

Dossier Projet

Discord Communautaire Simplon

Rédigé par Benjamin Gamache



Résumé.....	4
Présentation personnelle.....	5
Présentation du projet.....	6
Le client : Simplon.....	6
Contextualisation du projet.....	6
Recueil du besoin Client.....	6
L'équipe en charge du projet.....	7
Proposition de solution :	8
Gestion du projet.....	9
Méthodologie de travail Agile.....	9
Le framework SCRUM.....	10
Les quatre grandes valeur de SCRUM.....	10
Les rôles dans SCRUM.....	11
Le Product Owner.....	11
Le SCRUM Master.....	11
L'équipe de développement.....	11
Les rituels	12
Sprint planning.....	12
Dailys SCRUM.....	12
Sprint Review.....	12
Sprint Retrospective.....	12
Définition des rôles SCRUM au sein de l'équipe	12
Jira.....	13
Git.....	13
GitFlow.....	14
GitHub	14
Architecture du projet	14
Mono-Repo VS Multi Repo.....	14
GitFlow Vs Fork.....	15
Convention de formatage des commits.....	15
Critères des Pulls Requests	16
Qualité de code : ESLint.....	16
La convention AirBnb	16
Spécifications Fonctionnelles.....	17
Règles de Gestion du Bot.....	17
RBAC (Role Based Access Control).....	18
User Stories les plus représentatives.....	18
Spécifications Techniques.....	19
La base de donnée : PostgreSQL.....	19
Choix du langage.....	19
TypeScript.....	19

Architecture : API	20
Stateless (Sans état)	20
RESTful API	20
Choix du Framework Back End (Coté API)	21
Choix du Framework Back End (Coté Back)	21
Choix du Framework Front End & ORM	22
Pourquoi utiliser un ORM ?	22
Tableau récapitulatif	23
Testing	24
JEST	24
TDD (Test Driven Dev.)	25
Coverage	25
Le test unitaire	25
Le test d'intégration	25
Stratégie de test du projet	25
Annexe	85

Résumé

I chosed to present the Community Discord Simplon Server

This project consist of centralizing each entities of Simplon on a same Discord Server.

Our team discussed with Simplon teams members and as learner we saw some issues with Simplon communication.

The setup of communications supports of Simplon was extremly messy and so we had to provide a solution to make those process less tedious.

That was the main idea of this project.

I mainly focused on Back End developpement with NestJS & DiscordJS as Bot Developpement Framework

I firstly worked on this project as project for real client with my course mates.

We were 4 from 3 first months, working with Agile method using SCRUM.

When intership period arrived my mates gone in their company to work as intern.

I decided to take this project as internship project because it was an interesting project.

I refactored API side and Bot side from the first month of my internship.

And i'd worked on new features to implement those to Onboarding Bot whose goal was to simplify the Onboarding process of new learners.

This project was firstly designed to be a set of 1 API, 1 Database, 1 Web Management & Monitoring Dashboard & 7 Bots, but at this moment it's 1 finalized bot instead of 7.

Présentation personnelle

J'ai commencé à coder à l'âge de 8 ans, même s'il s'agissait uniquement de modifications mineures du code des pages Web. J'appréciais le fait que du code que je ne comprenais pas puisse donner forme à des pages Web.

Au fil du temps, ma compréhension de l'informatique a évolué, me permettant de me tourner vers des langages plus avancés. Ma passion pour le développement s'est intensifiée.

De plus, j'ai acquis mes compétences de manière autodidacte, sans suivre d'études dans le domaine de l'informatique. Ce qui était autrefois une passion est devenu un désir professionnel.

J'ai arrêté l'école à l'âge de 16 ans car le système scolaire ne me convenait pas.

C'est pourquoi, il y a quelques mois, j'ai décidé d'intégrer une formation de Concepteur Développeur d'Applications afin de renforcer mes compétences et de passer du simple développement à des travaux de conception et de réflexion préalables à la création de projets.

Grâce au projet que j'ai réalisé pendant ma formation, j'ai pris conscience des opportunités offertes par Discord dans le monde professionnel. On peut dire la même chose de Wordpress. Qui aurait pu imaginer que Wordpress deviendrait si lucratif ?

C'est pourquoi, à l'issue de ce projet, j'ai décidé de créer ma start-up spécialisée dans le développement de bots Discord.

Présentation du projet

Le client : Simplon

Dans ce projet, notre client était l'organisme de formation **Simplon Hauts-de-France** (Simplon HDF). Simplon compte 21 000 apprenants et dispose de 7 fabriques numériques dans la région Hauts-De-France.

La communication est un enjeu majeur pour Simplon HDF, et le principal support de communication utilisé est Discord, qui permet les échanges entre les équipes et les apprenants.

Contextualisation du projet

Dans le cadre de ce projet, nous nous sommes concentrés sur la manière dont Simplon HDF communique.

En tant qu'apprenants, nous avons remarqué que les équipes de Simplon HDF effectuent des **actions répétitives** qui pourraient être **automatisées facilement**.

De plus, la **communication**, notamment avec les apprenants, est **fastidieuse**.

En tant qu'utilisateurs réguliers de Discord, notre équipe s'est automatiquement tournée vers Discord en guise de support de communication principal pour ce projet.

Recueil du besoin Client

Pour recueillir les besoins du client, notre équipe a d'abord discuté avec les différents membres des équipes Simplon. Suite à ces discussions, nous avons dressé une liste des problèmes les plus fréquemment rencontrés par les membres des équipes Simplon.

Problématiques	Problèmes
	Multiplicité des outils
Utilisation de Discord inappropriée dans un cadre professionnel	Mauvaise identification lors des interactions
	Actions répétitives du personnel
	Manque d'ergonomie de Discord
	Multiplicité des messages inutiles (FLOOD)
Communication inefficace	Perte de contact avec les anciens apprenants
	Pas de mentorat
	Problème de sourcing

1. Multiplicité des supports de communication : **Concernant ce problème, notre équipe a constaté qu'il existait une multitude de serveurs Discord utilisés par les membres de Simplon pour communiquer avec différentes entités.** Par exemple, un formateur peut avoir accès à 12 serveurs Discord (sur une période d'un an et demi) réservés aux promotions auxquelles il participe, sans compter les autres serveurs nécessaires pour communiquer avec les équipes Simplon.
2. Mauvaise identification lors des interactions : **Discord utilise nativement un système de pseudonyme, ce qui n'est pas adapté à une utilisation professionnelle où l'identification claire des utilisateurs est nécessaire.** Cela peut entraîner des complications pour identifier les personnes avec lesquelles nous souhaitons communiquer via Discord.
3. Actions répétitives du personnel : **Il a également été observé que les membres des équipes Simplon, en particulier les formateurs, effectuent de nombreuses actions répétitives.** Par exemple, lors du démarrage d'une nouvelle promotion chez Simplon HDF, un formateur doit effectuer les étapes suivantes pour permettre aux apprenants de communiquer entre eux (Annexe 1) ces étapes doivent être répétées à chaque nouvelle promotion.
4. Manque d'ergonomie de Discord : **Les membres des équipes ont également exprimé leur malaise quant à l'utilisation de Discord, notamment dans un contexte professionnel.** Ils ont relevé des lacunes ergonomiques qui rendent l'utilisation de Discord moins confortable.
5. Multiplicité des messages inutiles : **Il a également été remarqué que les apprenants ont tendance à envoyer des messages de rappel aux membres des équipes dans des situations dans lesquelles ils n'ont pas reçu de réponse à leurs sollicitations.** Bien que cela puisse sembler anodin, cela génère des messages inutiles et **constitue une source de distraction pour les membres des équipes.**
6. Perte de contact avec les anciens apprenants : **Certains membres des équipes Simplon nous ont fait part de la difficulté à maintenir le contact avec les anciens apprenants.** En effet, dans la configuration actuelle de l'organisme de formation, il n'existe pas de solution concrète permettant de rester en contact avec les anciens apprenants. Une fois leur formation terminée, ils quittent le serveur Discord dédié à leur promotion, ce qui entraîne une perte de contact.
7. Problème de sourcing : **Les membres ont également signalé des problèmes de sourcing.** Selon eux, il est relativement difficile de trouver des profils correspondant aux critères des promotions et de les intégrer.

L'équipe en charge du projet

BOURREZ Bastien alias le pro Discord

PHILIPPE Nelson alias l'aventurier

LEROY Cédric alias l'encyclopédie / rédacteur en chef

GAMACHE Benjamin votre rédacteur

Proposition de solution :

Notre équipe s'est interrogée sur la meilleure solution entre l'utilisation de **Discord** et le développement d'une application spécifique pour **Simplon**.

Utiliser **Discord** présente les avantages suivants :

1. Il est déjà utilisé par la majorité des membres des équipes et des apprenants.
2. Il est gratuit.
3. Il constitue déjà un support de communication stable
4. Il est flexible et adapté à tout type d'équipe de développement.

En revanche, développer une application spécifique représente les contraintes suivantes :

1. Ajouter un nouveau support de communication pour tous les utilisateurs.
2. **Engendre des coûts** en termes de graphisme.
3. Nécessite un hébergement de serveur plus important.
4. Requiert des **connaissances avancées** dans certains langages de programmation.

C'est pourquoi nous avons choisi d'utiliser **Discord** plutôt que de développer une application à partir de zéro.

Gestion du projet

Avant de démarrer un projet, il est important de se poser les bonnes questions, telles que la façon dont le projet devra être géré, quels objectifs le projet se fixe, et bien d'autres questions auxquelles mon équipe a tenté d'apporter les réponses les plus adaptées possibles.

Méthodologie de travail Agile

La **méthodologie Agile** est une méthodologie **itérative** permettant le découpage des différents cycles de développement d'un projet en "**sprint**", chaque sprint à une durée moyenne d'1 à 4 semaines, à l'issue de ce sprint, une version fonctionnelle peut être présentée au client.

De plus, une équipe Agile interagira de façon **auto-gérée**, au sein d'une équipe Agile, la **collaboration** se fait entre tous les membres de façon étroite, le **partage des responsabilités** est aussi un des aspects clés de cette méthodologie, cette proximité au sein de l'équipe Agile permet une **communication continue**.

Agile apporte une livraison continue / itérative du produit au client, à chaque fin de sprint, le client peut récupérer une version du produit fonctionnelle et en tirer parti.

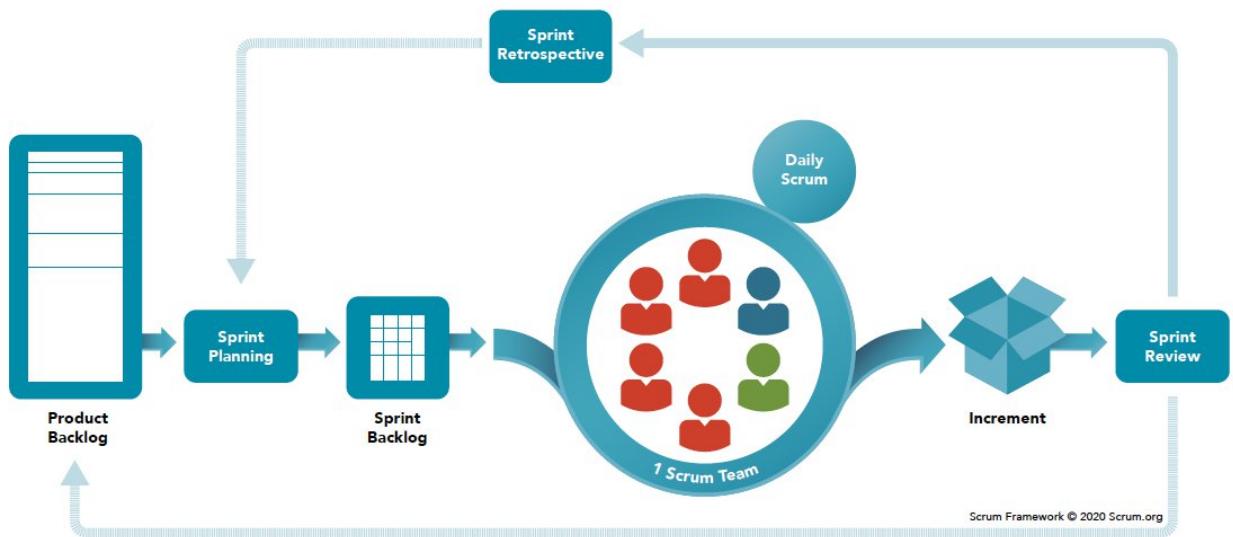
Les tâches d'un projet sont priorisées au sein d'un **Product Backlog** selon la valeur ajoutée pour le client, il est nécessaire d'établir un Product Backlog en priorisant les tâches à effectuer dès le début afin de permettre au client de tirer avantage au plus vite du produit.

Aussi, Agile considère que les besoins et les exigences du projet évoluent tout au long de la vie de ce dernier, avec Agile, l'**équipe peut s'adapter** à ces changements et effectuer des ajustements en fonction des nouvelles informations fournies par le client après qu'il ait reçu la dernière version fonctionnelle du produit.

Dû à ces délimitations des cycles de développement flexible, Agile permet à l'équipe de développement de s'améliorer de façon continue en mettant en place des **rituels** tels que les **Sprints Reviews** et les **Sprints Retrospectives** qui ont lieu à la fin de chaque sprint, permettant aux différents membres de l'équipe de discuter des succès et des difficultés rencontrées lors du dernier sprint.

De plus, Agile inclut une **métrique : La vitesse** qui est recalculée à chaque fin de sprints, de cette façon les membres de l'équipe sont capables d'évaluer la quantité de travail achevée lors du sprint.

Le framework SCRUM



SCRUM ou littéralement **Mêlée** est un framework permettant aux équipes qui le mettent en place d'adopter une approche Agile et d'apporter de la flexibilité au projet.

Le framework SCRUM permet à chaque membre d'une équipe d'avoir un rôle à jouer dans la prise de décision concernant un projet.

La taille d'une équipe SCRUM peut varier, mais il est important de savoir que SCRUM est plus adaptée à des équipes à effectif réduit, en effet, à partir d'un certain nombre de membre, peu importe la méthodologie utilisée au sein d'une équipe, **la communication devient de plus en plus complexe**.

Les quatre grandes valeur de SCRUM

1. **Individus & Interactions > Processus & Outils** : Une équipe engagée crée un produit de valeur estimable
2. **Un produit fonctionnel > Une documentation exhaustive** : La documentation d'un produit est essentielle, cependant une équipe doit se concentrer sur les fonctionnalités d'un produit plutôt que sur la rédaction d'une documentation.
3. **Collaboration avec le client > Négociation contractuelle** : Il est essentiel de placer le client au sein du processus de développement du produit, les retours du client sur le produit sont une source d'information précieuse, une bonne collaboration entre le client et l'équipe SCRUM permet l'élaboration d'un produit fini le plus fidèle aux exigences du client.
4. **Adaptation au changement > Suivi d'un plan** : Dans SCRUM l'apport de modifications aux projets n'est pas un obstacle, étant donné que le produit se construit de façon itérative, en plaçant le client au centre du processus de développement, il est plus facile pour les équipes d'effectuer des ajustements concernant le projet tout au long de ces cycles de développement, plutôt que d'avoir à revoir complètement certains aspects du produit à la fin du développement de ce dernier.

Les rôles dans SCRUM

Le Product Owner

Le Product Owner est le responsable de la **liaison entre le client et l'équipe de développement**, son rôle est de comprendre le client et de retranscrire sa demande aux équipes de développement, **les tâches** du Product Owner sont les suivantes :

- De créer et gérer un **Product Backlog** en prenant en compte les **tâches à prioriser**.
- **Collaborer avec les équipes** afin de s'assurer que tous les membres comprennent le Product Backlog du projet.
- Fournir à l'équipe les prochaines fonctionnalités à **livrer**.
- Décider de la **durée des sprints**.

Le SCRUM Master

Le SCRUM Master est le membre **responsable de l'application de SCRUM** au sein de son équipe, il analyse perpétuellement les **optimisations possibles** à apporter en termes d'application SCRUM au sein de son équipe, **il s'assure de comprendre ce que l'équipe doit réaliser** et aide celle-ci à **optimiser la transparence du projet** et le **flux de livraison** de ce dernier, il gère les **ressources** humaines et logistiques d'une équipe pour planifier les sprints.

Il organise les cérémonies SCRUM.

L'équipe de développement

L'équipe de développement est constituée de **5 à 7 développeurs** la plupart du temps, ces développeurs travaillent en **étroite collaboration**, ce qui permet d'aider chaque membre de l'équipe de développement en difficulté afin de ne pas se retrouver avec un **retard dans la livraison** du produit à la fin du sprint.

Les rituels

Sprint planning

Le Sprint Planning est organisé par le **SCRUM Master** afin d'établir la liste des **User Stories** à mettre dans le prochain sprint, l'équipe s'accorde à dire que toutes les User Stories sont en mesures **d'être complétées** à l'issue du sprint (notamment en se basant sur la **vélocité** calculée à la fin du dernier sprint)

Dailys SCRUM

Le Dailys Scrum a lieu **tous les jours**, il s'agit d'une réunion d'une durée moyenne de **15 minutes** qui permet à l'équipe de se remettre en phase avec les objectifs du Sprint en cours.

Les **questions indicatives à se poser** pour les membres de l'équipe afin d'établir si elles sont en phase avec les objectifs sont :

- Qu'est-ce que j'ai fait hier ?
- Qu'est-ce que je dois faire aujourd'hui ?
- Quels obstacles j'ai et je pourrai rencontrer ?

Sprint Review

La Sprint Review permet à l'équipe de présenter **le travail réalisé** lors du Sprint aux différentes parties prenantes, lors de cette réunion, le Product Owner peut décider de livrer ou non le travail réalisé.

Pour un Sprint d'une durée d'**1 semaine envisagez 1 heure** de Sprint Review

Sprint Retrospective

Cette réunion permet aux membres de l'équipe de **revenir sur le déroulement du Sprint**, ce qui a fonctionné et ce qui n'a pas fonctionné, l'idée de cette réunion est de **trouver des pistes d'améliorations** pour les prochains Sprints de l'équipe, et non **pas de mettre en lumière les échecs**.

Définition des rôles SCRUM au sein de l'équipe

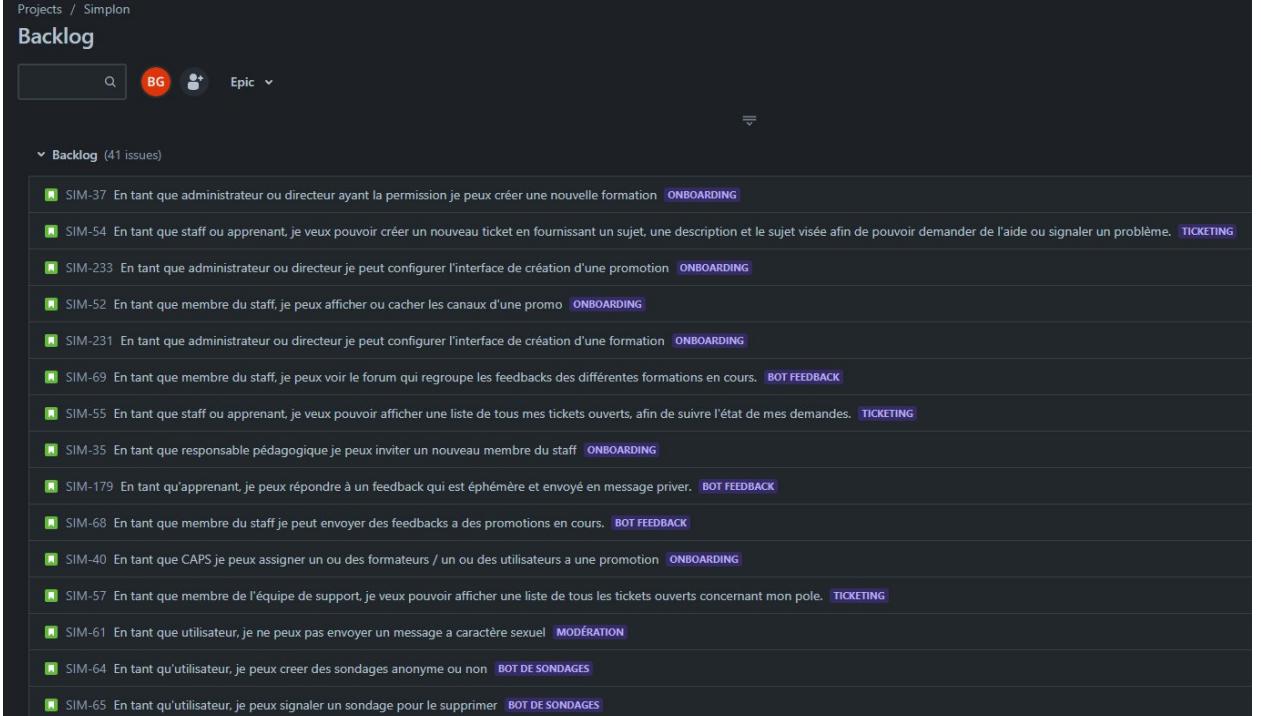
GAMACHE Benjamin occupe le rôle de **Product Owner**, qui consiste notamment à prendre en charge les interactions entre l'équipe en charge du projet et le client.

BOUREZ Bastien occupe le rôle de **Scrum Master**, qui consiste à diriger les membres de l'équipe en charge du projet et à assigner des tâches à chacun.

PHILIPPE Nelson et **LEROY Cédric** occupent tous deux le rôle de **Développeur**, qui consiste à développer les différentes solutions élaborées par tous les membres de l'équipe en charge du projet et validées par le client.

Jira

Pour nous **organiser** sur ce projet, mon équipe et moi-même avons choisi **Jira**, qui est une plateforme permettant d'organiser les différentes tâches d'un projet en les découplant en **Epic**, puis en **User Story**, et enfin en tâches.



The screenshot shows the Jira interface for the 'Simplon' project, specifically the 'Backlog' section. At the top, there are filters for 'BG' (Blocked), 'Epic', and a search bar. Below the header, a section titled 'Backlog (41 issues)' is expanded, showing a list of user stories. Each story is represented by a card with a green icon, a title, a description, and a category tag at the end. The categories visible include 'ONBOARDING', 'TICKETING', 'BOT FEEDBACK', 'MODERATION', and 'BOT DE SONDAGES'. The stories range from SIM-37 to SIM-65, detailing various permissions and features for staff and administrators.

Issue ID	Description	Category
SIM-37	En tant que administrateur ou directeur ayant la permission je peux créer une nouvelle formation	ONBOARDING
SIM-54	En tant que staff ou apprenant, je veux pouvoir créer un nouveau ticket en fournissant un sujet, une description et le sujet visé afin de pouvoir demander de l'aide ou signaler un problème.	TICKETING
SIM-233	En tant que administrateur ou directeur je peut configurer l'interface de création d'une promotion	ONBOARDING
SIM-52	En tant que membre du staff, je peux afficher ou cacher les canaux d'une promo	ONBOARDING
SIM-231	En tant que administrateur ou directeur je peut configurer l'interface de création d'une formation	ONBOARDING
SIM-69	En tant que membre du staff, je peux voir le forum qui regroupe les feedbacks des différentes formations en cours.	BOT FEEDBACK
SIM-55	En tant que staff ou apprenant, je veux pouvoir afficher une liste de tous mes tickets ouverts, afin de suivre l'état de mes demandes.	TICKETING
SIM-35	En tant que responsable pédagogique je peux inviter un nouveau membre du staff	ONBOARDING
SIM-179	En tant qu'apprenant, je peux répondre à un feedback qui est éphémère et envoyé en message privé.	BOT FEEDBACK
SIM-68	En tant que membre du staff je peut envoyer des feedbacks à des promotions en cours.	BOT FEEDBACK
SIM-40	En tant que CAPS je peux assigner un ou des formateurs / un ou des utilisateurs à une promotion	ONBOARDING
SIM-57	En tant que membre de l'équipe de support, je veux pouvoir afficher une liste de tous les tickets ouverts concernant mon rôle.	TICKETING
SIM-61	En tant que utilisateur, je ne peux pas envoyer un message à caractère sexuel	MODÉRATION
SIM-64	En tant qu'utilisateur, je peux créer des sondages anonyme ou non	BOT DE SONDAGES
SIM-65	En tant qu'utilisateur, je peux signaler un sondage pour le supprimer	BOT DE SONDAGES

Le Product Backlog du projet sous Jira

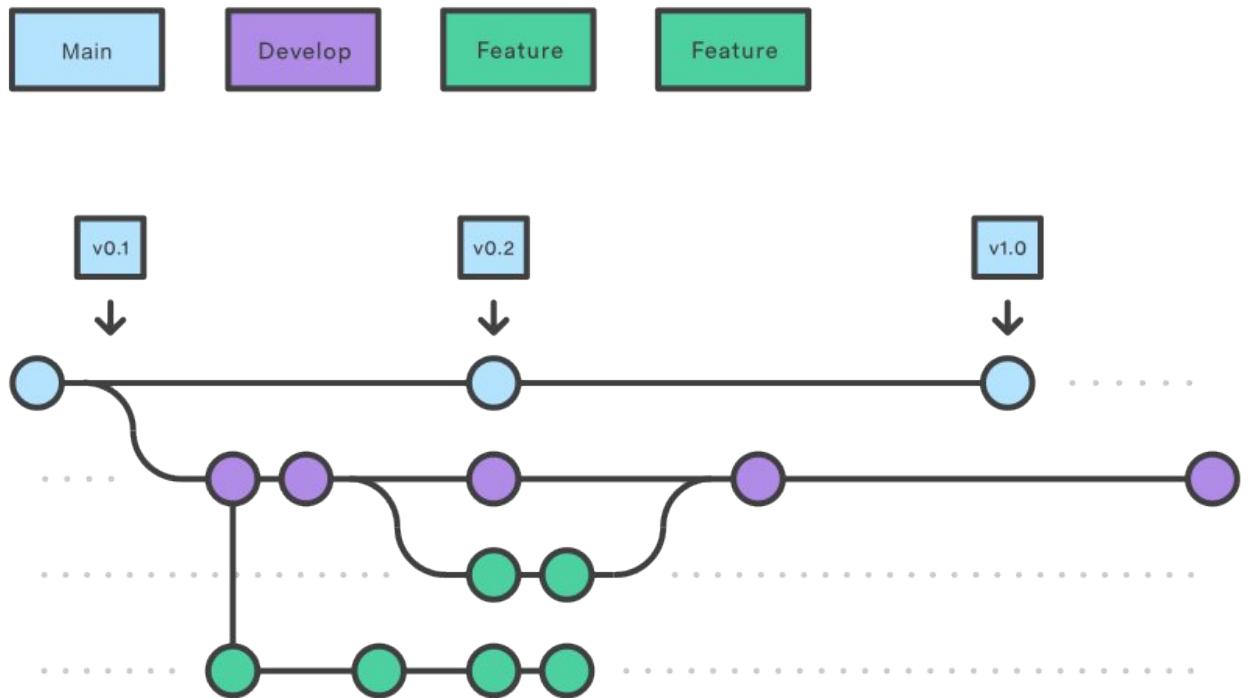
Git

Git est un outil permettant de **versionner son code**. Avec Git, il est possible d'écrire du code et de le séparer sur des **branches**, de revenir à des versions antérieures du code. Git permet aussi d'envoyer son code sur des **plateformes en ligne** comme **BitBucket**, **GitLab**, **GitHub**, ou de l'envoyer sur des plateformes de **self-hosting** pour héberger du code en ligne.

Git est **essentiel** à chaque équipe de développement afin de **collaborer** sur un projet, chaque membre d'une équipe de développement utilise Git afin d'apporter des modifications au code d'un projet.

GitFlow

GitFlow est un **workflow standardisé**, il permet à tous les membres d'une équipe d'adopter le même workflow.



Avec GitFlow, il est facile et rapide de définir un Workflow pour une équipe de développement, c'est pourquoi notre équipe s'est tournée vers GitFlow pour **collaborer**

GitHub

GitHub est une plateforme en ligne qui sert d'**outil collaboratif de versionning de code**. Elle permet d'**héberger du code** en ligne et de retrouver différentes versions du code.

GitHub permet donc de **partager** son code avec une **équipe restreinte** (comme c'est le cas ici) ou de le partager de **façon publique**. Nous avons donc utilisé GitHub afin de travailler en équipe sur ce projet.

Architecture du projet

Mono-Repo VS Multi Repo

Pour choisir la meilleure façon d'**organiser le projet** sur GitHub, nous nous sommes posés des questions sur la **solution optimale** entre le **Mono Repository** et le **Multi Repository**.

Le **Mono Repository** :

- **Centralise** toute la **conception** de l'application.
- **Centralise** tout le code de l'application.
- Mise en place de pipelines de déploiement (**CI/CD**) plus **complexes**.
- **Centralisation** de toutes les **dépendances** de l'application, qu'elles soient utiles à un module ou non.

Le Multi Repository :

- Décentralise la conception de l'application.
- Décentralise les sources de l'application.
- Mise en place de pipelines de déploiement (CI/CD) plus légères.
- Répartition des dépendances de chaque partie de l'application dans des dépôts séparés.

Nous avons adopté une **approche mixte** en hébergeant notre application avec **3 dépôts** :

- **1 dépôt pour l'API et la base de données.**
- **1 dépôt pour l'application Web (front-end).**
- **1 dépôt pour tous les bots** constituant l'application.

GitFlow Vs Fork

Bien qu'il était initialement prévu de travailler en forkant le dépôt du projet, notre équipe est revenue sur sa décision quant à la **méthodologie à adopter** pour travailler avec GitHub, car elle ne convenait pas à tous les membres de l'équipe.

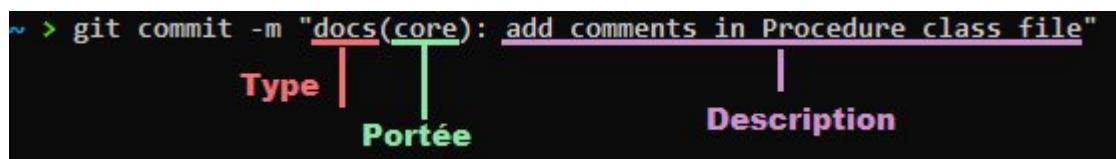
De plus, avec une **petite équipe de 4 membres**, le **fork n'est pas une obligation**. Nous pouvons tout à fait utiliser GitFlow en raison de la taille de l'équipe, créer des branches et les envoyer sur le dépôt distant.

Le fork est plus apprécié dans les projets Open-Source où la taille des équipes de développement n'est pas fixe, et dans lesquels les contributions sont émises par des collaborateurs qui ne sont pas à l'initiative du projet. Le fork permet une meilleure organisation dans les équipes de grande taille.

Convention de formatage des commits

Pour réaliser nos commits, notre équipe a adopté la convention "**Angular Style Commits**" qui permet de définir :

- Un type.
- Une portée (optionnelle).
- Une description.
- Un corps de commit (optionnel).
- Un footer de commit (optionnel).



Exemple de l'Angular Style Commit

Critères des Pull Requests

Pour qu'une Pull Request soit acceptée, nous avons mis en place des critères sur la branche de développement (develop). Ainsi, **avant chaque fusion de branche** vers develop, les modifications de la branche **doivent être approuvées** par un autre membre de l'équipe.

Critères d'acceptation des PR

- Que les commits soient **rédigés en anglais**.
- **Intitulés** des commits **clairs**
- Validation par un membre de l'équipe.

Qualité de code : ESLint

Afin d'assurer une qualité de code correcte, notre équipe a choisi ESLint en guise de **Linter**, car il prend en charge **TypeScript** et permet de vérifier certains critères concernant le code **de façon statique**, il notifie les développeurs lorsque :

- Une variable n'existe pas
- Une variable ou un paramètre n'existe pas
- Les doubles déclarations de variables
- Les doubles implémentation pour une fonction
- **La mauvaise organisation du code**
- Les erreurs de syntaxe

De plus ESLint peut être configuré selon certaines conventions, notre équipe s'est orientée vers la convention Airbnb.

La convention Airbnb

Voici quelques spécifications de la convention **ESLint Airbnb**

- Utilisez des guillemets simples pour les chaînes de caractères, sauf lorsque vous devez utiliser des guillemets simples à l'intérieur de la chaîne.
- Utilisez des parenthèses autour des arguments de fonction, même s'ils ne sont pas nécessaires dans certains cas.
- Utilisez l'indentation avec des espaces et configurez-la à 2 espaces.
- Déclarez les variables avec `const` ou `let` selon leur ré-assignabilité. Évitez d'utiliser `var`.
- Utilisez des accolades pour les blocs de code, même s'ils ne sont pas strictement nécessaires.

Ces points font partie des **recommandations générales** de la configuration ESLint Airbnb.

Spécifications Fonctionnelles

Règles de Gestion du Bot

Les règles de gestion sont une suite de règles que les membres d'une équipe définissent pour répartir les possibilités et les devoirs des entités constituant un système. Suite au recueil des besoins du client, notre équipe a établi les règles de gestion du projet.

- Le bot doit disposer d'un système de configuration.
- Le bot doit avoir une commande de génération d'embed (dans un canal) pour l'ajout des membres du staff.
- L'embed doit disposer d'une liste déroulante permettant de sélectionner le rôle et de générer un lien d'invitation.
- Le bot doit avoir une commande de génération d'embed (dans un canal) pour la création d'un nouveau type de formation.
- L'embed doit disposer d'un bouton permettant d'envoyer une demande de nom pour le nouveau type de formation.
- Le bot doit avoir une commande de génération d'embed (dans un canal) pour l'ajout de formations.
- L'embed doit disposer d'une liste déroulante permettant la sélection du type de formation.
- Une demande doit être envoyée pour demander de compléter le nom de la formation.
- Un nouvel embed doit être envoyé avec un bouton permettant de créer une nouvelle formation.
- Le bot doit envoyer un message demandant la date de début et de fin de la formation.
- Le bot doit avoir une commande de génération d'embed (dans un canal) pour l'ajout d'apprenants à une formation.
- L'embed doit disposer d'une liste déroulante permettant de générer un lien d'invitation pour un nouvel apprenant, pour une formation spécifique.
- Le lien d'invitation doit être valide pour une seule personne.
- Le bot doit avoir une commande de génération d'embed (dans un canal) pour l'ajout de nouveaux utilisateurs déjà présents sur le serveur Discord, à une formation.
- L'embed doit disposer d'une liste déroulante permettant de sélectionner une formation spécifique.
- Lors de la sélection de la formation, un nouvel embed doit être envoyé, avec un bouton permettant d'afficher un formulaire d'ajout d'utilisateur.
- Le bot doit avoir une commande de génération d'embed (dans un canal) pour l'ajout ou la modification de modèles de catégories de formation.
- Une catégorie de formation est un ensemble de canaux dédiés à une formation.
- Lors de la création d'une formation, le bot doit générer un embed de configuration dans un canal propre à sa catégorie.
- Le lien d'invitation généré par le bot ne doit fonctionner que pour une personne.
- Le lien d'invitation doit être temporaire.
- Le bot ne doit pas pouvoir créer deux fois le même embed de configuration.

- Le bot doit pouvoir détecter si un embed a été supprimé pour permettre la création d'un nouveau.
- L'administrateur peut supprimer un embed.
- Lors de l'ajout d'un utilisateur à une formation, le bot doit envoyer une demande de vérification (dans un canal dédié à cette formation).
- Le bot doit exiger une identification lors de l'arrivée d'un nouvel apprenant ou d'un nouveau membre du staff.
- Lors de l'arrivée d'un nouvel apprenant, le bot doit envoyer un message de demande de vérification (dans un canal dédié à cette formation).
- Lors de l'arrivée d'un nouveau membre du staff, le bot doit envoyer un message de demande de vérification (dans un canal dédié au staff).
- Une fois l'identité vérifiée, le rôle doit être attribué par le bot à l'utilisateur du lien.
- Le bot doit mettre en place un embed (dans un canal) permettant de sélectionner les formations visibles pour le staff.

RBAC (Role Based Access Control)

Le RBAC est un modèle permettant de définir les autorisations des utilisateurs dans un système, ce modèle permet de classer les **utilisateurs du système** dans des **rôles** et ainsi d'attribuer des permissions à ces rôles, de cette façon les utilisateurs ont les permissions strictement nécessaires aux actions que ces derniers ont à effectuer sur le système.

Notre RBAC étant trop massif, il m'est impossible de le présenter ici.

User Stories les plus représentatives

Voici une liste non exhaustive des User Stories représentatives du projet :

- En tant qu'Administrateur, Directeur ou CAP, je peux créer une nouvelle formation afin de permettre la création de promotion
- En tant que CAP, je peux démarrer une nouvelle promotion afin de permettre l'accueil des apprenants
- En tant que membre du Staff, je peux sélectionner les promotions que je souhaite voir afin de ne pas polluer mon espace de travail
- En tant que CAP, je peux assigner ou dés-assigner un Formateur à une promotion afin de gérer la promotion et les éventuels changements
- En tant que nouvel utilisateur, je dois m'identifier afin de rejoindre mon espace de promotion
- En tant que membre du Staff ou Formateur, je peux accepter la demande d'identification d'un utilisateur afin de lui assigner sa promotion ou son rôle de Staff

Spécifications Techniques

La base de donnée : PostgreSQL

Dans le cadre de ce projet, il nous fallait une base de donnée afin de permettre la persistance des données, nous avons donc réalisé des études comparatives entre différents types de SGBD et avons sélectionné PostgreSQL

PG est un **SGBDRO** (**Système de Gestion de Base de Donnée Relationnel et Objet**) ce qui signifie que PostgreSQL prend en charge les bases de données **relationnelles** ainsi que des bases de données non relationnelles (**NoSQL**)

De plus, PG **robuste**, il indique des **performances satisfaisantes** en termes d'accès aux ressources de l'application.

PG permet une approche **scalable** de la base de donnée et ainsi du projet.

De plus PostgreSQL met en place des systèmes de sécurité interne et permet d'appliquer des **sécurités supplémentaires** sur certaines tables selon certains critères (Row-Level Security).

PostgreSQL est open-source, ce qui en fait un SGBDRO flexible et adaptable aux besoins rencontrés pour le projet.

C'est pour toutes ces raisons que PostgreSQL a été sélectionné comme solution de persistance de nos données.

Choix du langage

Dans une équipe réduite comme la nôtre, il nous fallait un langage relativement simple à prendre en main par tous les membres de l'équipe, assez léger pour être mis en place côté serveur sans nécessiter de coûts de machine trop élevés.

TypeScript

TypeScript est un langage **plus robuste que JavaScript**. Il s'agit d'un "super-set" de JavaScript qui introduit de nombreux concepts que JavaScript vanilla n'implémente pas tels que la **POO** (Programmation Orientée Objet).

Dans un projet de cette envergure, il est absolument nécessaire de pouvoir utiliser la Programmation Orientée Objet.

Même si un bot peut sembler très basique de l'extérieur, à l'intérieur, c'est une autre affaire. Ce projet aurait pu être réalisé dans les délais avec du JavaScript vanilla, cependant, la **maintenabilité du code aurait été quasiment impossible** pour une équipe devant reprendre le projet.

TypeScript est un langage dit de "**Typage Fort**" qui ne permet pas l'assignation de valeur aux variables de façon dynamique, chaque variable à un type et il n'est pas possible de stocker n'importe quel type de donnée dans n'importe quelle variable.

De plus, par le biais de son **transpileur**, Typescript apporte **une gestion des erreurs** et un affichage de ces dernières **bien moins verbeux** et plus clair, ce qui permet de comprendre plus rapidement les erreurs de transpilation et ainsi de les **régler plus vite**.

C'est pour ces raisons que **TypeScript a été préféré à JavaScript**.

En termes de **pérennité du projet**, JavaScript n'était pas le choix adapté.

Architecture : API

Aussi, puisque nous avions besoin de **stocker des données**, nous avions besoin de **contrôler l'accès à ces données**. C'est pourquoi nous avons décidé de mettre en place une API afin de contrôler l'accès aux ressources de l'application.

Stateless (Sans état)

Notre API n'avait pas besoin d'être **stateful**. Nous n'avions pas besoin que les requêtes de l'API s'adaptent en fonction du **contexte** dans lequel elles étaient faites. C'est pourquoi nous avons décidé de **mettre en place une API stateless** plutôt qu'une API stateful.

RESTful API

Une API RESTful est une API qui respecte 5 principes fondamentaux de l'architecture REST :

- Architecture **client <-> serveur**
- Requête sans conservation d'état (stateless)
- Cacheable : Permettant de **mettre en cache** les ressources pour **optimiser les performances**
- Interface uniforme : Une interface uniforme qui permet la communication entre les composants du système (les composants peuvent interagir les uns avec les autres)
- Construction en couches : **Permet de hiérarchiser les différents composants de l'API**

Nous avons également choisi une architecture RESTful pour notre API, car il fallait qu'elle soit accessible par un large éventail de clients, et donc qu'elle fournisse des réponses structurées de façon standardisée pour chaque client souhaitant la consommer.

La mise en place d'une architecture RESTful semblait être la solution la plus adaptée dans le cadre de ce projet.

Choix du Framework Back End (Coté API)

Critères	Koa	Fastify	Nest	Adonis
Personnalisation	0	0	2	0
Rapidité	1	2	2	1
Popularité	1	1	2	0
Maturité	2	1	0	1
Releases	0	2	1	1
Bonnes pratiques	0	0	2	0
Stars Github	1	1	2	0
Equipe Développement	0	2	1	1
Communauté Github	1	1	1	0
Communauté StackOverflow	1	1	2	0
Sécurisé By Design	0	0	2	2
Documentation	0	1	2	2
Magie	0	0	0	0
Mariage librairies	2	2	2	0
Prise Politique	2	2	0	2
License	MIT	MIT	MIT	MIT
Total	11	16	19	8

Choix du Framework Back End (Coté Back)

Notre équipe a décidé d'utiliser le framework DiscordJS pour développer les bots Discord. DiscordJS est un framework très complet, maintenu par une grande communauté de développeurs et surtout, le seul framework parfaitement compatible avec TypeScript. C'est pourquoi, pour le choix du framework Back-End concernant les bots, notre équipe s'est tournée vers DiscordJS.

Choix du Framework Front End & ORM

0 = Moins Intéressant 2 = Plus intéressant				0 = Moins Interestant 2 = Plus Intéressant		
Critères	ReactJS	Angular	Vue.js	Critères	TypeORM	Prisma
Personnalisation	2	0	1	Rapidité	2	1
Rapidité	1	1	2	Popularité	2	1
Popularité	2	1	0	Maturité	2	1
Maturité	1	1	1	Releases	1	2
Releases	0	2	1	Bonnes pratiques	1	2
Bonnes pratiques	1	2	2	Stars Github	2	1
Stars Github	2	1	2	Equipe Développement	1	2
Equipe Développement	2	1	0	Communauté Github	2	1
Communauté Github	1	1	0	Dernier commit	1	2
Communauté StackOverflow	1	2	0	Sonsors	2	1
Qualité Documentation	1	2	1	Communauté StackOverflow	2	2
Côté Magique	2	2	2	Documentation	1	2
Mariage librairies	0	0	0	Prise Politique	1	0
Prise Politique	0	2	2	License	Apache2	MIT
License	MIT	MIT	MIT	Total	20	18
Total	16	18	14			

Si l'on s'attarde sur le benchmark de l'ORM, on peut voir que TypeORM semblait être préféré à la place de Prisma. Cependant, notre équipe a dû choisir la solution la plus adaptée au projet, et il est apparu que, pour une équipe aussi réduite que la nôtre et dans les délais imposés, TypeORM semblait être trop lourd pour être intégré de manière optimisée au projet. C'est pourquoi, malgré l'analyse des caractéristiques de ces ORM, notre équipe a préféré utiliser Prisma plutôt

Pourquoi utiliser un ORM ?

- **La sécurisation** : En utilisant un ORM nous déléguons le travail de sécurisation de l'accès aux données à l'ORM qui va se charger d'exécuter des requêtes préparées permettant ainsi l'échappement des caractères et rendant de fait impossible l'exploitation d'une **Injection SQL**
- **L'abstraction** : En effet, un ORM est très bon dans le domaine de l'abstraction, il permet d'exécuter des requêtes SQL préparées (et donc sécurisées) de façon plus confortable pour les développeurs qui n'ont pas à écrire manuellement les requêtes à la main et qui peuvent simplement utiliser l'ORM dans le langage dans lequel ils développent.

Tableau récapitulatif

Question	Réponse
Machine Learning	Non
Architecture	API
Asynchrone	Oui
SEO	Non
Rendu	CSR
Hébergement	Serveur Perso
Environnement	NodeJS
Langage	TypeScript
Framework Back End	NestJS
Framework Front End	Angular
BDD	PostgreSQL (SQL Relationnelle)
ORM	Prisma

Dans le cadre de ce projet, nous n'avions aucunement besoin de **Machine Learning**. Il nous fallait une application construite autour d'une architecture API, car cette architecture est plus modulable qu'une architecture monolithique. Le type de programmation qui a été sélectionné a été le développement asynchrone, car il ne fallait pas que notre API ait un fil d'exécution bloquant.

SEO signifie **Search Engine Optimization**, il s'agit d'une liste de critères à remplir lors de la création d'un site Web afin de permettre un **bon référencement** de celui-ci par les **moteurs de recherche**. En voici une liste non exhaustive :

- Rédaction de contenu de qualité pour les pages
- Bonne **structuration sémantique** des pages
- Utilisation de **mots-clés dans les métadonnées** des pages
- Présence de liens internes et externes significatifs dans les pages
- Rendu des pages en SSR (**Server-Side Rendering**)

Notre application Web, ayant pour but de gérer les différents bots depuis un **panel d'accès Web**, il était inutile de se préoccuper du référencement du site Web dans le cadre de ce projet. L'application Web n'a pas vocation à être ouverte au public et **est restreinte à une communauté Simplonniene**.

Le SSR est un **mode de rendu** des pages Web **côté serveur**, il est intéressant de mettre en place ce mode de rendu lorsque le référencement d'un site Web est nécessaire.

Cependant, lorsqu'il n'est pas intéressant d'améliorer son SEO, le SSR n'est pas recommandé car ce mode de rendu s'effectue côté serveur et donc consomme les ressources de ce dernier.

Le CSR est un mode de rendu des pages Web côté client, c'est-à-dire que le rendu des pages Web est effectué côté client et non côté serveur, ce qui **impacte directement les performances** d'affichage d'une page Web selon la machine du client voulant la consulter. Il n'était pas utile de mettre en place un mode de rendu SSR dans le cadre de ce projet, car cela aurait nui aux performances du serveur qui aurait alors dû gérer le rendu des pages.

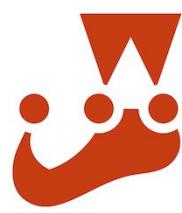
Il était plus intéressant de laisser le **rendu des pages Web au client** plutôt qu'au serveur dans le cadre de ce projet.

Note: Il n'existe plus de framework qui utilise purement l'un ou l'autre de ces modes de rendu, les frameworks utilisent un hybride en fonction des besoins de l'application.

Nous avons choisi d'héberger l'application sur des serveurs dédiés internes à Simplon HDF. Cependant, à l'heure actuelle, nous nous sommes trompés, car l'application est hébergée sur des **VPS**.

Testing

JEST



JEST est un framework Javascript permettant d'exécuter des **tests unitaires** pour des applications de tout type.

Avec JEST il est facile d'écrire des tests et de les exécuter.

TDD (Test Driven Dev.)

TDD signifie **Test Driven Développement**, ce qui se traduit par "Le développement piloté par les tests".

Le TDD est un concept selon lequel un développeur décrit un test pour une fonction, lance ce test qui échoue FORCEMENT.

Ensuite le développeur écrira la fonction à tester.

De cette façon, une fonction sera écrite en fonction du test qu'elle doit passer et le développeur.

Coverage

Le Coverage est une métrique permettant de connaître le taux de code exécuté lors des tests d'une application.

Cette metric permet de savoir à combien de pourcent le code est testé.

Le test unitaire

Le test unitaire est un test réalisé sur une partie non divisible du code, un test unitaire est écrit pour tester une fonction du code, on décrit la valeur attendue à la sortie de la fonction et on vérifie si la donnée est bien celle attendue, le cas échéant, le test est un succès sinon le test échoue et le développeur modifie sa fonction tant que le test ne se solde pas par un succès.

Le test d'intégration

Le test d'intégration permet de tester l'intégration des composants au sein du code déjà existant, il assure que les composants individuels s'intègrent correctement au sein du code déjà existant sans que ce dernier n'ait à subir de modification afin de permettre l'intégration du composant.

Stratégie de test du projet

Il était initialement prévu de développer en TDD et d'incorporer ces différents tests au sein de notre processus de développement, cependant notre organisation, ou désorganisation dans ce cas ne nous a pas permis de mettre en place de test unitaire de façon structurée.

Solutions Techniques

Sécurisation de l'application

Dans cette section, je parlerai de la **sécurité globale de l'application**, pour des raisons de délais **je n'ai pas eu le temps de mettre en place toutes les sécurités** dont je parle dans cette section, cependant je donnerai les solutions techniques que je mettrai en place à terme.

XSS (Cross Site Scripting)

La faille XSS est une faille majeure, l'utilisation de faille XSS consiste à injecter du code malveillant d'un site vers un autre afin de récupérer les informations de connexion de l'utilisateur par exemple.

L'utilisation de faille XSS peut être dévastatrice et de nombreuses pratiques frauduleuses peuvent en découler.

Prenons un exemple d'utilisation de faille XSS:

Un utilisateur nommé Gérard reçoit un mail provenant de sa banque lui indiquant un besoin de rectification,

Gérard clique alors sur ce lien, malheureusement, il est déjà trop tard.

L'émetteur de ce mail frauduleux avait glissé du code malveillant dans ce lien par le biais d'une faille XSS inconnue sur le site de la banque de Gérard, l'attaquant a alors été en mesure de faire exécuter du code malveillant au site de la banque de Gérard, l'attaquant a redirigé Gérard par le biais d'un lien récupérant ses cookies de connexion à sa banque, l'attaquant a récupéré le cookie de connexion de Gérard et a maintenant la possibilité de se connecter au compte de ce dernier.

La plupart du temps, les attaques XSS si elles sont bien faites ne sont même pas visibles par la victime.

Contre-mesure relative

Il est relativement compliqué de se protéger d'une faille XSS, cette faille est plutôt coriace à éliminer de par sa nature, il suffit d'une erreur à un endroit d'un des développeurs et pouf.

Je vais toute fois apporter des éléments de réponse à cette question.

NTUI (Never Trust User Input) :

L'un des principes fondamentaux en termes de développement est de ne **jamais** faire confiance aux utilisateurs.

Dans cette optique il est très important de considérer que l'utilisateur va chercher à casser votre système ou à s'y introduire, en partant de ce postulat, il faut toujours traiter avec le plus grand soin les données soumises par un utilisateur.

De ce concept découle une contre-mesure

Possibilité technique :

Côté Front-End :

L'implémentation d'une fonction du type

```
function sanitize(dataToSanitize: string): string {
  return dataToSanitize.replace( /(<[^>]+>)/ig, '' );

const html = '<strong>Hello</strong>';

console.log(html);
console.log(sanitize(html));
```

Dont le résultat est

```
> src@1.0.0 start
> tsc --project tsconfig.json && node build/index.js

<strong>Hello</strong>
Hello
```

De cette façon le client ne pourrait pas envoyer code HTML au Back-End

Côté Back End :

Implémenter la sanitisation côté Back-End en passant par le DTO

```
@ApiProperty()
@IsNotEmpty({ message: 'This user need a name' })
@IsString()
@Transform((value: any) => Utils.sanitize(value))
name: string;
```

On voit d'ailleurs que le DTO permet aussi gérer le type de donnée attendue ce qui, fait partie de la sanitisation.

CSRF

L'attaque par CSRF ou Forge de Requête par site Interposés est un type d'attaque semblable à une attaque par XSS, le but de l'attaque par CSRF est en fait d'exécuter des actions sur un site Web légitime à l'insu de l'utilisateur, poster un message sur un forum par exemple, il est aussi possible de récupérer les informations de connexion de l'utilisateur par le biais d'attaque CSRF.

DDos (Distributed Denial of Service / Attaque par déni de service)

Une attaque DDos est une attaque de grande envergure, elle consiste à bombarder un système exposé au réseau de requête massive de façon à ce que le composant logiciel entraîne la chute de l'infrastructure système.

Les attaques DDos n'ont pas de réelle solution en termes de développement, il est possible d'optimiser les sécurités au niveau des composants logiciels afin de réduire la charge de ces derniers sur les infrastructures, cependant dans 99.9% des cas une attaque DDos se solde par un succès si des mesures de protection au niveau de la couche réseau ne sont pas prises.

Possible contre-mesure :

Une alternative à considérer en termes de sécurité de l'API est le Request Rate Limit ou Limite des Requêtes.

Il est possible de définir un nombre limité de requêtes à l'API dans un temps donné, ce qui constitue une sécurité en termes de structure système, sur un serveur de petite composition, il est intéressant d'évaluer cette possibilité afin d'empêcher l'API d'être surchargé et de faire tomber toute l'infrastructure réseau, cependant cette solution n'est pas une solution absolue en cas d'attaque DDos (Distributed Denial of Service).

Mise en place

```
@Module({
  imports: [
    ThrottlerModule.forRoot({
      ttl: 60,
      limit: 10,
    }),
  ],
})
export class AppModule {}
```

Importation du module « Throttler » de NestJS dans le module globale de l'API

@SkipThrottle(false)

Ajout du décorateur pour activer le RRL sur les routes voulu de l'API

Moindre privilège

Le principe de moindre priviléges consiste à restreindre les permissions d'un composant système aux permissions strictement nécessaires à son bon fonctionnement, ce qui signifie qu'un composant n'aura pas de permissions supérieures à ses fonctions.

Ce qui permet d'empêcher une potentielle exploitation des composants système mis en place afin de réaliser des actions malveillantes au sein d'un système.

Défense en profondeur

La défense en profondeur est un concept selon lequel il prévaut d'aborder un aspect de sécurité dans chaque composant d'un système plutôt que de concentrer la sécurité en un seul et même point.

En sécurisant chaque composant individuellement, ces composants participent à la sécurité générale du système.

Réduction de la surface d'attaque



Exposition d'un système

La réduction de la surface d'attaque est un concept selon lequel il n'est nécessaire d'exposer les composants d'un système uniquement et uniquement si cela est obligatoire.

Qu'il s'agisse d'exposer des composants ou des sous-services de ses composants il est important de se poser la question de savoir si un choix a été fait de façon réfléchie et de comprendre les causes qui mènent à l'exposition des composants.

Par exemple, dans une architecture comme la notre nous avons, 1 BackEnd API, 1 FrontEnd, 1 Base de Donnée

Dans ce cas, nous savons que nous allons exposer les ports :

- 80 : **HTTP**
- 443 : **HTTPS**
- 22 : **SSH**
- 3000 : **Port de connexion de l'API**

Et exposer notre machine sur le réseau uniquement par ces ports, nous avons réduis la surface

CORS

Cross Origin Ressources Sharing (CORS) ou Partage de Ressources Inter-Origin est une sécurité permettant d'échanger des ressources entre différentes origines de cette façon, il sera possible de récupérer des informations sur des sites partenaires, cependant grâce à CORS il est possible d'établir une liste de domaine avec lesquels échanger des informations.

Pour qu'un échange de donnée entre origines puisse avoir lieu avec CORS il est nécessaire que les 2 origines acceptent l'échange l'un avec l'autre sans quoi l'échange de donnée n'aura pas lieu.

Politique des mots de passe

Une politique de mot de passe est un ensemble de recommandations à suivre afin d'établir une certaine sécurité concernant les mots de passe possiblement entrés par les utilisateurs.

Dans le cadre de cette stratégie, la politique des mots de passes sera la suivante :

- Une longueur comprise entre 8 et 80 caractères.
- Une composition (A-z, 0-9, caractères spéciaux sans exception) des mots de passe.
- La vérification de la robustesse du mot de passe entré en temps réel par la vérification des facteurs ci-dessus.
- Un temps d'attente de 2 minutes par tranche de 3 tentatives de connexions échouées.
- Une stratégie de recouvrement par le biais d'un lien envoyé par mail afin de régénérer un mot de passe.

Chiffrement

Un algorithme de chiffrement est utile lorsque l'on veut qu'une information ne soit pas découverte, on insère alors cette information dans un algorithme de chiffrement afin que cette information ne soit plus compréhensible en l'état.

Il était initialement prévu d'utiliser le module BCrypt afin de chiffrer les informations de connexion des utilisateurs au Dashboard Web, cependant nous n'avons pas eu le temps de mettre en place ce système de connexion.

Hashage Bcrypt

BCrypt utilise un système de coût (qui n'est ni plus ni moins que le nombre de fois que l'opération de hashage sera effectuée sur l'information)

Cependant, le hashage en lui-même ne représente pas une sécurité suffisante en elle-même puis ce qu'un algorithme de chiffrement est une simple opération mathématique.

Salage

Le salage est simplement le fait de rajouter une information (le sel) avant ou après l'information à hasher.

Le sel peut être une phrase telle que 'Je suis le sel, et je suis utilisé dans le but de rendre indéchiffrable le hash auquel je vais être appliqué' sur laquelle un traitement aléatoire de mélange sera appliqué afin de rendre chaque sel unique et par la même occasion chaque empreinte de hashage différente.

RGPD (Règlement Général sur la Protection des Données)

Le RGPD est un règlement en vigueur au sein de l'Union Européenne, grâce au RGPD les utilisateurs dont les informations sont stockées sur un système afin d'assurer un service voient la protection de leurs données respectées.

Le RGPD traitent certains sujets tels que les suivants :

- Un traitement de donnée (Récupération, Modification, Extraction, etc) ne peut pas être effectué sans but précis.
- Les données d'un utilisateur ne peuvent pas être stockées indéfiniment, il faut que les données soient supprimées à un moment.
- Un utilisateur dont les données sont stockées sur un serveur doit avoir un droit de suppression, modification, regard sur ces dernières.

UUID (Universally Unique Identifiers)

Un UUID est un identifiant unique délivré par le serveur pour chaque utilisateur ayant besoin d'un enregistrement en base de donnée.

A l'inverse d'un simple ID pouvant être composé de digits allant de 0 à 9

l'UUID se compose d'un ensemble de caractères alphanumériques : 863cebfd-7875-45af-9e83-cd1e43aa1be4

De cette façon, il n'est pas envisageable qu'un attaquant puisse faire appelle à la base de donnée afin d'en récupérer les informations par le biais du champ 'ID' puis ce qu'il faudrait être incroyablement chanceux pour tomber sur un UUID identique.

SQLInjection

La faille SQLi (SQL Injection) est un type de faille permettant d'exécuter des requêtes SQL non prévues.

Prenons un exemple d'exploitation de faille SQLi :

Un attaquant se trouve sur une page de connexion d'un site Web vulnérable à l'injection SQL

Il tape un nom d'utilisateur aléatoire et tape un mot de passe tel que celui-ci : SQL ' OR 1=1 .

Le serveur voit simplement le mot de passe et l'interprète comme suit : .. WHERE
USERAME='nom utilisateur bidon' AND PASSWORD=" OR 1=1 .

Le serveur exécute la requête SQL et donne l'accès au premier compte contenu dans la base de donnée, l'attaquant est donc connecté sur le compte d'un utilisateur qui est dans 99% des cas le compte un compte à priviléges.

Il existe des déclinaisons à l'injection SQL, comme la suppression de toutes les données contenues de la base de données par l'interposition d'un ';' afin de faire exécuter toutes les requêtes envisageables à la base de donnée.

Afin de palier à ce type de faille, il existe des outils tels que des ORM permettant d'endiguer ce type de faille.

ORM

Un ORM permet de **préparer des requêtes SQL** et de préparer des requêtes lors de leur exécution en base de données ce qui permet d'empêcher l'insertion de caractère d'échappement dans les requêtes, de cette façon chaque information sera écrite **AS DATA** et non plus **AS LITERAL** et ne permettra plus l'exécution de code SQL arbitraire.

Dans le pire des cas, avec un ORM bien configuré mais sans la gestion d'erreur relative la réponse de la base de donnée sera une succession d'erreur et le script plantera.

Le Back End

La base de donnée : Conception

Formes Normales

Les formes Normales sont une liste de critères qu'une base de donnée doit remplir afin d'être considérée comme performante voici la liste des formes normales que notre base de donnée respectent :

- 1NF : Clé primaire (Pas de doublon)
- 2NF : Valeurs atomiques (Chaque cellule qui compose une table est une unité indivisible)
- Exemple : Une adresse est divisible par un numéro, un nom de rue, une ville et un code postal.
- Pour respecter la 2ème forme normale, on créer une table adresse qui décompose ces différents champs.
- 3NF : (Les champs d'une table correspondent à une seule et unique clé primaire)
Exemple : Les champs : Numéro de Commande, Adresse de livraison, Client, Produit dans une table Commande sont uniquement dépendants de la clé primaire "Numéro de Commande"

Dictionnaire de donnée

Le dictionnaire de donnée est un recueil de toutes les clés d'une base de donnée, le dictionnaire de donnée est la première étape préalable à la conception d'une base de donnée, il servira de référentiel lors de la conception afin d'évaluer quelles clés vont dans quelles tables
Le dictionnaire de donnée pour notre application est le suivant :

Colonne	Type	Description
id	INT	Identifiant numérique entier
position	INT?	Position numérique entière (facultatif)
uuid	VARCHAR	Chaîne de caractères
name	VARCHAR	Chaîne de caractères
created_at	DATETIME	Date et heure de création
updated_at	DATETIME	Date et heure de mise à jour
type	INT?	Type numérique entier (facultatif)
id_category	INT?	Identifiant numérique entier (facultatif)
id_config	INT?	Identifiant numérique entier (facultatif)
message_uuid	VARCHAR	Chaîne de caractères
message_content	VARCHAR	Chaîne de caractères
active	BOOLEAN	Valeur booléenne (true/false)
code_request	BOOLEAN	Valeur booléenne (true/false)
town_name	VARCHAR	Chaîne de caractères
link	VARCHAR	Chaîne de caractères
id_learner	INT?	Identifiant numérique entier (facultatif)
id_trainer	INT?	Identifiant numérique entier (facultatif)

Merise (Méthode d'étude et de réalisation informatique pour les systèmes d'entreprise)

Merise est une méthode d'analyse permettant la conception et la visualisation d'une base de donnée, il s'agit de la version UML de la base donnée, bien qu'on ne parle pas ici de diagramme même si les Modèles que Merise met en place y ressemblent

Cardinalités

Les cardinalités sont les synonymes des multiplicités d'UML, elle permettent d'identifier les relations entre les différentes entités du décrite dans le MCD, il y'a différents type de cardinalités :

- 0, 1 : Au minimum 0, au maximum 1 (On parle ici de CIF)
- 1, 1 : Au minimum 1, au maximum 1 (On parle ici aussi de CIF)
- 0, n : Au minimum 0, au maximum plusieurs valeurs (n représente ici un entier variable)
- 1, n : Au minimum 0, au maximum plusieurs valeurs

CIF et CIM

Parlons dans un premier temps de la CIF (Contrainte d'Intégrité Fonctionnelle) on parle d'une association CIF lorsque cette dernière est strictement fonctionnelle, c'est à dire que l'une des 2 entités associées par cette relation dépend de l'autre, sans l'une de ces 2 entités, l'autre n'existe pas.

La CIF ne se traduit pas par une nouvelle entité, elle permettra seulement d'exporté la clé primaire d'une entité dans l'autre au moment du passage en MPD / MLD, cette clé primaire deviendra alors une clé étrangère dans sa table d'accueil (Foreign Key).

Voyons maintenant la CIM (Contrainte d'Intégrité Multiple) une CIM se caractérise par une relation dont les cardinalités sont soit 0, n soit 1 / n, la CIM représente un couple unique auquel on peut affecter des propriétés particulières. Une CIM est constitué des clés auxquelles elle est liée.

De plus, une CIM se traduit par une nouvelle entité lors du passage en MPD / MLD

MCD (Modèle Conceptuel de Donnée)

Le Modèle Conceptuel de Donnée permet de visualiser les différentes entités (table) d'une base de donnée, il permet de définir les relations entre ces tables, les cardinalités de ces relations et donc d'établir un vue d'ensemble sur le base de donnée.

Dans le MCD, nous définissons les différentes clés des différentes entités décrite par le dictionnaire de donnée, et relions ces différentes entités par des relations comprenant des cardinalités qui permettent de définir le type de relation entre les entités.

Annexe [2]

MLD (Modèle Logique de Donnée)

Le Modèle Logique de Donnée est une retranscription plus proche de la mise en place de la Base de donnée, on ne parle plus d'entité, mais de table.

Le MLD permet de représenter la façon dont sera agencée chaque table de la base de donnée avec les clés primaires et les foreign key dans les différentes table.

Les CIM du MCD se transforment en table dans le MLD. (Annexe 3)

MPD (Modèle Physique de Donnée)

Le Modèle Logique de Donnée retranscrit le MLD en instructions SQL, qui permettra de générer la Base de Donnée conceptualisée en Base de Donnée réelle.

Voici le MPD de notre Base de Donnée :

```
CREATE TABLE "Category" (
    "id" SERIAL PRIMARY KEY,
    "id_guild" INTEGER NOT NULL,
    "position" INTEGER,
    "uuid" VARCHAR(255) NOT NULL,
    "name" VARCHAR(255) NOT NULL,
    "created_at" TIMESTAMP DEFAULT NOW(),
    "updated_at" TIMESTAMP DEFAULT NOW(),
    "channelsStockId" INTEGER,
    "promoid" INTEGER,
    FOREIGN KEY ("id_guild") REFERENCES "Guild" ("id") ON DELETE CASCADE,
    FOREIGN KEY ("channelsStockId") REFERENCES "ChannelStock" ("id") ON DELETE SET NULL,
    FOREIGN KEY ("promoid") REFERENCES "Promo" ("id") ON DELETE SET NULL
);

CREATE TABLE "Channel" (
    "id" SERIAL PRIMARY KEY,
    "name" VARCHAR(255) NOT NULL,
    "uuid" VARCHAR(255) UNIQUE NOT NULL,
    "type" INTEGER NOT NULL,
    "position" INTEGER NOT NULL,
```

```

"id_guild" INTEGER NOT NULL,
"id_category" INTEGER,
"created_at" TIMESTAMP DEFAULT NOW(),
"updated_at" TIMESTAMP DEFAULT NOW(),
FOREIGN KEY ("id_guild") REFERENCES "Guild" ("id") ON DELETE CASCADE,
FOREIGN KEY ("id_category") REFERENCES "Category" ("id") ON DELETE SET NULL
);

CREATE TABLE "Config" (
"id" SERIAL PRIMARY KEY,
"config" JSON,
"guildId" INTEGER,
"courselId" INTEGER UNIQUE,
FOREIGN KEY ("guildId") REFERENCES "Guild" ("id") ON DELETE SET NULL,
FOREIGN KEY ("courselId") REFERENCES "Course" ("id") ON DELETE SET NULL
);

CREATE TABLE "Course" (
"id" SERIAL PRIMARY KEY,
"name" VARCHAR(255) NOT NULL,
"id_role" INTEGER UNIQUE NOT NULL,
"id_guild" INTEGER NOT NULL,
"id_config" INTEGER UNIQUE,
"created_at" TIMESTAMP DEFAULT NOW(),
"updated_at" TIMESTAMP DEFAULT NOW(),
FOREIGN KEY ("id_role") REFERENCES "Role" ("id") ON DELETE CASCADE,
FOREIGN KEY ("id_guild") REFERENCES "Guild" ("id") ON DELETE CASCADE,
FOREIGN KEY ("id_config") REFERENCES "Config" ("id") ON DELETE SET NULL
);

CREATE TABLE "Guild" (
"id" SERIAL PRIMARY KEY,
"uuid" VARCHAR NOT NULL,

```

```

"name" VARCHAR(50) NOT NULL,
"member_count" INTEGER NOT NULL,
"config_id" INTEGER UNIQUE,
"created_at" TIMESTAMP DEFAULT NOW(),
"updated_at" TIMESTAMP DEFAULT NOW(),
FOREIGN KEY ("config_id") REFERENCES "Config" ("id") ON DELETE SET NULL
);

CREATE TABLE "Link" (
"id" SERIAL PRIMARY KEY,
"link" VARCHAR(50) NOT NULL,
"created_at" TIMESTAMP DEFAULT NOW(),
"updated_at" TIMESTAMP DEFAULT NOW()
);

CREATE TABLE "Message" (
"id" SERIAL PRIMARY KEY,
"message_uuid" VARCHAR NOT NULL,
"message_content" VARCHAR(255) NOT NULL,
"id_user" INTEGER NOT NULL,
"id_ticket" INTEGER,
"created_at" TIMESTAMP DEFAULT NOW(),
"updated_at" TIMESTAMP DEFAULT NOW(),
FOREIGN KEY ("id_user") REFERENCES "User" ("id") ON DELETE CASCADE,
FOREIGN KEY ("id_ticket") REFERENCES "Ticket" ("id") ON DELETE CASCADE
);

CREATE TABLE "Promo" (
"id" SERIAL PRIMARY KEY,
"name" VARCHAR NOT NULL,
"active" BOOLEAN NOT NULL,
"code_request" BOOLEAN DEFAULT true,
"id_course" INTEGER NOT NULL,

```

```

"id_role" INTEGER UNIQUE NOT NULL,
"id_factory" INTEGER NOT NULL,
"id_category" INTEGER UNIQUE NOT NULL,
"created_at" TIMESTAMP DEFAULT NOW(),
"updated_at" TIMESTAMP DEFAULT NOW(),
FOREIGN KEY ("id_course") REFERENCES "Course" ("id") ON DELETE CASCADE,
FOREIGN KEY ("id_role") REFERENCES "Role" ("id") ON DELETE CASCADE,
FOREIGN KEY ("id_factory") REFERENCES "Factory" ("id") ON UPDATE NO ACTION,
FOREIGN KEY ("id_category") REFERENCES "Category" ("id") ON DELETE CASCADE
);

CREATE TABLE "Factory" (
"id" SERIAL PRIMARY KEY,
"town_name" VARCHAR(255) NOT NULL,
"id_guild" INTEGER NOT NULL,
"id_role" INTEGER UNIQUE,
"created_at" TIMESTAMP DEFAULT NOW(),
"updated_at" TIMESTAMP DEFAULT NOW(),
FOREIGN KEY ("id_guild") REFERENCES "Guild" ("id") ON DELETE CASCADE,
FOREIGN KEY ("id_role") REFERENCES "Role" ("id") ON DELETE CASCADE
);

CREATE TABLE "resources_sharing" (
"id" SERIAL PRIMARY KEY,
"up_vote" INTEGER NOT NULL,
"down_vote" INTEGER NOT NULL,
"id_channel" INTEGER NOT NULL,
"id_user" INTEGER NOT NULL,
"created_at" TIMESTAMP DEFAULT NOW(),
"updated_at" TIMESTAMP DEFAULT NOW(),
FOREIGN KEY ("id_channel") REFERENCES "Channel" ("id") ON DELETE NO ACTION,
FOREIGN KEY ("id_user") REFERENCES "User" ("id") ON DELETE NO ACTION

```

```

);

CREATE TABLE "Role" (
    "id" SERIAL PRIMARY KEY,
    "uuid" VARCHAR NOT NULL,
    "name" VARCHAR(255) NOT NULL,
    "color" VARCHAR(20) NOT NULL,
    "position" INTEGER NOT NULL,
    "id_guild" INTEGER NOT NULL,
    "courseld" INTEGER,
    "promold" INTEGER,
    "factoryId" INTEGER,
    "created_at" TIMESTAMP DEFAULT NOW(),
    "updated_at" TIMESTAMP DEFAULT NOW(),
    FOREIGN KEY ("id_guild") REFERENCES "Guild" ("id") ON DELETE CASCADE,
    FOREIGN KEY ("courseld") REFERENCES "Course" ("id") ON DELETE NO ACTION,
    FOREIGN KEY ("promold") REFERENCES "Promo" ("id") ON DELETE CASCADE,
    FOREIGN KEY ("factoryId") REFERENCES "Factory" ("id") ON DELETE CASCADE
);

CREATE TABLE "Signature" (
    "id" SERIAL PRIMARY KEY,
    "id_learner" INTEGER,
    "id_trainer" INTEGER,
    "created_at" TIMESTAMP DEFAULT NOW(),
    "updated_at" TIMESTAMP DEFAULT NOW(),
    FOREIGN KEY ("id_learner") REFERENCES "User" ("id") ON DELETE NO ACTION,
    FOREIGN KEY ("id_trainer") REFERENCES "User" ("id") ON DELETE NO ACTION
);

CREATE TABLE "Template" (
    "id" SERIAL PRIMARY KEY,
    "id_course" INTEGER NOT NULL,

```

```

"id_channel" INTEGER NOT NULL,
"created_at" TIMESTAMP DEFAULT NOW(),
"updated_at" TIMESTAMP DEFAULT NOW(),
FOREIGN KEY ("id_course") REFERENCES "Course" ("id") ON DELETE CASCADE,
FOREIGN KEY ("id_channel") REFERENCES "Channel" ("id") ON DELETE CASCADE
);

CREATE TABLE "Ticket" (
"id" SERIAL PRIMARY KEY,
"ticket_tag" VARCHAR(255) NOT NULL,
"ticket_state" VARCHAR NOT NULL,
"id_user" INTEGER NOT NULL,
"id_message" INTEGER,
"created_at" TIMESTAMP DEFAULT NOW(),
"updated_at" TIMESTAMP DEFAULT NOW(),
FOREIGN KEY (id_user) REFERENCES User (id) ON DELETE CASCADE,
FOREIGN KEY (id_message) REFERENCES Message (id) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS "User" (
"id" SERIAL PRIMARY KEY,
"username" VARCHAR(255),
"mail" VARCHAR(255),
"uuid" VARCHAR(255),
"id_guild" INT,
"created_at" TIMESTAMP DEFAULT NOW(),
"updated_at" TIMESTAMP,
FOREIGN KEY ("id_guild") REFERENCES "Guild" ("id") ON DELETE CASCADE
);

CREATE TABLE "ChannelStock" (
"id" SERIAL PRIMARY KEY,
"id_guild" INT UNIQUE,

```

```

"id_category" INT UNIQUE,
FOREIGN KEY ("id_guild") REFERENCES "Guild" ("id") ON DELETE CASCADE ON UPDATE NO ACTION,
FOREIGN KEY ("id_category") REFERENCES "Category" ("id"),
CONSTRAINT "channelsStock_guilds_fk" FOREIGN KEY ("id_guild") REFERENCES "Guild" ("id") ON DELETE CASCADE ON UPDATE NO ACTION,
CONSTRAINT "channelsStock_category_fk" FOREIGN KEY ("id_category") REFERENCES "Category" ("id")
);
CREATE TABLE "Define" (
"id" INT,
"id_channelsStock" INT,
FOREIGN KEY ("id") REFERENCES "ChannelStock" ("id") ON DELETE NO ACTION ON UPDATE NO ACTION,
FOREIGN KEY ("id_channelsStock") REFERENCES "ChannelStock" ("id") ON DELETE NO ACTION ON UPDATE NO ACTION,
CONSTRAINT "define_channels_fk" FOREIGN KEY ("id") REFERENCES "Channel" ("id") ON DELETE NO ACTION ON UPDATE NO ACTION,
CONSTRAINT "define_channelsStock_fk" FOREIGN KEY ("id_channelsStock") REFERENCES "ChannelStock" ("id") ON DELETE NO ACTION ON UPDATE NO ACTION,
PRIMARY KEY ("id", "id_channelsStock")
);
CREATE TABLE "Participants" (
"id" SERIAL PRIMARY KEY,
"id_promo" INT,
"user_id" INT,
FOREIGN KEY ("user_id") REFERENCES "User" ("id") ON DELETE CASCADE,
FOREIGN KEY ("id_promo") REFERENCES "Promo" ("id") ON DELETE CASCADE,
CONSTRAINT "participate_user_fk" FOREIGN KEY ("user_id") REFERENCES "User" ("id") ON DELETE CASCADE,
CONSTRAINT "participate_promo_fk" FOREIGN KEY ("id_promo") REFERENCES "Promo" ("id") ON DELETE CASCADE
);

```

L'API

Le Crud

Le CRUD est les 4 opérations d'une API qui permettent respectivement de

- Créer une ressource à l'aide de la Method HTTP POST
- Récupérer une ressource à l'aide de la Method HTTP GET
- Mettre à jour une ressource à l'aide de la Method HTTP PATCH / PUT
- Supprimer une ressource à l'aide de la Method HTTP DELETE

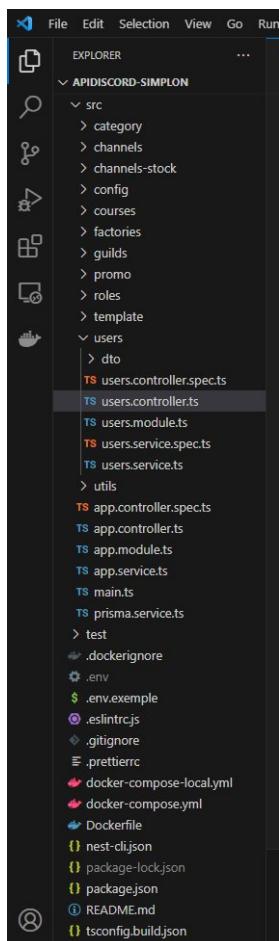
NestJS & Express

Il est important de savoir que NestJS met en place le squelette pour le développeur et délègue la gestion des requêtes à Express ou Fastify (en l'occurrence Express).

Architecture de l'API : NestJS

NestJS est un framework qui permet par le biais du module CLI de créer des composants d'API de façon structurée.

```
$ nest g resource
```



Cette commande permet de générer un module qui se décompose comme le dossier 'users'

Qui comprends :

- 1 Dossier portant le nom de la ressource
- 1 Dossier DTO
- 1 fichier de classe controller
- 1 fichier de test du controller
- 1 fichier de classe service
- 1 fichier de test du service
- 1 fichier de classe moduleLa commande génère un module CRUD.

Chaque module NestJS sera donc séparé de la même façon, cela permet de séparer chaque classe en fonction de leur utilisation.

Le module User

Les modules NestJS permettent de définir les spécifications pour ce module.

```
1 import { Module } from '@nestjs/common';
2 import { UsersService } from './users.service';
3 import { UsersController } from './users.controller';
4 import { PrismaService } from '../prisma.service';
5 import { GuildsService } from '../guilds/guilds.service';
6 import { CoursesService } from 'src/courses/courses.service';
7 import { RolesService } from 'src/roles/roles.service';
8 import { PromoService } from 'src/promo/promo.service';
9 import { FactoriesService } from 'src/factories/factories.service';
10 import { CategoryService } from 'src/category/category.service';
11
12 @Module({
13   controllers: [UsersController],
14   providers: [
15     UsersService,
16     PrismaService,
17     GuildsService,
18     CoursesService,
19     FactoriesService,
20     PromoService,
21     RolesService,
22     CategoryService,
23   ],
24 })
25 export class UsersModule {}
```

On peut voir un décorateur Nest intitulé "Module" qui abstrait une grande quantité de code pour le développeur et qui est laissé à NestJS

Dans l'objet envoyé au décorateur, on peu observer certaines clés.

- La propriété controllers envoyée au décorateur est un tableau de Type :

Avec la propriété 'controllers', le développeur définit le (ou LES) contrôleur(s) du module, de plus, le fait que cette propriété soit un tableau de Type et non juste un Type permet de définir plusieurs contrôleurs pour un même module, et donc permet sur des projets de grande envergure de séparer les routes en fonctions de leur champ d'action, ce qui apporte une certaine clarté dans les fichiers de code.

- La clé providers : Permet de définir les Providers du module, c'est-à-dire, les classes dont ce module (Ou le module l'important) utilise. Provider signifie Fournisseur, on peut en comprendre que les Providers NestJS sont donc des fournisseurs de code.

Le contrôleur User

Un contrôleur permet de recevoir des requêtes HTTP et les traiter en appliquant la logique adaptée à la requête.

Conventions d'écriture des contrôleurs

J'ai décidé de standardiser la structure des contrôleurs afin de permettre une lecture optimale du code, la structure que j'ai choisi est la suivante :

- Les routes POST
- Les routes GET
- Les routes DELETE
- Les routes UPDATE

De cette façon, chaque contrôleur est construit de la même façon et il est facile de retrouver une route en particulier.

En plus de cet agencement du code, j'ai décidé d'écrire chaque méthode comme suit :

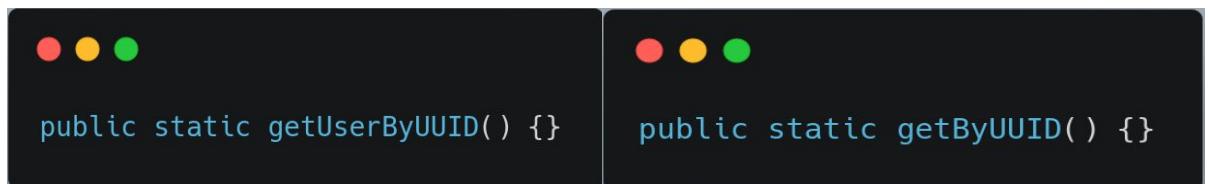


Afin que n'importe qui puisse comprendre que la méthode est publique est en exécution asynchrone

De plus, j'ai adopté une convention de nommage qui abstrait le module dans lequel la méthode se trouve par exemple la méthode :

Sans la convention

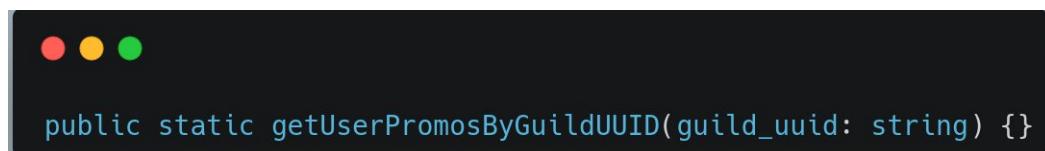
Avec la convention



Puis ce que la méthode se trouve dans le contrôleur 'User' les méthodes n'ont pas besoin de spécifier que ces dernières se rapportent à l'utilisateur.

Il existe cependant des exceptions à cette convention, lorsque la route impliquera par exemple une logique qui ne correspond pas uniquement à l'utilisateur.

Par exemple, une route du contrôleur qui implique une logique selon laquelle on cherche à récupérer les promotions liées à un utilisateur sera nommée de façon explicite :



Au premier coup d'œil, on comprend que la méthode sert à récupérer les promotions liées à l'utilisateur en fonction du UUID de la guilde passé en argument.

Contenu des contrôleurs

Le contenu de chaque méthode contient :

- Un bloc de code Try-Catch : J'ai utilisé des Blocs Try Catch dans chaque contrôleur pour permettre à l'exécution de ne pas s'arrêter en cas d'erreur non gérée ou gérée, je m'explique. La logique de chaque route peut renvoyer une erreur déclenchée volontairement afin de mettre fin au fil d'exécution, pour l'instant, cela peut sembler étrange, cependant nous verrons plus tard pourquoi j'ai designé les contrôleurs de cette façon.

Amélioration à apporter : Globaliser les blocs de try catch au sein d'un middleware pour éliminer cette invasion.

- 2 appels à une fonction de standardisation des réponses. : J'ai développé une méthode permettant de retranscrire les objets ressortis par la logique de façon uniforme.

```
public static build_response_object(
    request_status_code: HttpStatus,
    data: any,
    message: string,
) {
    return {
        statusCode: request_status_code,
        data: this.exclude_ids(data),
        message: message,
    };
}

public static build_error_response_object(error: any) {
    return {
        statusCode: error.statusCode ?? HttpStatus.INTERNAL_SERVER_ERROR,
        data: null,
        message: error.message,
    };
}

private static exclude_ids(data: any) {
    for (const key in data) {
        if (typeof data[key] == 'object') {
            this.exclude_ids(data[key]);
        }
        if ([
            key.toLowerCase().includes('_id') ||
            key.toLowerCase().includes('id_') ||
            key.toLowerCase() == 'id'
        ]) {
            delete data[key];
        }
    }
    return data;
}
```

On peut voir dans la méthode "build_response_object" que le code prends en paramètre :

- Un status code (Code HTTP de réponse à la requête)
- La donnée renvoyée suite à l'execution de la logique
- Le message à renvoyé suite à l'execution de la logique

Cette fonction se charge uniquement de "parser" toutes ces données en un objet JSON et de normaliser la sortie de chaque reponse.

De plus, on peut constater que les données passe dans une fonction qui exclues les champs ID des objets retournés.

La route POST

```
@Controller('users')
export class UsersController {
    constructor(private readonly usersService: UsersService) {}

    @Post('register')
    public async register(@Body() register_user_dto: RegisterUserDto) {
        try {
            return Utils.build_response_object(
                HttpStatus.CREATED,
                await this.usersService.register(register_user_dto),
                `User ${register_user_dto.uuid} on guild ${register_user_dto.guild_uuid} has been registered`,
            );
        } catch (error: any) {
            return Utils.build_error_response_object(error);
        }
    }
}
```

La route est caractérisée par la méthode `register() {}` on peut voir sur cette méthode un décorateur NestJS qui permet de définir que la méthode prendra en charge la requête HTTP vers la route `users/register` en Method HTTP POST.

Notez la présence du décorateur `@Controller('users')` qui permet de définir un préfixe aux routes du contrôleur.

On peut voir que dans les paramètres de cette route qu'il y a un DTO (Data Transfer Object), il s'agit de la partie "Body" (ou "Payload" dans le cas d'une requête POST) des requêtes HTTP.

On retrouve dans le bloc Try Catch le retour de l'appel du service dédié (la logique concernée) normalisé au client.

La route GET

```
@Get('guild/:guild_uuid')
public async getByGuildUUID(@Param('guild_uuid') guild_uuid: string) {
  try {
    return Utils.build_response_object(
      HttpStatus.OK,
      await this.userService.getByGuildUUID(guild_uuid),
      `Users on guild ${guild_uuid} has been found`,
    );
  } catch (error: any) {
    return Utils.build_error_response_object(error);
  }
}
```

Ici encore la même construction que précédemment, sauf que l'on peut voir dans le décorateur `@Get('guild/:guild_uuid')`.

Ici le `:` permet à NestJS de splitter et d'identifier les paramètres de la requête.

Comme on peut le voir en dessous dans les arguments de la méthode `@Param('guild_uuid')` NestJS sait qu'il doit attribuer la valeur du paramètre à cette variable en Runtime, il établit en quelque sorte qu'il remplace la valeur du paramètre en 'guild_uuid' et identifie qu'il s'agit de ce paramètre et attribue donc la valeur du paramètre à l'argument concerné.

Les routes DELETE & Patch

```
@Patch('guild/:guild_uuid/:user_uuid')
public async updateByUUID(
    @Param('guild_uuid') guild_uuid: string,
    @Param('user_uuid') user_uuid: string,
    @Body() update_user_dto: UpdateUserDto,
) {
    try {
        return Utils.build_response_object(
            HttpStatus.OK,
            await this.userService.updateByUUID(
                guild_uuid,
                user_uuid,
                update_user_dto,
            ),
            `User ${user_uuid} has been updated`,
        );
    } catch (error: any) {
        return Utils.build_error_response_object(error);
    }
}
```

Route UPDATE (PATCH)

```
80  @Delete('guild/:guild_uuid/:user_uuid')
81  public async deleteByUUID(
82      @Param('guild_uuid') guild_uuid: string,
83      @Param('user_uuid') user_uuid: string,
84  ) {
85      try {
86          return Utils.build_response_object(
87              HttpStatus.OK,
88              await this.userService.deleteByUUID(guild_uuid, user_uuid),
89              `User ${user_uuid} on guild ${guild_uuid} has been deleted`,
90          );
91      } catch (error: any) {
92          return Utils.build_error_response_object(error);
93      }
94 }
```

La route DELETE

Les services d'User

La méthode d'enregistrement des utilisateurs

```
public async register(register_user_dto: RegisterUserDto) {
  const targeted_user = await this.getByUUID(
    register_user_dto.guild_uuid,
    register_user_dto.uuid,
    false,
  );

  if (targeted_user !== undefined) {
    throw new DataAlreadyRegError(
      `User ${register_user_dto.uuid} is already register for guild ${register_user_dto.guild_uuid}`,
    );
  }

  return await this.prisma.user.create({
    data: {
      mail: register_user_dto.mail,
      uuid: register_user_dto.uuid,
      username: register_user_dto.name,
      guild: {
        connect: {
          id: (await this.guild.getByUUID(register_user_dto.guild_uuid)).id,
        },
      },
      include: this.includeValues(),
    });
}
```

Pour enregistrer un nouvel utilisateur Discord côté API, la méthode `register() {}` décrit la logique.

On stock d'abord la valeur de retour de l'appel de la méthode `getByUUID() {}` qui prend les mêmes paramètres et qui sert à récupérer un utilisateur enregistré dans la base de donnée.

Si la valeur de la variable `targeted_user` est indéfini, cela veut dire que l'utilisateur n'existe pas, alors on peut demander à Prisma de le créer et la méthode retournera l'objet créé (en plus de certaines autres données) au contrôleur.

Si l'utilisateur est déjà enregistré alors on coupe le fil d'exécution en déclenchant une erreur et le contrôleur renvoie le message d'erreur adapté.

La méthode de récupération par UUID

```
public async getByUUID(
    guild_uuid: string,
    user_uuid: string,
    throw_enable = true,
) {
    const targeted_user = await this.prisma.user.findFirst({
        where: {
            uuid: user_uuid,
            AND: {
                guild: {
                    id: (await this.guild.getByUUID(guild_uuid)).id,
                },
            },
        },
        include: this.includeValues(),
    });

    if (targeted_user === undefined && throw_enable)
        throw new DataNotFoundError(`No user found for ${user_uuid}`);

    return targeted_user;
}
```

Dans la méthode de récupération, on stock l'utilisateur récupéré dans une variable par le biais de Prisma.

Notez que l'on demande à Prisma de récupérer l'utilisateur en fonction de son identifiant unique mais aussi de l'identifiant unique de la guild (Serveur Discord)

En invoquant la méthode de récupération de Guild par son UUID, on permet en fait au code de retourner une erreur dans le cas où la Guild n'existe pas.

Et donc de renvoyer le message d'erreur : "La Guild spécifiée n'existe pas"

Si tout se passe bien, on renvoie l'utilisateur.

Les méthodes de suppression & Méthode de mise à jour

```
public async deleteByUUID(guild_uuid: string, user_uuid: string) {
  return await this.prisma.user.delete({
    where: {
      id: (await this.getByUUID(guild_uuid, user_uuid)).id,
    },
    include: this.includeValues(),
  });
}

public async updateByUUID(
  guild_uuid: string,
  user_uuid: string,
  update_userDto: UpdateUserDto,
) {
  return await this.prisma.user.update({
    where: {
      id: (await this.getByUUID(guild_uuid, user_uuid)).id,
    },
    data: {
      mail: update_userDto.mail ?? undefined,
    },
    include: this.includeValues(),
  });
}
```

This.includeValues() ?

```
// Internal methods

private includeValues() {
  return {
    guild: {
      select: {
        uuid: true,
      },
    },
  };
}
```

Il s'agit seulement d'une méthode permettant de définir ce que les objets de retours vont inclurent.

En l'occurrence, on demande à Prisma d'inclure la Guild (son UUID ici) d'un User lorsqu'il le récupère.

On centralise seulement les ressources nesteds au sein d'une seule et même méthode.

DTO

Le DTO ou Data Transfer Object est un type de classe mis en place par NestJS qui permet de formater un objet JSON voici un exemple :

```
import { ApiProperty } from '@nestjs/swagger';
import { IsNotEmpty, IsString } from 'class-validator';

export class RegisterUserDto {
    @ApiProperty()
    @IsNotEmpty({ message: 'This user need a UUID' })
    @IsString()
    uuid: string;

    @ApiProperty()
    @IsNotEmpty({ message: 'This user need a name' })
    @IsString()
    name: string;

    @ApiProperty()
    @IsNotEmpty({ message: 'This user need to be attached to a guild' })
    @IsString()
    guild_uuid: string;

    @ApiProperty()
    @IsNotEmpty({ message: 'This user need a mail' })
    @IsString()
    mail: string;
}
```

NestJS met en place cette fonctionnalité afin de structurer les données reçues dans la requête.

De plus le système de DTO permet d'exécuter une vérification préalable à l'exécution de la logique.

Le décorateur `@IsString()` en est un bon exemple, il permet de vérifier que cette propriété dans les objets de la requête correspondent aux types décrit dans le DTO.

Il existe d'autre décorateur de vérification.

Chaque décorateur de vérification permet de définir un message d'erreur spécifique.

Swagger

Swagger est un outil permettant de mettre en place une documentation de l'API, il permet de décrire chaque route, ses paramètres, ses données attendues.

Nous avons utilisé Swagger pour documenter notre API, bien qu'elle ne soit pas complète elle apporte une vue d'ensemble sur l'architecture de l'API.

Contexte du Bot d'Onboarding

Le Bot d'Onboarding doit permettre de gérer de façon globale l'import d'un écosystème de Centre de Formation au sein d'un seul et unique serveur Discord, dans cette optique, le Bot doit permettre d'accueillir de nouveaux membres internes ou externes au Centre de Formation.

Il doit permettre de définir qu'un utilisateur est un Apprenant ou un Membre des différentes équipes du Centre de Formation de la façon la plus simple possible, pour permettre l'accueil des différentes promotions, le Bot doit mettre à disposition un espace dédié à une promotion.

Aussi, le Bot doit permettre la création de nouvelles formations ainsi que la création de nouvelles promotions de façon la plus simple possible, il doit aussi permettre aux membres concernés de consulter ces différents espaces si nécessaire tout en conservant une navigation fluide au sein du serveur Discord

Conception

UML (Unified Modeling Language)

UML est un langage de modélisation utilisé pour modéliser des processus commerciaux ou analyser, concevoir et implémenter des composants logiciels

Le langage UML est un standard à suivre quant à la conception d'une application, nous avons utilisé UML afin de concevoir notre application de façon globale.

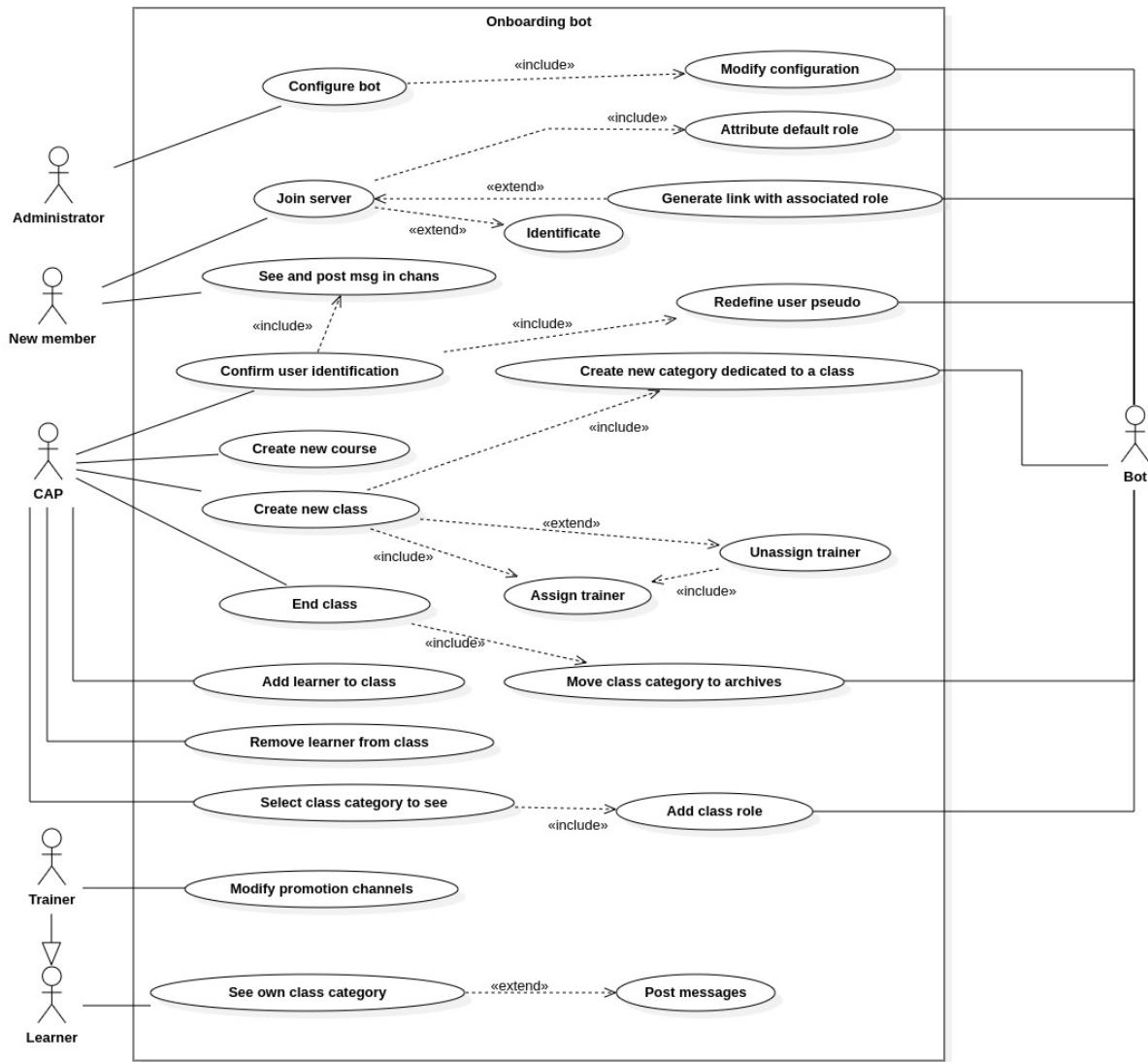
Dans lequel chaque diagramme apporte une vision différente de l'application, chaque diagramme est important et peut être pris indépendamment des autres, il s'agit d'une ligne directrice à suivre lors de l'étape de réalisation d'une application.

Use case (Diagramme de cas d'utilisation) : Zoom x1

Le diagramme de cas d'utilisation d'une application permet aux développeurs de comprendre :

- Quels sont les acteurs impliqués au sein du système
- Quelles sont les utilisations possibles du système par ces acteurs
- Quelles sont les interactions entre ces acteurs

Il apporte une vue d'ensemble et très peu détaillée de l'application, cependant il est nécessaire d'établir un diagramme de cas d'utilisation correcte afin de construire les autres diagrammes qui rentreront plus en profondeur dans les cas d'utilisation de l'application.



Dans ce présent diagramme :

L'administrateur (représenté ici par Administrator) peut configurer le Bot ce qui implique que le Bot modifie sa configuration.

Un nouveau membre (représenté ici par New Member) peut rejoindre le serveur ce qui implique que le Bot lui attribut un rôle par défaut, aussi, il est possible que le Bot ait généré un lien d'invitation auquel il a associé un rôle à attribuer à l'utilisateur. Aussi, le nouveau membre peut ou doit s'identifier selon le lien d'invitation avec lequel il a rejoint le serveur Discord.

Le nouveau membre peut poster des messages dans les canaux globaux, et il peut poster des messages dans les canaux liés à son rôle ce qui implique, qu'il se soit identifié.

Le CAP peut confirmer les demandes d'identification des nouveaux membres, ce qui implique que le Bot doive redéfinir le pseudo du nouvel utilisateur, le cap peut aussi créer un nouveau type de formation, il peut créer une nouvelle promotion, il doit pour cela assigner au moins un formateur à la promotion, pour créer une nouvelle promotion le Bot doit créer un espace dédié à la promotion.

Le CAP peut aussi mettre fin à une promotion en cours, ce qui implique que le Bot doit déplacer la catégorie liée à la promotion vers une zone d'archivage, le cap peut aussi ajouter ou

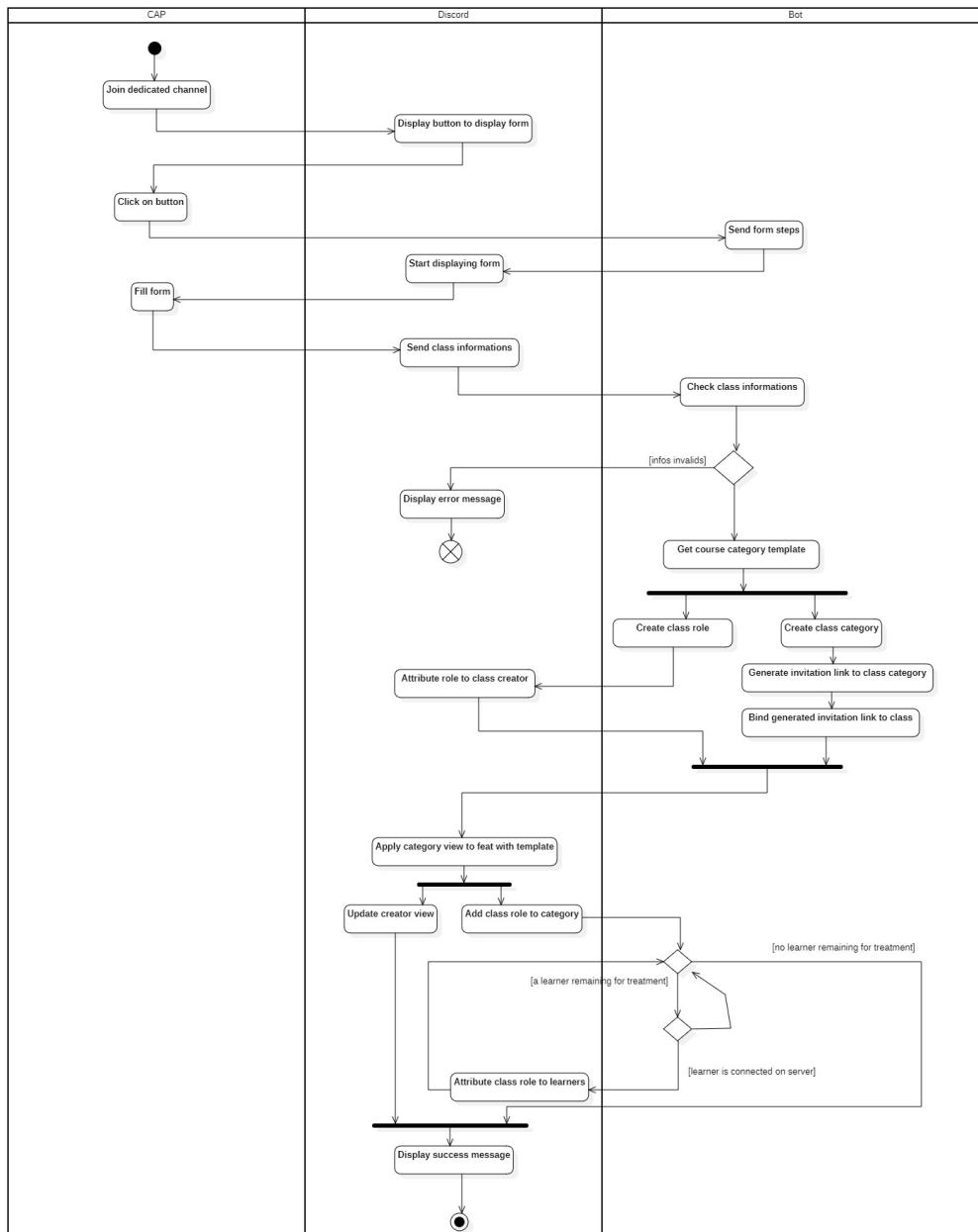
supprimer un apprenant d'une promotion. Il peut aussi sélectionner les promotions qu'il veut consulter ce qui implique que le Bot doit ajouter ou retirer un rôle au CAP.

Le formateur (représenté ici par Trainer) peut quant à lui consulter l'espace dédié à sa promotion et y discuter librement. L'apprenant (représenté ici par Learner) peut lui aussi consulter son espace de formation et y discuter librement.

Activity Diagram (Diagramme d'Activités) : Zoom x2

Le diagramme d'activités représente les différentes activités d'une application de façon générale et peu détaillée, il est conçu à partir du diagramme de cas d'utilisation

Il apporte cependant une meilleure visibilité quant au flux d'exécution des actions

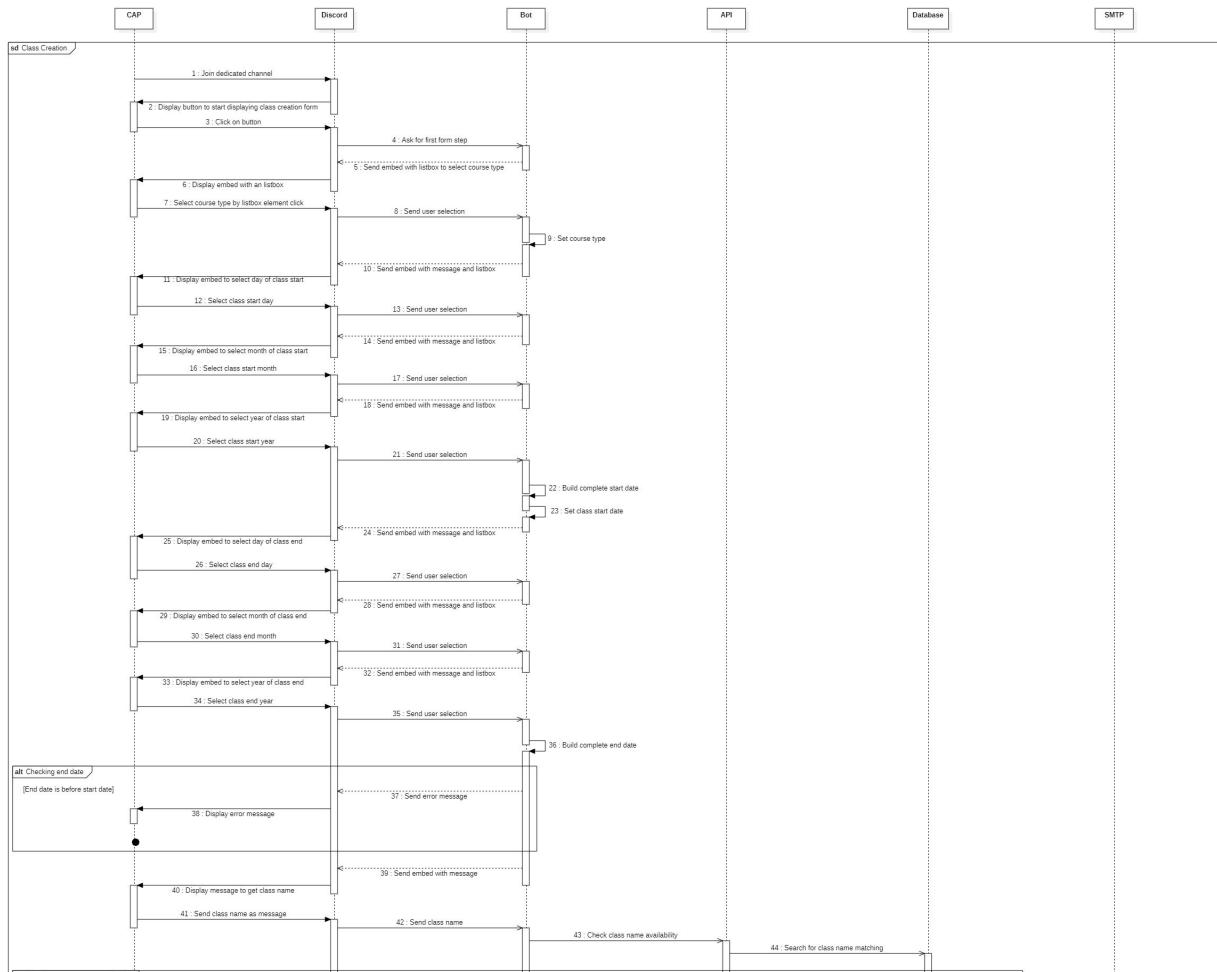


Ce présent diagramme décrit les activités possibles lors de la création d'une nouvelle promotion :

Un CAP rejoint le canal dédié à la création d'une promotion, Discord affiche donc un bouton afin de donner au CAP une interface de création de promotion. Le CAP clique sur le bouton et entre les différentes caractéristiques de la promotion puis valide la demande de création de promotion. Le Bot vérifie les informations entrées par le CAP, si une erreur est détectée, il le notifie au CAP, sinon le Bot récupère le template du type de formation, puis il crée un rôle de promotion ainsi qu'une catégorie (un espace) dédié à la promotion à laquelle il lie un lien d'invitation. Il attribue le rôle de promotion au CAP qui a créé la promotion. Alors Discord applique le template de formation à la catégorie de la promotion et lie le rôle de promotion à la catégorie. Ensuite le Bot va lier le rôle de promotion à chaque apprenant. Enfin Discord va notifier le succès de la création de la promotion au CAP

Diagramme de Séquence : Zoom x3

Le diagramme de séquence représente les différentes activités d'une application de façon très détaillée, il est conçu à partir du diagramme d'activités



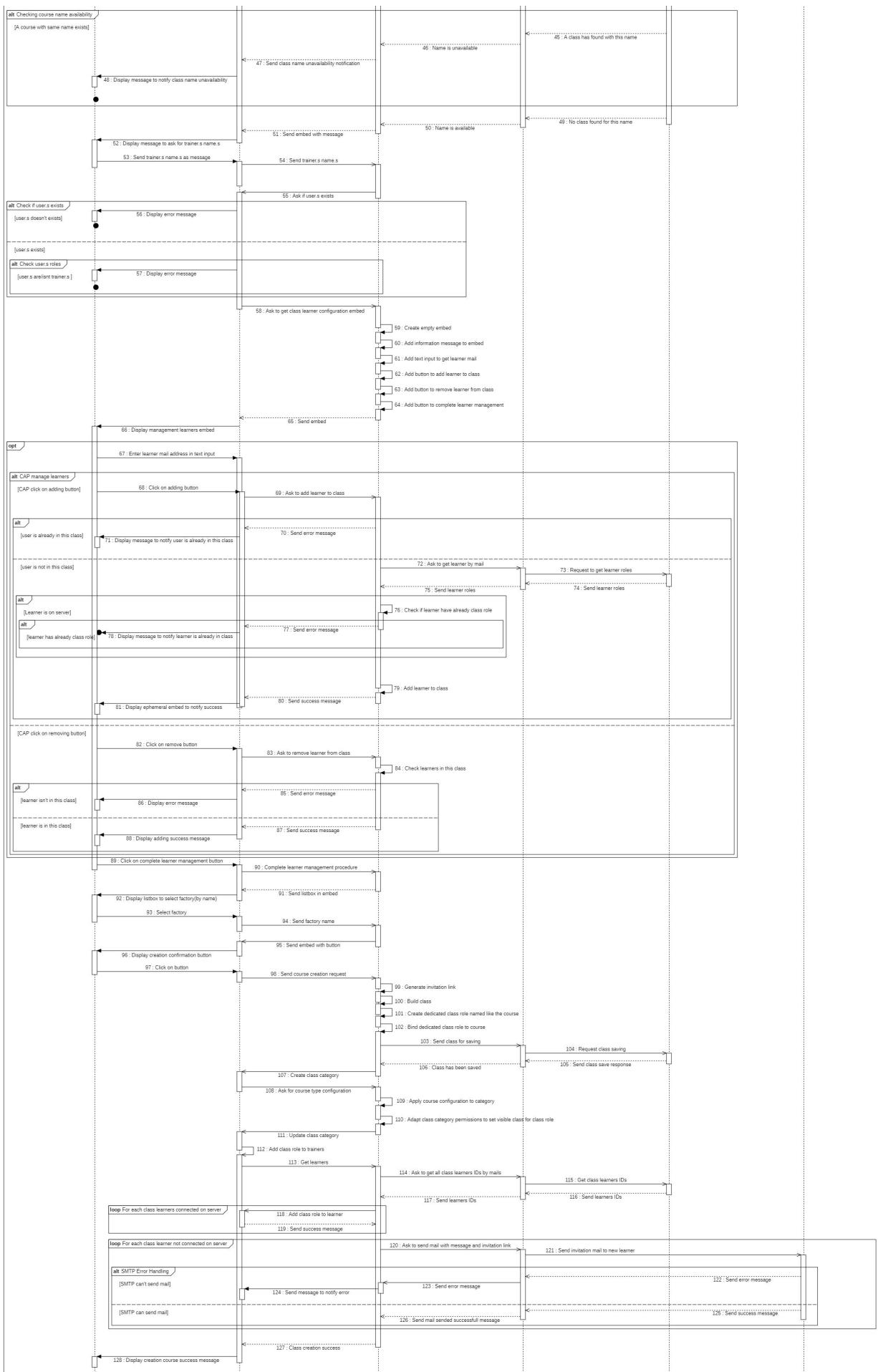
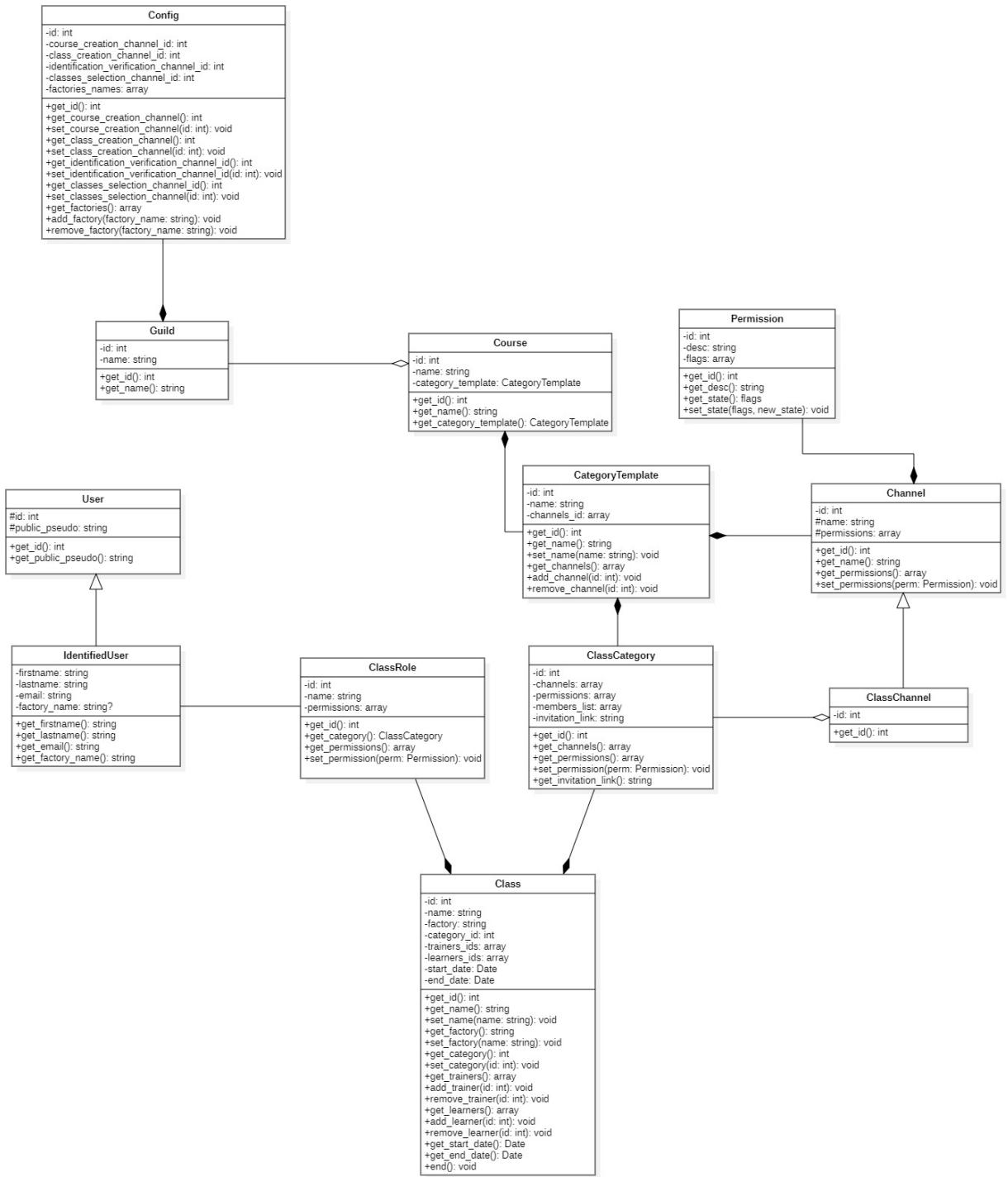


Diagramme de Classe : Zoom x4

Le diagramme de classe représente les différentes classes d'une application, ce diagramme doit être utilisé lors de la phase de développement afin de permettre aux développeurs de savoir la façon dont doit être développée l'application.



Dans ce diagramme, nous avons une classe nommée Config qui est utile afin de configurer le Bot, elle compose la classe Guild qui représente le Serveur Discord qui agrège la classe Course qui représente une formation.

La classe CategoryTemplate compose cette même classe Course et représente la façon dont doit-être agencé une catégorie Discord selon son type de Course (Formation) La classe Course représente une formation La classe ClassCategory compose la classe CategoryTemplate et est l'instanciation de la classe CategoryTemplate La classe ClassCategory agrège la classe ClassChannel qui est l'instanciation des canaux pour la promotion La classe ClassChannel généralise (hérite) de la classe Channel qui est la représentation d'un canal Discord La classe Permission compose la classe Channel, cette classe permet de définir les permissions des différents rôles selon le canal Discord La classe ClassCategory compose la classe Class qui représente une promotion dans son ensemble La classe ClassRole compose elle aussi la classe Class, elle représente le rôle associé à chaque promotion La classe ClassRole est associé à la classe IdentifiedUser qui représente un utilisateur identifié sur le serveur Discord La classe IdentifiedUser généralise (hérite) User qui représente un utilisateur lambda

Déploiement

Docker



L'utilisation première de Docker répond à une problématique, l'environnement d'exécution.

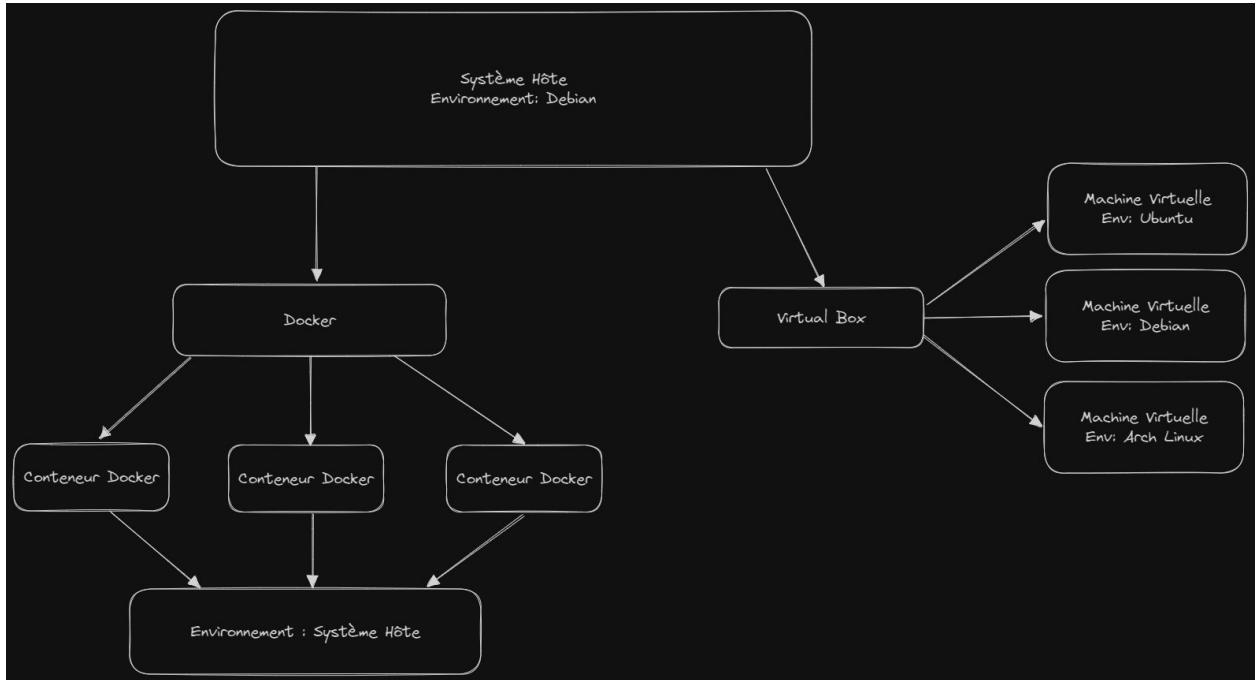
En effet, il est courant d'entendre un développeur (j'en ai fait partie) dire que son application fonctionne sur sa machine, mais lors du déploiement de l'application, les clients rencontrent des problèmes d'exécution, de dépendance.

Docker permet d'installer n'importe quel environnement d'exécution dans ce qu'on appelle des conteneurs qui permettront d'avoir un environnement absolument identique en tout point entre la version de l'application en phase de test et l'application en phase de livraison, voilà à quoi sert Docker.

Docker apporte une couche de sécurité supplémentaire en compartimentant chaque conteneur bien distinctement des autres, les conteneurs ne peuvent pas communiquer entre eux (sauf dans des cas spécifiques dans lesquels il est possible de faire communiquer des conteneurs) ce qui apporte une isolation parfaite des composants systèmes.

Par exemple, si une API se trouve dans un conteneur et qu'une base de donnée se trouve dans un autre, ces 2 composants ne communiquent pas directement entre eux, ils sont séparés dans des conteneurs.

Un conteneur n'est PAS une Machine Virtuelle !



Conteneur != Machine Virtuelle

GitHub Actions

Afin de déployer notre application nous avons utilisé GitHub Actions.

Nous étions limité en termes de ressources serveurs et avions 2 serveurs VPS pour héberger notre application, il existe donc 2 CI/CD pour notre application.

Le découpage que j'ai décidé d'adopter pour notre application est le suivant :

- L'api et la base de donnée sur 1 serveur
- Le Bot d'Onboarding et Redis sur 1 serveur

La Pipeline CI / CD (Continuous Integration & Continuous Deployment) de l'API

```
● ● ●

name: CI/CD

on:
  push:
    branches:
      - prod

jobs:
  clear-docker-context:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Clear docker context
        uses: appleboy/ssh-action@master
        with:
          host: ${{ secrets.SERVER_HOST }}
          username: ${{ secrets.SERVER_USERNAME }}
          password: ${{ secrets.SERVER_PASSWORD }}
          script: |
            docker stop postgres-prod
            docker stop nest-prod
            docker system prune -a -f

  deploy:
    runs-on: ubuntu-latest
    needs: clear-docker-context
    if: always()
    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Send files to server
        uses: appleboy/scp-action@master
        with:
          host: ${{ secrets.SERVER_HOST }}
          username: ${{ secrets.SERVER_USERNAME }}
          password: ${{ secrets.SERVER_PASSWORD }}
          source: .
          target: api

      - name: Build API Image
        uses: appleboy/ssh-action@master
        with:
          host: ${{ secrets.SERVER_HOST }}
          username: ${{ secrets.SERVER_USERNAME }}
          password: ${{ secrets.SERVER_PASSWORD }}
          script: |
            cd /home/${{ secrets.SERVER_USERNAME }}/api
            docker build -t simplon_api:1.0 .
            docker network create api-network-prod

      - name: Mount Postgres Container
        uses: appleboy/ssh-action@master
        with:
          host: ${{ secrets.SERVER_HOST }}
          username: ${{ secrets.SERVER_USERNAME }}
          password: ${{ secrets.SERVER_PASSWORD }}
          script: |
            docker run -d \
              --network=api-network-prod \
              --name=postgres-prod \
              -e POSTGRES_USER=${{ secrets.POSTGRES_USER_PROD }} \
              -e POSTGRES_PASSWORD=${{ secrets.POSTGRES_PASSWORD_PROD }} \
              -p 5432:5432 \
              -v postgres:/var/lib/postgresql/data \
              postgres:12.2

      - name: Mount Nest Container
        uses: appleboy/ssh-action@master
        with:
          host: ${{ secrets.SERVER_HOST }}
          username: ${{ secrets.SERVER_USERNAME }}
          password: ${{ secrets.SERVER_PASSWORD }}
          script: |
            docker run -d \
              --network=api-network-prod \
              --name=nest-prod \
              -e DATABASE_URL=${{ secrets.URI_POSTGRES_PROD }} \
              -p 3000:3000 \
              --restart always \
              simplon_api:1.0
```

Cette pipeline assure que l'application sera déployée à chaque fois que des modifications sont apportées sur la branche Prod (Lors de Merge suite à une pull request par exemple)

Voici dans l'ordre ce que fait ce script YAML :

- Nettoyage de contexte Docker distant : Lors de la mise en place de cette pipeline, j'ai rencontré un problème, chaque déploiement que j'effectuais rajoutait une image Docker et donc prenez de l'espace Disque, si bien qu'à un moment, ma base de donnée ne pouvait plus écrire. Je ne comprenais pas d'où venait le problème, bien que Postgres me disait qu'il manquait d'espace et que j'ai constaté que le Disque était saturé à l'aide de la commande (df -H) pour obtenir des informations sur l'espace de stockage, je n'avais pas fait le lien entre Docker et l'espace Disque. J'ai fini par débugger en utilisant cherchant méticuleusement quel dossier prenait de la place et ai compris que cela venait de docker. J'ai alors rajouté ce Job dans ma CI / CD afin de supprimer toutes les images à chaque déploiement.
 - Envoie des fichiers sources au serveur distant
 - Construction de l'image de l'application :

Cette étape consiste à créer l'image de l'application par le biais du DockerFile

```
FROM node:slim
COPY . .
RUN npm install
RUN npx prisma generate
EXPOSE 3000
CMD ["./bin/bash", "-c", "npx prisma db push && npm run start"]
```

Ce DockerFile installe une image de base (Node:Its-slim).

Ensuite on copie la racine (L'endroit où se trouve le DockerFile) à la racine du futur conteneur (monté à partir de cette image).

On exécute l'instruction ```npm install``` afin de faire installer les dépendances du projet dans l'image.

On génère les migrations à partir du schéma prisma à l'aide de l'instruction `npx prisma generate` ce qui permet de synchroniser la base de donnée avec le Schéma Prisma.

On expose le port 3000 du futur conteneur (monté à partir de l'image que l'on construit) afin de permettre la consommation de cette API depuis l'extérieur

Et enfin on lance l'API à l'aide de l'instruction ```CMD .. npm run start```.

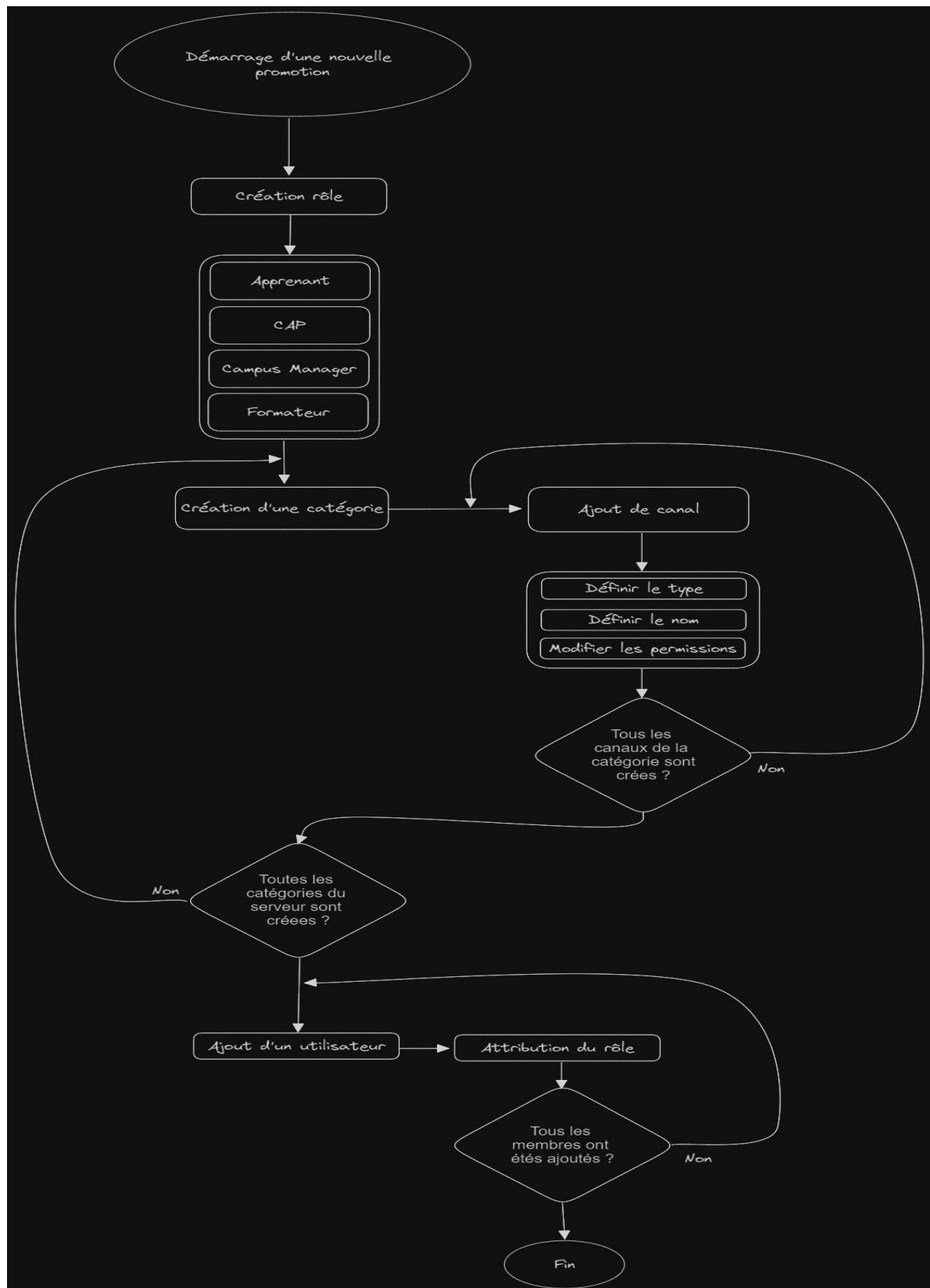
Une fois que l'image est construite plus qu'à la monter.

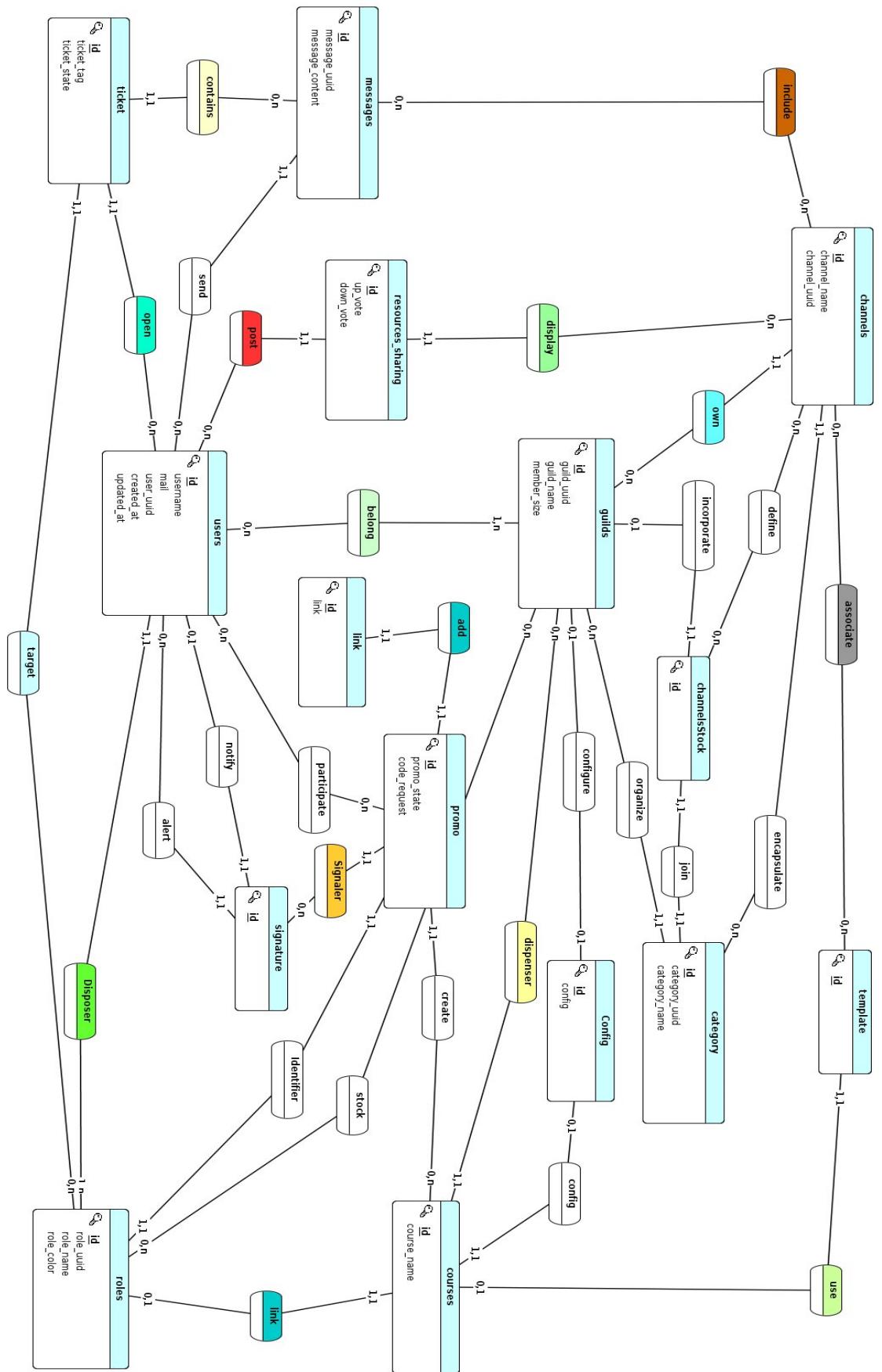
- Montage de l'image Postgres : L'image n'existe jamais localement donc Docker la télécharge et la monte
- Montage de l'image de l'API : L'image à été construire en local, on la lance dans un conteneur.

Et voilà, l'application est déployée !

Annexe

[1] : Algorigramme concernant la création de promo





[3] : MLD

