pas de backend à proprement parlé puisque nous utilisons la solution cloud de Google : Firebase. Elle permet la création rapide d’une application sans se poser les questions du « langage » de la base de données. J’ai utilisé le package AngularFire2 qui permet une intégration simple de Firebase avec Angular. Cette solution est pratique lorsque l’on souhaite développer une petite application rapidement. Mais dès lors que l’application devient plus complexe, on arrive rapidement aux limites de Firebase qui ne se retrouve pas suffisamment mature dans son fonctionnement pour nous satisfaire. Malgré beaucoup d’optimisations faites niveau code, on se retrouve forcément bloqué par la version gratuite de Firebase, et nous sommes alors obligé de passer par la case porte-monnaie. Pour une entreprise comme Amiltone, qui a développé ses propres solutions pour la gestion des collaborateurs, gestions des voitures de fonctions, ce n’est pas dans la philosophie de l’entreprise. De plus, cela permet aux collaborateurs en inter-contrat ou aux nouveaux arrivants de se former.

Comme Amilapp est un projet interne, un projet sans deadline, qui permet de progresser, le but est d’utiliser de nouvelles technologies, que les collaborateurs n’ont pas encore l’habitude d’utiliser. C’est le cas des micro-services. Le backend sera donc fait en micro-services et remplacera complètement Firebase. Le projet servira de base de lancement à cette façon de créer un projet.

Chez amiltone, le département web, la web factory, a créé un starter-kit, un projet de base avec énormément de fonctionnalité de base, tels que les différentes façons de se connecter en tant qu’utilisateur, une gestion de rôle, d’envoie de mail, de planificateur de tâches, qui peuvent potentiellement servir sur n’importe quel projet client. L’objectif du starter-kit est d’éviter de repartir de zéro à chaque nouveau projet.

Amilapp permettra la création d’un nouveau starter-kit, basé sur l’ancien, fonctionnant grâce aux micro-services.

Analyse du contexte

Acutalité dans le secteur de l’informatique :

Microservices de plus en plus en vogue, le cloud pareil.

Problématique : Comment adapter une application existante en marque blanche grâce aux micro-services.

Possible nouvelle problématique ? Comment créer un starter-kit backend en NodeJS grâce aux micro-services. OU Mise en place d’un starter-kit évolutif (en fonction des besoins) compatible avec les micro-services

Connaissances spécifiques nécessaires pour traiter le sujet du mémoire :

(18 juillet 2020) : <https://blog.octo.com/larchitecture-microservices-sans-la-hype-quest-ce-que-cest-a-quoi-ca-sert-est-ce-quil-men-faut/>

« Microservices are small, autonomous services that work together » (18 juillet 2020) <https://www.oreilly.com/library/view/building-microservices/9781491950340/ch01.html>

Expliquer l’achitecture microservice avec un schéma peut être pour que ce soit plus simple et plus visuel, et du coup, montrer quels partie je dois développer pour le starter-kit.

Méthodes habituellement utilisées pour une situation présentant des similitudes.

Parler de springboot (le starterkit java) et le .NET (démarrage d’un projet vide, possibilité de sélectionner une partie authentification soit déjà faite.

(11 Juillet 2020) <https://github.com/sahat/hackathon-starter>

(11 Juillet 2020) <https://github.com/w3tecch/express-typescript-boilerplate>

Exposé des décisions prises et des interventions menées par le stagiaire pour résoudre le problème

Utiliser cette façon d’écrire pour donner une définition des microservices, du cloud, des starter-kits : <https://medium.com/the-mission/the-greatest-sales-deck-ive-ever-seen-4f4ef3391ba0>

(31 juillet) <https://www.itpro.fr/les-microservices-sont-ils-lavenir-des-applications-en-entreprise/>

Les micro-service ?

**Définition des micro-services** :  
Le but de l’architecture est de découper une application en plusieurs petits services, autonomes et indépendants les uns des autres, contrairement à une architecture monolithique où l’application forme un tout indissociable. La plupart du temps, chaque service expose une API REST qui sera accessible par l’interface utilisateur ou n’importe quel autre micro-service. Chaque service aura alors son propre environnement et pourra être complètement différent des autres, que ce soit au niveau des technos utilisés ou des versions de celles-ci. On peut très bien imaginer, par exemple, un service codé en NodeJS avec une base de données MongoDB qui tourne grâce à Nginx et un autre service codé en Java avec une base de données Postgresql qui tourne grâce à Apache. Pour la plupart du temps, Docker est utilisé pour les micro-services.

**Avantage des micro-services :**

Est adapté au cloud, qui est de plus en plus utilisé.  
meilleure scalabilité : petit composants, autonomes et légers, simple à conteneuriser et à répliquer.  
Lors d’une mise à jour, il sera possible de cibler directement le micro-services en question

Inconvénients :  
très récent, pas de définition claire, évolue très rapidement, à besoin d’une veille technologique plus importante que pour les autres starterkit (spring, etc)

Pourquoi les micro-services ?

Nouvelle techno que personne ne connait dans l’entreprise.  
Sur le starterkit actuel, on se retrouve avec énormément de fonctionnalité et on devra supprimer celle qu’on ne veut pas utiliser, avec le risque casser des fonctionnalités. Avec les micro-services, on commencera un projet avec une brique de base très petites, et on pourra réutiliser d’autres micro-services pour les modifier ou les réutiliser comme on le souhaite.

Parler du fait que les micro-services ont été mal compris au départ et donc il manque des fonctionnalités importantes tel que le service de bus de message, celui de gestion du cache etc, et du coup, expliquer que la décision a été prise de déployer l’application actuelle en version de dev en gardant l’architecture actuelle parce qu’elle est bloquante pour la mobile. L’application reste fonctionnelle

Parler des objectifs et contrainte du projet, expliquer ce qu’on attend de chaque microservice

* Module de connexion à l’AD : sécurisé, accessible depuis l’extérieur
* Utiliser l’architecture microservices pour plus de modularité
* Remplacer Firebase

Expliquer les technologies choisies (on reprend les mêmes que le starter-kit : Tsed pour l’injection de dépendance, Docker pour le déploiement des containers, Kong pour l’API gateway) décrire l’architecture du projet, avec un schéma

Parler de Koa pour remplacer Express, mais pas assez matûre.  
Pourquoi OpenID plutôt qu’un autre système de connexion  
Comparer Tsed avec d’autre package npm qui permettent de faire de l’injection de dépendance : comparer avec NestJS qui fait globalement la même chose (mais en mieux ?) Demander à Damien pourquoi avoir choisi Tsed au lieu de NestJS

* Utilisation des annotations
* Génération de la doc swagger grâce aux décorateurs
* Ses fonctionnalités : auth middleware, injection de dépendance, génération d’erreur pour les requêtes en entrée et sortie
* Codé en typescript de base donc typé
* (12 aout 2020) <https://github.com/ebataille/express-annotations>  
  ce package pourrait être équivalent aux annotations que propose TSed pour le routing

Pourquoi Kong plutôt qu’un autre ?

(12 aout 2020) <https://www.moesif.com/blog/technical/api-gateways/How-to-Choose-The-Right-API-Gateway-For-Your-Platform-Comparison-Of-Kong-Tyk-Apigee-And-Alternatives/>

Déroulement du projet :

- phase de recherche sur les microservices avec Charles-Henri (lead technique) (comparer les différentes technos ici)

- Etant seul sur le projet au début, développement de chaque microservice en réutilisant le starterkit existant avec pour but d’être ISO avec la V1 d’Amilapp

* + Tri des fonctionnalités utile pour notre application + séparation des fonctionnalité nécessaires soit en microservices, soit en package npm
  + Parler de ma prise de décision de faire un microservice par fonctionnalité de l’application (news, sondages, événement, etc) plutôt que de faire un seul gros bloc qui aurait permi d’éviter la redondance de code, dire que c’est pour la modularité et que si on veut potentiellement vendre l’appli, tous les clients n’auront pas forcément besoin de toutes les fonctionnalités
* Création de package npm pour éviter la redondance de code comme par exemple la gestion de la connexion à mongo, les fichiers d’erreurs, nats
* Ajout de la connexion avec l’active directory avec un microservice dédié (parler du fait que c’est CAS qui devait être utilisé mais que CH est parti en mission et que seul lui avait avancé sur le projet (au moins 6 mois de travail) et qu’il aurait fallu beaucoup plus de temps pour que quelqu’un reprenne le truc et rattrape CH, du coup, utilisation d’OpenID connect)
* Ajout d’un mock de donnée pour la base collab qui permettra de gérer les utilisateurs (crud)
* Décision de Déploiement du projet sur une VM de dev avec Docker et gitlab CI pour que la mobile factory puisse développer l’application avec de vraies données malgré le fait qu’il manque des microservices.
* Développer les fonctionnalité V2 directement sans essayer d’être ISO V1
* Un collègue qui bossait sur le backend avec moi est parti en mission, décision que Haroun ou Yann me rejoigne sur le sujet pour accélérer le développement
* Parler du microservices message queue, pourquoi c’est utile, etc : <https://dzone.com/articles/microservice-architecture-best-practices-messaging>
* Parler du CQRS pour l’aggregation des données : <https://microservices.io/patterns/data/cqrs.html>

Parler de **TARGET** dans les news, les events, les sondages : est-ce que je vais en faire un micro-service à part entière ou est-ce que j’en fais un package amiltone (je pense plus pour l’option d’un microservice à part entière) (est-ce qu’il faut vraiment en parler ici ? c’est une fonctionnalité d’Amilapp et non pas d’un starter-kit microservice avec des fonctionnalités basiques

Démonstration d’une originalité dans l’élaboration et la mise en œuvre de la solution :

Différents des solutions existantes sur le marché, adapté au besoin de l’entreprise.

Analyse de l’approche choisie :

Demander aux chefs de projet le temps que prenait un nouveau projet sans starter-kit, puis avec starter-kit. Va permettre d’analyser les avantages et inconvénients de chaque méthode.

Comment serait réutilisable la solution dans une autre infrastructure ou sur un autre projet.

## Auto-évaluation du travail réalisé

Pendant la durée de l’alternance, mes semaines étaient découpées en 3 jours de cours et 2 jours en entreprise.

**4.3.4 Pourquoi les avoir choisis ? (Les micro-services)**

Aujourd’hui, le « Cloud Computing » est devenu un standard dans l’utilisation de nos applications ainsi que dans leur développement. Celles-ci doivent être disponible pour tous les utilisateurs à tout moment et depuis n’importe où. Pour une grosse application, choisir le cloud c’est permettre de l’adapter rapidement en fonction de l’utilisation qu’en font les utilisateurs et c’est aussi permettre aux équipes de développement de se focaliser sur le produit final. Ne pas l’utiliser, c’est devoir prendre le temps pour concevoir une architecture serveur robuste mais c’est aussi prendre le risque de voir augmenter les factures du matériel informatique, qui peut potentiellement ne pas être amorti en fonction des utilisateurs encore une fois.

Pour utiliser le cloud, il vaut mieux s’y préparer en développant directement des projets adaptés au cloud computing. C’est ici que les micro-services nous intéressent.

Une fois déployée, les micro-services permettent à une application d’être plus scalable, c’est-à-dire qui s’adapte d’un point de vue dimensionnel, tant vers des tailles inférieures que vers des tailles supérieures, en fonction des pics d’utilisation de celle-ci, et d’être plus tolérante à la panne.

Mais les micro-services permettent aussi aux entreprises d’optimiser les ressources consacrées au développement des applications. En effet, utiliser ce type d’architecture permet de diviser une grosse équipe en plusieurs équipes de tailles réduites, qui se consacrent seulement à un seul service, indépendamment des autres équipes, et est plus adapté au fonctionnement en mode agile. « *Les organisations qui conçoivent des systèmes […] tendent inévitablement à produire des designs qui sont des copies de la structure de communication de leur organisation »* (Loi de Conway), autrement dit, l’organisation des équipes découpées en sous-équipes indépendantes représente l’architecture globale de l’application, l’architecture micro-services. Comme chaque service est indépendant, chaque équipe peut prendre les décisions les plus adaptées et optimisées pour répondre au cahier des charge imposé. Enfin, la taille relativement petite de chaque service permet de les réécrire aisément avec des technologies plus récentes et plus pertinentes : la complexité globale du projet s’en retrouve réduite.

Contrairement à l’architecture monolithique, les micro-services sont optimisés pour la mise à l’échelle horizontale, c’est-à-dire que lorsque les capacités de la machine qui héberge l’application sont trop limitées pour supporter le nombre de requêtes utilisateurs, l’application est facilement déployable sur plusieurs serveurs distincts de façon à réduire la charge de travail entre eux. Et, contrairement à la mise à l’échelle verticale, l’application reste toujours disponible pendant l’ajout de ressources supplémentaires : si un des serveurs tombe en panne, l’application reste disponible.

Globalement, le projet devient plus facile à appréhender pour les développeurs puisqu’ils se focalisent sur une seule fonctionnalité. Chaque service est donc plus petit en termes de code et est plus rapidement testable. De plus, étant donné qu’ils peuvent être déployés indépendamment des autres, les équipes n’ont pas besoin de coordonner le déploiement des modifications spécifiques à leur service. Le déploiement continu devient donc possible là où il ne l’est pas avec une architecture monolithique.

Malgré des avantages indéniables, l’architecture micro-services n’est pas à la portée de toutes les entreprises. Elle implique de lourds changements au niveau de l’organisation des équipes d’un projet et une entreprise de petite taille, ou un projet avec une équipe déjà réduite, aura plus de mal à diviser ses équipes. Celles-ci devront apprendre de nouvelles pratiques de communications, voir même un changement de culture lorsque la méthode agile n’était pas du tout appliquée.

Etant encore une architecture récente et évoluant très rapidement, les développeurs devront faire une veille technologique plus régulière et plus importante que pour une architecture monolithique. Les micro-services restent complexes à mettre en œuvre et demande aux développeurs de nouvelles connaissances, de nouvelles bonnes pratiques, et est donc plus couteuse en temps et en conception.

Parler du changement dans l’interface d’un service qui est lourd à gérer

Enfin, tous les projets n’ont pas l’utilité d’être développée en micro-services. Par exemple, une application CRUD (Create, Read, Update, Delete, autrement dit, une application de gestion de données) spécifique à un seul type de données n’aura pas besoin de ce type d’architecture et pourra largement se contenter d’une architecture monolithique qui sera beaucoup plus simple et rapide à mettre en place.

Netflix, application dont la popularité n’est plus à prouver, a décidé en 2008 d’utiliser les Amazon Web Service, solution cloud d’Amazon, et a dû adapter son architecture pour mieux correspondre au cloud. Ce sont ces deux décisions qui ont façonné le succès de l’entreprise aujourd’hui. Celle-ci est même devenue un acteur majeur dans la définition des micro-services et des bonnes pratiques à mettre en place.

Amiltone a choisi d’utiliser les micro-services pour son application AmilApp afin de pouvoir potentiellement déployer la solution pour d’autres clients en mode édition logicielle Saas, Softwaire as a Service. En effet, chaque client n’a pas les mêmes besoins et peut n’avoir envie que d’une partie des fonctionnalités que propose AmilApp. Enfin, l’utilisation des micro-services permettra la création de services réutilisables sur d’autres projets internes tels que l’authentification.