Rapport de Stage

BERRICHI Yassine

BTS SIO 2B OPTION SLAM

LYCÉE TURGOT

INTRODUCTION

Dans le cadre de ma formation en BTS SIO , option SLAM (Solutions Logicielles et Applications Métiers), j'ai effectué un stage de développement web au sein d'une entreprise commençant le 4 Décembre 2023 qui s'est conclu le 27 Janvier 2024 , pour une durée totale de 8 semaines.

J'ai effectué ce stage au sein de l'entreprise HDM Network ASBL , en intégrant leur équipe et leur pôle développement web.

Durant ce stage, j'ai pu acquérir de multiples connaissances par le biais de formation sur des technologies émergentes utilisées au sein de l'entreprise , ainsi que part un gros projet où j'ai pu avoir une part de responsabilité et de travail.

Réussir à compléter chacune de mes missions me donnerait alors ma nouvelle expérience professionnelle ainsi que de renforcer mon niveau en programmation web , ma compréhension du code ainsi que les fondamentaux que j'ai acquis durant ces deux années de BTS SIO .

HDM NETWORK ASBL

HDM Network ASBL, basée à Charleroi en Belgique, a été fondée par Quentin Mousset et David Van Goidtsnoven. Cette entreprise se spécialise dans la proposition de formations de qualité visant à développer les compétences professionnelles dans les divers domaines du numérique. Grâce à son engagement en faveur de l'excellence, HDM Network offre une expérience de formation enrichissante.

L'équipe de HDM Network compte entre 50 et 60 professionnels travaillant dans différents pôles clés. Parmi ces pôles, on retrouve les RH spécialisées dans le recrutement et le placement, ainsi que des experts en Web Development, Web Design et graphisme (illustrateurs). De plus, l'entreprise dispose d'une équipe dédiée aux partenariats B2B et d'une équipe spécialisée en SEO (Optimisation pour les moteurs de recherche).

Un aspect essentiel de l'activité de HDM Network est de mettre en relation ces différents pôles avec des entreprises intéressées à travailler avec elles sur différentes missions, en fonction des besoins spécifiques de chaque domaine. L'entreprise facilite ainsi la collaboration entre ses experts et les clients potentiels, garantissant une réponse adaptée aux demandes des entreprises en matière de compétences numériques.

Grâce à son approche centrée sur la qualité des formations et à sa capacité à créer des synergies entre les différentes équipes, HDM Network ASBL est reconnue comme un acteur majeur dans le développement des compétences numériques en Belgique.

MON ROLE DANS HDM

HORAIRES :
-Lundi à Jeudi : 8h30-12h / 13h-16h30

-Vendredi : 8h30-11h30 / 12h30-16h30

MES MISSIONS :

Durant ce stage au sein du pôle WebDev de HDM que j'ai pu intégrer pour 8 semaines , voici le travail que j'ai effectué :

-Formations sur de nouvelles technologies liées au développement web
-La réalisation d'application web front-end et back-end
-La réalisation d'API

TECHNOLOGIES UTILISÉES :

-HTML
-NestJS

-GraphQL
-JavaScript
-NodeJS
-React

FORMATIONS

NESTJS

Au cours de ma formation, j'ai eu l'opportunité de plonger dans l'univers passionnant de NestJS, un framework Node.js qui m'a permis d'explorer de nouvelles perspectives dans le développement d'applications web. Cette expérience a été particulièrement enrichissante, ouvrant la voie à une compréhension approfondie des concepts clés de NestJS et de son utilisation dans des projets concrets.

NestJS : Fondements et Principes :

NestJS, construit sur la base de TypeScript, a marqué une transition significative dans ma compréhension du développement backend. Sa structure modulaire basée sur des modules, des contrôleurs et des services offre une organisation claire et maintenable du code. J'ai rapidement appris à apprécier la facilité avec laquelle NestJS intègre des bibliothèques tierces et permet une gestion transparente des dépendances.

L'Architecture Orientée Module :

L'architecture orientée module de NestJS m'a permis de concevoir des applications modulaires et évolutives. En divisant mon code en modules fonctionnels, j'ai constaté une amélioration significative de la lisibilité du code et de la facilité de maintenance. Les modules ont également facilité l'intégration de fonctionnalités supplémentaires sans compromettre la structure globale de l'application.

Les Decorators et les Pipes :

L'utilisation intensive des decorators et des pipes a été l'un des aspects les plus intrigants de mon apprentissage. Les decorators, tels que ceux utilisés pour définir les routes dans les contrôleurs, simplifient la configuration des endpoints API. Les pipes, quant à eux, offrent un moyen élégant de valider et de transformer les données entrantes, renforçant ainsi la robustesse de l'application.

Le Système de Middleware :

Le système de middleware de NestJS m'a permis de comprendre comment gérer efficacement le flux de requêtes au sein de l'application. En utilisant des middleware personnalisés, j'ai pu implémenter des fonctionnalités de gestion des erreurs, d'authentification et d'autorisation de manière centralisée, améliorant ainsi la cohérence et la sécurité de mes applications.

La Gestion des Exceptions :

La gestion des exceptions dans NestJS m'a ouvert les yeux sur la nécessité d'une gestion robuste des erreurs. J'ai appris à créer des filtres d'exception personnalisés pour traiter spécifiquement certains types d'erreurs, améliorant ainsi la qualité des messages d'erreur renvoyés aux clients.

La Communication entre Modules avec GraphQL :

L'intégration de GraphQL dans NestJS a été une expérience fascinante. En utilisant le système de resolvers, de queries et de mutations, j'ai pu créer des API flexibles et optimisées pour répondre aux besoins spécifiques de mes applications. GraphQL a également facilité la

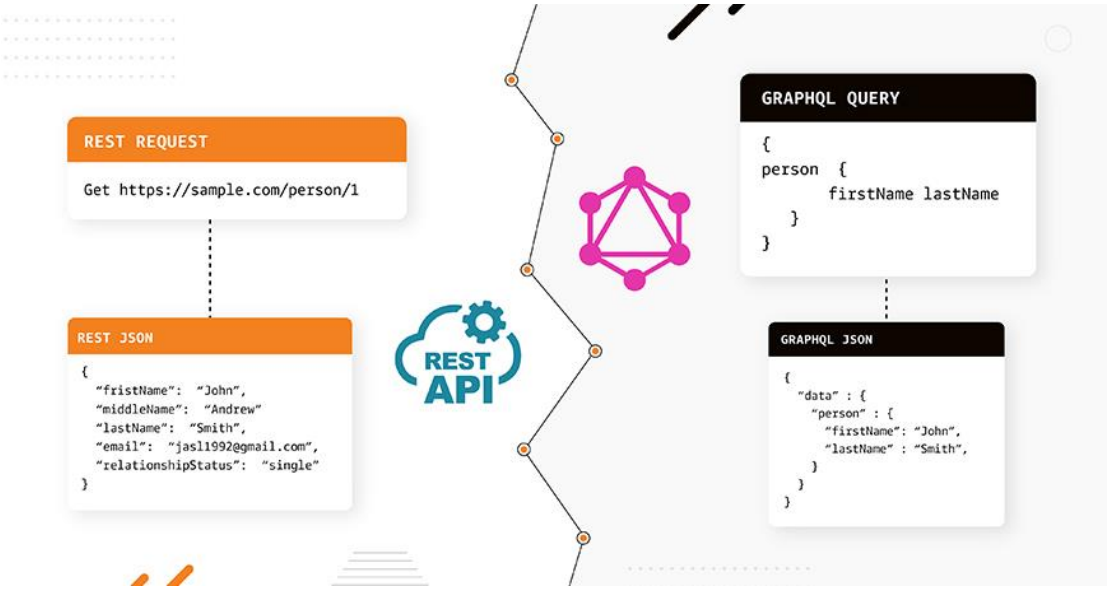
communication entre les modules, permettant une interaction fluide entre différentes parties de l'application.

La Sécurité avec NestJS :

La sécurisation des applications a été un aspect clé de ma formation. En utilisant les decorators et les pipes, j'ai mis en place des mécanismes de validation des données et de contrôle d'accès, renforçant ainsi la sécurité des API que j'ai développées.

En conclusion, ma formation approfondie sur NestJS a été une aventure captivante. J'ai acquis une compréhension approfondie de l'architecture, des principes et des fonctionnalités offerts par ce framework innovant. Cette expérience a grandement enrichi mes compétences en développement backend et a renforcé ma confiance dans la création d'applications web robustes et évolutives. Je suis impatient(e) d'appliquer ces connaissances dans mes projets futurs, convaincu(e) que NestJS continuera à jouer un rôle central dans mon parcours professionnel.

GRAPHQL



Au cours de ma formation, j'ai plongé dans l'univers novateur de GraphQL, une technologie qui a considérablement transformé la manière dont les données sont gérées et échangées

dans le développement d'applications web. Cette immersion dans GraphQL a été une expérience éclairante, me permettant de découvrir les principes fondamentaux et de mettre en pratique ses fonctionnalités révolutionnaires.

GraphQL : Un Aperçu Fondamental

GraphQL, conçu par Facebook, se distingue par son approche flexible et déclarative pour la récupération de données. À la différence des API REST traditionnelles, GraphQL permet aux clients de spécifier exactement les données dont ils ont besoin, réduisant ainsi le surcoût lié à la surcharge de données inutiles. Ce paradigme centré sur le client a été une révélation pour moi, offrant une expérience utilisateur plus efficace et rapide.

Le Langage de Requête Puissant :

L'une des caractéristiques les plus puissantes de GraphQL est son langage de requête expressif. Les clients peuvent définir précisément les champs et les relations qu'ils souhaitent récupérer, éliminant ainsi la surcharge d'informations inutiles. Cette capacité à demander uniquement ce dont le client a besoin a des implications significatives en termes d'efficacité et de performances.

Les Résolveurs et les Schémas :

La mise en œuvre de resolvers et de schémas dans GraphQL m'a permis de comprendre comment organiser et structurer les données dans une API GraphQL. Les resolvers agissent comme des fonctions pour chaque champ déclaré dans le schéma, fournissant ainsi la logique pour récupérer les données correspondantes. Cette approche modulaire et décentralisée facilite l'évolutivité des API GraphQL.

La Flexibilité des Mutations :

GraphQL offre une approche cohérente pour effectuer des opérations de modification de données via des mutations. J'ai pu apprécier la flexibilité de définir des mutations sur mesure pour répondre aux besoins spécifiques de chaque application. La validation des données et la gestion des erreurs sont simplifiées, contribuant à une expérience de développement plus fluide.

Les Avantages de la Communication entre Clients et Serveurs :

L'aspect conversationnel de GraphQL, où les clients et les serveurs peuvent négocier les données à échanger, a été un point fort. Cette communication bidirectionnelle permet aux clients de récupérer rapidement des données spécifiques sans dépendre des modifications du côté serveur. La version interactive du Playground GraphQL a été particulièrement utile pour explorer et tester les requêtes.

La Gestion Intuitive des Erreurs :

La gestion des erreurs dans GraphQL est transparente et intuitive. Plutôt que d'obtenir une multitude de codes d'erreur, GraphQL renvoie une réponse structurée avec des messages d'erreur spécifiques à chaque champ en cas de problème. Cela simplifie le processus de débogage et améliore la qualité des retours d'erreur pour les développeurs et les utilisateurs finaux.

GraphQL et Prisma : Un Duo Puissant :

L'intégration de Prisma avec GraphQL a ouvert de nouvelles perspectives dans la gestion des bases de données. La synchronisation entre le modèle de données dans Prisma et le schéma GraphQL simplifie la création, la modification et la récupération de données. Cette synergie entre GraphQL et Prisma a considérablement amélioré la productivité du développement backend.

Conclusion sur GraphQL :

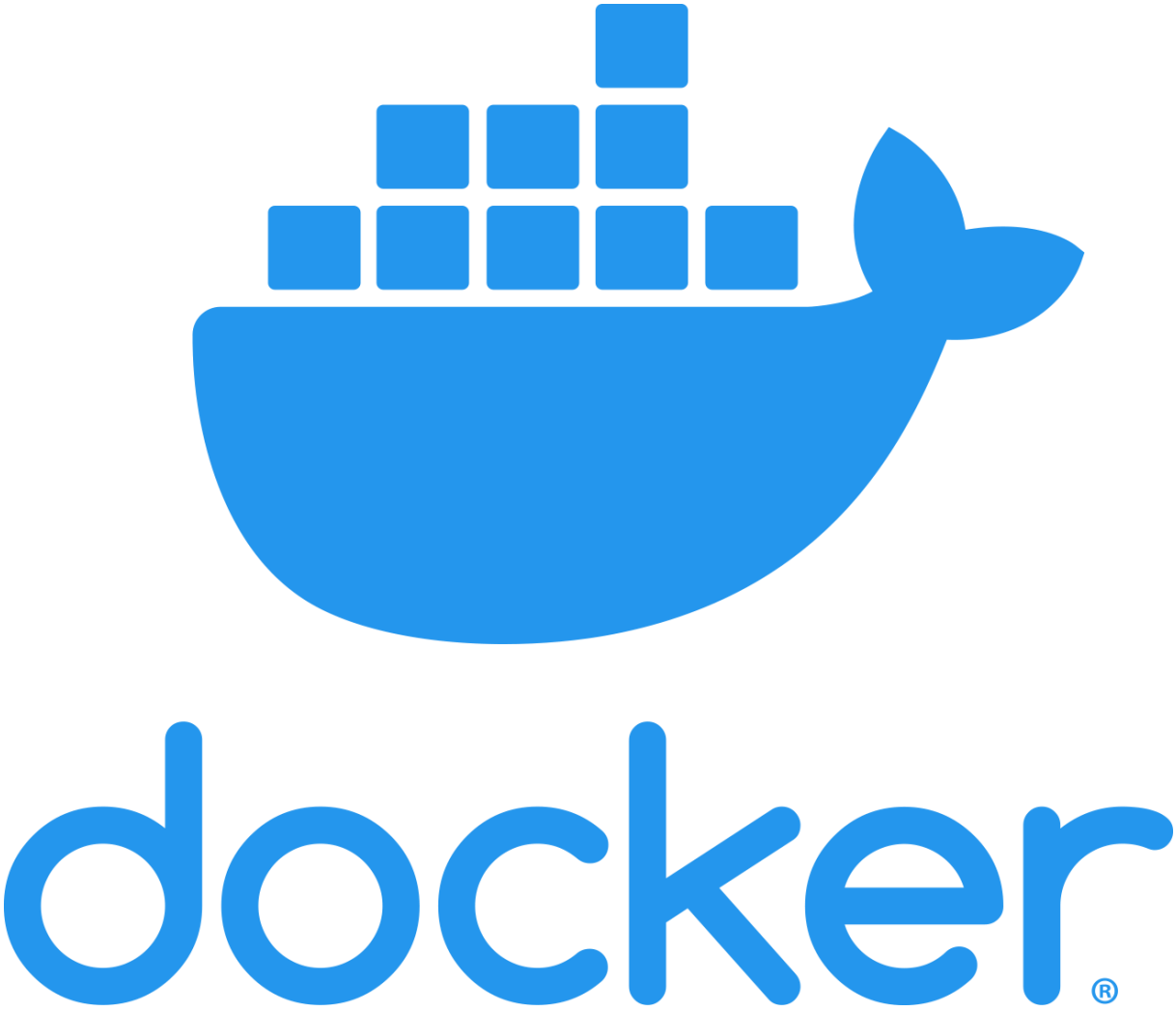
GraphQL a été une révélation dans ma formation, offrant une approche radicalement différente de la gestion des données dans le développement web. Sa flexibilité, sa puissance de requête, et sa capacité à faciliter la communication entre clients et serveurs ont considérablement enrichi mes compétences. J'ai désormais une compréhension approfondie de la manière dont GraphQL peut optimiser les performances des applications et simplifier la récupération des données.

GraphQL dans le Paysage Technologique :

L'utilisation croissante de GraphQL dans l'industrie témoigne de son impact significatif sur le développement web moderne. En intégrant GraphQL dans mes projets futurs, je suis

convaincu que je pourrai continuer à offrir des expériences utilisateur améliorées et optimisées.

DOCKER



Docker, une plateforme de conteneurisation qui a considérablement simplifié le déploiement d'applications et la gestion des environnements. Mon aventure avec Docker s'est concentrée sur la création d'un conteneur pour abriter une base de données MySQL, une expérience qui a élargi mes compétences dans le domaine de la gestion d'infrastructures logicielles.

Docker : Les conteneurs :

Docker, avec son approche de conteneurisation, offre une méthode efficace pour encapsuler des applications et leurs dépendances dans des conteneurs légers et portables. Cela m'a permis de créer un environnement isolé et reproductible, indépendant du système d'exploitation sous-jacent, simplifiant ainsi le déploiement et l'évolutivité des applications.

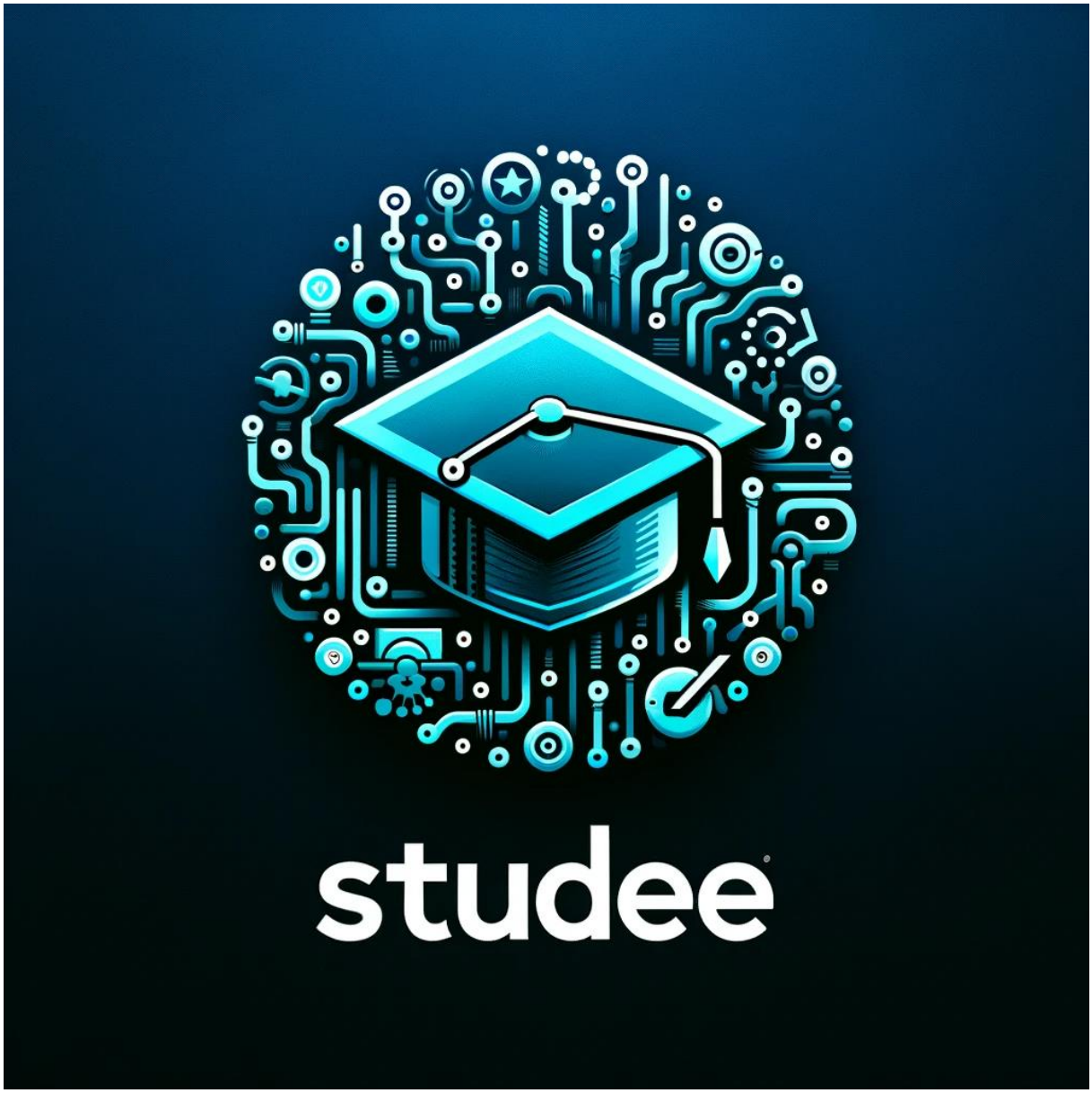
Création d'un Conteneur MySQL :

Ma première tâche dans Docker a consisté à créer un conteneur pour héberger une base de données MySQL. J'ai pu profiter de la richesse des images officielles de Docker Hub pour MySQL, qui fournissent un point de départ solide pour la configuration.

Configuration et Paramétrage :

La flexibilité de Docker m'a permis de configurer le conteneur MySQL en fonction des besoins spécifiques de mon application. La gestion des variables d'environnement et des volumes a été particulièrement utile pour définir des paramètres tels que les mots de passe, les bases de données et la persistance des données.

PROJET Studee



Résumer du projet

Studee se positionne comme une application mobile novatrice visant à faciliter la recherche et la mise en relation entre les étudiants (ou toute personne cherchant à effectuer un stage) et les entreprises proposant des opportunités de stage. Cette plateforme offre aux entreprises la possibilité de publier leurs offres de stage, créant ainsi une interface directe pour interagir avec des candidats potentiels. L'objectif principal de Studee est de simplifier le processus de recherche de stages en favorisant une communication directe entre les deux parties, améliorant ainsi l'efficacité du placement des stagiaires dans divers domaines d'activité.

Le projet est mené par le chef de Projet général d'HDM , Quentin Mousset qui a défini les technologies utilisées , la structure du code que l'on doit suivre ainsi que les fonctionnalités à développer.
La partie front-end elle est réalisée et mise en place par Cédric et Zakaria , deux développeurs front.
La partie back-end quant à elle est réalisée par mes soins ainsi que de l'aide des deux dev front lorsqu'ils le peuvent , car être seul sur la partie back-end de ce projet reste une tâche compliquée.

Technologies utilisées :

-Langages :
TypeScript/JavaScript :
TypeScript est un sur-ensemble de JavaScript qui ajoute le support des types statiques au langage. L'intérêt de son utilisation réside dans la possibilité d'ajouter des types aux variables, ce qui peut améliorer la robustesse du code en détectant les erreurs potentielles plus tôt dans le processus de développement.

-Frameworks :
NestJS :
NestJS est un framework Node.js pour la construction d'applications serveur. NestJS facilite la création d'applications robustes et évolutives en suivant une architecture modulaire. Son utilisation dans une application mobile permet de créer une backend solide et maintenable qui peut gérer les demandes de l'application de manière efficace.

GraphQL :

GraphQL est un langage de requête pour les API, et il offre une alternative aux API REST traditionnelles. Il ne s'agit pas d'une bibliothèque ou d'un framework en soi, mais plutôt d'une spécification de requête. Son utilisation dans une application mobile permet de récupérer exactement les données nécessaires, réduisant ainsi la surcharge de données et améliorant les performances de l'application.

React :

React est une bibliothèque JavaScript pour la construction d'interfaces utilisateur, principalement pour des applications à page unique (SPA). Bien que React ne soit pas un framework complet, il offre des composants réutilisables qui facilitent le développement d'interfaces utilisateur interactives. Son utilisation dans une application mobile permet de créer une interface utilisateur réactive, offrant une expérience utilisateur fluide et agréable.

-Base de donnée :

MySQL 5.7 :

MySQL est un système de gestion de base de données relationnelles (SGBDR) Son utilisation dans une application mobile permet de stocker et de récupérer efficacement les données nécessaires à partir du backend, assurant une gestion fiable des informations liées aux stages, utilisateurs, etc.

POUR LE MAQUETTAGE ET GESTION DE PROJET :

Figma est une application de conception d'interfaces utilisateur (UI) et d'expérience utilisateur (UX) basée sur le cloud. Il ne s'agit ni d'une bibliothèque ni d'un framework, mais plutôt d'un outil de conception collaboratif. Figma permet aux équipes de créer des maquettes interactives, de collaborer en temps réel et de partager facilement des designs. Son utilisation dans la phase de maquettage d'un projet permet une conception visuelle et interactive, tout en favorisant la collaboration entre les membres de l'équipe.

Miro est une plateforme de tableau blanc collaboratif en ligne. Il ne s'agit pas d'une bibliothèque ou d'un framework, mais plutôt d'un outil de gestion visuelle de projet. Miro offre des fonctionnalités telles que le maquettage, le brainstorming, la planification de projets, etc. Son utilisation facilite la collaboration à distance, permettant à l'équipe de travailler ensemble de manière synchronisée sur des diagrammes, des maquettes et d'autres aspects visuels du projet.

Jira Software est un outil de gestion de projet agile développé par Atlassian. Il ne s'agit pas d'une bibliothèque ou d'un framework, mais plutôt d'un logiciel de suivi de projets et de gestion des tâches. Jira Software offre des fonctionnalités telles que la création de tickets, la planification de sprints, le suivi des problèmes, etc. Son utilisation dans la gestion de projet permet de suivre et d'organiser les tâches, d'assurer la transparence et la collaboration au sein de l'équipe de développement.

Cahier des charges

Pour mener ce projet au mieux , il a fallu établir un cahier des charges.
Cahier des charges pour l'application Studee:

1. Login:

Type: Login obligatoire pour accéder à l'ensemble des fonctionnalités.

Détails: Les utilisateurs doivent s'inscrire avec une adresse e-mail et un mot de passe. L'accès libre peut être envisagé plus tard, une fois que des données significatives sont collectées.

2. Menu "Qui sommes-nous":

Emplacement: Page des paramètres (settings).

Détails: Un lien vers une page "Qui sommes-nous" doit être inclus dans le menu des paramètres pour fournir des informations sur l'application.

3. Inscription:

Informations nécessaires: Adresse e-mail, mot de passe.

Détails: Les utilisateurs doivent fournir des informations de base lors de l'inscription. D'autres détails peuvent être ajoutés plus tard, après la connexion.

4. Connexion:

Action après connexion: Affichage d'une page pour compléter le profil (étudiant/entreprise).

Détails: Les utilisateurs seront redirigés vers une page pour ajouter des informations supplémentaires à leur profil après la connexion. Le système doit alerter les utilisateurs qui ont omis ces informations lorsqu'ils tentent de poster ou candidater.

5. Notifications par e-mail:

Cas particulier: Si l'utilisateur supprime l'application et une offre correspondante est disponible, l'informer par e-mail.

Détails: Mise en place d'un système d'e-mails pour informer les utilisateurs des opportunités correspondantes, même s'ils ont désinstallé l'application.

6. Affichage des offres d'entreprises:

Format: Sliders.

Détails: Les offres d'entreprises doivent être présentées de manière attrayante via des sliders pour faciliter la navigation.

7. StageFinder et Filtres:

Caractéristiques: Système de hashtag pour les offres avec filtres additionnels.

Inspiration: Modélisation de la fonctionnalité en s'inspirant de LinkedIn.

Détails: Mise en place d'un système de hashtag pour faciliter la recherche et filtres additionnels pour affiner les résultats.

8. Secteurs d'activité:

Gestion: Deux listes de secteurs (déclaré par entreprise, certifié par l'application).

Validation: Autocomplétion basée sur les secteurs validés par l'application.

Inspiration: Analyse de la gestion des secteurs d'activité sur LinkedIn.

Détails: Les entreprises déclarent leur secteur, mais la certification est gérée par l'application. Utilisation d'autocomplétion basée sur les secteurs validés.

9. Intégration des Réseaux Sociaux:

Fonctionnalité: Afficher les réseaux sociaux avec une vérification d'URL.

Détails: Au clic sur un icône de réseau social, une modal doit s'afficher avec l'URL du réseau. Si valide, le logo devient actif sur le profil.

10. Monétisation:

Options de monétisation:

Dépôt de nouvelles offres.

Activation de push notifications pour l'auto-sourcing (crédits pour un certain nombre de notifications).

Crédits de l'application pour acheter divers services.

Fonctionnalité payante pour avoir plus d'un collaborateur disponible.

11. Administration:

Rôle administratif: Ajout d'un administrateur pour gérer les secteurs d'activité certifiés.

12. Documentation:

Création: Élaboration d'une documentation détaillée pour l'utilisation de l'application et la gestion de ses fonctionnalités.

13. Sécurité:

Mesures de sécurité: Intégration de mesures de sécurité pour protéger les données sensibles des utilisateurs et assurer la confidentialité.

14. Tests:

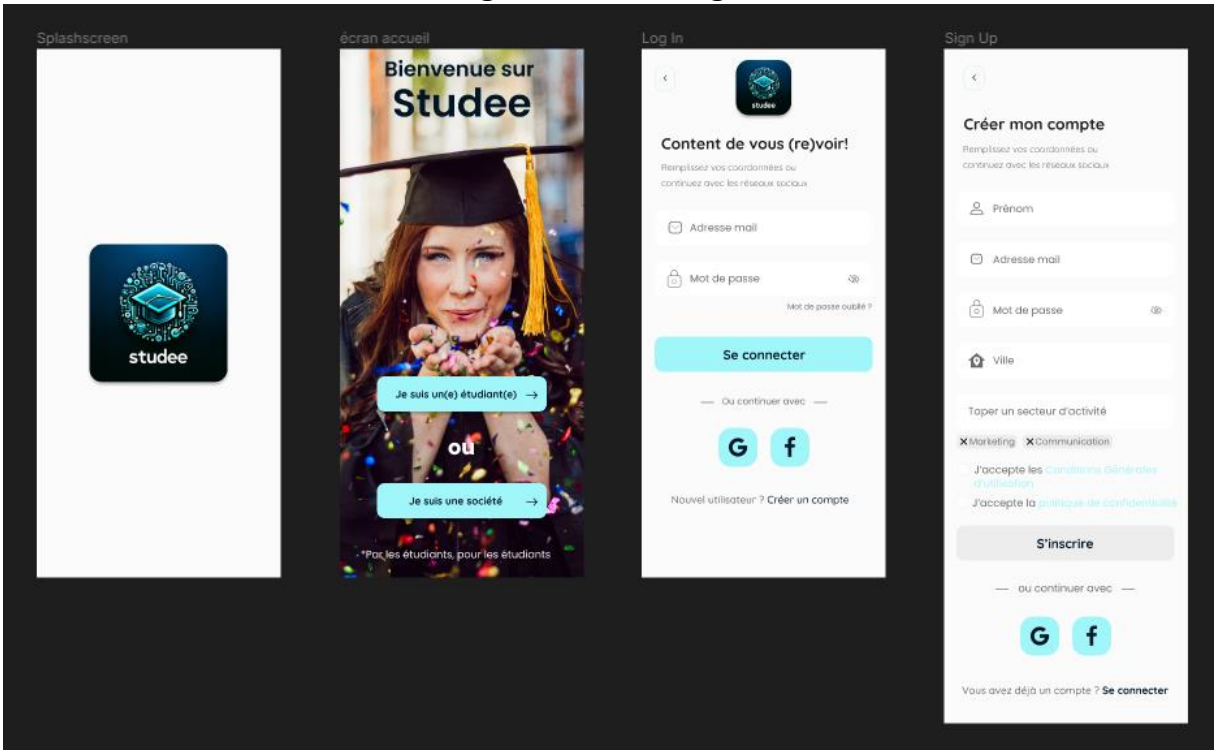
Tests unitaires et d'intégration: Réalisation de tests approfondis pour garantir la stabilité et la fiabilité de l'application.

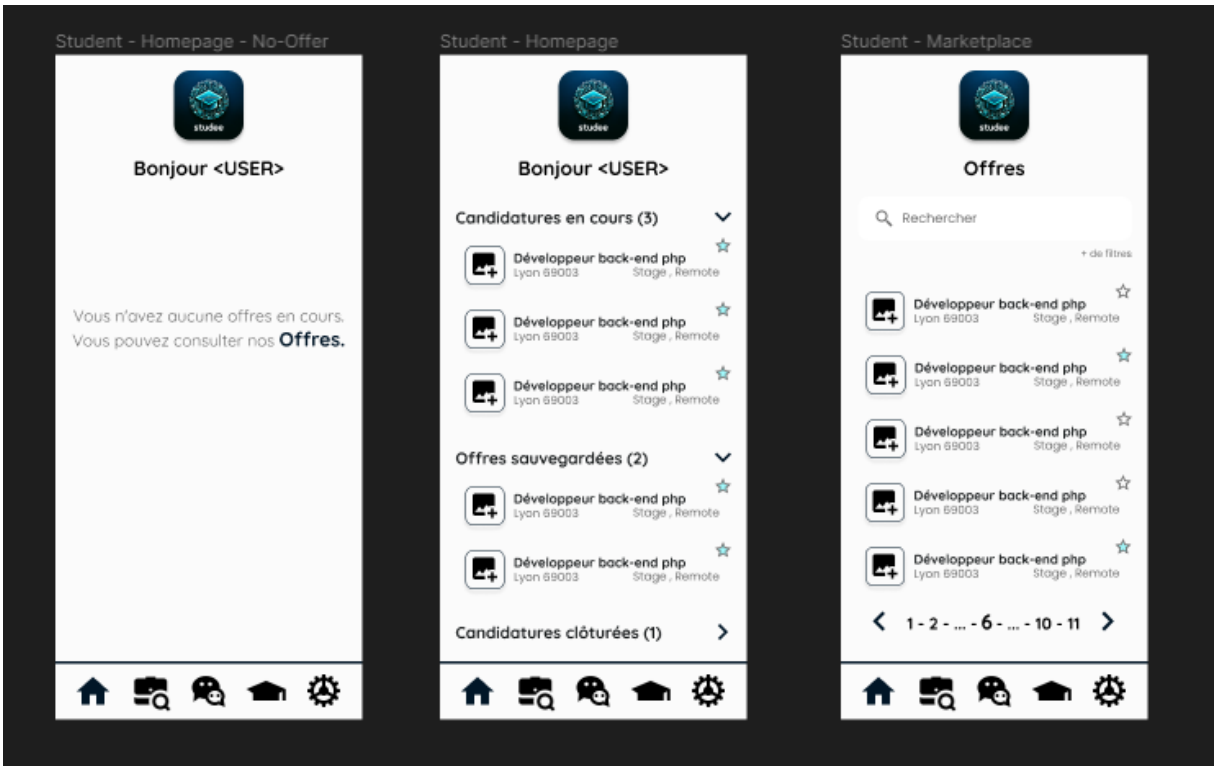
15. Support:

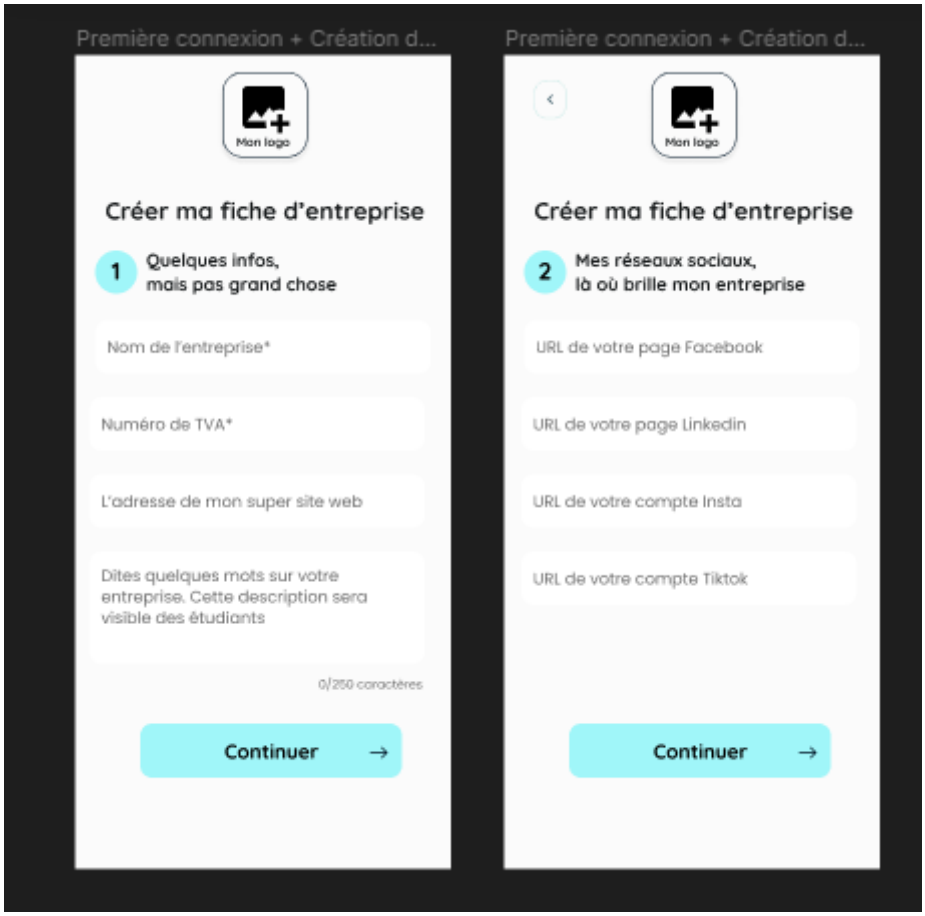
Canal de support: Mise en place d'un système de support pour les utilisateurs avec des canaux clairs de communication et de résolution des problèmes.

Maquettage de l’application :

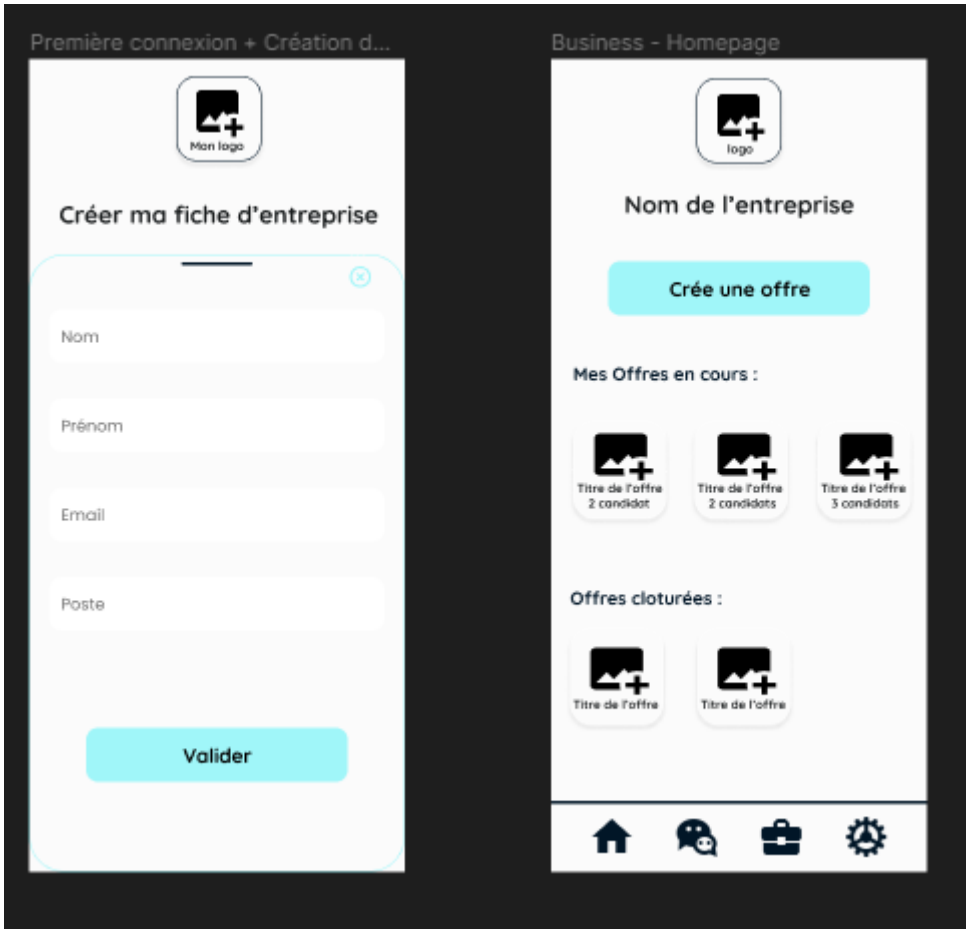
Logiciel utilisé : Figma

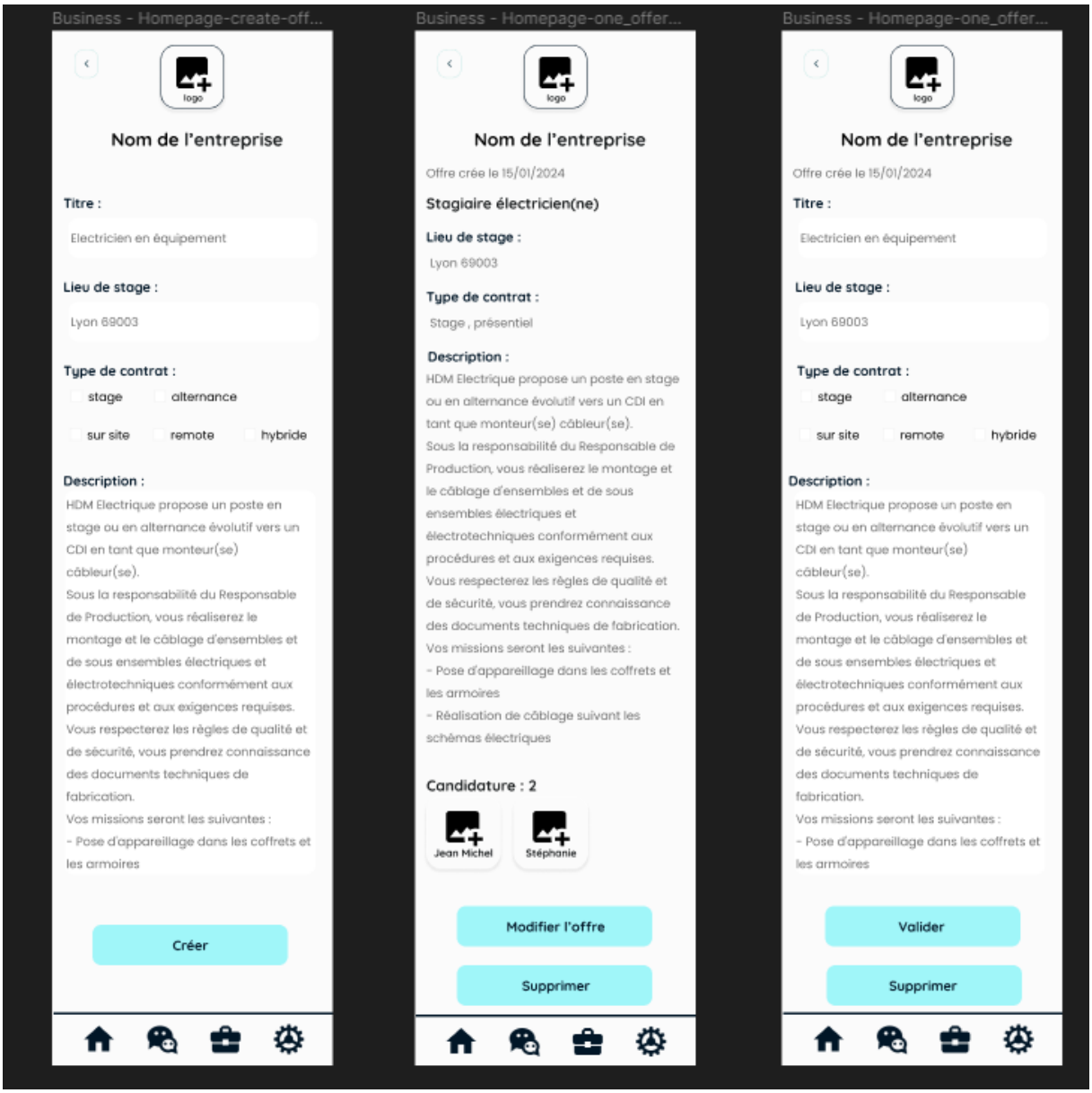


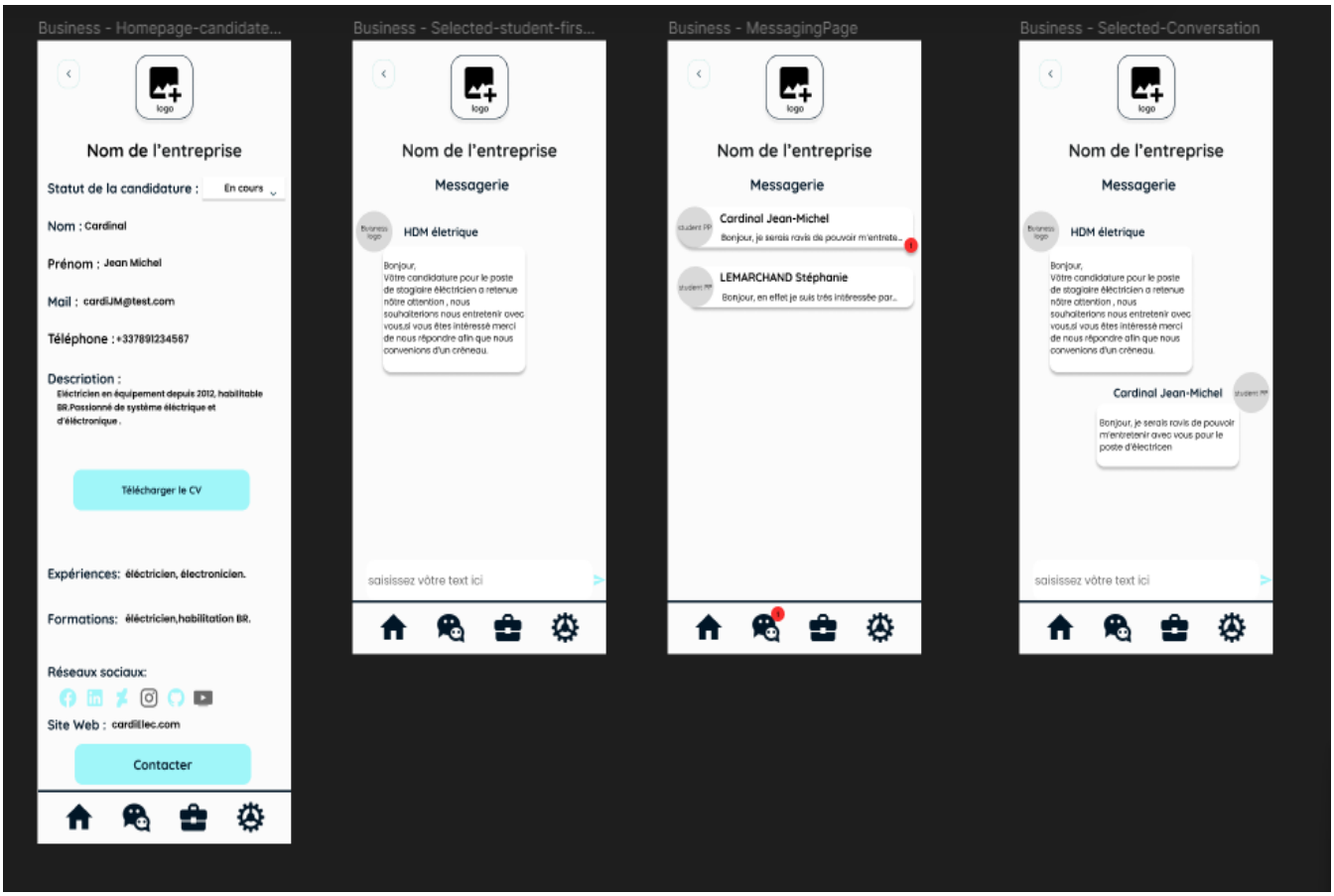

Student - Homepage - No-Offer
Bonjour <USER>
Vous n'avez aucune offres en cours.
Vous pouvez consulter nos Offres.
Student - Homepage
Bonjour <USER>
Candidatures en cours (3)
Développeur back-end php
Lyon 69003
Stage , Remote
Offres sauvegardées (2)
Candidatures clôturées (1)
Student - Marketplace
Offres
Rechercher
+ de filtres
1 - 2 - ... - 6 - ... - 10 - 11


Première connexion + Création d...
Mon logo
Créer ma fiche d'entreprise
1
Quelques infos,
mais pas grand chose
Nom de l'entreprise*
Numéro de TVA*
L'adresse de mon super site web
Dites quelques mots sur votre entreprise. Cette description sera visible des étudiants
0/250 caractères
Continuer
Première connexion + Création d...
Mon logo
Créer ma fiche d'entreprise
2
Mes réseaux sociaux,
là où brille mon entreprise
URL de votre page Facebook
URL de votre page Linkedin
URL de votre compte Insta
URL de votre compte Tiktok
Continuer


Première connexion + Création d...
Mon logo
Créer ma fiche d'entreprise
3
Seul on va plus vite,
ensemble on va plus loin
Quentin Mousset
Compte 2
J'ai besoin de plus de comptes
Terminer


Première connexion + Création d...
Mon logo
Créer ma fiche d'entreprise
Nom
Prénom
Email
Poste
Valider
Business - Homepage
logo
Nom de l'entreprise
Crée une offre
Mes Offres en cours :
Titre de l'offre
2 candidat
Titre de l'offre
2 candidats
Titre de l'offre
3 candidats
Offres cloturées :
Titre de l'offre
Titre de l'offre

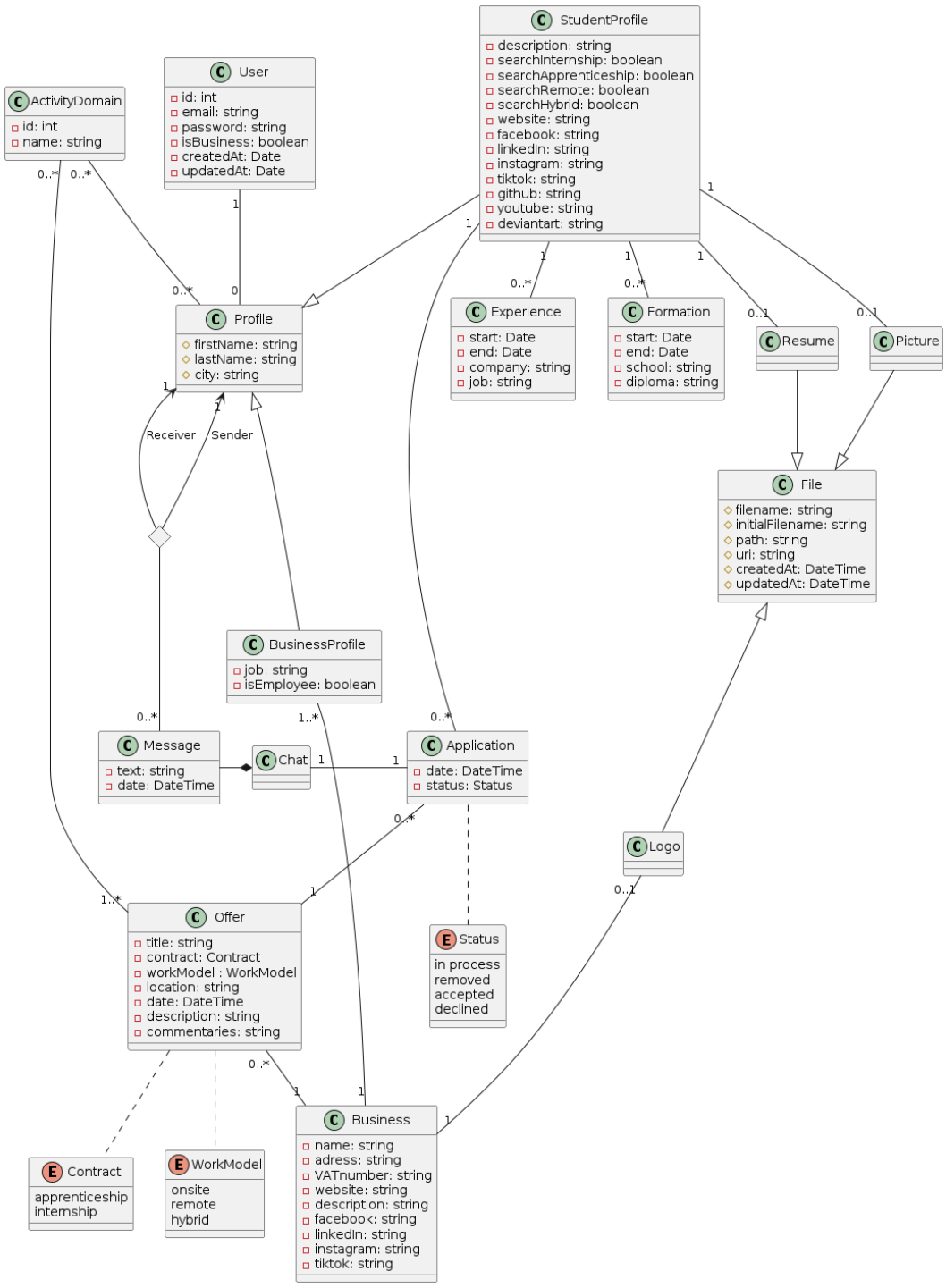

Business - Homepage-create-off...
logo
Nom de l'entreprise
Titre :
Electricien en équipement
Lieu de stage :
Lyon 69003
Type de contrat :
stage
alternance
sur site
remote
hybride
Description :
HDM Electrique propose un poste en stage ou en alternance évolutif vers un CDI en tant que monteur(se) câbleur(se).
Sous la responsabilité du Responsable de Production, vous réaliserez le montage et le câblage d'ensembles et de sous ensembles électriques et électrotechniques conformément aux procédures et aux exigences requises.
Vous respecterez les règles de qualité et de sécurité, vous prendrez connaissance des documents techniques de fabrication.
Vos missions seront les suivantes :
- Pose d'appareillage dans les coffrets et les armoires
Créer
Business - Homepage-one_offer...
logo
Nom de l'entreprise
Offre crée le 15/01/2024
Stagiaire électricien(ne)
Lieu de stage :
Lyon 69003
Type de contrat :
Stage , présentiel
Description :
HDM Electrique propose un poste en stage ou en alternance évolutif vers un CDI en tant que monteur(se) câbleur(se).
Sous la responsabilité du Responsable de Production, vous réaliserez le montage et le câblage d'ensembles et de sous ensembles électriques et électrotechniques conformément aux procédures et aux exigences requises.
Vous respecterez les règles de qualité et de sécurité, vous prendrez connaissance des documents techniques de fabrication.
Vos missions seront les suivantes :
- Pose d'appareillage dans les coffrets et les armoires
- Réalisation de câblage suivant les schémas électriques
Candidature : 2
Jean Michel
Stéphanie
Modifier l'offre
Supprimer
Business - Homepage-one_offer...
logo
Nom de l'entreprise
Offre crée le 15/01/2024
Titre :
Electricien en équipement
Lieu de stage :
Lyon 69003
Type de contrat :
stage
alternance
sur site
remote
hybride
Description :
HDM Electrique propose un poste en stage ou en alternance évolutif vers un CDI en tant que monteur(se) câbleur(se).
Sous la responsabilité du Responsable de Production, vous réaliserez le montage et le câblage d'ensembles et de sous ensembles électriques et électrotechniques conformément aux procédures et aux exigences requises.
Vous respecterez les règles de qualité et de sécurité, vous prendrez connaissance des documents techniques de fabrication.
Vos missions seront les suivantes :
- Pose d'appareillage dans les coffrets et les armoires
Valider
Supprimer


Business - Homepage-candidate...
Nom de l'entreprise
Statut de la candidature :
En cours
Nom : Cardinal
Prénom : Jean Michel
Mail : cardiJM@test.com
Téléphone : +337891234567
Descriotion :
Eléctricien en équipement depuis 2012, habilitable BR.Passionné de système éléctrique et d'éléctronique .
Télécharger le CV
Expériences: éléctricien, électronicien.
Formations: éléctricien,habilitation BR.
Réseaux sociaux:
Site Web : cardillec.com
Contacter
Business - Selected-student-firs...
Nom de l'entreprise
Messagerie
HDM életrique
Bonjour,
Vôtre candidature pour le poste de stagiaire éléctricien a retenue nôtre attention, nous souhaitions nous entretenir avec vous,si vous êtes intéressé merci de nous répondre afin que nous convenions d'un créneau.
saisissez vôtre text ici
Business - MessagingPage
Nom de l'entreprise
Messagerie
Cardinal Jean-Michel
Bonjour, je serais ravis de pouvoir m'entrete...
LEMARCHAND Stéphanie
Bonjour, en effet je suis très intéressée par...
Business - Selected-Conversation
Nom de l'entreprise
Messagerie
HDM életrique
Bonjour,
Vôtre candidature pour le poste de stagiaire éléctricien a retenue nôtre attention, nous souhaitions nous entretenir avec vous,si vous êtes intéressé merci de nous répondre afin que nous convenions d'un créneau.
Cardinal Jean-Michel
Bonjour, je serais ravis de pouvoir m'entretenir avec vous pour le poste d'électricien
saisissez vôtre text ici

Base de donnée :

La base de donnée sera hébergée via un conteneur docker MySQL 5.7.
Afin de schématiser celle-ci , avec mes collègues sur le projet nous effectuerons une longue tâche de réflexion afin de mettre en place un diagramme de classe (UML) pour que l'on suive tous la même structure et idée dans le code.

Voici donc le diagramme de classe de l'application Studee réalisé via PlantUML



Ma responsabilité dans le projet

Je devais me charger du back-end , donc du traitement de données ainsi que des diverses fonctionnalités. Marqué par le manque de temps du stage et le temps d'adaptation et de compréhension de la structure du projet , après la mise en place de la base de donnée via Prisma j'ai alors été uniquement assigné aux fonctionnalités d'authentification.

Du côté de l'API donc , je devais structurer mon code de la manière suivante :

Dans un dossier « User » pour toute les fonctionnalités concernant donc l'utilisateur :

1 . Entity

Le dossier Entity contenant des fichiers entity.ts , représente le modèle de données de notre base de données de notre domaine. Elle contient les propriétés principales qui caractérisent , dans notre cas un utilisateur dans le système.
Exemple d'entity dans le projet STUDEE :

```
1  import { Field, GraphQLISODateTime, Int, ObjectType } from '@nestjs/graphql';
2  import { ContextualGraphqlRequest } from '../../../index';
3  import Profile from '../../Profile/Entity/Profile';
4  import { Exclude } from 'class-transformer';
5
6  @ObjectType()
7  export default class User {
8    @Exclude()
9    @Field(() => Int)
10   id: number;
11
12   @Field()
13   email: string;
14
15   password: string;
16
17   @Field()
18   isBusiness: boolean;
19
20   @Field(() => Profile, { nullable: true })
21   profile?: Profile;
22
23   @Exclude()
24   @Field(() => GraphQLISODateTime)
25   createdAt: Date;
26
27   @Exclude()
28   @Field(() => GraphQLISODateTime)
29   updatedAt: Date;
30
31   context?: ContextualGraphqlRequest;
32 }
```

On y retrouve bien toute les propriétés composant un Utilisateur. Sachant qu'on peut voir qu'on lie une entité a une autre , a la ligne 21 en appelant une autre entity appelée « Profile » qu'on importe plus haut dans le script.

2 . DTO

Les DTO (Data object transfer) , sont des fichiers utilisés pour transférer les données entre différents parties de l'application , principalement entre le client et le serveur. Dans mon cas

par exemple , j'ai donc un DTO pour le processus d'enregistrement. En voici un exemple :

```
@InputType()
export class RegisterDto {
  @Field()
  email: string;

  @Field()
  password: string;

  @Field()
  isBusiness: boolean;
```

On reconnaît généralement les Dto grâce au décorateur graphql « InputType() »

3 . Repository

Le Repository est un fichier (contenu dans un dossier général d'User appeler Repository), qui est responsable de l'interaction avec la source de donnée qui peut être une base de données ou un autre système de stockage. Dans mon cas, mon repository gère donc les opérations liées a l'enregistrement et authentification des utilisateurs.

En voici un exemple extrait du projet Studee :

```
import { BadRequestException, Injectable } from '@nestjs/common';
import { PrismaService } from '../../../Core/Datasource/Prisma';
import Bcrypt from 'src/Core/Security/Service/encryption/Bcrypt';
import User from '../Entity/User';
import { RegisterDto } from '../UseCase/Register/RegisterUserDto';
import ProfileCreateInput from '../UseCase/Register/RegisterUserDto';

@Injectable()
export default class AuthRepository {
  constructor(private readonly prisma: PrismaService, private readonly bcrypt: Bcrypt) {}

  validatePassword(password: string): void {
    // Vérifie la longueur
    if (password.length < 8 || password.length > 40) {
      throw new BadRequestException('Le mot de passe doit contenir entre 8 et 40 caractères.');
    }

    // Vérifie la présence d'au moins une majuscule, un chiffre et un caractère spécial
    const passwordRegex = /^(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]+$/;
    if (!passwordRegex.test(password)) {
      throw new BadRequestException('Le mot de passe doit contenir au moins une majuscule, un chiffre et un caractère spécial.');
    }
  }

  async save(dto: RegisterDto, profileInput: ProfileCreateInput): Promise<User> {
    this.validatePassword(dto.password); // Valider le mot de passe

    const hashedPassword = await this.bcrypt.hash(dto.password);
    // Création de l'utilisateur
    const createdUser = await this.prisma.user.create({
      data: {
        email: dto.email,
        password: hashedPassword,
        isBusiness: dto.isBusiness,
        profil: {
          create: {
            firstname: profileInput.firstName,
            lastName: profileInput.lastName,
            city: profileInput.city,
          },
        },
      },
```

Extrait du fichier AuthRepository. Il y contient la logique de vérification et sauvegarde d'un utilisateur , ainsi que de sa création.

4 . UseCase

Le dossier UseCase contient divers fichiers usecase , qui chacun contient sa logique métier de l'application. Dans notre cas , j'ai alors un fichier RegisterUserUseCase , qui gère le processus d'enregistrement d'un nouvel utilisateur.
En voici un exemple direct :

```
import { BadRequestException, Injectable } from '@nestjs/common';
import AuthRepository from '../../repository/AuthRepository';
import { RegisterDto } from './RegisterUserDto';
import ProfileCreateInput from './RegisterUserDto';
import { ContextualGraphqlRequest, UseCase } from 'src';
import User from '../../Entity/User';


@Injectable()
export default class RegisterUserUsecase implements UseCase<Promise<User>, [dto: RegisterDto, profileInput: ProfileCreateInput]> {
  constructor(private readonly authRepository: AuthRepository) {}

  async handle(context: ContextualGraphqlRequest, dto: RegisterDto, profileInput: ProfileCreateInput): Promise<User> {
    try {
      return this.authRepository.save(dto, profileInput);
    } catch (error) {
      throw new BadRequestException(error.message);
    }
  }
}
```

On y voit d'ailleurs un appel a la méthode save , contenue dans le repository d'authentification ainsi que nos DTO dans les paramètres de la méthode.

5 . Resolver

Le Resolver lui , est responsable de la gestion des requêtes du client et de l'appel aux usecase appropriés. Dans notre cas , le resolver d'authentification donc expose une mutation graphql pour enregistrer un nouvel utilisateur.

En voici un exemple :

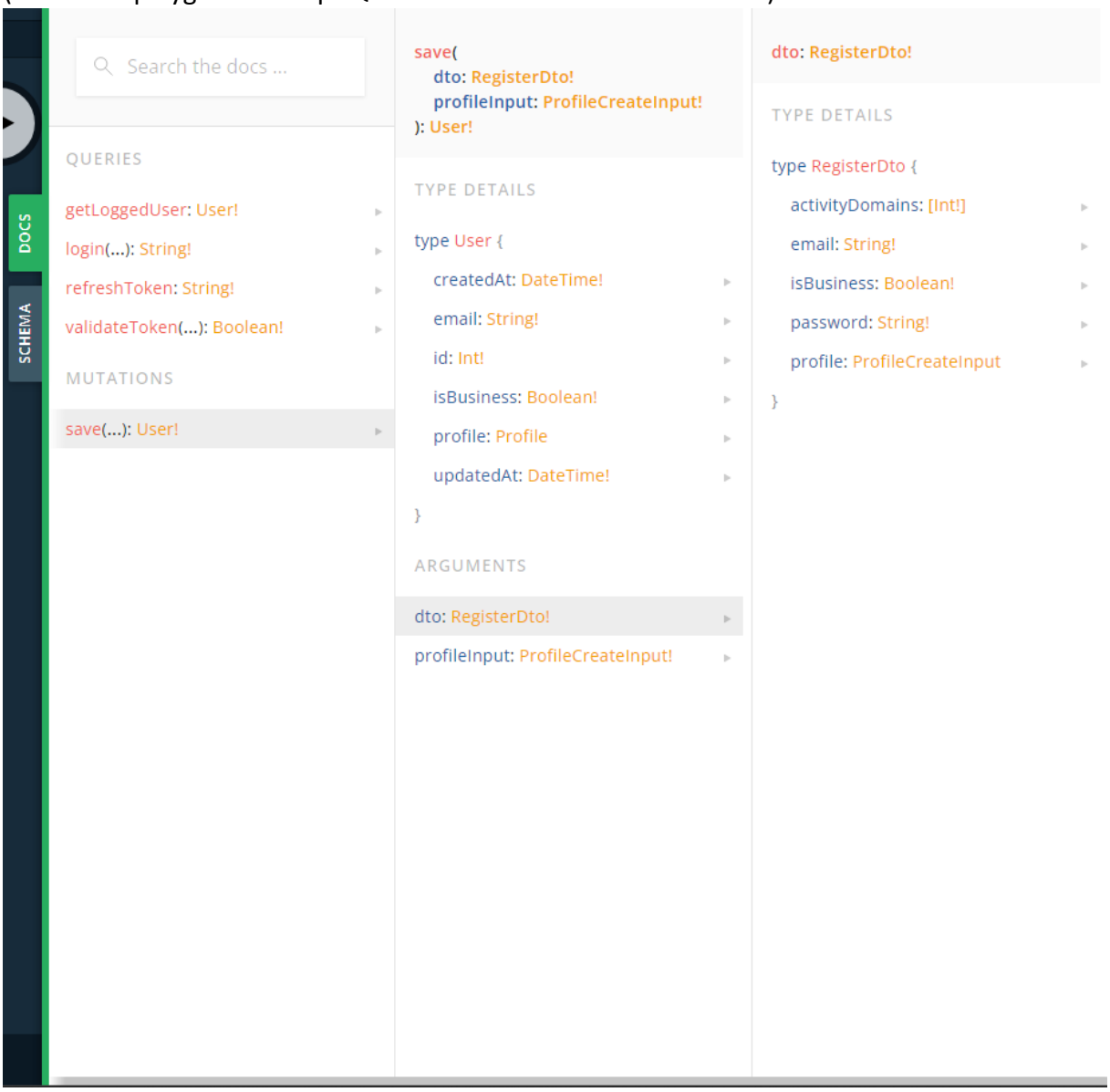
```
import { Resolver, Mutation, Args, Context } from '@nestjs/graphql';
import RegisterUserUsecase from '../UseCase/Register/RegisterUserUseCase';
import User from '../Entity/User';
import { ContextualRequest } from 'src/Core/Decorator/ContextualRequest';
import { RegisterDto } from '../UseCase/Register/RegisterUserDto';
import ProfileCreateInput from '../UseCase/Register/RegisterUserDto';
@Resolver()
export default class AuthResolver {
  constructor(private readonly registerUserUseCase: RegisterUserUsecase) {}

  @Mutation(() => User)
  async save(
    @ContextualRequest() context,
    @Args('dto') dto: RegisterDto,
    @Args('profileInput') profileInput: ProfileCreateInput,
  ) {
    return this.registerUserUseCase.handle(context, dto, profileInput);
  }
}
```

Voici donc la structure globale du projet pour implémenter une fonctionnalité. Chaque composant joue un rôle spécifique et suit le principe de séparation des préoccupations.

En utilisant GraphQL ainsi que Prisma pour la gestion de la base de donnée , j'y découvre alors une nouvelle manière de programmer. En effet , grâce a GraphQL , comparé aux API REST traditionnelles , on ne surcharge pas la bande passante avec des données inutiles et utilise uniquement les données spécifiées , réduit le nombre de requêtes puisqu'il récupère tout ce qui est nécessaire en une requête , facilite l'évolution du projet car on peut implémenter de nouvelles fonctionnalités sans impacter les précédentes ainsi que leur requêtes , le typage de graphql est STRICTE et garanti alors une sécurité exemplaire , et pour finir GraphQl a une documentation interactive basée sur le schéma de ma base de donnée me facilitant ainsi la compréhensionde mon API.

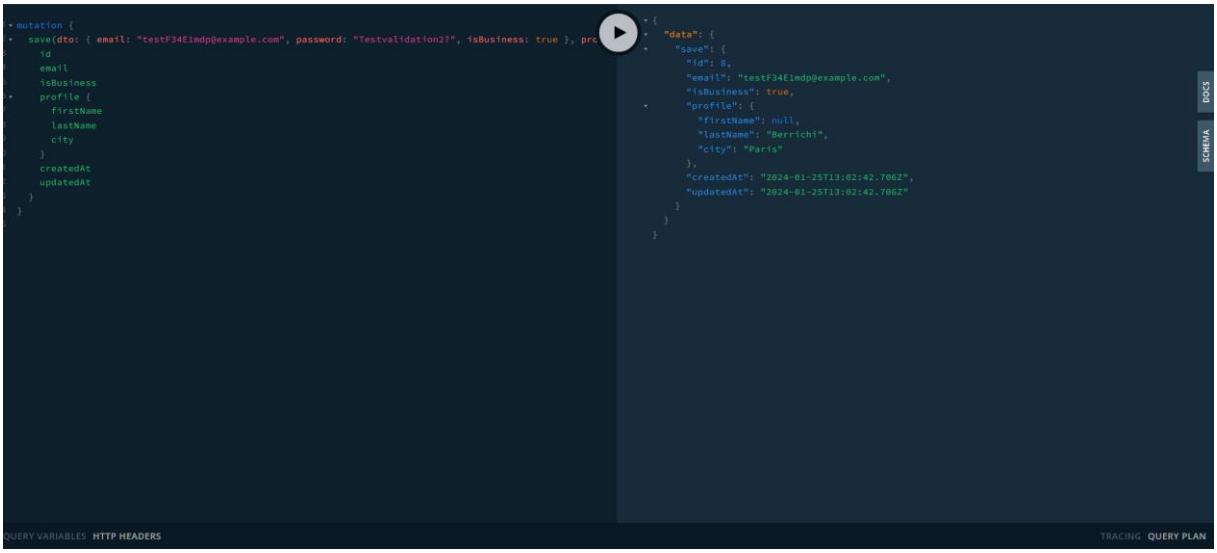
(Extrait du playground GraphQL de la documentation interactive) :



Concernant Prisma , j'ai pu découvrir l'utilisation d'ORM (Object Relational Mapping) qui simplifie l'interaction avec ma base de donnée. Il évite d'écrire manuellement les requêtes SQL , donc ça évite une immense faille concernant les injections SQL et donc allège la charge de travail côté sécurisation. De plus , Prisma génère des type-safes queries , c'est-à-dire qu'il génère des types typescript pour le schéma de la base de donnée pour garantir la sécurité des types lors de l'écriture de requêtes.

Test de ma fonctionnalité

Afin d'essayer ma fonctionnalité d'authentification , je peux aller dans le « playground GraphQL » , qui se lance en serveur local lorsque je lance l'API , pour tester ma mutation. Cela se fait de la manière suivante :



Nous pouvons voir a gauche ma mutation que j'ai implémentée dans l'API , avec les champs nécessaires.
Sur la deuxième ligne , j'écris toute les valeurs que je souhaite. S'il en manque une obligatoire , j'aurais une erreur. Si j'utilise le même mail , j'aurais une erreur de contrainte d'unicité sur la clé email. Si le mot de passe ne respecte pas le regex de même , une erreur me le montrera.

A droite , j'ai décidé de retourner l'utilisateur dans ma logique métier pour vérifier que ça a bien fonctionné , et nous pouvons donc y voir l'utilisateur qui vient d'être créer.

Afin d'en être sûr et certain , nous pouvons utiliser un outil appelé PRISMA STUDIO , qui s'exécute avec la commande npx prisma studio dans le terminal . Cela affichera un IDE tel que PHPMYADMIN avec toute les tables , et donc je peux y vérifier dans la table User s'il y a bien mes valeurs.(Je pouvais très bien aussi vérifier depuis un terminal de commande grâce a mon conteneur docker)

Extrait de PRISMA STUDIO :

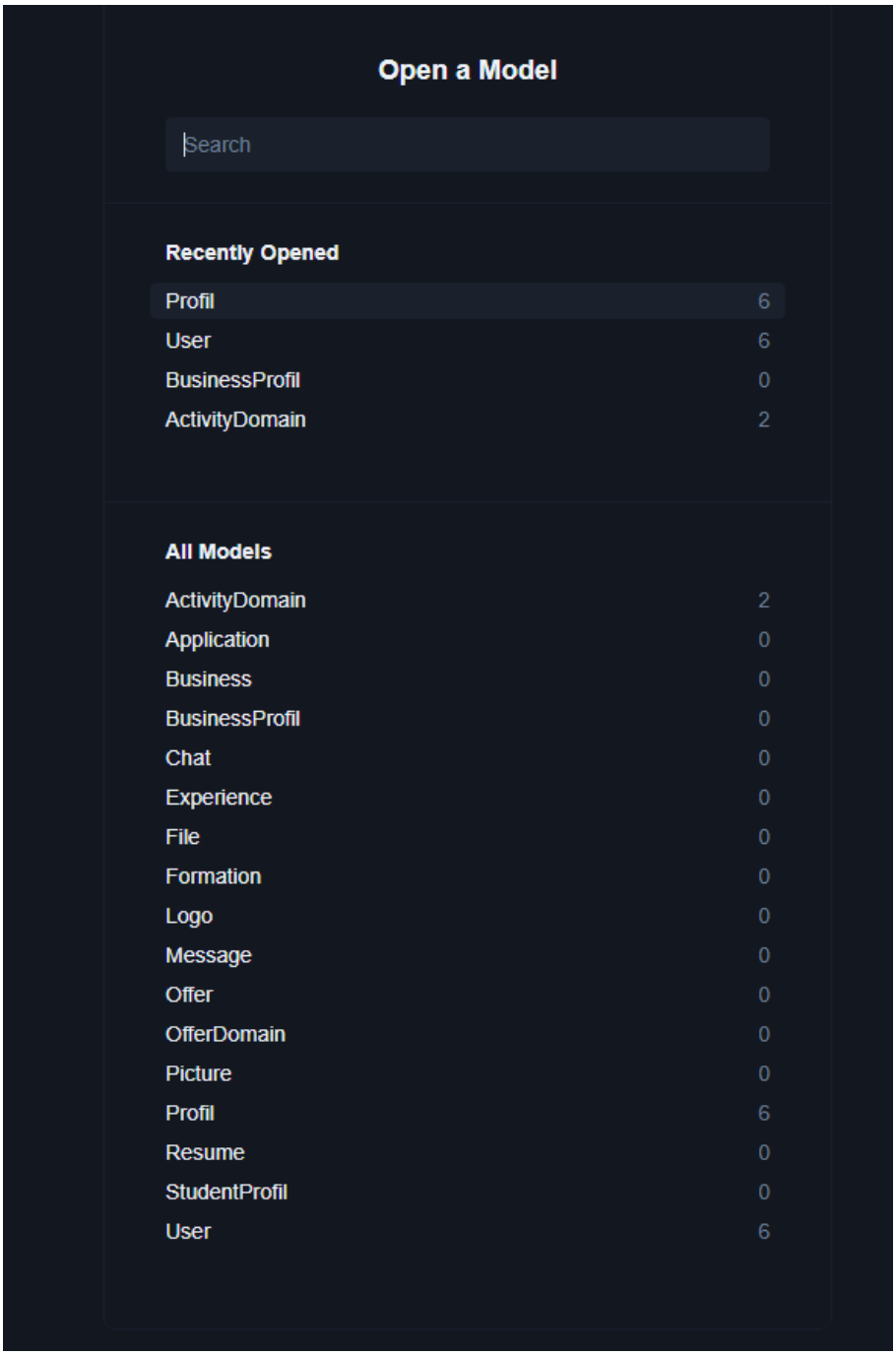


Table User :

On peut également voir qu'un profile y est lié , et donc si on click sur « Profil » , le studio prisma nous emmènera sur la table studio a la ligne concernant l'utilisateur sur lequel on a clicker.

GITHUB

Pour travailler en équipe , et consulter le code de nos collègues , nous avons utiliser github.

Après chaque journée , il a fallu que je « push » mon code sur le github du projet dans ma branche. Chaque personne travaillant sur le projet avait une branche sur laquelle il travaillait donc.
L'intérêt est de récupérer le travail des autres , de le review ensemble et de garder « à jour le projet ».
Une fois que nous avions tous finis notre partie du travail sur une fonctionnalité , côté front et back , nous effectuons un « merge » de nos branches pour tout mettre en commun.
Les risques sont évidemment d'y avoir des problèmes de compatibilités entre les codes , les bases de données etc.

Je n'ai pas échappé à cette règle ,et lors du merge pour essayer l'authentification il y a eu des conflits dans le code par rapport aux champs requis et champs renseignés , etc. Il y avait donc des problèmes dans la logique du respository , ainsi que dans les entity et le schéma prisma de la base de donnée.
Il a fallu régler tout ceci en équipe, mais heureusement pour nous le problème n'était pas d'une immense complexité et nous avons pu en une après midi tout faire fonctionner.

Alors, le système d'authentification est fonctionnel avec le front et le back combiné, la première fonctionnalité de l'application mobile était définitivement fonctionnelle !

Suite à ça, puisque le temps manquait et que je n'allais plus pouvoir faire d'autre sprints, on m'a simplement demandé de coder tous les fichiers des entités pour les prochaines fonctionnalités qu'ils feront après mon départ, une tâche pas très amusante mais plutôt simple.

CONCLUSION

La réalisation de ce projet a été une expérience enrichissante, couvrant divers aspects du développement web moderne. L'objectif principal était de mettre en place un système d'authentification sécurisé utilisant des technologies telles que NestJS, GraphQL et Prisma.

Le choix d'utiliser NestJS, un framework Node.js, a été motivé par sa structure modulaire, sa scalabilité et son intégration native avec TypeScript. L'utilisation de GraphQL a offert une flexibilité remarquable en permettant aux clients de spécifier les données exactes nécessaires, évitant ainsi le sur-fetching et le sous-fetching. Prisma, en tant que couche d'accès aux données, a facilité les opérations CRUD en fournissant une interface intuitive pour interagir avec la base de données.

Les points forts du projet résident dans la mise en place d'une architecture propre et modulaire. L'utilisation de decorators dans NestJS a permis de créer des résolvers et des DTOs clairs et concis. La séparation des responsabilités entre les résolvers, les use cases et les repositories a amélioré la maintenabilité du code.

Cependant, plusieurs défis ont rendu le parcours compliqué. L'intégration initiale des dépendances, notamment NestJS et Prisma, a nécessité une compréhension approfondie de la configuration et de l'initialisation. Les erreurs de configuration et de compréhension, bien que formatrices, ont parfois été source de frustration.

Le passage à l'utilisation de GraphQL a nécessité une adaptation, notamment dans la création de schémas GraphQL, mais a offert une souplesse et une clarté dans la définition des types d'opérations.

En conclusion, ce projet et ce stage a été une plongée profonde dans le développement backend « moderne », mettant en lumière les avantages et les défis de l'utilisation de technologies émergentes. Les compétences acquises dans la mise en place d'un système d'authentification sécurisé et dans la gestion des bases de données avec Prisma et GraphQL constituent une base solide pour aborder des projets plus vastes et complexes à l'avenir.