

Mémoire d'activité

Thème du mémoire

Oliwer SKWERES

31/10/2023

Abstract: Ce document est mon mémoire d'activité et permet d'illustrer les travaux que j'ai pu effectuer chez IMGT GROUP au cours de mon alternance pendant l'année scolaire 2023-2024.

1 - Certificat d'authenticité

2 - Remerciements (1 page)

Avant tout de chose, je voudrais remercier ma famille qui m'a soutenu dans cette épreuve difficile qu'est la vie d'adulte. Vivre tout seul était très compliqué mais j'ai pu surmonter cette épreuve grâce à leur présence au quotidien via les réseaux.

Je tiens également à remercier l'entreprise IMGT-GROUP pour m'avoir accordé leur confiance et de m'avoir offert l'opportunité de travailler avec eux. Je remercie Monsieur Jocelyn Marchadier, mon tuteur qui m'a confié plusieurs projets et qui a partagé son savoir, ainsi que Rodolphe Bart, le business manager de l'entreprise qui était présent tous les jours dans les locaux pour répondre à mes interrogations sur la vie en entreprise ainsi que pouvoir échanger de tout et de rien.

Je souhaite remercier l'école IPSSI qui m'a mise en relation avec l'entreprise IMGT-GROUP, sans laquelle je n'aurais pas pu effectuer mon alternance, mais je voudrais aussi remercier les étudiants que j'ai pu rencontrer à l'IPSSI et avec lesquels j'ai pu travailler et échanger au cours de cette année.

Enfin je remercie toutes les personnes qui m'ont apporté de l'aide en ligne sur les différents projets, pour avoir répondu à mes questions et m'avoir aidé à résoudre les problèmes auxquels j'étais confronté.

Merci à vous tous.

3 - Résumé (1.5 page)

Rien de plus agréable pour un développeur que d'avoir des ressources mis à sa disposition et ne pas avoir à développer une solution de A à Z sur chaque nouveaux projets. Plus de 3.1 Millions de packages npm sont disponibles sur Internet gratuitement pour les développeurs du monde entier. Les packages permettent un gain de temps considérable sur le développement car ils incluent directement des directement réutilisables. "React", "Express", "Axios", ces packages sont la pour nous faciliter la vie en tant que développeur. Chez IMGT, on voudrait aussi simplifier le processus de création d'application, les rendant personnalisable et augmentant efficacement le temps de développement. On a donc décidé de transformer nos applications, front et back, en application modulaire, avec des packages et modules spécifiques afin de mieux gérer les ressources de nos applications, mais aussi de pouvoir rendre la création de futures applications plus simple et rendre l'intégration de l'API plus simple sur d'autres plateformes. L'objectif pour moi et tout d'abord de m'approprier l'application afin de mieux comprendre le fonctionnement et de transformer les différents modules existants vers des packages

4 - Abstract (1.5 page)

5 - Sommaire

6 - Dictionnaire

7 - Introduction problématique (2 pages)

En 2023, une étude menée sur internet par Stackoverflow a démontré que la technologie la plus utilisée est le langage de programmation Javascript, utilisé par 64% des développeurs dans le monde. De nombreux sites web intègrent des scripts Javascript afin de les rendre dynamiques en permettant par exemple de gérer tout événement utilisateur (toute interaction provenant de l'utilisateur) ainsi que la communication asynchrone avec le serveur (upload de contenus additionnels, mise à jour des données tout en garantissant la continuité, maintenance simplifiée, gestion dynamique de l'affichage) . Les standards des utilisateurs ont tellement évolué au fil des années qu'il devient désormais impensable de développer un site statique et de le mettre en ligne en 2024.

En plus de Javascript, les développeurs se tournent vers des frameworks. Un framework est une bibliothèque mettant à disposition aux développeurs de multiples fonctions, des modules ou des composants totalement réutilisables permettant d'accélérer et faciliter le développement d'une application ou d'un site web. Les fonctions pré-écrites peuvent être réutilisées par les développeurs. Un framework permet aussi de mieux structurer l'application grâce à un design pattern qui est souvent expliqué dans la documentation du framework ou alors mis en place lors de l'installation du framework.

Parmi les frameworks web les plus utilisés, on retrouve le trio: React (40,58%), Angular (17,46%) et VueJS (16,38%). Ces 3 frameworks sont des frameworks "Front-end", c'est à dire qu'ils vont gérer l'affichage du site, et permettre de communiquer avec notre serveur via une API. Il existe aussi des frameworks pour les API, comme Express (19,28%), FastAPI (7,42%) ou NestJS (5,13%). Certains frameworks sont plus populaires que d'autres, cette popularité est notamment attribuée à plusieurs facteurs : - de meilleures performances et une meilleure adaptabilité pour des applications web données, - des outils proposés, - la renommée du propriétaire du framework, gage de qualité - la contribution active de la communauté pour les mises à jours, corrections de bugs, maintenance.

Au sein d'IMGT GROUP nous cherchons une solution pour créer une application modulaire. Ce type d'application permet de développer une application centrale à laquelle nous allons pouvoir ajouter différents modules développés en parallèle et qui ne sont pas obligatoires pour faire fonctionner cette application. On obtient alors un grand potentiel de personnalisation de l'application en fonction des besoins du client, mais aussi un grand gain de performance en allégant la charge du serveur.

Cette méthode de développement soulève des problématiques à différents niveaux : - choix du framework, que ce soit pour le framework back-end et front-end, - l'hébergement des modules qui devra être pris en compte, pour permettre notamment le versionnage des modules et faciliter le déploiement via des méthodes CI/CD.

IMGT Group a choisi deux frameworks. Pour le front, nous utiliserons Angular, framework créé et

maintenu par Google, et pour le back, nous utiliserons NestJS, framework se rapprochant de la structure d'Angular. Nous reviendrons plus tard sur les raisons de ces choix.

En s'appuyant sur le contexte, on peut être amené à se demander comment développer une application modulaire front-to-back avec Angular et NestJS?

Dans un premier temps nous nous intéresserons au projet choisi en détaillant les choix effectués, la conception des applications ainsi que leur fonctions. Nous nous attarderons ensuite sur la société IMGT Group en présentant ses domaines d'activités, certains projets sur lesquels nous serons amenés à travailler, ainsi que ses clients historiques qui lui ont accordé sa confiance. Pour conclure, nous reviendrons sur les différentes étapes et les points clés de ce projet.

8 - Partie 1 - Le sujet

8.1 - Le sujet

Tout d'abord, nous allons présenter plus longuement le projet concerné par notre problématique: Virtual Buro. Virtual Buro est une application web écrite avec Angular, (récemment mise à jour vers React) qui permet de gérer son entreprise virtuellement, avec une interface web. En plus de la gestion de l'entreprise, elle permet aux employés de gérer eux-mêmes leurs tâches, leur suivi du temps de travail ou encore les différents projets auxquels ils participent.

L'application Virtual Buro est construite sur une base modulaire, c'est à dire que chaque fonctionnalité de l'application (comme la gestion des utilisateurs, la gestion des groupes, la gestion des tâches, ...) est développée comme un projet distinct qui viendra compléter l'application, c'est ce que l'on appelle les modules. Ces modules ne seront pas intégrés de base dans l'application, ils pourront être rajoutés ultérieurement.

Cette approche modulaire permet de diviser l'application en plusieurs unités fonctionnelles et indépendantes (les modules). Ces modules peuvent être utilisés seuls ou combinés afin de former une application avec plus ou moins de fonctionnalités. Comme les modules communiquent entre eux, ils peuvent interagir. Cette structure autorise donc non seulement une réduction de la charge du serveur, une possibilité de personnalisation complète par l'utilisateur mais aussi une réutilisation du code par les développeurs avec notamment l'utilisation d'un module "Template" et une meilleure organisation du code. En effet l'ensemble des fonctionnalités d'un module est contenu dans un seul et unique dossier avec une arborescence dont la structure permet une navigation aisée dans le code mais évite également les répétitions inutiles du code.

8.2 - Les technologies

Pour réaliser cette application, IMGT Group a opté pour l'utilisation d'Angular et NestJS.

8.2.1 - Angular

Angular (ou Angular 2+) est un framework utilisé avec Typescript, un langage similaire à Javascript qui offre la possibilité de typer les variables. Créé par Google, le but de ce framework est de concevoir plus facilement des applications web dynamiques grâce à sa structure modulaire. Il ne faut pas le confondre avec AngularJS (Angular 1) qui est la version antérieure d'Angular, et qui utilise Javascript, mais nous y viendrons plus tard. Actuellement, Angular en est à sa version

17, les mises à jours sont très régulières: au cours des 3 dernières années, 2 mises à jour majeures sortent chaque année afin de corriger les différents bugs et améliorer l'application.

Angular permet aussi l'utilisation de composants. Les composants sont des éléments réutilisables dans une application ce qui permet de rendre le développement plus efficace et le code plus lisible en ré-écrivant par exemple seulement une ligne. Un autre avantage de l'utilisation des composants est qu'ils peuvent être très personnalisables en fonction des paramètres pré-définis dans le composant. Grâce à cette méthode, on peut alors créer un composant "Button" totalement personnalisable et qui sera réutilisable dans l'application.

Angular est parfait pour le genre d'application qu'IMGT veut créer, le framework est conçu pour créer des applications complexes et de grande échelle.

a. Structure basique

Comme chaque framework, Angular possède ses propres particularités et sa propre structure.

La structure de base de l'application est la suivante (annexe 1). A cette structure peuvent venir s'ajouter différents dossiers: - pages - composants - routing - services - models - interceptors - ...

b. Fonctionnement d'une page / composant

Les 4 fichiers `app.component` sont les fichiers qui représentent une page / un composant.

Le fichier `.ts` contient toute notre logique pour gérer la page dynamiquement, comme par exemple les requêtes, les variables ou les fonctions. C'est le fichier le plus important car il permet de définir tous les paramètres que notre composant peut prendre, comme une couleur, un nom, des classes, ce qui le rend personnalisable via le hook `[Input?]()`. Il existe aussi un hook `[Output?]()` qui permet de faire remonter des informations depuis le composant enfant vers le composant parent, ce qui est très utile.

Le fichier `.html` est l'affichage de notre page / composant. C'est ici que nous allons mettre notre code HTML pour construire les pages. On pourra aussi y importer les différents composants créés ou implémenter un affichage conditionnel. Le fichier `.spec.ts` représente le fichier test de notre composant. Dans ce fichier, nous pouvons ajouter les tests unitaires à effectuer pour s'assurer que notre page/composant fonctionne bien. Pour finir, le fichier `.scss` permet de gérer le style de notre page.

c. Cycle de vie

Le cycle de vie d'un composant Angular est un ensemble d'étapes exécutées par le composant jusqu'à sa destruction. Angular fournit de base les hooks pour gérer le cycle de vie de l'application.

Constructor

Le constructor est la première étape du cycle de vie, c'est là que le composant est instancié. Le constructeur permet de déclarer les variables locales (appels des services par exemple) ou encore d'effectuer des calculs complexes et asynchrones.

Initialisation

Une fois le composant instancié, on peut utiliser le hook **ngOnInit()** afin d'effectuer des requêtes d'API ou de gérer la configuration du composant. Cette fonction est très importante car elle permet de faire des appels d'API ou d'instancier des variables nécessaires pour le fonctionnement du composant. ##### Rendu

Après avoir initialisé nos composants, nous pouvons appeler le hook **ngAfterViewInit()**. Comme son nom l'indique, il va exécuter les calculs nécessaires une fois que le composant et les enfants de ce composant ont été initialisés.

Destruction

Le dernier hook important est **ngOnDestroy()**. Dans cette fonction, on peut nettoyer les différents variables, observables et/ou les listeners pour vider la mémoire.

Il existe d'autres hooks mais qui sont pas autant utilisés que les hooks cités ci-dessus.

d. Routing

Le routing est très simple à mettre en place sur Angular. Dans l'arborescence du template "my module" se trouve le fichier `app.routes.ts`. Grâce à ce fichier, nous pouvons tout simplement importer notre composant et lui associer une route ou plusieurs routes.

e. Modules

Un autre fichier figurant dans l'arborescence du template "my module" est le fichier `app.module.ts`. Ce fichier permet de créer un container, qui va contenir la même structure que le dossier `apps`. C'est ici que nous allons importer nos différents composants, et exporter les composants propres à ce module.

f. Services

Le dernier fichier figurant dans l'arborescence du template "my module" est le fichier `app.service.ts`. Ce fichier contient essentiellement les requêtes d'API vers notre serveur en utilisant le module HTTP. ##### Models

Sachant qu'Angular utilise Typescript, on peut alors typer nos variables, ce qui est très pratique pour vérifier nos données lors du développement et s'assurer ainsi que le format des données est correct. On peut aussi accéder plus facilement aux propriétés de nos objets.

g. Commandes

Contrairement à d'autres frameworks populaires comme React ou Vue, Angular peut être utilisé via des commandes dans le terminal. Par exemple, pour créer une application, on peut utiliser la commande suivante:

```
ng new my-app
```

Cette commande permet de générer la structure basique. Tous les fichiers cités ci-dessus pourront également être créés grâce à d'autres commandes, c'est à dire les composants (annexe 2), les modules (annexe 3), les services (annexe 4)

h. Avantages d'Angular

Angular présente plusieurs avantages comparé à un site statique.

Les composants

Comme stipulé plus haut, Angular facilite le développement d'une application. L'utilisation de ces composants est également très pratique lors de projets de groupe Avec plusieurs personnes travaillant sur le même projet, le code peut vite se différencier, et les éléments d'un site comme les boutons par exemple peuvent vite varier d'un utilisateur à un autre (taille, couleur, police d'écriture, ...) : les composants permettent d'éviter ce problème. Grâce à la mise en place des composants dès le début du projet, la charge de travail lors du code review sera allégée puisqu'il y aura moins d'éléments à revoir. C'est également le cas pour d'autres éléments comme les cards, les modals ou les champs d'entrées qui peuvent être réutilisés en guise de composant.

Le DOM "Incrémental"

Google a développé la technologie du DOM Incrémental pour optimiser l'utilisation de la mémoire par rapport au modèle traditionnel du DOM Virtuel. Cette approche consiste à mettre à jour le DOM directement, ce qui conduit à une amélioration des performances grâce à une diminution des opérations de ramassage des ordures (garbage collections), surtout dans le contexte des applications mobiles.

Cependant, il faut noter que le DOM Incrémental présente également un inconvénient: sa vitesse de traitement peut être inférieure à celle du DOM Virtuel. La raison principale de cette différence est que le processus de calcul des modifications nécessaires pour synchroniser le DOM avec le document HTML requiert plus de temps.

i. Une structure stricte

La structure d'Angular est assez stricte. Chaque composant est initialisé de la même manière, possède les mêmes fichiers, et globalement les applications basiques se ressemblent. C'est tout à fait différent des autres frameworks plus permissifs comme React, le développeur doit élaborer sa propre structure de fichiers. Cette structure stricte et connue d'Angular, autorise l'intégration rapide et aisée d'un nouveau développeur sur le projet.

j. SWOT

Forces

- Framework populaire: Angular est un framework très populaire comme on a pu le voir avec une communauté très grande et active.
- Composants: l'architecture proposée par Angular permet de développer plus efficacement avec des composants réutilisables.
- Utilisation de Typescript: Typescript permet de mieux contrôler et de typer nos variables pour rendre les vérifications plus simples.
- Framework complet: Angular est un framework très complet offrant toute l'expérience nécessaire pour le développement d'une application web.

Faiblesses

- Complexité: le framework est très compliqué à appréhender dans sa totalité, si on souhaite l'exploiter de manière efficace.
- Taille: la taille d'un projet Angular augmente énormément avec l'avancement du projet.
- Mise à jour: Angular bénéficie de 2 mises à jour majeures par an, ce qui peut rendre le code difficile à maintenir.

Opportunités

- Support de Google: le support de Google permet de donner de la crédibilité au framework et permet d'assurer une qualité au framework.
- Demande croissante: au fil des années, la demande et la popularité du framework a augmenté (13,76% en 2023, contre 7,18% en 2022, et seulement 5,8% en 2021).
- Développement modulaire: Angular permet de travailler efficacement avec la mise en place possible de plusieurs modules.

Menaces

- Forte concurrence: il existe différents frameworks qui peuvent concurrencer Angular, comme React, Vue ou encore Svelte. De plus, de nouveaux frameworks sont développés tous les ans.
- Dépendance de Google: en cas d'abandon d'Angular par Google les mises à jour et la popularité pourraient être impactées.

k. Comparaison avec d'autres frameworks Front

(annexe 5)

On peut voir sur ce tableau que globalement que React revient souvent en avant sur ce tableau. On distingue que React ressort en avant surtout grâce à sa vitesse de compilation et à sa popularité. La popularité de React est due à différents facteurs, comme la courbe d'apprentissage qui est plus légère comparée à Angular qui nécessite plus de temps pour le maîtriser correctement. La popularité est aussi due à la date de sortie plus ancienne de React par Facebook, qui a su attirer les développeurs dès le début, ce qui est visualisable sur Github, avec la différence du nombre d'étoiles attribuées par les utilisateurs aux deux projets.

8.2.2 - NestJS

NestJS est un framework OpenSource basé sur Typescript sous licence MIT créé par Kamil Mysliwiec, développeur polonais, en Mai 2017, et qui permet de mettre en place très facilement une API REST sous NodeJS. Le framework a été conçu dans le but de proposer une structure modulaire et le typage statique avec Typescript. NestJS est basé sur Express mais il peut être utilisé sur d'autres frameworks comme Fastify. Le framework devient de plus en plus populaire au fil des années et est soutenu par une communauté active qui contribue à sa croissance et une documentation détaillée pour les personnes voulant commencer à l'utiliser.

a. Structure basique

La structure de base de l'API NestJS ressemble énormément à celle d'Angular. Contrairement à d'autres frameworks, NestJS propose directement les fichiers de base lors de l'installation. (annexe 6)

b. Fonctionnement de l'API

Les 5 fichiers du dossier `app` cités ci-dessus permettent de lancer l'API et définir son fonctionnement.

Le fichier `app.controller` permet de définir les routes d'API et effectuer des vérifications. (annexe 7) Lors de la création d'une route, on commence par préciser la méthode REST appropriée (comme POST, GET, UPDATE, DELETE, ou PUT). Ces méthodes définissent l'action qui va être effectuée par la suite. La méthode **POST** permet d'envoyer vers l'API les données que l'on souhaite insérer dans notre base de données. Les données envoyées sont transmises dans le corps de la requête (body) garantissant une + sur la sécurité. La méthode **GET** permet de récupérer des données depuis le serveur, incluant ou non des paramètres passés dans l'URL de la requête, comme un id, le nombre d'éléments à récupérer etc... La méthode **UPDATE** permet de mettre à jour les données dans la base de données, en passant en paramètres dans le body, les informations à modifier et accessoirement, l'id dans l'URL. La méthode **DELETE** permet la suppression des informations dans la base de données en passant un id dans l'URL. Pour finir, la méthode **PUT** permet de remplacer entièrement un objet dans la base de données. Cette méthode est souvent confondue avec l'UPDATE, pourtant les deux se différencient bien.

De plus, chaque nouvelle route déclarée peut avoir des codes de réponse, des codes d'erreur, des redirections, et bien d'autres fonctionnalités avancées, grâce à l'utilisation de decorators (fonctions permettant d'ajouter des actions supplémentaires à effectuer lors du contact de la route). Cela signifie que les développeurs ont la liberté de concevoir leurs propres decorators pour réaliser des validations ou des actions spécifiques à leur application, comme le test des permissions ou de rôles.

Pour finir, on peut définir notre fonction. Dans la fonction, on peut préciser les paramètres pris par notre route. Cette fonction va renvoyer vers le fichier `app.service.ts`, qu'il faudra importer. On ne communique pas directement avec notre base de données depuis notre controller afin de garder une structure propre et ne pas tout mélanger ensemble. (annexe 8) Voici un exemple de route pour récupérer un utilisateur via son ID. `:id` signifie que nous allons passer une variable dans l'URL (`'/1'`, `'/2'`) ce qui permet de rendre les routes dynamiques. Le decorator `HttpCode` permet de déterminer le status de la requête lors de la réponse. Le decorator `IsAuth` permet de savoir si l'utilisateur est connecté ou non. Ceci est un exemple basique de la déclaration de route dans NestJS. On peut y ajouter des codes d'erreurs, des messages d'erreurs et d'autres vérifications si nécessaire. ##### TypeORM

Dans notre cas, NestJS est couplé à TypeORM, une bibliothèque Object Relational Mapping (ORM) écrit en Typescript, permettant la manipulation des données afin de les stocker dans la base de données. TypeORM inclut la création de table automatiquement dans la base de données, une facilité de gestion des relations et des moyens simples de mettre à jour les objets en base de données. Dans notre cas, 3 fichiers supplémentaires sont à ajouter dans notre architecture. (annexe 9)

c. Répositories

Avec TypeORM, les repositories sont des fichiers permettant d'interagir avec les tables de la base de données. Les repositories fournissent des méthodes prédéfinies comme `.findOne`, `findOneBy`, `create` etc. qui facilitent les interactions avec la base de données. De plus, il est tout à fait possible de créer ses propres méthodes pour répondre à nos besoins.

d. Managers

Les managers sont des fichiers qui permettent de gérer les requêtes nécessitant des vérifications sur les valeurs ou nécessitant des calculs. Dans ces fichiers nous effectuerons des tests supplémentaires pour vérifier si les données existent par exemple.

e. Entity

Le fichier entity représente notre table dans la base de données tout simplement. C'est dans ce fichier qu'on va définir les colonnes de notre table, en précisant le type, la longueur etc... ou encore faire les relations vers d'autres entités. Nous allons aussi définir des méthodes accesseurs (get) et mutateurs (set) permettant la modification des données d'un objet une fois récupéré depuis la base de données.

Le fichier `app.service.ts` permet alors d'effectuer des requêtes simples avec des vérifications simples, et de rediriger les requêtes plus compliquées vers le manager. Que ce soit le manager ou le service, les deux fichiers peuvent importer le repository pour pouvoir communiquer avec notre base de données.

f. Swot

Forces

- Utilisation de Typescript: Typescript permet de mieux contrôler et de typer nos variables pour rendre les vérifications plus simples grâce aux typages.
- Architecture modulaire: L'architecture modulaire permet de mieux organiser son code dans des modules réutilisables.
- Injection de dépendances: NestJS permet l'injection de dépendances dans les modules, améliorant l'aspect modulaire du code.

Faiblesses

- Courbe d'apprentissage élevée: NestJS n'est pas un framework maîtrisable du jour au lendemain, rendant l'accès à cette technologie un peu plus limitée.
- Dépendances circulaires: Lors du développement avec NestJS, on peut très vite être amené à faire des dépendances circulaires, surtout avec le développement modulaire.

Opportunités

- Popularité montante: au fil des années la popularité de NestJS augmente, plus de développeurs l'utilisent. Cela permet une assistance plus efficace.

- Open Source: sachant que NestJS est OpenSource, les utilisateurs peuvent contribuer au projet et faire évoluer l'application en apportant de l'aide pour corriger les failles et problèmes existants.

Menaces

- Risque de sécurité: comme tout framework, NestJS est exposé à des risques de sécurité sur les injections, l'accès à la base de données, ce qui nécessite une vigilance accrue sur l'aspect sécuritaire.
- Compétition sur le marché: NestJS est en compétition avec d'autres frameworks très utilisés comme Express ou Fastify, et il peut-être menacé par une nouvelle technologie.

g. Comparaison avec d'autres frameworks Backend

(annexe 10)

Sur ce tableau, on peut voir qu'il n'existe pas beaucoup de frameworks/bibliothèque permettant de réaliser des API NodeJS. Express reste de loin le framework le plus utilisé pour réaliser des API, grâce à sa popularité, sa date de sortie et sa simplicité.

8.3 - Que est-ce que l'intégration continue ?

L'intégration continue (CI: Continuous Integration) est une pratique DevOPS qui vise à maintenir l'application en cours de développement à jour en y ajoutant les fonctionnalités développées par les autres utilisateurs. Cette méthode permet de garder une évolution de l'application et de garder le déploiement continu (CD: Continuous Deployment). Afin d'appliquer l'intégration continue, il faut vérifier que: - Le code source soit accessible à tous (Un repository Gitlab par exemple) - Les personnes qui travaillent sur le projet envoient les modifications effectuées - Test automatique de l'application via des tests unitaires

L'intégration continue permet alors de gagner en productivité et en efficacité lors des développements. Les tests unitaires s'exécutent automatiquement à chaque fois, permettant de garder une application fonctionnelle tout le temps. Si des tests échouent, l'utilisateur est notifié du problème. En plus des avantages techniques, l'intégration continue permet d'augmenter les communications entre les membres de l'équipe. En cas de conflit lors des merges, les membres de l'équipe peuvent travailler ensemble afin de résoudre les problèmes. L'intégration continue et le déploiement continu permettent aussi la distribution de packages npm automatiquement sur Gitlab. Cette fonctionnalité sera utile pour publier les modules sur Gitlab et rendre l'application facilement modulaire.

9 - Partie 2 - Cas réel

9.1 - Qui est IMGT-GROUP ?

9.1.1 - Une jeune société à l'esprit start-up

L'entreprise dans laquelle j'effectue mon alternance se nomme IMGT Group. Cette jeune société à l'esprit start-up exerce dans le conseil en stratégie et en management de l'innovation, intervenant

sur des projets innovants à forte valeur ajoutée. Cette société par actions simplifiées à associé unique fût fondée le 4 Septembre 2012 par Jocelyn Marchadier, ancien étudiant et docteur de l'ESIEE PARIS. Les bureaux se situent dans le 16ème arrondissement de Paris et au cœur de la Cité Descartes à Champs sur Marne, son siège social est localisé à Paris 16.

9.1.2 - 5 à 10 employés

Au lancement, la société ne comptait qu'une seule personne, aujourd'hui 5 personnes travaillent pour IMGT Group. De plus, l'entreprise accueille régulièrement des stagiaires, non comptabilisés dans l'effectif.

9.1.3 - Présentation de l'organigramme de l'entreprise

Jocelyn MARCHADIER préside IMGT Group, il s'occupe de parrainer tous les projets au sein de l'entreprise mais exerce également les fonctions de commercial, consultant et participe activement à la R&D de la société. Le Business Manager s'occupe de la gestion courante de l'entreprise et de ses moyens. Deux chargés d'études au sein de l'entreprise s'occupent de certains projets IMGT Group et assurent également les fonctions de consultants. Pendant mon alternance, j'occupe le poste de développeur web au sein de l'entreprise. Mon but est de mettre en place des solutions d'application web en interne pour l'entreprise mais aussi pour les clients qui contactent l'entreprise.

9.1.4 - Les différents domaines d'activités

Le cœur d'activité de la société porte sur la finance quantitative, plus précisément le conseil en stratégie, finance et le management. La société travaille également dans le réseau, le développement en modélisation, la robotique et la vision par ordinateur. Le dernier domaine dans lequel IMGT Group exerce est Internet : en effet l'entreprise développe des sites, des plateformes web et mobiles pour la gestion du e-commerce.

9.1.5 - Les différents produits phares

IMGT propose différents produits :

- L'architecture Cloud et la sélection de solutions. IMGT analyse les besoins de systèmes informatiques (SI) et met en place un cahier des charges, une définition de l'architecture à utiliser ainsi que les choix de solutions logicielles et matérielles.
- Virtual Buro permet aux entreprises de toutes tailles de disposer d'un cloud d'entreprise, d'espaces de coworking, d'une domiciliation, d'une assistance administrative et de matériels informatiques télé-gérés. C'est notamment l'un des projet sur lequel j'ai le plus travaillé durant cette alternance.
- IMGT procède à l'édition des sites web, et de plateformes de e-commerce ainsi que des applications SAS.

9.1.6 - Les clients

IMGT Group a plusieurs clients, le principal étant le groupe Crédit Agricole. De plus la société jouit d'un partenariat avec le cabinet de conseil A CHORD PARTNERS, dont le cœur de métier est

le conseil dans les domaines de la finance et l'assurance. IMGT Group profite ainsi de leur savoir-faire tout comme A CHORD PARTNERS profite des recherches et logiciels développés par IMGT Group. A travers ce partenariat, IMGT Group ajuste et spécifie ses outils en fonction des besoins de la clientèle de A CHORD PARTNERS, s'assurant ainsi que ses outils/logiciels soient toujours en adéquation avec l'évolution des marchés.

9.2 - Ma place au sein de IMGT-GROUP ?

9.2.1 - Mon poste

Au sein de l'entreprise IMGT-GROUP, je travaille en tant que développeur web junior. Avec Monsieur Jocelyn MARCHADIER en tant que tuteur, je suis accompagné dans la réalisation de plusieurs projets importants au sein de l'entreprise. Après avoir finalisé le développement de la solution, j'organise une réunion avec M. Marchadier pour discuter de sa validation et potentiellement apporter des améliorations. L'objectif est de s'assurer que la solution répond effectivement aux besoins exprimés.

9.2.2 - Mes projets

Au cours de cette année en alternance, j'ai eu l'opportunité de participer à plusieurs projets importants; ces expériences m'ont permis non seulement d'approfondir mes connaissances et compétences, tant sur le plan théorique que pratique mais aussi d'acquérir de nouvelles connaissances, compétences et savoir-faire. Néanmoins, malgré la complexité de certains de ces projets, ma détermination et mon intérêt n'ont jamais faibli.

a. Virtual Buro

Virtual Buro est une application web permettant la gestion virtuelle de l'entreprise. Elle permet une gestion administrative des différents éléments d'une entreprise, comme le traitement des documents, la gestion des utilisateurs ou le suivi de tâches. L'application est basée sur Angular, qui est un framework TypeScript développé par Google permettant la décomposition des éléments graphiques du site en composants réutilisables afin de rendre le développement plus agréable et efficace. La décision d'IMGT GROUP de se tourner vers Angular repose sur la cohérence de son architecture à travers tous les projets ainsi que sur sa capacité à gérer efficacement des applications complexes et volumineuses.

L'application utilise aussi NestJS, un autre framework Javascript. NestJS est une API permettant de faire le lien entre notre application et notre serveur. NestJS se rapproche de la structure d'Angular en gardant la logique de composant. Le but final de l'application est de proposer des modules optionnels qui pourront être intégrés grâce à des packages NPM. ##### Déploiement de paquets NPM via Gitlab

Ce projet consiste à maintenir un gestionnaire de paquets privés NPM afin de devenir indépendants des paquets npm publics. Pour IMGT-GROUP, il y a plusieurs avantages à maintenir un gestionnaire de paquets privés plutôt que d'aller chercher les paquets en ligne. Le but est d'avoir la main sur les paquets que l'on installe et pouvoir les mettre à jour en fonction des besoins de l'entreprise.

Pour répondre à ce besoin, IMGT-GROUP a opté pour l'utilisation de Gitlab. Gitlab est un outil

qui permet de gérer les projets au sein d'une entreprise et de déployer des applications depuis ce domaine. Gitlab permet aussi de gérer un gestionnaire de paquets privés et permet la distribution des paquets. Gitlab permet aussi la gestion du développement CI/CD qui est directement intégré dans l'application.

IMGT-GROUP se concentre sur la sécurisation des packages utilisés pour éviter tout risque de piratage lié à une vulnérabilité présente dans ces paquets. L'entreprise s'efforce de garantir la compatibilité de l'application parmi les utilisateurs, afin que chacun dispose de la même version et des mêmes packages sur ses projets.

Ma mission est donc de mettre en place ce système de téléchargement de paquets en utilisant Gitlab qui est déjà installé sur le réseau. L'utilisation d'une solution maison permet d'avoir la main sur ce que l'on fait et que l'on utilise. Des solutions comme Verdaccios existent, cependant l'analyse du code et la vérification de l'intégrité de l'application prendrait beaucoup de temps.

b. Prestashop

Pour finir, le dernier projet sur lequel je travaille est une boutique Prestashop. Ce projet consiste à mettre à jour une boutique Prestashop déjà existante de la version 1.7 vers la version 8. Cette mise à jour permet un départ sur de bonnes bases. En plus de cette mise à jour, je dois transférer l'entièreté des produits de la boutique vers la nouvelle boutique, ce qui implique : - la migration vers une nouvelle version de la base de données, en préparant un script de migration, - la mise à jour des différents modules déjà présents pour la version 1.7, - la possibilité de l'insertion en base de données des nouveaux produits via l'API de Prestashop afin de faciliter l'insertion des fichiers CSV.

La mise à jour du site web de cette boutique est cruciale car le site précédent jouissait d'une grande popularité auprès de ses clients. IMGT-Group vise à reconquérir son public en offrant du mobilier de haute qualité dans la nouvelle boutique.

9.2.3 - Mes tâches

10 - Partie 3 - Solution : développement et application

10.1 - La solution modulaire

10.1.1 - Qu'est-ce que le développement modulaire ?

Dans le domaine de la programmation, le développement modulaire est une approche où la résolution d'un problème est divisée en segments plus petits, appelés modules. Chacun de ces modules s'occupe d'une tâche précise, comme la gestion des utilisateurs ou la gestion du temps, et peut opérer indépendamment des autres composants. Ce type de développement présente plusieurs bénéfices, notamment : - la possibilité de réutiliser le code, - faciliter sa maintenance, - permettre l'injection de dépendances et - optimiser les performances globales.

a. Réutilisation

Les différents modules qui seront développés par IMGT Group pourront être réutilisables dans plusieurs applications différentes, que ce soient les modules front ou back. On verra plus tard

comment implémenter ces différents modules dans différentes applications. La réutilisation des modules fronts sous Angular permet de réutiliser les pages adaptées à notre API, ce qui économise un temps de développement considérable.

La réutilisation des modules back permet de créer des API simplement avec NestJS, car les tables, les routes et les vérifications sont mises en place directement avec le module. La combinaison des modules front/back permet la mise en place d'une solution rapidement car la liaison entre les 2 modules s'effectue automatiquement, les modules sont complémentaires.

b. Maintenance

Les applications modulaires sont plus faciles à maintenir et mettre à jour. Chaque module est modifié et mis à jour séparément des autres, permettant d'assurer la fonctionnalité du reste de l'application.

c. Injection des dépendances

L'injection des dépendances est un concept important dans le développement modulaire. L'injection de dépendances est un design pattern qui vise à fournir les dépendances nécessaires à un objet plutôt que de les créer à l'intérieur de ce dernier. Les dépendances sont alors "injectées", via un constructeur.

L'injection des dépendances permet de : - Réduire la complexité du code, - Améliorer la modularité et - Faciliter les tests de l'application.

d. Performances

La modularité permet d'améliorer les performances des applications, particulièrement des applications web et applications mobile. La modularité permet de réduire les lignes de codes et les duplications de celui-ci, en écrivant une fois seulement le code nécessaire, réduisant ainsi par extension la taille des fichiers.

e. Hébergement

Dans le cadre de VirtualBuro, on a séparé les modules dans différents projets Gitlab afin de permettre de créer des packages installable via commande NPM, que ce soit pour le front ou le back. Le but est ensuite de pouvoir créer un réseau avec des conteneurs Docker avec Kubernetes afin d'améliorer les performances du backend.

10.1.2 - Structure d'un module

Un module doit être structuré correctement afin qu'il respecte les principes cités au dessus. Lors du développement des différents modules, nous avons pensé à des structures "simples", que ce soit pour le front et back, afin de rendre la création de module simple et compréhensible et assez flexible pour assurer la réutilisabilité, la maintenance et les performances.

Les structures choisies permettent de définir les dépendances nécessaires pour le bon fonctionnement des modules avec les providers et imports du décorateur `[NgModule?]` / `[Module?]`. Elle permet aussi l'injection des dépendances pour injecter les services dont le module a besoin.

Pour mieux organiser le module, on a décidé d'avoir un schéma de nommage très simple, permettant de respecter la cohérence du code entre deux modules, on sait où se trouve quel fichier. On utilise aussi des noms explicites, en ajoutant les mots clés “components”, “services” ou autres devant le nom du fichier pour définir sa fonction et améliorer la lisibilité.

Dans les deux cas, nos modules Frontend et Backend ont une structure un petit peu différente, mais restent tous les deux modulables.

10.1.3 - Structure avancée: FrontEnd

Grâce aux différents fichiers que nous avons examinés et les enjeux de l'application, la structure de base peut être améliorée pour optimiser l'application. Afin de répondre à notre problématique, nous allons créer un dossier `pages` dans le répertoire `src/apps`, dans lequel nous positionnerons un dossier `my_module-management` qui va gérer nos pages fronts. On peut ensuite créer un dossier `my_modules`, qui va contenir l'ensemble des éléments réutilisables par tous les modules mais aussi nos différents modules avec leurs services, composants et modèles.

Le résultat côté Angular serait quelque approchant : (annexe 11)

Voici l'architecture de notre application. Elle se divise en 2 dossiers, `apps` et `back_modules`. Dans le dossier `apps`, on retrouve le dossier `pages`, contenant les pages de nos modules (liste, détails, ect.), ou encore les fichiers qui permettent de charger l'application et les routes.

Dans le dossier `back_modules`, on va retrouver nos modules front, contenant des fichiers spécifiques à ce module (modèles, éléments réutilisables, etc...), les fichiers services nous permettant de communiquer avec la base de données et d'autres fichiers si nécessaire pour le bon fonctionnement du module.

Globalement, dans notre dossier `src`, notre module se sépare dans les dossiers `apps/pages` et `back_modules/api` afin de mieux organiser le projet. Regardons de plus près le fonctionnement des modules.

Cette structure respecte la logique modulaire, en prenant cette partie, l'intégrant dans une autre application en installant les dépendances nécessaires, on va pouvoir réutiliser toutes les pages. En reprenant les particularités de ce qu'est un module, remarque bien que l'on peut réutiliser le code, si on copie le code vers une autre application et en installant les dépendances nécessaires, on peut faire fonctionner ces éléments. La maintenance est simplifiée car on peut totalement développer les nouvelles fonctionnalités de ce module sans directement affecter l'application. Pour finir, les composants créés sont réutilisables partout dans l'application. Le module peut être enlevé facilement si d'autres modules ne dépendent pas de celui-ci. On peut alors supprimer les modules si les utilisateurs en ont pas besoins ou qu'ils ne sont pas inclus dans l'offre.

On a donc réussi à créer notre un module front. On verra juste après comment transformer ce module en module NPM, mais avant, ça on va revenir sur le fonctionnement de notre module.

a. Fonctionnement

Commençons par le dossier `pages`.

pages.module.ts / pages-routing.module.ts

Le fichier `pages.module.ts` est une sorte de rond point où vont se joindre tous nos modules. Dans ce fichier, nous allons importer tous les fichiers `*.module.ts`, permettant de gérer le routing vers les différents modules.

Dans le fichier `pages-routing.module.ts`, nous allons aussi importer les modules afin de pouvoir créer le routing vers nos pages.

Cette manière de fonctionner nous permet de facilement désactiver les modules afin de donner une application vierge au client, et qu'il choisisse lui-même les modules à inclure dans son application.

Le module

Intéressons nous maintenant au dossier `my_module-management`. Pour chaque module, nous avons donc un fichier `my_module-management.module.ts` et `my_module-routing.ts` ainsi qu'un dossier contenant `my_module-management` contenant l'ensemble des pages du module.

Le fichier `my_module-management.module.ts` est l'un des fichiers les plus importants d'un module. Ce fichier agit comme un conteneur pour regrouper les composants, pages créés pour ce module et importer les modules tiers fournis par des bibliothèques externes ou ceux développés par IMGT Group.

Le fichier `my_module-routing.module.ts` permet de gérer le routing du module afin de rediriger l'utilisateur vers les bonnes pages.

Pour finir, il y a toutes les pages à l'intérieur du dossier. Les pages respectent la structure d'un composant Angular comme on a pu le voir avant. Dans nos pages, nous pouvons effectuer des appels vers notre API en faisant appels à nos services qui se trouvent dans notre module dans le dossier `back`.

Maintenant que nous avons vu la structure d'une page, nous pouvons aller voir dans le dossier `api`. Dans le dossier `api`, pour chaque module nous retrouvons les services, modèles ou encore formulaire.

10.1.4 - Structure avancée: BackEnd

(annexe 12)

a. Authentification

Le dossier `authentification` contient une mini-API d'authentification combinée à un serveur Redis.

Le dossier contient une API avec la structure basique, avec le minimum nécessaire pour fonctionner.

Pour authentifier les utilisateurs automatiquement, lors de la connexion, l'utilisateur reçoit un token JWT (Json Web Token) lui permettant de s'identifier au serveur sur une certaine durée.

Redis

Un serveur redis fonctionne comme une base de données clé-valeur en mémoire, offrant des performances exceptionnelles grâce à son stockage en mémoire RAM. Le stockage en mémoire volatile permet de rapidement supprimer des sessions actives des utilisateurs une fois déconnecté.

b. Api

Comme pour le frontend, notre module backend se sépare en 2 parties. Un fichier se trouve dans le dossier `controllers` et `@web`. Le dossier `controllers` accueillera tous les controllers de l'API. Le dossier `[web?]` quant à lui va contenir l'ensemble des modules que l'on va installer dans l'application.

app.module.ts

Le fichier `app.module.ts` est l'équivalent du fichier `pages.module.ts`. Dans ce fichier, on importera nos modules afin qu'ils puissent être chargés dans notre API.

Dans ce fichier, on importera aussi les controllers que l'on va créer. Dans l'exemple ci-dessus, pour faire fonctionner notre module, on devra importer le fichier `my_module.controller.ts` et le fichier `my_module.module.ts`.

La structure mise en place respect

Fonctionnement

Comme vu plus tôt, le fichier `my_module.controller.ts` permet de créer les routes d'API.

10.1.5 - Préparation d'un module front au format packages

Pour créer un module Angular et en faire un package NodeJS, Angular possède une commande pour générer une base de library.

Pour transformer notre module en package NodeJS, on doit créer un nouveau projet Angular avec la commande `ng new <nom> --create-application=false`. Cette commande permet de créer un environnement de travail pour travailler sur nos modules. Une fois le projet créé, on doit exécuter la commande `ng g library <nom> -p <prefix>`. Elle crée une library dans laquelle on va pouvoir développer nos composants, services et les autres dépendances nécessaires pour notre module. Une fois la commande exécutée, on se retrouve avec cette structure (annexe 13). Les fichiers dans le répertoire `src/lib` sont les fichiers principaux de notre package. Cependant, on va pouvoir les supprimer pour pouvoir créer des modules.

Le fichier `public-api.ts` contient l'ensemble des ressources que l'on va exporter pour les rendre accessibles aux utilisateurs, comme les composants, modules ou services.

En se plaçant dans notre dossier `src/lib`, on va pouvoir créer un fichier module avec la commande `ng g m notre-module`. On va ensuite pouvoir générer les composants, services et autres fichiers et les importer dans notre fichier module et `public-api.ts`. Pour l'exemple, on peut créer un module `calendar` avec un composant et service. Une fois les commandes nécessaires exécutées, on a ce résultat (annexe 14).

Après avoir fait les déclarations, imports et exports nécessaires dans le fichier `calendar.module.ts`, on va exporter nos fichiers de cette manière dans le fichier `public-api`. Dans le fichier `public-api`, on va alors exporter nos fichiers de cette manière (annexe 15). Maintenant que l'on a créé notre module, on va pouvoir le build.

Dans le fichier `package.json` à la racine de l'environnement de travail, on va rajouter le script suivant (annexe 16)

En exécutant ce script, Angular va compiler le module et créer un dossier `/dist` avec notre module, on va pouvoir le publier. Cependant, avant de le publier, on pourrait le tester localement via la commande `npm pack`. Cette commande va générer un fichier `.tgz` qui est installable via la commande `npm i` et permet d'installer le package sur notre application. Une fois exécuté, on peut se rendre dans notre projet Angular et installer le package

```
npm i ../path/to/package
```

Une fois installé, on doit importer notre `CalendarModule` dans le fichier `app.module.ts`. A partir de là, on peut utiliser les composants et les services.

10.1.6 - Préparation d'un module back au format package

Maintenant qu'on a vu comment mettre en place le front, on peut créer notre package back avec NestJS.

Pour commencer, on doit les packages de NestJS et TypeORM via la commande NPM. Une fois installé, on peut commencer à créer notre arborescence du projet. On commence par créer un dossier `src` avec dedans un fichier `index.ts`. Ce fichier nous permettra d'exporter les fichiers, à l'image du fichier `public-api` pour Angular.

On peut reprendre la structure de notre module (annexe 17)

Grâce au fichier `calendar.module.ts`, on va pouvoir importer très facilement nos modules dans les différentes applications que nous allons créer. En déclarant nos services, entités et tous les autres fichiers du module, Nest va pouvoir les lire et faire le nécessaire pour faire fonctionner le module.

Une fois que nous avons fini de créer et configurer nos fichiers, nous pouvons tous les exports dans le fichier `index.ts`. (annexe 18)

Une fois que nous avons fini de développer le module, on va pouvoir le build et le publier sur le registry de Gitlab, avant tout ça on va devoir modifier notre fichier `package.json` et y rajouter le script pour build et les chemins d'accès au module. (annexe 19)

En lançant la commande `npm run build`, notre module est créé, et on peut le publier.

En lançant la commande `npm pack`, un fichier `.tgz` est créé, on peut donc essayer d'installer notre package via la commande `npm i`.

Publier un module sous forme de package

Une fois que nos modules sont prêts, nous devons les publier sur Gitlab. Par défaut, lorsque l'on publie les packages, ils sont publiés automatiquement sur le registry de NPM, le gestionnaire de packages de Gitlab. Notre but ici est de publier ces packages sur le Gitlab de l'entreprise, ce qui permettrait de garder les packages uniquement pour l'entreprise sans les publier en ligne.

Cette méthode rajoute une couche de sécurité pour plusieurs raisons. Si on publie les packages en ligne sur le registry de NPM, on rend le travail publique. Ces packages ne sont pas destinés à être rendus publics et doivent donc rester privés. De plus, si on publie ces packages en ligne, des personnes malveillantes peuvent découvrir des failles de sécurité dans certains modules. Les personnes pourraient ensuite exploiter ces failles pour effectuer des actions malveillantes envers l'entreprise.

Nous allons voir comment publier un package sur Gitlab.

a. Publication

Il existe différents moyens de publier notre package sur le serveur Gitlab de l'entreprise. Lors de mes recherches, j'ai opté pour deux de ces solutions. La première méthode permet de publier le package sur Gitlab via un fichier `.npmrc`. La deuxième méthode est celle du Pipeline CI/CD intégrée par Gitlab.

Avant la publication, il faut s'assurer que le fichier `package.json` contient les informations nécessaires, comme le nom ou la version. Il faut changer le paramètre "private" et le passer de l'état `true` à `false` afin qu'il puisse être publié.

Méthode `.npmrc`

La méthode du fichier `.npmrc` permet de publier manuellement notre code sous forme de package sur registry NPM ou Gitlab. Cette méthode est efficace car elle est simple et rapide car nous ne sommes pas obligés de passer par une configuration complète de Pipeline CI/CD. Le déploiement via ce fichier est aussi portable, facilitant sa distribution et son utilisation dans différents environnements.

Pour commencer, nous devons créer un fichier `.npmrc`. Ce fichier est un fichier de configuration utilisé par NPM pour gérer l'authentification de l'utilisateur, la gestion du proxy ou la publication de packages en ligne.

Pour commencer, nous devons créer ce fichier, car il n'est pas généré de base par les commandes `npm`. Ce fichier doit être créé dans le même dossier que le fichier `package.json`. Dans ce fichier, nous devons mettre la configuration nécessaire pour publier un package. Voici la configuration à inclure :

```
@scope/registry=https://{GITLAB_URL}/api/v4/projects/{ID_PROJECT}/packages/npm/  
://{GITLAB_URL}/api/v4/projects/{ID_PROJECT}/packages/npm/:_authToken="$  
{NPM_TOKEN}"
```

- `@scope` permet d'associer les packages à un registre npm particulier. Ce paramètre n'est pas obligatoire.
- `registry` permet de définir la source d'où sera téléchargé le package.
- `{GITLAB_URL}` est l'url de notre serveur Gitlab.
- `{ID_PROJECT}` est l'id du projet sur lequel on va publier le package.
- `$_{NPM_TOKEN}` est le token d'authentification permettant la connexion entre Gitlab et les commandes NPM. C'est un token personnel qui doit rester privé.

Dans notre cas, la configuration ressemble à ceci:

```
//gitlab.extra.imgt-group.com/api/v4/projects/79/packages/npm/:_authToken="$  
{NPM_TOKEN}"
```

On évite de mettre notre `$_{NPM_TOKEN}` en brut dans le fichier pour des raisons de sécurité. Ce

token doit être généré sur Gitlab. Dans les paramètres de notre compte, nous pouvons nous rendre dans l'onglet "Access Token". Une fois le token généré, on peut le garder de côté pour l'utiliser juste après. Il est aussi conseillé d'ajouter le fichier `.npmrc` dans le `.gitignore` dans ce cas pour des mesures de sécurité.

Pour la dernière chose, on doit changer le registry de npm. Il y a 2 manières de le faire, soit en précisant directement en paramètre le registry, ou en changeant totalement la configuration des commandes npm.

On peut changer la configuration via la commande suivante:

```
npm config set registry  
https://gitlab.extra.imgt-group.com/api/v4/packages/npm/  
npm config get registry # Pour vérifier si le changement a été effectué
```

Une fois ces paramétrages effectués, nous pouvons publier les packages via la commande `npm publish`. Il faut aussi préciser en paramètres le token que l'on a généré juste avant.

```
NPM_TOKEN=token_genere npm publish
```

Dans le cas où nous avons pas changé la configuration du registry, nous devons le préciser en paramètres

```
NPM_TOKEN=token_genere npm publish --registry https://gitlab.extra.imgt-group.com/api/v4/packages/npm/
```

Lorsque nous nous rendons dans le repository de notre projet, nous pouvons voir que le package a été publié.

Méthode Pipeline CI/CD

La deuxième méthode de publication est la méthode Pipeline CI/CD permet d'effectuer la même chose que la méthode d'avant, c'est à dire publier des packages node sur Gitlab, mais cette fois-ci de manière automatisée. Cette manière permet d'automatiser le déploiement avec des tests unitaires et test d'intégration avant la distribution du package. On peut exécuter la publication à chaque fois que nous mettons à jour le repository ou à des heures précises.

Pour commencer, comme pour la méthode d'avant, il faut créer un fichier `.npmrc` mais la configuration sera un peu modifiée.

```
@scope:registry=https://${CI_SERVER_HOST}/api/v4/projects/${CI_PROJECT_ID}/  
packages/npm/  
://${CI_SERVER_HOST}/api/v4/projects/${CI_PROJECT_ID}/packages/npm/:_authToken=${  
CI_JOB_TOKEN}
```

La configuration ici est différente, ici on utilisera directement des variables disponibles sur Gitlab directement. Dans ce cas là, on a besoin d'envoyer ce fichier vers Gitlab.

Une fois ceci fait, on doit se rendre sur notre projet Gitlab, dans l'onglet Settings > CI/CD > Variables. Ici, on doit définir les 3 variables `CI_PROJECT_ID`, `CI_SERVER_HOST` et `CI_JOB_TOKEN` (pour ce dernier, le champs doit rester vide, Gitlab va le générer tout seul).

Ensuite, nous devons créer un fichier `gitlab-ci.yml` qui va être le fichier d'instruction que doit exécuter Gitlab lorsque il va publier le fichier. La configuration minimale est celle-ci. (annexe 20)

On pourra modifier ce fichier par la suite pour ajouter des tests unitaires par exemple.

Pour finir, on va devoir mettre en place un runner. C'est un application qui va pouvoir exécuter des

tâches Gitlab CI/CD automatiquement. C'est un élément essentiel de Gitlab pour l'automatisation des tâches.

La création d'un runner nécessite les accès administrateur au serveur, je n'ai donc pas pu le créer mais mon tuteur s'en est occupé à ma place.

Une fois ceci fait, nous pouvons envoyer les fichiers que nous avons créés vers notre repository. Gitlab va détecter tout seul le fichier et va automatiquement lancer les tests et la publication du package.

Le package sera disponible sous peu si toutes les étapes du build sont validées.

b. Installation

Une fois que notre package est disponible via l'une des méthodes précédentes, nous pouvons l'installer via la commande npm.

Si nous n'avons pas défini le registry dans la config, nous devons le préciser avec le paramètre `--registry`.

Nous pouvons alors effectuer la commande suivante:

```
npm i <my-package>
# npm i <my-package> --registry
https://gitlab.extra.imgt-group.com/api/v4/packages/npm/
```

Le module est alors installé, il suffit de l'importer là où nous en avons besoin.

c. Configuration du code

La commande NPM Link

La commande `npm link` permet de créer un "symlink" vers un package npm installé ou dans un répertoire, que ce soit sur un serveur ou en local. Il y a plusieurs avantages à mettre en place un symlink.

Avec `npm link`, nous pouvons lier un package local à une application. Durant tout le processus de développement, lorsque l'on va modifier notre package, les modifications seront directement apportées dans le package qui est dans notre projet. Il n'y a pas besoin de reconstruire l'application et l'installer à chaque fois, et d'autant moins la publier. Le temps de développement est considérablement réduit et plus fluide car on peut effectuer tous nos tests rapidement. En mettant notre package à disposition sur un serveur auquel plusieurs utilisateurs ont accès, tout le monde peut collaborer sur ce même package sans passer par Gitlab, et donc ne pas faire de pull à chaque fois qu'un package est publié.

En se plaçant dans notre package, on exécute la commande `npm link` dans le terminal pour créer un symlink. Sur Windows, le package est disponible dans `C:\Users\you\AppData\Roaming\npm\node_modules\`.

Backend

Pour illustrer les propos, nous allons utiliser le package "User".

Une fois notre package disponible sur Gitlab, nous pouvons nous rendre dans le répertoire de notre projet et installer le package via la commande `npm i` pour l'installer depuis Gitlab, ou créer un symlink via la commande `npm link`. Si tout s'est bien passé, notre module est déclaré dans notre fichier `package.json` et on le retrouve aussi dans l'arborescence du dossier `node_modules`. Pour commencer à l'utiliser, on se rend dans notre fichier `app.module.ts`, et on importe notre module. (annexe 21)

Le module est installé et configuré, nous pouvons lancer notre application. Si tout se passe bien, notre application se lance et les routes de notre applications sont chargées.

Frontend

Comme pour le backend, on va utiliser en guise d'exemple le module user.

Après la configuration de notre package et après l'avoir envoyé sur notre repository Github, on peut utiliser la commande `npm i` ou `npm link` pour pouvoir commander à l'utiliser dans notre application React. Après l'installation, on peut directement utiliser les composants et services que nous avons créés directement dans notre application, sur les pages ou juste en important le composant nécessaire.

11 - Partie 4 : Bilan et perspectives

Pour ce bilan, je voudrai revenir sur les difficultés que j'ai pu rencontrer pendant mon travail au sein d'IMGT Group sur ce projet. La plus grande difficulté était la compréhension de la structure du projet entier, que ce soit pour le frontend ou backend.

En arrivant dans l'entreprise, je pensais retrouver une structure similaire à des projets que j'avais déjà fait auparavant tout seul en suivant le guide d'Angular ou pendant mes stages, mais les choses étaient très différentes. Je n'avais jamais fait d'application modulaire, donc la compréhension du fonctionnement d'un module était compliquée, mais malgré la difficulté, j'ai su me frayer un chemin dans cette jungle.

Au cours du développement de l'application, on s'est rendu compte qu'Angular n'était peut être pas la bonne solution. La barrière d'entrée pour comprendre le fonctionnement de l'application n'est pas évidente, malgré que le framework lui même est strict quant à sa structure. C'est la raison principale de la migration vers React. Sur notre application React, le développement est plus rapide, ce qui a été développé en + de 6 mois avec Angular a été refait et amélioré avec React en seulement 2 mois ** JMA: pas trop d'accord avec l'analyse, Les temps de dev ne sont pas comparables, car l'application existait en angular et nous n'avons fait que la porter, ce qui est beaucoup plus simple . **On a remarqué que le développement était plus léger** JMA: de combien ?** et facile et surtout la compréhension de la structure et du code en général est plus accessible aux futures personnes qui viendront travailler pour IMGT. ** JMA: cela restera à prouver ;) **

Ce changement impacte aussi d'autres aspects de l'application, comme le poids, qui a presque doublé sur React à cause des nombreux packages NPM, mais la vitesse de compilation et déploiement est 5 fois plus rapide qu'Angular, ce qui n'est pas négligable.

Compilation: 12s React, 1min01s Angular => 5x plus rapide

Taille: 1.7Go React, 1.1Go Angular => 0.6x plus lourd

Build: 32s React, 1min22 Angular => 2.75x plus rapide

Taille module Template: 100 Ko React, 264 Ko => 2.64x moins lourd

**** JMA:** Les temps de compilation et de build ne sont pas pertinents pour la performance de l'application. Par contre, la taille l'est et tend à avantager angular. ******

12 - Nombre de caractères

13 - Tableau de fonctionnalité

Fonctionnalité	Angular	React
Composants	✓	✓
Liaison de données bidirectionnelle	✓	✓
Rendu côté serveur	✓	✓
Routage	✓	✓
Injection de dépendances	✓	✗
CLI (Interface en ligne de commande)	✓	✓
DOM virtuel	✗	✓
JSX	✗	✓
TypeScript intégré	✓	✓
Gestion d'état	✓	✓
Hooks	✗	✓
Lazy loading	✓	✓
Tests unitaires intégrés	✓	✓
Formulaires réactifs	✓	✗
PWA (Progressive Web App) support	✓	✓
Contexte (Context API)	✗	✓
Fragments	✗	✓

Nous avons fait fausse route au début car on pensait qu'Angular serait un bon choix stratégique, finalement cette erreur nous a coûté du temps de développement, car on devait développer de nouveau l'application depuis le début. Ce temps aurait pu être investi pour développer d'autres modules de l'application. **** JMA:** nous n'avons pas arrêté angular ni tout migré en react, les deux fonctionnant pour le moment sur la même API, et nous avons plusieurs frameworks dans le backbone pour le moment, donc pas forcément d'accord avec l'analyse ******

Une autre partie qui m'a posé une colle était le déploiement du module sur Gitlab. N'ayant jamais utilisé Gitlab et n'ayant jamais déployé d'application et de package npm, cette tâche s'est révélée très compliquée pour différentes raisons. De plus, les règles de sécurité chez IMGT Group étant très strictes, je ne pouvais pas tester tout ce que je voulais sans l'autorisation de mon tuteur. Celui-ci devait aussi transférer les fichiers que je voulais du serveur extra vers le serveur intra. En plus de ça, venaient s'ajouter les problèmes sur le serveur Gitlab, les pages de déploiement n'étaient pas disponibles pendant un certain moment **JMA: JMA: à ajouter pour le contexte : du fait du démantèlement de IMGT à Champs sur Marne, car le SI n'est pas encore redevenu complètement fonctionnel**, il m'était donc impossible de tester si la publication de package marchait.

” er, j’ai fait un point environ toute les semaines et j’aurais préféré qu’on en parle plutôt que je le lise dans ton rapport, ce qui ne correspond pas à l’encadrement qui t’a été fourni **—>

14 - Conclusion

15 - Bibliographie

15.1 - | A

[Angular](#)

[Ambient-it - React vs Angular](#)

[Atlassain Intégration Continue](#)

[Axios](#)

15.2 - | B

[Bitsrc - DOM Incrémental](#)

[Bootstrap](#)

15.3 - | C

[Channel News - Avantage architecture modulaire](#)

[CERN - Premier site web](#)

15.4 - | D

[dev.to - NPM pack](#)

15.5 - | F

[FakerJS](#)

[FreelanceRepublic - Différence Angular/AngularJS](#)

15.6 - | G

[Github - local-npm Repository](#)

[Github - React Preline](#)

[Github - Verdaccio Repository](#)

[Gitlab](#)

[Gitlab - Déploiement automatique](#)

[Gitlab Forum - Runner stuck](#)

[Gitlab Registry](#)

[Gitlab - Gitlab Runner](#)

[Gitlab - Gitlab Runner](#)

15.7 - ** | I **

[IBM Intégration Continue](#)

15.8 - ** | K **

[Kinsta - Angular vs React](#)

[Krausest - Benchmark JS Framework](#)

15.9 - | N

[NestJS](#)

[npmjs](#)

[npmjs - local-npm](#)

[npmjs - npm-pack](#)

15.10 - | M

[Markdown guide](#)

[Medium - npm-link](#)

15.11 - | O

[Octoverse Github - Top programming languages](#)

[Organisation Performante - Architecture Modulaire](#)

15.12 - | Q

[QImfo - Angular vs React](#)

15.13 - | R

[Remixicon](#)

15.14 - | S

[Stackoverflow - Gitlab Runner stuck](#)

[Stackoverflow - npm i from local registry](#)

[Stackoverflow - npm local registry](#)

[Stackoverflow - Developer statistics 2023](#)

[Stackoverflow - Developer statistics 2022](#)

[Stackoverflow - Developer statistics 2021](#)

15.15 - | V

[VM.PL - Angular](#)

15.16 - | W

[Wikipedia Angular](#)

[Wikipedia npm](#)

16 - Annexes

1. Structure applicaiton Angular



2. Commande pour générer un composant Angular

```
ng generate component my-component  
# ou  
ng g c my-component
```

3. Commande pour générer un module Angular

```
ng generate module my-module  
# ou  
ng g c my-module
```

4. Commande pour générer un service Angular

```
ng generate service my-service  
# ou  
ng g c service
```

5. Tableau de comparaisons des différents Frameworks Front

Comparaison avec d'autres frameworks Front

Le meilleur résultat est en gras

Framework	Vitesse compilation (1ère compilation en ms)	Vitesse compilation (en ms)	Taille (en Mo)	Popularité (en 2023)	Date de sortie	Type de DOM
Angular	4198 ms (+ 1462,5%)	1282 ms (+ 842,7%)	320 Mo (+ 406,3%)	17.46%	2016	Incrémental
React	279 ms (+ 3,7%)	137 ms (+ 0,7%)	88.8 Mo (+ 40,5%)	40.58%	2013	Virtuel
React + SWC	267 ms	136 ms	137 Mo (+ 116,7%)	40.58%	2013	Virtuel
NextJS	1498 ms (+ 457,6%)	1469 ms (+ 979,4%)	284 Mo (+ 349,3%)	16.67%	2014	Virtuel
Nuxt	Client: 2060 ms / Server: 1890 ms (~+ 633,5%)	Client: 1450 ms / Server: 1260 ms (~+ 896,3%)	231 Mo (+ 265,5%)	3.69%	2018	Universal Rendering
Vue	335 ms (+ 24,5%)	195 ms (+ 43,4%)	66.9 Mo (+ 5,8%)	16.38%	2014	Virtuel
Svelte	556 ms (+ 106,7%)	410 ms (+ 201,5%)	63.2 Mo	6.62%	2016	Physique

6. Structure module NestJS

```

- app/
  - app.controller.ts
  - app.controller.spec.ts
  - app.module.ts
  - app.service.ts
  - main.ts
- package.json
- angular.json

```

7. Exemple route API

```
@Get('') // @Post, @Update, @Delete
getHello(): string {
  return this.appService.getHello();
}
```

8. Exemple route API avancé

```
@Get('/:id') // @POST, @UPDATE, @DELETE
@HttpCode(200) // Exemple de decorator NestJS
@IsAuth() // Exemple de decorator personnalisé
getHello(@Param('id') id: number): string {
  return this.appService.getUser(id);
}
```

9. Structure module avec TypeORM

```
- app/
  - app.controller.ts
  - app.controller.spec.ts
  - app.module.ts
  - app.service.ts
  + - app.manager.ts
  + - app.repository.ts
  + - app.entity.ts
  - main.ts
  - package.json
  - angular.json
```

10. Comparaisons de différents frameworks Backend





























Comparaison avec d'autres frameworks Backend

Le meilleur résultat est en gras





















Framework	Vitesse compilation	Taille	Popularité (2023)	Date de sortie
Express	76 ms	2.03 Mo (+ 63,7%)	19.28%	2010
Express + Required*	80 ms (+ 5,2%)	4.95 Mo (+ 298,4%)	19.28%	2010
Fastify	143 ms (+ 88,1%)	7.53 Mo (+ 507,3%)	2.05%	2016
Hapi	111 ms (+ 46,1%)	1.24 Mo	N/D	2012
Nest	315 ms (+ 314,5%)	147 Mo (+ 11744,9%)	5.13%	2017
FastAPI (Python)	249 ms (+ 227,6%)	N/D	7.42%	2018

**Express avec les bibliothèques body-parser, cors, nodemon et mysql2*
















11. Structure module front complet

```
-  src
  -  apps
    -  pages
      -  my_module-management
        -  my_modules
          -  my_module_page
            -  my_module_page.component.ts
            -  my_module_page.component.html
            -  my_module_page.component.spec.ts
            -  my_module_page.component.scss
          -  my_modules.component.ts
          -  my_modules.component.html
          -  my_modules.component.spec.ts
          -  my_modules.component.scss
          -  my_module-management.module.ts
          -  my_module-management.routing.ts
        -  pages-routing.module.ts
        -  pages.module.ts
      -  back_modules
        -  api
          -  my_module
            -  services
              -  my_module.service.ts
            -  models
              -  my_module.ts
            -  components
              -  ...
            -  my_module.module.ts
```

















12. Structure module backend complet

```
-  api
  -  src
    -  @web
      -  my_module <-- Package NPM
        -  src
          -  data
            -  data-launch.data.ts
          -  entities
            -  my_module.entity.ts
            ...
          -  dto
            -  entity-create.dto.ts
            -  entity-update.dto.ts
            ...
          -  repository
            -  my_module.repository.ts
            -  index.ts
            -  my_module.module.ts
        -  controllers
          -  my_module.controller.ts
          ...
        -  app.module.ts
      -  authentication
```

13. Package front end

```
-  projects
  -  notre-library
    -  src
      -  lib
        -  notre-library
          -  notre-library.component.html
          -  notre-library.component.spec.ts
          -  notre-library.component.css
          -  notre-library.component.ts
        -  public-api.ts
      -  ng-package.json
      -  package.json
      -  tsconfig.lib.json
      -  tsconfig.lib.prod.json
      -  tsconfig.spec.json
```

14. Structure package front end

```
-  projects
  -  calendar
    -  src
      -  lib
        -  cal
          -  components
            -  calendar
              -  calendar.component.html
              -  calendar.component.spec.ts
              -  calendar.component.css
              -  calendar.component.ts
            -  services
              -  calendar.service.ts
              -  calendar.module.ts
          -  public-api.ts
```













15. Export éléments de la library

```
export * from './lib/cal/components/calendar/calendar.component';  
export * from './lib/cal/services/calendar.service.ts';  
//...
```

16. Script build

```
"scripts":  
  "build-calendar": "ng build calendar",
```

17. Structure package backend

```
-  src  
  -  dto  
    -  create-calendar.dto.ts  
    -  update-calendar.dto.ts  
  -  entities  
    -  calendar.entity.ts  
  -  repositories  
    -  calendar.repository.ts  
  -  services  
    -  calendar.service.ts  
  -  calendar.module.ts  
  -  index.ts
```

v

18. Export éléments package backend

```
export * from "../dto/create-calendar.dto.ts";  
export * from "../dto/update-calendar.ts";  
//...
```

19. Script build backend

```
"main": "dist/index.js",  
"types": "dist/index.d.ts",  
"scripts": {  
  "build": "nest build",  
},
```

20. Fichier déploiement automatique

```
image: node:latest  
  
stages:  
  - deploy  
  
publish-npm:  
  stage: deploy  
  script:  
    - echo "@scope:registry=https://${CI_SERVER_HOST}/api/v4/projects/${CI_PROJECT_ID}/package  
    - echo "//${CI_SERVER_HOST}/api/v4/projects/${CI_PROJECT_ID}/packages/npm/:_authToken=${CI  
.npmrc  
  - npm publish
```

21. Import module backend

```
import { UserModule } from "@imgt/user-module";  
//...  
@Module({  
  imports: [UserModule],  
  //...  
})
```

17 - Glossaire

17.1 - | A

API: Une API (Application Programming Interface) est une application permettant l'échange de données entre une base de données et une base de données, en utilisant le protocole HTTP et la méthode CRUD (Create, Read, Update, Delete). En utilisant une route vers notre serveur (par exemple https://entreprise.api.gouv.fr/v3/douanes/etablissements/id_etablissement) permet de récupérer, auprès du gouvernement, les données concernant un établissement. Les routes sont similaires à un site web, mais renvoient du contenu au format JSON et non des pages web.

17.2 - | F

Framework: Un framework est un outil structuré permettant un développement plus efficace grâce à une structure précise et organisée. Chaque framework est plus ou moins différent des autres, proposant des méthodes et une structure qui lui sont spécifiques. On peut considérer un framework comme le squelette de notre application.