

# Dossier de Projet professionnel



**Realisation d'un application mobile**  
Johan Bouguermouh

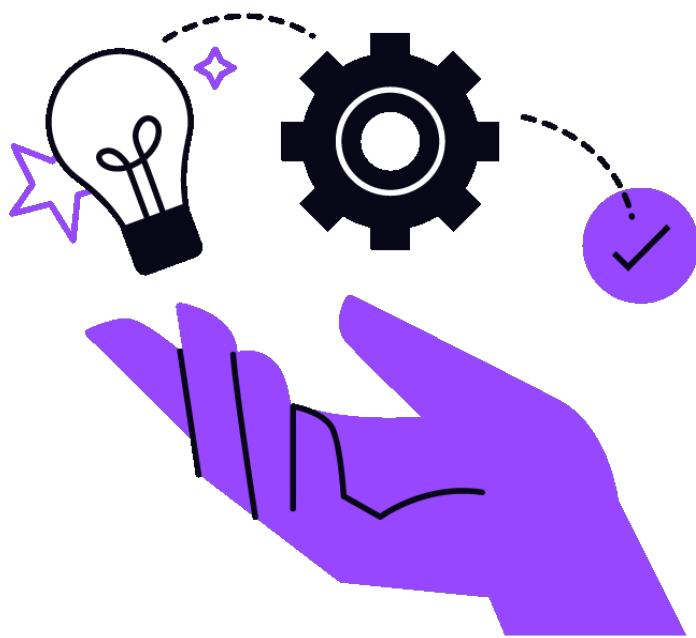
# Sommaire

## Table des matières

<b>Liste des compétences du référentiel qui sont couvertes par le projet .....</b>	<b>3</b>
<b>Définition du projet .....</b>	<b>4</b>
Résumé du projet .....	4
Cahier des charges   Contexte et objectifs du projet .....	5
Définition du Minimum Viable Product .....	7
<b>Phase d'Initialisation &amp; conception du projet .....</b>	<b>9</b>
Aux abords de la conception .....	9
Planification du projet .....	10
Interrogation sur les Designs Patterns & Framework de gestion de projet .....	13
Détail de notre organisation .....	14
Aligner les technologies : Le rôle crucial des décisions de stack .....	18
Architecture backend : Choix stratégiques pour un développement performant .....	20
Sélectionner la base de données optimale entre SQL et MongoDB .....	23
<b>Modélisation de la base de donnée .....</b>	<b>25</b>
Objectifs de la définition générale du système .....	25
MCD   Modèle Conceptuel de Base de Donnée .....	28
MLD   Modèle logique de base de donnée .....	30
<b>Maquettage de l'application .....</b>	<b>32</b>
Un support intuitif qui se révèle être un formidable outil technique .....	32
Le wireframe   Concevoir c'est aussi s'aménagé du temps .....	40
Gestion des Parcours utilisateur dynamique avec Figma .....	44
Maquette haute fidélité .....	45
<b>Premier Sprint   Développement de l'API .....</b>	<b>47</b>
Mise en place de conventions pour une approche plus cohérente .....	49
Mise en place d'express et de l'architecture dossier Back-End .....	50
L'élaboration des tickets et formalisation de la demande .....	55
Développement des routes nécessaires .....	57
Créer des tests unitaires associés aux routes .....	62
Test Fonctionnel des routes avec Thunder Client .....	63
<b>Deuxième Sprint   Développement du Front-End .....</b>	<b>65</b>
Organisation de notre git-Hub Project .....	66
Mise en place du Backlog pour le développement front-End .....	69
Architecture générale du dossier Front-end .....	71
Développement du Front-end .....	71
Exemple de Jeu d'essaie sur la fonctionnalité d'authentification automatique .....	76

# Liste des compétences du référentiel qui sont couvertes par le projet

- Maquetter une application
- La maquette prend en compte les spécificités fonctionnelles décrites dans les cas d'utilisation ou les scénarios utilisateur
- L'enchaînement des écrans est formalisé par un schéma
- La communication écrite, en français ou en anglais, est rédigée de façon adaptée à l'interlocuteur et sans faute
- La maquette respecte les principes de sécurisation d'une interface utilisateur
- Développer des composants d'accès aux données
- Développer la partie front-end d'une interface utilisateur web
- Développer la partie back-end d'une interface utilisateur web
- Concevoir une base de données
- Mettre en place une base de données
- Développer des composants dans le langage d'une base de données
- Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement
- Concevoir une application
- Construire une application organisée en couches
- Développer une application mobile



# Définition du projet

## Résumé du projet

**Sent** is a mobile application aimed at enhancing our skills as designer-developers. It is a comprehensive academic project that allowed us to apply our knowledge in design, organization, and application development, in accordance with the requirements of the RNCP certification. Sent goes beyond being a simple chat application; it can be likened to a photograph capturing the expression of a carefully thought-out and conceptualized internal organization, while adhering to time and feasibility constraints. This collaborative experience provided us with an opportunity to work as a team. Our main objective was to design, develop, and deploy an application that meets the business specifications.

To optimize our efficiency, we adapted our approach by clearly defining the application's scope of feasibility. Drawing inspiration from Agile methodologies, we assigned roles and responsibilities, with the ultimate goal of creating a functional proof of concept that meets the MVP requirements set by the team.

Beyond being just a school project, Sent is a tangible example of the importance of organization and design within a coherent framework. Through effective project management, we created an environment conducive to productivity and commitment. As a result, we delivered a robust, scalable, and maintainable solution capable of adapting to future challenges.

# Cahier des charges | Contexte et objectifs du projet

## Contexte et définition du projet

Dans le cadre d'un projet de fin d'études, une équipe de quatre membres est chargée de concevoir et développer une application mobile de chat. L'objectif est de créer une plateforme permettant aux utilisateurs d'échanger des messages sur un channel public, avec la possibilité de créer et de modifier leur profil. Le projet implique la prise en charge complète de la conception et de la réalisation de l'application mobile. Le délai de réalisation du projet est d'un mois.

## Objectif du projet

Offrir une application mobile fonctionnelle et agréable à utiliser. Possibilité d'administrer l'application via un panel admin utilisable via une web app. Ce projet a pour but de remplir la plus grande partie des compétences demandées pour le titre de Concepteur Développeur d'Application Périmètre

Le projet comprendra un Proof of Concept (POC) avec la possibilité de tester l'application sur un appareil mobile pour évaluer sa fonctionnalité et sa convivialité.

## Description fonctionnelle des besoins

- Le système devra comprendre un panel administrateur permettant la gestion des utilisateurs et des messages.
- Le chat offrira aux utilisateurs la possibilité de créer un compte, de modifier leur profil, et d'accéder à un annuaire pour trouver d'autres utilisateurs.
- Les utilisateurs pourront créer des canaux de discussion individuels ou de groupe, avec le créateur du canal ayant la possibilité de paramétrier et d'administrer les droits du groupe. Les discussions se feront en temps réel.
- Le système devra également inclure un historique de conversation

## Contraintes techniques

- Utilisation d'un langage de programmation adapté pour le développement d'une application mobile.
- Utilisation d'un langage de programmation approprié pour le développement du back-end. Utilisation d'outils de contrôle de version et de collaboration pour la gestion du code source. Gestion de projet efficace en utilisant une méthodologie adaptée au domaine informatique.
- Modélisation de la base de données selon les pratiques courantes, en assurant une structure solide et des mises à jour en cas de changements.

## **Contraintes techniques**

- Livraison de la maquette de l'application : 06/01/2023
- Livraison de la modélisation de la base de données : 06/01/2023
- Livraison de la version POC de l'application : 01/02/2023

## **Rôles et responsabilités**

- Johan Bouguermouh & Ahmed Magassouba seront en charge de la conception et de la réalisation de la maquette de l'application WireFrame
- Boris TIKHOMIROFF & Cyril Porez seront en charge de la modélisation de la base de données et de sa mise à jour en cas de changements
- L'intégralité de l'équipe sera en charge du développement de l'application mobile avec React Native
- L'intégralité de l'équipe sera en charge du développement du back-end avec Node.js
- L'intégralité de l'équipe sera seront successivement en charge de la gestion de projet et de la coordination des différentes tâches avec Trello et les diagrammes de Gantt
- L'intégralité de l'équipe sera en charge de la gestion de version avec Git et GitHub

## **Critères de qualité**

1. L'application doit être fonctionnelle et agréable à utiliser
2. Le panel admin doit permettre une gestion efficace des utilisateurs.
3. Les maquettes doivent être professionnelles et bien détaillées
4. La modélisation de la base de données doit être complète et à jour en cas de changements.
5. L'utilisation de Trello et des diagrammes de Gantt doit permettre une gestion efficace du projet.
6. L'utilisation de Git et GitHub doit être professionnelle et organisée.
7. Les fonctionnalités principales doivent être fonctionnelles
8. Deux utilisateurs peuvent échanger des messages instantanément.

# Définition du Minimum Viable Product

## La méthode MVP | Une approche rationnelle pour concevoir efficacement

À la suite de la rédaction du cahier des charges, nous avons évalué les objectifs à prioriser afin de répondre au mieux à la demande dans le temps imparti. L'objectif principal était de valider de manière pragmatique les fonctionnalités clés du produit.

Pour cela, nous nous sommes inspirés de la méthode introduite par l'entrepreneur américain Eric Ries dans son ouvrage «[The Lean Startup](#)», à savoir la méthode du MVP (Minimum Viable Product). Le MVP peut être considéré comme une version de travail du produit qui peut être testée et validée avant d'ajouter des fonctionnalités.

À ce stade de la réflexion, il est fondamental de se pencher sur la cohésion et l'utilisation de notre application afin d'extraire, un socle stable à développer. Cela garantit une intégrité compacte et homogène suffisante pour son utilisation.

Bien entendu, dans le cadre d'un travail scolaire, la méthode mentionnée n'avait pas pour prétention de permettre une pénétration sur le marché à moindre coût, mais plutôt d'éclairer la possibilité des objectifs que nous pouvons nous fixer. Idéalement, cela nous permet de comprendre et de définir de manière commune les aspirations de notre application et son potentiel de développement. En limitant ainsi les fonctionnalités à l'essentiel, nous évitons des dépenses excessives de temps et de ressources sur des éléments qui pourraient ne pas être essentiels ou qui pourraient être modifiés ultérieurement.

## Liste des fonctionnalités essentielles

En nous basant sur les enseignements de Mike Cohn et sa méthodologie présentée dans «[User Stories Applied: For Agile Software Development](#)», nous avons adopté une approche de fragmentation des éléments du projet en catégories «Must-have», «Should-have» et «Nice-to-have». Cela nous permet de prioriser les fonctionnalités essentielles, importantes et facultatives, facilitant ainsi la prise de décisions et la planification du développement.

### Must-have (Doit avoir) :

- Authentification : Permettre aux utilisateurs de créer un compte et de se connecter à l'application.
- Profil utilisateur : Permettre aux utilisateurs de créer et de gérer leur profil, y compris la modification des informations personnelles.

- Messagerie instantanée : Mettre en place un système de chat en temps réel permettant aux utilisateurs d'envoyer et de recevoir des messages.
- Annuaire des utilisateurs : Fournir une fonctionnalité de recherche ou d'affichage des utilisateurs enregistrés dans l'application.

## Should-have (Devrait avoir) :

- Canal public : Permettre aux utilisateurs d'accéder à un canal public commun pour envoyer des messages visibles par tous.
- Gestion des utilisateurs : Implémenter un panneau d'administration permettant de gérer les utilisateurs, y compris la création, la modification et la suppression de comptes.
- Donner la possibilité à l'utilisateur de quitter le channel
- Gestion des droits d'accès : Permettre à l'utilisateur créateur d'un canal de discussion de définir les droits d'accès et les autorisations pour les autres utilisateurs.

## Nice-to-have (Souhaitable) :

- Gestion des messages : Permettre à l'administrateur ou au modérateur de gérer les messages, y compris la suppression des messages inappropriés ou indésirables.



La catégorie «Must-have» comprend les éléments essentiels pour fournir une expérience de base fonctionnelle. La catégorie «Should-have» contient des fonctionnalités importantes qui ajoutent de la valeur à l'application mais qui pourraient être prioritaires selon les besoins spécifiques du projet. Enfin, la catégorie «Nice-to-have» regroupe des fonctionnalités supplémentaires qui améliorent l'expérience utilisateur mais qui peuvent être considérées comme des ajouts optionnels.

# Phase d'Initialisation & conception

## Aux abords de la conception

La conception minutieuse d'une application en amont est un élément clé pour garantir une production fluide et moins fastidieuse. En élaborant une vision claire du produit, en identifiant les besoins des utilisateurs et en planifiant les fonctionnalités de manière stratégique, on maximise les chances de réussite du projet. Une conception bien pensée permet d'éviter les pièges courants, de réduire les risques et de faciliter la collaboration entre les membres de l'équipe de développement. Dans ce chapitre, nous explorerons notre processus d'élaboration et comment cela contribue à simplifier et optimiser le processus de production.

## Phase d'Initialisation & conception du projet

En ce sens nous avons défini un corpus d'outils qui nous seront favorables au partage d'information et au bon déroulement de notre organisation. En voici quelques uns qui nous ont été utiles lors de la phase de conception :



### Git | comme Système de contrôle de version distribué

- Intégration rapide du projet en locale & Intégrité des données
- Mise à jour simplifié du répertoire commun
- Facilité de travail sur des fonctionnalités ou des correctifs de bugs isolés
- Visibilité rapide de l'arborescence de travail



### GitHub Project | comme interface de gestion du Projet

- Établissement de délais et de jalons
- Partage du code sur répertoire commun
- Peer Review lors des merge
- intrication des issues vs branches
- interface de projet sous forme de Kaban
- détail des issues en MarkDown



### Google Drive | espace de stockage collaboratif

- stockage des assets
- transfert d'informations annexes



### Google Agenda | Organisation de notre calendrier

- Prise de rendez-vous
- Daily Meeting programmés



### **lucidChart |** conception de Diagramme UML & Meurise

- Réalisation de Diagramme d'activité lors de procédé complexes
- Réalisation du MCD/MLD



### **Notion |** comme application collaborative de prise de note

- facilitation de la hiérarchisation de l'information
- prise de note collaborative
- documentation liés au technologies associées

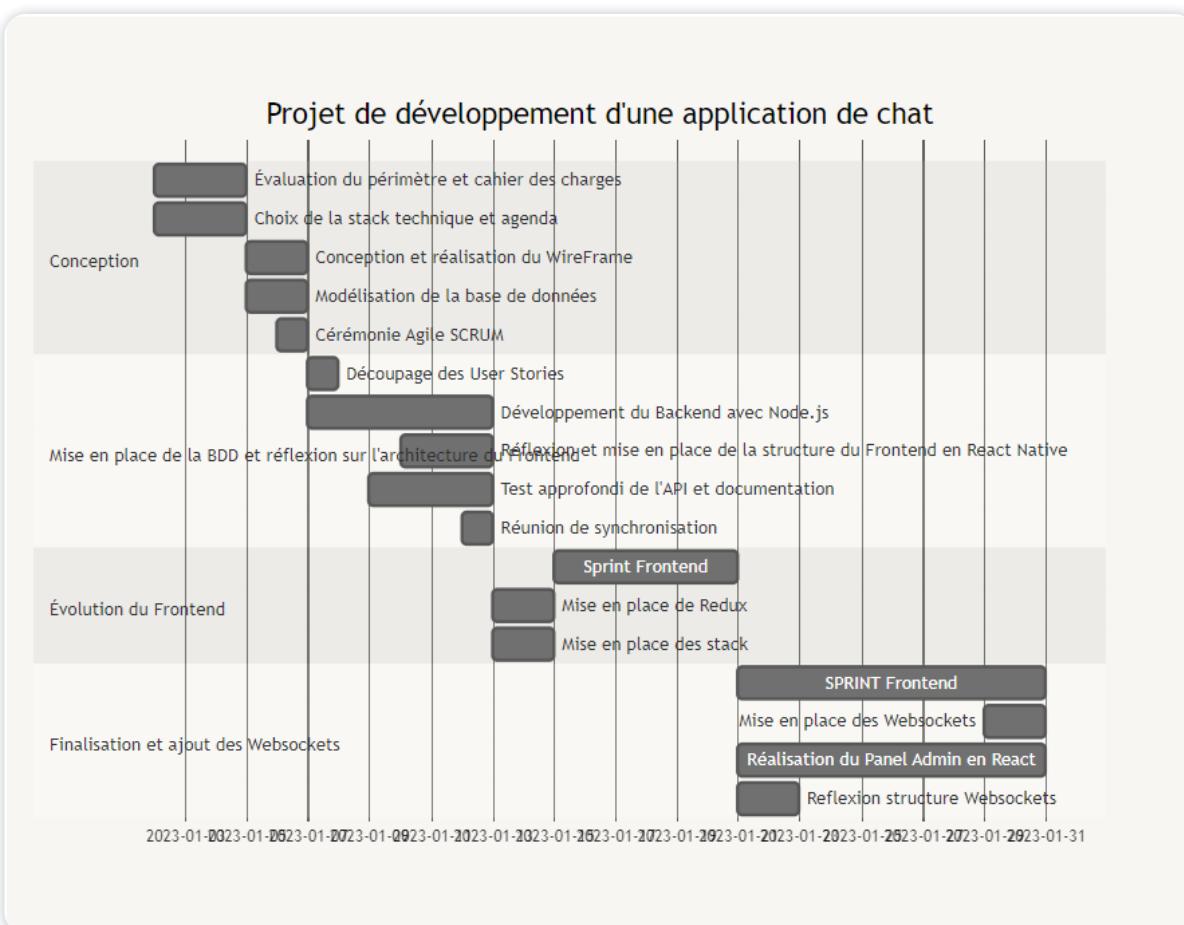


### **Figma |** comme interface collaborative de maquette

- Mise en place du Design Système
- Création du Wireframe
- Création de la maquette

## Planification du projet

Nous avons décidé de suivre un plan structuré en utilisant des diagrammes de Gantt. Cette approche nous a permis de s'assurer de la compréhension de tout les collaborateurs sur les objectif a atteindre.



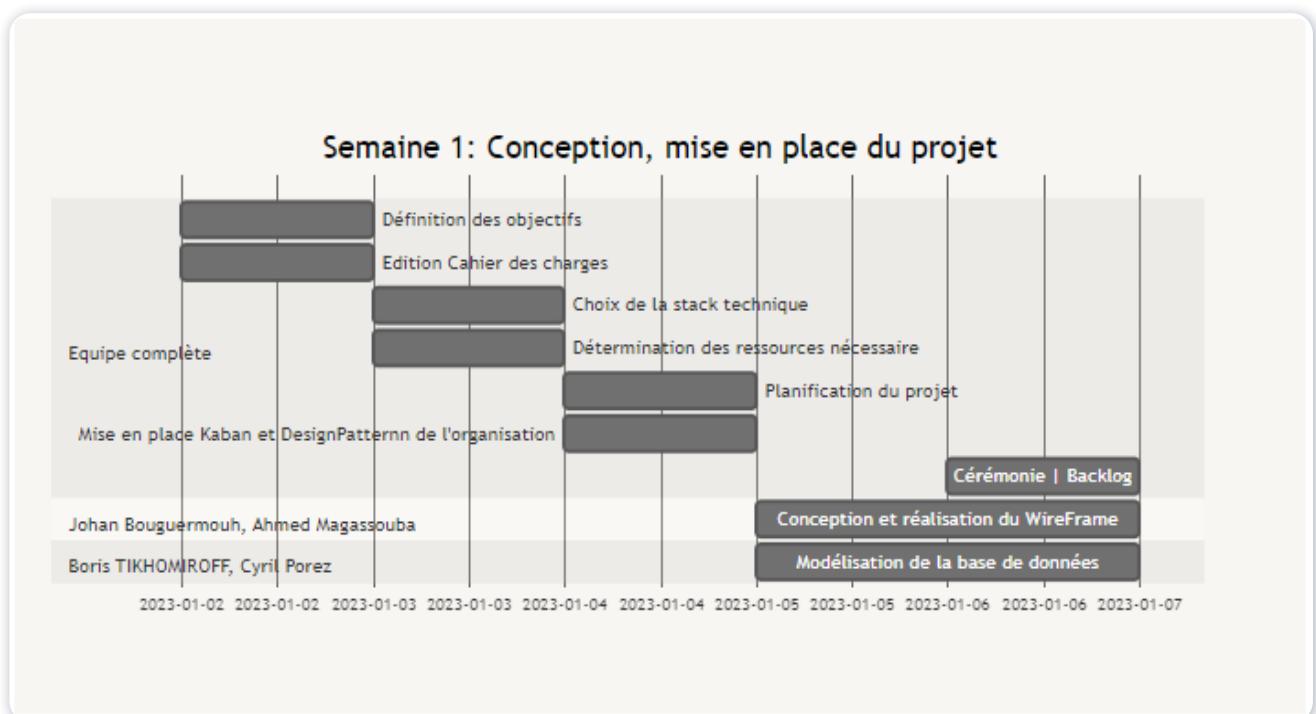
Ce plan commence par une semaine de conception où l'équipe évalue le périmètre, établit un cahier des charges et choisit la stack technique. En parallèle, des membres spécifiques de l'équipe travaillent sur la conception du wireframe et la modélisation de la base de données.

La deuxième semaine se concentre sur la mise en place de la base de données, le développement du backend et la réflexion sur l'architecture du frontend. Des tests approfondis de l'API sont effectués, et une réunion de synchronisation est organisée.

La troisième semaine est dédiée à l'évolution du frontend, avec un sprint spécifique. Tandis que la quatrième semaine est réservée à la finalisation du projet et à l'ajout des websockets.

Dans l'ensemble, le projet suit une approche agile, où chaque membre de l'équipe a des tâches spécifiques à accomplir, conformément à la planification établie.

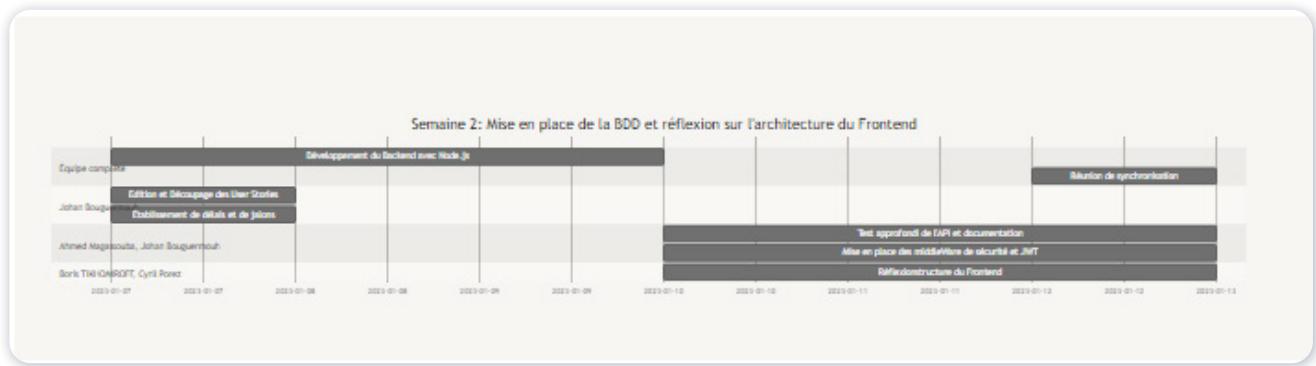
## Première Semaine | Organisation de l'équipe



### La première semaine ce concentre sur deux étapes majeurs :

- Dans un premier temps nous discutons avec la totalité des collaborateurs sur les enjeux de notre applications. Nous évaluons ainsi le périmètre de faisabilité selon les délais et les ressources nécessaires à sa réalisation et nous élaborons un plan détaillé pour la réalisation du projet, en identifiant les différentes tâches, les dépendances, les ressources requises et les échéances.
- La deuxième partie de cette semaine a été destiné à l'édition simultané du Zoning et du Wireframe ainsi que l'élaboration du Model conceptuel de base de donnée et du modèle logique.

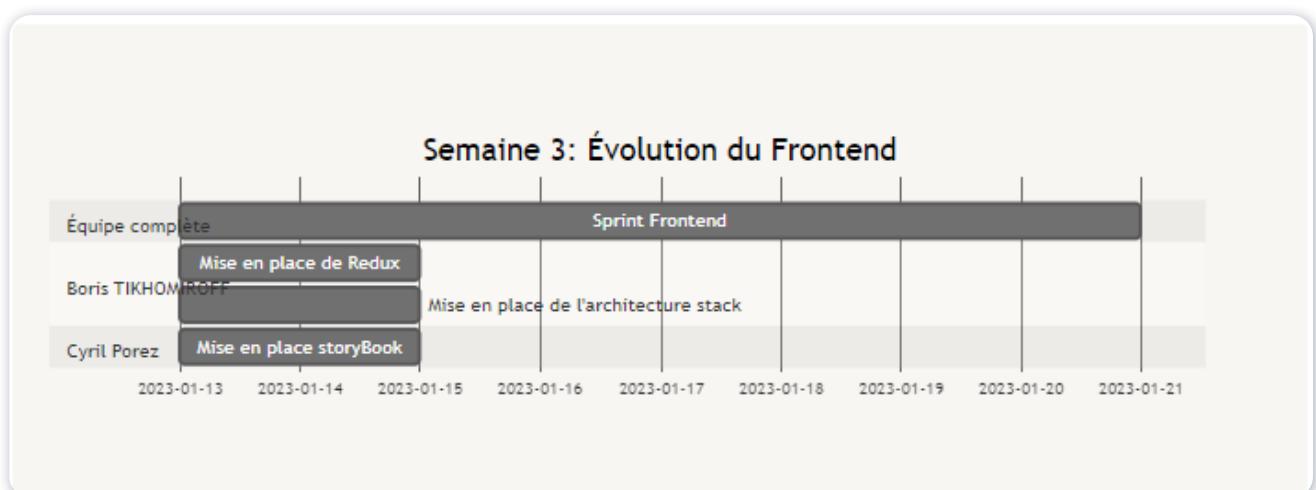
## Deuxième Semaine | Réalisation de L'api



Dans la deuxième semaine nous avons aussi décidé de scindé l'équipe en deux afin de nous assuré une meilleur maniabilité au niveau de l'organisation.

- Après la réalisation du backlog et la compréhension mutuelle des données nécessaires à l'hydratation de notre application nous avons décidé de réalisé un premier sprint afin développer le backEnd de notre application.
- En évaluant cette étape à 3 jours à quatre collaborateurs, nous avons estimé que 2 des collaborateurs de l'équipe pourrait éventuellement prendre de l'avance sur la conception de l'architecture Front-End afin de facilité l'arrivé du sprint de la semaine suivante et les deux autres continué à évalué la fiabilité de l'API existante.

## Troisième semaine | Réalisation des Screens Front-End



Lors de la troisième semaine nous ouvrons les hostilités avec un sprint complet pour l'ensemble de l'équipe. Tandis que 3 s'occupe à développer les screens le dernier organise la mise en place de redux et de l'architecture des stack.

## Quatrième semaine | Fin du sprint et mise en place des webSockets



Nous consacrons la dernière semaine à la finition du développement et la mise en place des webSocket. Un des collaborateurs se lance sur sa réflexion en début de semaine et le reste de l'équipe fini et teste la validité des applicatifs développé.

## Interrogation sur les Designs Patterns & Framework de gestion de projet

### Optimiser la gestion de projet avec une approche agile efficiente

Afin d'organiser au mieux notre projet, il nous a paru essentiel d'adopter un cadre de travail commun qui réduirait les risques tout en favorisant la communication et une gestion optimale des ressources. Les Frameworks Agiles offrent une visibilité, une flexibilité et une transparence qui facilitent la prise de décisions éclairées et la livraison régulière de résultats tangibles. Ils jouent un rôle clé dans la gestion de projet en favorisant l'efficacité, la réactivité et la réussite globale des équipes.

En s'appuyant sur nos connaissances personnelles et notre expérience professionnelle, nous avons entrepris une réflexion collective pour déterminer la méthode la plus appropriée à notre situation spécifique. Face à des contraintes telles qu'un délai très court d'un mois, une équipe composée de quatre développeurs juniors et une hiérarchie homogène sans niveaux distincts, nous avons cherché une approche qui permettrait une collaboration étroite, une prise de décision rapide et une adaptation agile aux changements imprévus.

### La méthode idéale

En prenant en compte les contraintes spécifiques de notre projet, nous avons évalué les avantages et les limitations de différentes approches agiles. Scrum, avec ses itérations courtes appelées «sprints» et sa focalisation sur la collaboration et la transparence, semblait adapté à notre équipe de développeurs juniors. Cepen-

dant, compte tenu de notre hiérarchie homogène et du besoin de flexibilité accrue, nous avons également considéré Crystal Clear, qui se concentre sur la simplicité et l'adaptabilité. En analysant ces facteurs, nous pourrons déterminer quelle approche agile correspond le mieux à notre situation spécifique.

## **Scrum : Une méthode agile puissante, mais avec des limites à considérer**

La méthode Scrum offre plusieurs avantages en favorisant la flexibilité, la collaboration et la transparence. Elle permet de diviser le projet en itérations appelées «sprints», avec des objectifs clairs et des livraisons fréquentes. Cela permet une meilleure adaptation aux changements et une meilleure réactivité aux besoins du client. De plus, les rôles définis dans Scrum favorisent la collaboration et la responsabilisation de l'équipe.

## **Scrum + Crystal Clear : Une combinaison gagnante pour notre projet**

Cependant, par rapport à la méthode Crystal Clear, Scrum peut sembler plus structuré et nécessiter plus de ressources pour sa mise en place. Les cérémonies, les rôles et les artefacts de Scrum peuvent être perçus comme contraignants pour les équipes plus petites et moins expérimentées. De plus, Scrum peut nécessiter une planification plus détaillée et une gestion plus rigoureuse des tâches, ce qui peut être perçu comme une charge supplémentaire dans certaines situations où la simplicité et la flexibilité de Crystal Clear seraient plus adaptées.

## **Détail de notre organisation**

En considérant les différents facteurs qui influencent notre projet, nous avons identifié plusieurs éléments qui guideront notre choix d'approche agile. Tout d'abord, notre proximité physique et la possibilité de travailler ensemble dans la même pièce pendant cette période d'un mois favorisent une communication osmotique, permettant un échange d'informations rapide et continu.

De plus, en adaptant nos propres conventions de travail régulières, nous facilitons la compréhension mutuelle en tirant parti de notre expérience commune en tant que développeurs juniors. L'amélioration réflexive est également un atout précieux, car elle nous encourage à réfléchir collectivement à nos pratiques et à apporter des ajustements réguliers. Les réunions quotidiennes (daily meetings) nous permettent de faire le point chaque jour sur nos avancées, d'identifier les éventuels problèmes et de trouver des solutions rapidement.

Enfin, l'utilisation d'un backlog nous permet de définir facilement nos priorités et de prendre des décisions en fonction de l'évolution du projet au fil du temps. Tous ces éléments contribuent à créer un environnement propice à la collaboration et à l'efficacité, ce qui sera pris en compte lors de notre choix.

## **Le rôle du Scrum Master et la rotation au sein de l'équipe**

L'une des particularités de notre approche de gestion de projet est la rotation du rôle de Scrum Master au sein de l'équipe. Chaque membre de l'équipe a l'opportunité de prendre en charge ce rôle, ce qui favorise une meilleure compréhension de la méthodologie Scrum et permet à chacun de développer ses compétences en leadership et en gestion de projet.

## **Le rôle du Scrum Master et la rotation au sein de l'équipe**

L'une des particularités de notre approche de gestion de projet est la rotation du rôle de Scrum Master au sein de l'équipe. Chaque membre de l'équipe a l'opportunité de prendre en charge ce rôle, ce qui favorise une meilleure compréhension de la méthodologie Scrum et permet à chacun de développer ses compétences en leadership et en gestion de projet.

## **Organisation interne pour favoriser la collaboration et l'efficacité**

Afin de maximiser la collaboration et l'efficacité au sein de l'équipe, nous avons décidé de fragmenter les équipes de développement de manière à ce que certains membres puissent se concentrer sur la recherche de solutions tandis que d'autres continuent à avancer sur les sprints. Cette approche nous permet de résoudre rapidement les problèmes et d'optimiser notre progression tout en maintenant un flux de travail fluide.

## **Les Daily Meetings : Une communication fluide et une coordination efficace**

Les Daily Meetings jouent un rôle essentiel dans notre approche de gestion de projet. Ces rencontres quotidiennes de 15 minutes nous permettent de faire le point sur nos avancées, de partager nos réussites et de discuter des obstacles éventuels. Grâce à ces réunions régulières, nous favorisons une communication fluide, une coordination efficace et une meilleure visibilité sur l'état d'avancement du projet. Cela nous permet d'identifier rapidement les éventuels problèmes et de prendre des mesures correctives de manière proactive.

En combinant les Daily Meetings pour une communication fluide et la communication osmotique pour surmonter les défis de la fatigue et de la compréhension, nous renforçons notre capacité à travailler de manière collaborative et à maintenir une compréhension globale du projet, même dans des conditions exigeantes.

## **La Cérémonie du Backlog pour ajuster les priorités**

Chaque début de sprint, nous organisons la Cérémonie du Backlog, une réunion où nous passons en revue les tâches à accomplir et redéfinissons les priorités en fonction de notre progression. Cette approche nous permet d'ajuster nos objectifs et de prendre des décisions éclairées pour maximiser notre efficacité tout au long du projet.

## **La rétrospective hebdomadaire pour tirer les leçons**

En fin de semaine, nous nous réunissons pour la rétrospective hebdomadaire. Cette séance nous permet de faire le point sur ce qui a été réalisé, les obstacles rencontrés et les leçons apprises. En réfléchissant sur notre travail, nous identifions les améliorations à apporter et mettons en place des actions correctives pour optimiser notre travail dans les prochaines étapes.

## **Des sprints d'une semaine pour une progression itérative**

Nous avons adopté des sprints d'une semaine pour structurer notre projet. Chaque semaine est dédiée à une phase spécifique, allant de la conception au développement back-end, puis au développement front-end, suivi de tests poussés et d'améliorations. Cette approche itérative nous permet de progresser de manière incrémentale tout en restant réactifs aux changements et ajustements nécessaires.

## **Une compréhension partagée du backlog et des fonctionnalités clés**

Lors du démarrage du projet, nous avons consacré du temps à nous assurer que le backlog était bien compris par tous les membres de l'équipe. Nous avons clarifié les objectifs, défini les fonctionnalités clés et partagé une vision commune pour garantir une compréhension par tous de la feuille de route du projet.

## **Les User Stories : Une approche centrée sur les besoins des utilisateurs**

Les User Stories constituent un élément essentiel de notre méthode de gestion de projet. Elles nous permettent de décomposer les fonctionnalités de notre application en scénarios utilisateurs clairs et compréhensibles. Lors de la conception de nos User Stories, nous nous assurons de les découper en fonctionnalités cohérentes et indépendantes, afin de faciliter la planification et l'exécution des tâches. Chaque User Story est accompagnée de critères d'acceptation clairs et mesurables, permettant ainsi de valider facilement son implémentation.

## **Les User Stories : Une approche centrée sur les besoins des utilisateurs**

Les User Stories constituent un élément essentiel de notre méthode de gestion de projet. Elles nous permettent de décomposer les fonctionnalités de notre application en scénarios utilisateurs clairs et compréhensibles. Lors de la conception de nos User Stories, nous nous assurons de les découper en fonctionnalités cohérentes et indépendantes, afin de faciliter la planification et l'exécution des tâches. Chaque User Story est accompagnée de critères d'acceptation clairs et mesurables, permettant ainsi de valider facilement son implémentation.

## **Alignment sur l'environnement de travail : une approche concertée pour un développement harmonieux**

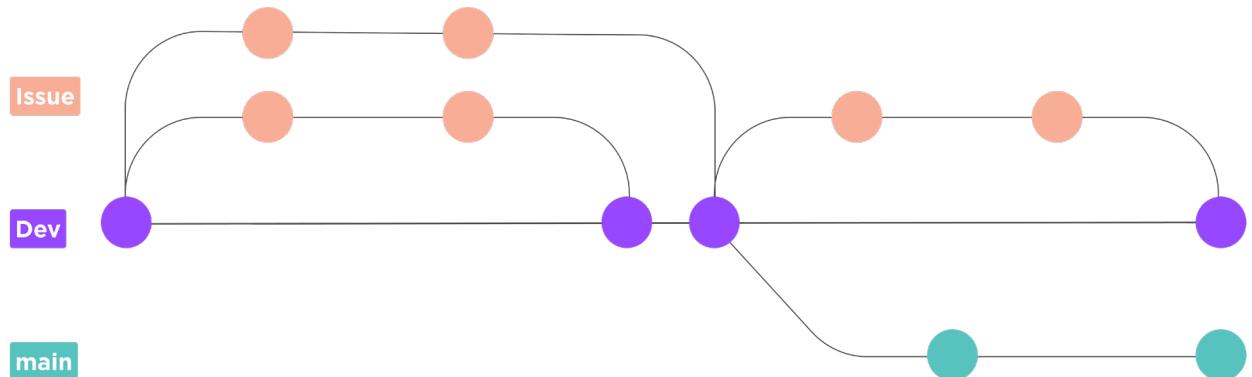
Nous avons opté pour un environnement de développement basé sur Windows, en utilisant Visual Studio Code comme IDE principal. Pour gérer notre base de données SQL. Pour interagir avec la base de données, nous avons utilisé phpMyAdmin, une interface conviviale qui nous a permis d'exécuter des requêtes et de gérer les tables.

Afin de maintenir la cohérence entre les membres de l'équipe, nous avons utilisé Git comme système de contrôle de version. Chaque membre disposait du même fichier package.json et package-lock.json, qui spécifient les dépendances et les versions des packages utilisés dans le projet. Cela nous a permis de garantir la reproductibilité de l'environnement de développement et d'éviter les problèmes liés aux incompatibilités de versions.

Il était également important de maintenir une approche collaborative lors des mises à jour des librairies. Aucune mise à jour n'était effectuée sans le consentement de tous les membres de l'équipe. Cela nous permettait de prévenir les éventuelles incompatibilités et de maintenir la stabilité du code tout au long du développement.

Afin de maximiser le temps de développement, nous avons adopté un workflow basé sur les principes du «Trunk-Based Development» (TBD). Ce workflow simplifie l'accès aux branches en utilisant une branche commune sur laquelle nous créons des branches spécifiques pour chaque issue. Lorsqu'un problème est résolu, une demande de fusion (Merge Request) est créée sur la branche commune, et tous les membres du groupe agissent en tant que relecteurs. Tous les commentaires, demandes ou remarques doivent être résolus avant la fusion sur la branche commune. Dans notre cas, nous avons désigné la branche «Dev» comme branche commune, en laissant inactive jusqu'au premier déploiement.

## Illustration du workflow Trunk-based Development



source: <https://www.toptal.com/software/trunk-based-development-git-flow>

En mettant en place ces différentes approches dans notre organisation interne, nous visons à optimiser notre gestion de projet, à favoriser une communication fluide, à structurer notre travail de manière itérative et à maximiser notre efficacité tout en restant flexibles face aux éventuels ajustements nécessaires

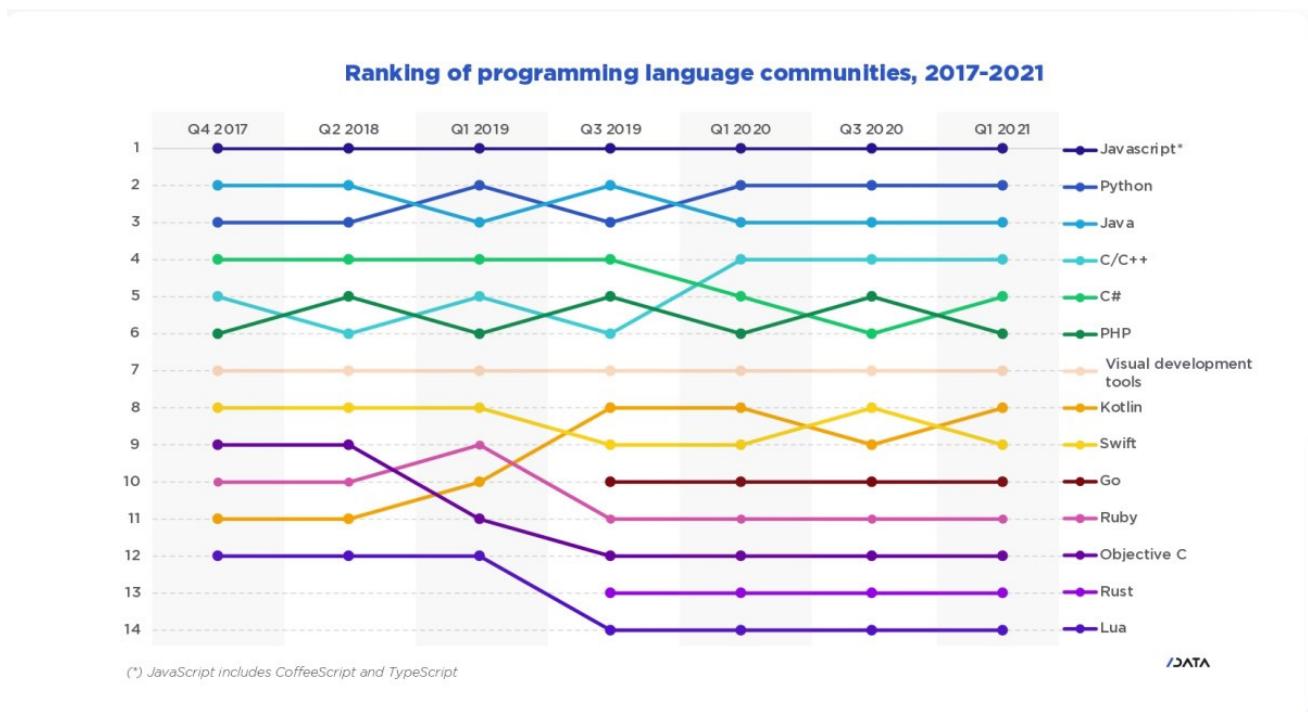
## **Aligner les technologies : Le rôle crucial des décisions de stack**

Pour une équipe de développeurs telle que la nôtre, le choix de la stack est une décision stratégique. Il s'agit de trouver le bon équilibre entre la familiarité des membres de l'équipe avec les technologies sélectionnées, la compatibilité avec les besoins spécifiques du projet, ainsi que les avantages et les inconvénients des différentes options disponibles sur le marché.

Une stack technologique bien alignée permet une meilleure collaboration entre les développeurs, facilite la communication et favorise l'échange d'informations. Elle permet également de capitaliser sur l'expertise préexistante de l'équipe, d'accélérer le développement grâce à des outils et des frameworks bien établis, et d'assurer la pérennité du projet en choisissant des technologies adaptées et évolutives.

## **JavaScript | une communauté active comme levier de production**

Notre équipe possède une sensibilité naturelle envers JavaScript, un langage de programmation populaire et largement adopté par plus de 14 millions de développeurs à travers le monde. Cette popularité se traduit par une communauté dynamique et active, offrant un soutien précieux et des ressources abondantes. Nous avons constaté que cette communauté nous permet d'accéder rapidement à des solutions et des bonnes pratiques, ce qui est essentiel pour résoudre efficacement les problématiques rencontrées lors du développement d'une application.



## React Native | une librairie modulaire cross-plate-forme

De plus, notre équipe avait déjà acquis une solide expérience dans l'utilisation de React. Certains membres de l'équipe, comme Ahmed Magassouba et Cyril Porez, avaient déjà collaboré sur un projet de développement d'une PWA lors du [projet liberty](#). Dans ce même projet j'avais à charge le développement d'une application en React Native. Ces expériences positives ont renforcé notre confiance dans les capacités de React à fournir une architecture solide et modulaire pour notre application.

C'est ainsi que nous avons pris la décision éclairée de choisir React Native comme solution de développement cross-plateforme. En capitalisant sur notre expertise préexistante en React, nous avons trouvé en React Native un cadre idéal pour créer des applications mobiles natives pour iOS et Android en utilisant un code JavaScript commun. Cette approche nous permet de maximiser l'efficacité du développement, en réduisant les efforts nécessaires pour créer et maintenir deux bases de code distinctes pour chaque plateforme.

## Déployer rapidement avec Expo

Pour faciliter davantage le développement avec React Native, nous avons également opté pour l'utilisation d'Expo. Cette plateforme apporte une couche d'abstraction supplémentaire en fournissant un ensemble d'outils et de fonctionnalités prêts à l'emploi, simplifiant ainsi des tâches courantes telles que la gestion des ressources, la compilation et le déploiement de l'application. Grâce à Expo, nous pouvons nous concentrer davantage sur la logique métier de notre application, sans nous soucier des aspects plus techniques et fastidieux de la configuration initiale.

En résumé, en tenant compte de notre sensibilité avec JavaScript, de notre expérience positive avec React et de notre volonté de développer une application multiplateforme de manière efficace, ces choix nous paraissaient judicieux. Cette combinaison nous a offert la possibilité de tirer parti des avantages du développement avec JavaScript, de bénéficier d'une large communauté de développeurs et d'accélérer le processus de développement en utilisant une base de code commune pour les plateformes iOS et Android.

## Architecture backend : Choix stratégiques pour un développement performant

### Le choix de Node.js et l'asynchronicité pour une gestion efficace des opérations simultanées

Dans notre quête d'un environnement de développement cohérent et performant, nous avons fait le choix judicieux d'utiliser Node.js comme technologie backend. Cette décision s'inscrit dans la continuité de notre choix sur JavaScript pour le développement frontend, ce qui facilite grandement la collaboration au sein de notre équipe.

L'un des avantages majeurs de Node.js réside dans sa gestion asynchrone des entrées-sorties. Grâce à cette caractéristique, notre application peut traiter efficacement les opérations simultanées et les requêtes parallèles, ce qui est particulièrement important pour notre projet de chat où la fluidité et la réactivité sont essentielles. Node.js nous permet d'offrir une expérience utilisateur fluide en gérant simultanément un flux continu de messages et une interaction en temps réel entre les utilisateurs.

### Express : Un framework web minimaliste pour une mise en place rapide d'une API REST

Pour faciliter le développement de notre backend, nous avons choisi Express, un framework web minimaliste et flexible pour Node.js. Express simplifie considérablement la création d'une API REST en fournissant des fonctionnalités prêtes à l'emploi pour le routage, la gestion des requêtes et des réponses, ainsi que la mise en place d'une architecture multicouche.

L'utilisation d'une architecture multicouche est cruciale pour assurer une séparation claire des responsabilités et faciliter la maintenance et l'évolutivité de notre application. Grâce à Express, nous pouvons découpler les différentes fonctionnalités de notre système en modules distincts, ce qui offre une meilleure modularité et facilite les mises à jour et les améliorations futures tout en respectant les principes REST.

## Les avantages de l'API REST : Flexibilité et intégration aisée

En adoptant une approche orientée ressources avec une API REST, nous créons une interface bien définie pour interagir avec notre application. Cette approche offre une flexibilité et une extensibilité accrues, permettant une intégration fluide avec d'autres systèmes ou services externes. Grâce à cette architecture multicouches, nous pouvons facilement évoluer et ajouter de nouvelles fonctionnalités à notre application, tout en maintenant une base solide et cohérente.

Cette approche repose sur X principes fondamentaux :

- **Client-serveur** : Les responsabilités sont séparées entre le client et le serveur.
- **stateless** : La communication client-serveur s'effectue sans conservation de l'état de la session côté back-end.
- **Gestion de mise en cache** : améliore les performances en évitant aux clients de récupérer des données obsolètes, augmentant ainsi l'extensibilité du système.
- **Modèle de système multicouches** : permet d'affecter des ressources à des couches de fonctionnalité pour des capacités de service partagées et un traitement complexe des données.
- **Interface uniforme** :

**Identification des ressources dans les requêtes** : Les ressources sont identifiées par des URI dans les requêtes. Les représentations renvoyées peuvent varier (HTML, XML, JSON) tout en étant distinctes des ressources elles-mêmes.

**Manipulation des ressources par des représentations** : Chaque représentation d'une ressource fournit suffisamment d'informations au client pour modifier ou supprimer la ressource

**Messages auto-descriptifs** : Chaque message contient assez d'information pour savoir comment l'interpréter. Par exemple, l'interpréteur à invoquer peut être décrit par un type de médias.

**Hypermédia comme moteur d'état de l'application** : le client doit être en mesure d'utiliser dynamiquement les hyperliens fournis par le serveur pour découvrir toutes les autres actions possibles et les ressources dont il a besoin pour poursuivre la navigation.

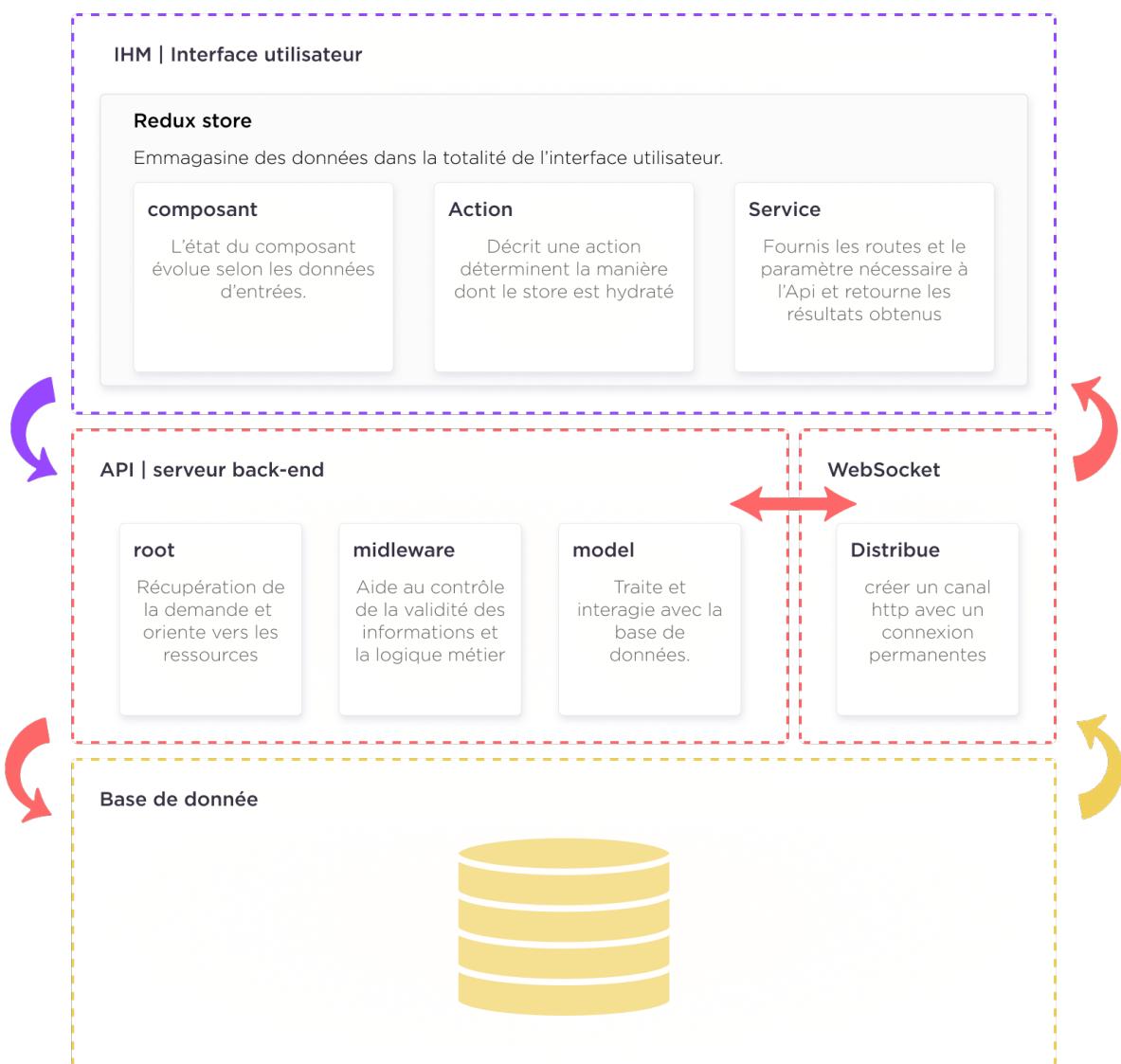
## Les WebSockets : Une communication en temps réel pour une expérience interactive

Dans notre projet, nous avons choisi d'intégrer les WebSockets pour leur utilité essentielle. Ces derniers jouent un rôle central dans notre application de chat en permettant une communication en temps réel et une interaction fluide entre les utilisateurs. Grâce à cette technologie, nous avons établi une connexion persistante

entre le serveur et l'application, ce qui nous permet de mettre à jour instantanément les messages sans avoir besoin que le récepteur envoie une requête pour les réceptionner. Cette approche offre une expérience immersive et interactive, où les conversations se déroulent de manière dynamique et réactive. De plus, les WebSockets optimisent les performances en réduisant la consommation de bande passante, assurant ainsi une fluidité de communication sans compromis. En intégrant les WebSockets à notre projet React Native, nous avons créé un environnement de discussion en ligne véritablement engageant et captivant.

## Architecture générale de notre application Mobile

Nous avons opté pour une architecture bien définie, séparant clairement les responsabilités entre le serveur et l'interface utilisateur. Cette approche nous offre une meilleure gestion de chaque élément et permet également à notre serveur d'être consulté par l'interface administrateur en version web.



## Pourquoi les webSockets hors de l'API ?

Mettre les websockets en dehors de l'API REST peut être une bonne pratique dans certaines situations. Voici quelques raisons pour lesquelles cela peut être bénéfique :

1. Séparation des responsabilités : Les websockets sont souvent utilisées pour des fonctionnalités en temps réel telles que les notifications en direct, le chat en temps réel, etc. Ces fonctionnalités ont souvent des exigences spécifiques en termes de performances et de temps de réponse. En les séparant de l'API REST, vous pouvez optimiser l'architecture et les ressources pour répondre à ces besoins spécifiques.
2. Scalabilité : Les websockets peuvent nécessiter une mise à l'échelle différente de l'API REST. En les isolant, vous pouvez les mettre à l'échelle indépendamment de l'API REST, ce qui permet une meilleure gestion des charges et une évolutivité plus efficace.
3. Gestion des connexions persistantes : Les websockets nécessitent des connexions persistantes entre le serveur et le client. Cela peut être géré plus efficacement en utilisant une infrastructure dédiée pour les websockets plutôt que de les intégrer directement à l'API REST

## Sélectionner la base de données optimale entre SQL et MongoDB

### SQL vs. MongoDB : Un dilemme complexe

Lorsqu'il s'agit de choisir la base de données pour notre projet, nous nous sommes retrouvés confrontés à un dilemme complexe : SQL ou MongoDB ? Chacune de ces options présente des avantages et des inconvénients uniques, ce qui a rendu notre décision difficile.

#### SQL : La puissance de la relation

SQL, une base de données relationnelle bien établie, offre une approche solide pour modéliser des données complexes. Sa structure tabulaire et ses relations pré-définies garantissent la cohérence des données et facilitent la gestion des informations structurées. En exploitant la puissance de SQL, nous pouvons effectuer des requêtes avancées et optimisées pour des performances élevées.

#### MongoDB : Souplesse et scalabilité

MongoDB, une base de données NoSQL, se démarque par sa flexibilité et sa scalabilité. En utilisant un modèle de données basé sur des documents JSON, il permet de stocker et de gérer facilement des structures de données complexes. De plus, sa conception orientée document permet une évolutivité horizontale, ce qui est essentiel pour un projet en pleine croissance.

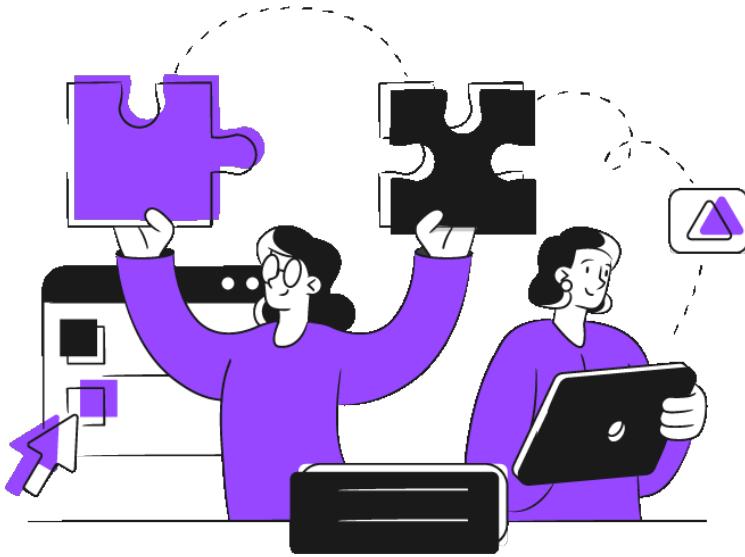
## **Les choix cornéliens : Avantages et inconvénients**

Chacun de ces choix présente des avantages et des inconvénients à prendre en considération. SQL offre un écosystème mature avec un large soutien communautaire et une grande stabilité. En revanche, MongoDB offre une gestion souple des schémas et une évolutivité horizontale, ce qui en fait un choix attrayant pour les projets nécessitant une grande flexibilité.

## **Notre choix réfléchi : MySQL**

Après une évaluation approfondie, nous avons finalement opté pour SQL pour notre projet. Sa structure relationnelle correspondait parfaitement à nos besoins en matière de gestion de données structurées. De plus, ses fonctionnalités avancées, telles que les requêtes complexes et les performances optimisées, nous ont convaincus de choisir cette solution éprouvée.

Nous avons notamment opté pour l'utilisation de MySQL comme SGBD, principalement en raison de sa nature totalement open source et gratuite. En plus d'être disponible sans coût, MySQL offre d'excellentes performances, notamment grâce à son architecture multi-threadée et multi-utilisateurs. Cette caractéristique nous a permis de gérer efficacement les requêtes concurrentes et de garantir des performances optimales pour notre application.



## Modélisation de la base de donnée

Notre choix s'étant porté sur MySQL, il nous paraissait judicieux de s'employer à concevoir la structure de notre base de donnée de manière à définir clairement ses entités et les relations pouvant y être exploité. Nous avons choisi de suivre le méthode Merise mis en place dans les année 1970 par René Colletti, Arnold Rochfeld et Hubert Tardieu.

## Objectifs de la définition générale du système

Nous avons déterminé dix objectifs nous permettant par la suite de définir les fondations du système avant de passer à la modélisation détaillée.

1. **Gestion efficace des utilisateurs :** Le système doit permettre la création, la modification et la suppression des utilisateurs de manière efficace, tout en assurant la sécurité des informations personnelles et en respectant les règles d'authentification.
2. **Organisation des utilisateurs en channels :** Le système doit permettre de regrouper les utilisateurs en fonction de leurs rôles et de leurs relations, facilitant ainsi la gestion des permissions, des communications et des interactions au sein de ces groupes.
3. **Communication fluide :** Le système doit faciliter la communication entre les utilisateurs au sein des groupes en permettant l'échange de messages, la visualisation des conversations.
4. **Gestion des autorisations et des rôles :** Le système doit prendre en charge la gestion des autorisations et des rôles pour contrôler l'accès aux fonctionnalités et aux informations sensibles, garantissant ainsi la confidentialité et la sécurité des données.

5. **Flexibilité et évolutivité :** Le système doit être conçu de manière à pouvoir s'adapter facilement aux besoins changeants de l'organisation, en permettant l'ajout de nouveaux utilisateurs, de nouveaux groupes et de nouvelles fonctionnalités sans compromettre la stabilité et la performance.
6. **Intégration avec d'autres systèmes :** Le système peut nécessiter une intégration avec d'autres systèmes existants, tels que des systèmes de messagerie ou des bases de données externes, afin de faciliter l'échange d'informations et d'assurer une expérience utilisateur transparente.
7. **Convivialité :** Le système doit être convivial et intuitif, offrant une interface utilisateur claire et facile à utiliser, afin de favoriser l'adoption par les utilisateurs et de réduire la courbe d'apprentissage.
8. **Sécurité des données :** Le système doit garantir la sécurité des données, en mettant en œuvre des mécanismes de protection tels que le chiffrement des données sensibles.
9. **Performance et fiabilité :** Le système doit être performant et fiable, offrant des temps de réponse rapides, une disponibilité élevée et la capacité de gérer de grandes quantités de données et de trafic utilisateur sans compromettre les performances.
10. **Simplicité de prise en main :** Le système doit accès la rédactions au plus simple. L'utilisateurs doit avoir accès aux informations selon le stricte nécessité pour une contrainte de minimalité.

## Analyse et conception des entités : Les fondamentaux de notre base de donnée

La conception de base de données est cruciale pour un système d'information performant, fiable et sécurisé. Elle commence par l'identification précise des entités, objets essentiels qui stockent et organisent les données. Dans notre application chat, nous avons identifié trois entités clés : **User**, **Channel** et **Conversation**, chacune jouant un rôle spécifique. En structurant la base de données de manière logique et cohérente, nous garantissons un fonctionnement optimal.

### Présentation de nos entités clefs



**User** | Cette entité et la représentation de notre utilisateurs.

*Chaque utilisateur est une entité distincte avec des attributs tels que le nom, l'adresse e-mail et le numéro de téléphone. Les utilisateurs sont au cœur de notre application, car ils sont la raison pour laquelle les conversations ont lieu. Ils peuvent converser avec une ou plusieurs personnes présentes dans l'application.*



**Channel** | Cette entité représente une collectivité d'utilisateurs regroupé sous un même nom.

*Un canal permet de circonscrire une communauté spécifique au sein de l'application. Les canaux facilitent la communication et l'interaction entre les utilisateurs partageant des intérêts communs présentes dans l'application.*



**Conversation** | Cette entité représente la transmission d'un message écrit.

*Elle permet de conserver un message pouvant être lu par les utilisateurs d'un même canal. Les conversations sont essentielles pour permettre aux utilisateurs de communiquer entre eux*

L'analyse des entités révèle des interactions clés : les utilisateurs participent à des canaux et s'engagent dans des conversations. Cette relation entre les entités est cruciale pour une base de données cohérente et efficace. Les canaux regroupent des utilisateurs partageant des intérêts communs, tandis que les conversations facilitent les échanges. Une approche relationnelle permet d'organiser les données de manière optimale. Le Modèle Conceptuel de Base de Données (MCD) permettra de définir ces interactions et les attributs nécessaires pour caractériser les entités de manière optimale.

## Définition détaillé de nos entités clefs



### User

- **firstname** : Prénom de l'utilisateur.
- **lastname** : Nom de famille de l'utilisateur.
- **email** : Adresse e-mail de l'utilisateur, agent de connexion.
- **phone** : Numéro de téléphone de l'utilisateur.
- **role** : Rôle de l'utilisateur dans le système.
- **password** : Mot de passe de l'utilisateur.
- **createdAt** : Date de création de l'utilisateur.
- **updatedAt** : Date de mise à jour de l'utilisateur.
- **deletedAt** : Date de suppression de l'utilisateur (utilisé pour la suppression logique).



### User

- **name** : Nom du canal.
- **createdAt** : Date de création de l'utilisateur.
- **updatedAt** : Date de mise à jour de l'utilisateur.
- **deletedAt** : Date de suppression de l'utilisateur (utilisé pour la suppression logique).



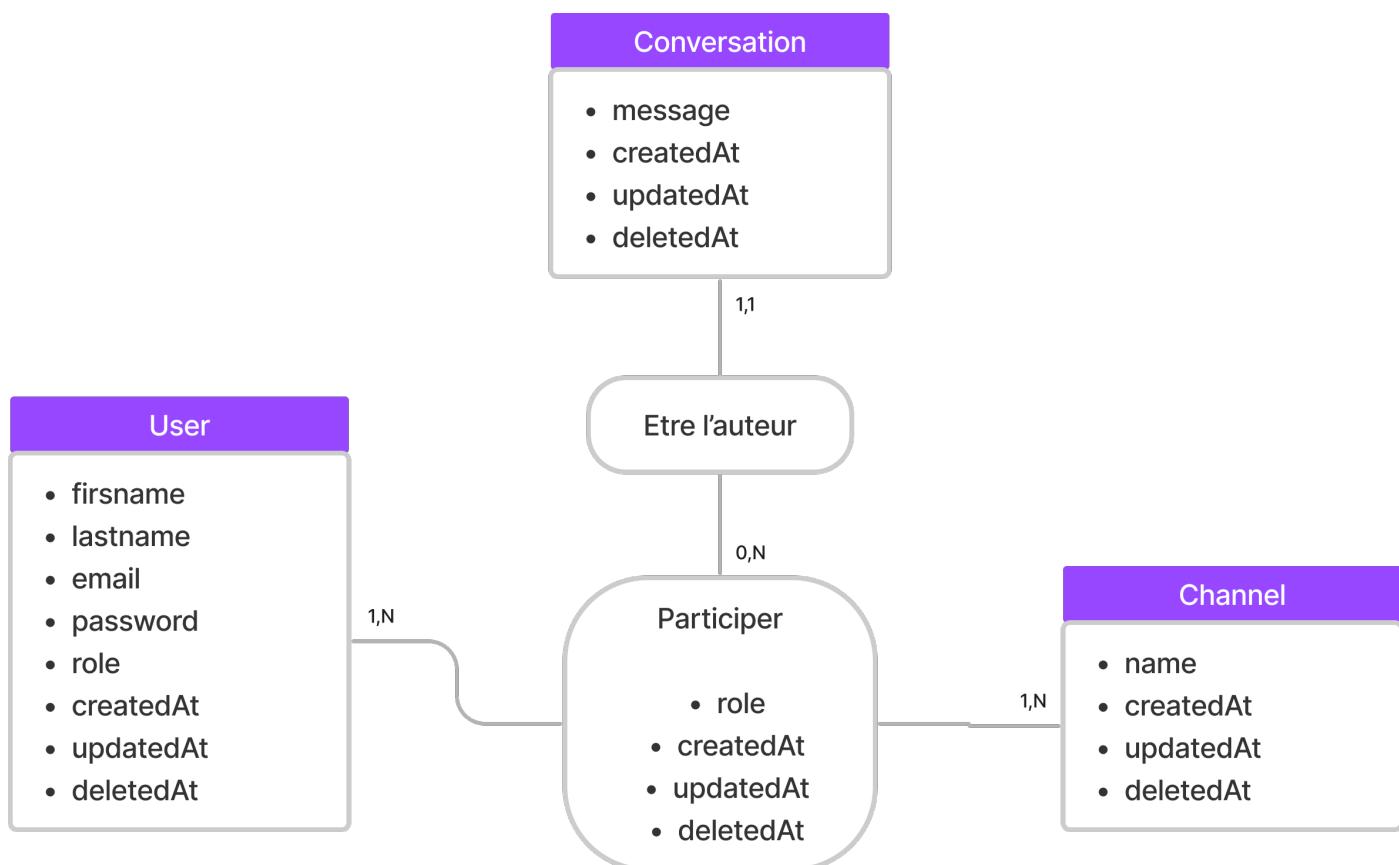
## Conversations

- **message** : Message de la conversation.
- **createdAt** : Date de création de l'utilisateur.
- **updatedAt** : Date de mise à jour de l'utilisateur.
- **deletedAt** : Date de suppression de l'utilisateur (utilisé pour la suppression logique).

# MCD | Modèle Conceptuel de Base de Donnée

Une fois que nous avons établi les informations nécessaires au sein de notre équipe, nous avons pu réfléchir aux relations pour assurer le bon fonctionnement de notre application. En suivant les bonnes pratiques de normalisation, nous avons cherché à limiter la multiplication des clés étrangères au sein de nos tables afin de faciliter les relations entre elles.

Dès le départ, nous avons identifié une relation Many-to-Many entre nos entités utilisateurs et channels. En effet, plusieurs utilisateurs peuvent participer à un canal, tandis qu'un utilisateur peut faire partie de plusieurs canaux. Cette constatation nous a conduit à envisager la création d'une table relationnelle qui recenserait les associations entre utilisateurs et canaux. Cette nouvelle entité, appelée «participant», pourrait également être en relation avec les conversations. Ainsi, un participant pourrait être l'auteur de plusieurs message (conversations), établissant ainsi une relation one-to-many.



Modèle Conceptuel de la base de donnée de l'application sent

## Dans le modèle suivant nous constatons que :

1. Chaque utilisateur (User) peut participer à un ou plusieurs canaux (Channel), représenté par la relation «PARTICIPER» avec une cardinalité de (1,N) du côté de User et (1,N) du côté de Channel.
2. Chaque canal peut avoir la participation de un ou plusieurs utilisateurs.
3. Chaque participation (Participer) peut être l'auteur de zéro ou plusieurs messages (Conversation), représenté par la relation «Être l'auteur» avec une cardinalité de (0,N) du côté de Participer et (1,1) du côté de Conversation.
4. La relation «Participer» est une relation d'association entre User et Channel, représentant la participation d'un utilisateur à un canal.
5. La relation «Être l'auteur» est une relation d'association entre Participer et Conversations, représentant le fait qu'un participant peut être l'auteur de plusieurs conversations.

Il est important de noter que la table «Participer» agit comme une table de jonction qui enregistre les relations entre les utilisateurs, les canaux et les conversations. Elle permet de gérer la relation many-to-many entre les utilisateurs et les canaux, ainsi que la relation one-to-many entre les participants et les conversations.

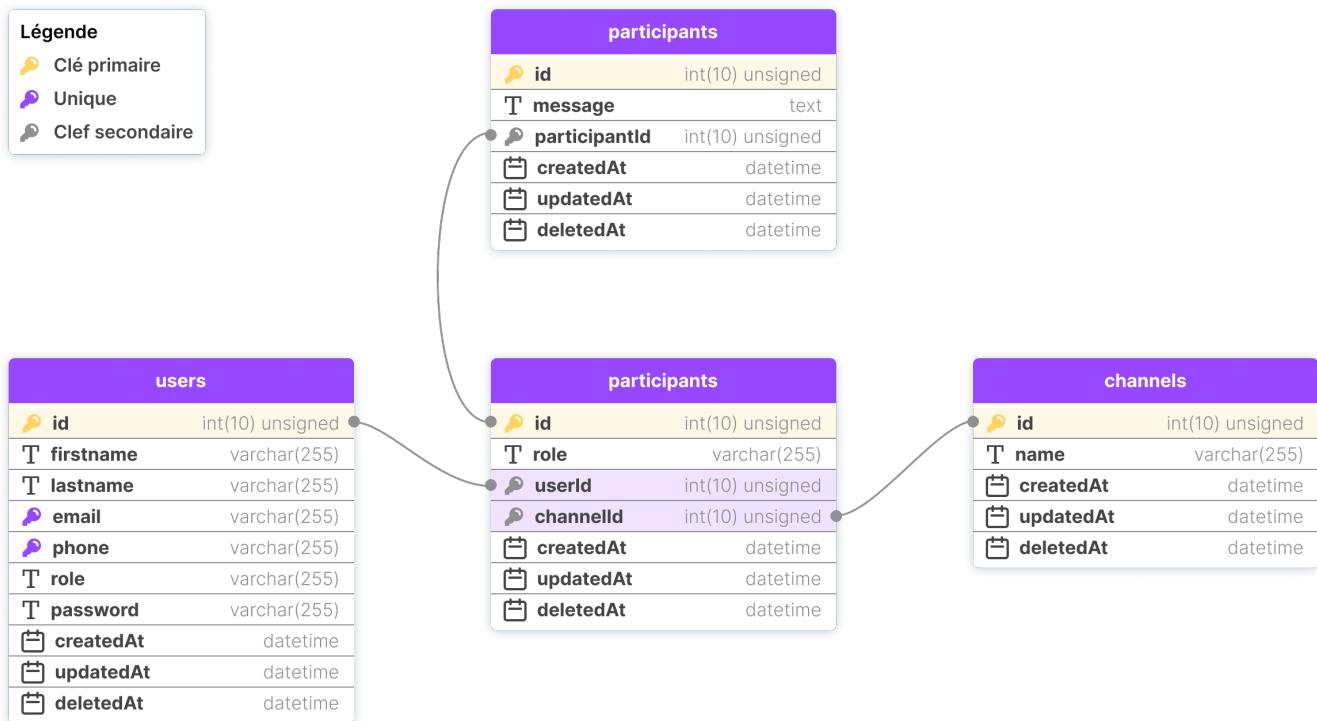


### A noter

- Nous avons choisi d'insérer directement l'utilisateur en tant que participants sur le canal général lors de son inscription afin d'assurer qu'il y ait toujours un canal associé à cet utilisateur dès le départ. Cela garantit que chaque utilisateur dispose d'au moins un canal auquel il peut participer dès son inscription.
- Il est techniquement possible qu'un utilisateur soit la seule personne liée à un canal. Cependant l'interface utilisateur obligera l'utilisateur à sélectionner au moins un autre utilisateur.
- - le terme conversation est discutable. Au-delà de l'approche sémantique, le terme message permettrait une compréhension plus aisée à son usage.

# MLD | Modèle logique de base de donnée

Dans le chapitre précédent, nous avons analysé les entités clés de notre application de chat : User, Channel et Conversation. Nous avons identifié leurs attributs et interactions, et construit un MCD. Maintenant, dans le chapitre suivant, nous nous concentrerons sur le modèle logique de base de données. Nous définissons les tables, les clés primaire et secondaire et les relations entre les entités pour structurer notre base de données. Ce chapitre traduit les concepts du MCD en une structure concrète de tables et de relations pour optimiser notre système d'information.



Modèle logique de la base de donnée de l'application sent

## Clés primaires et secondaires

- Les clés primaires sont utilisées pour identifier de manière unique chaque enregistrement dans une table. Dans notre base de données, les clés primaires des tables «users», «channels», «participants» et «conversations» sont de type INT(10) UNSIGNED. Soit des nombre entier à hauteur maximal de 4 294 967 295 sans possibilité de valeur négatif pour conserver une cohérences.
- Les clés secondaires sont des clés étrangères qui établissent des relations entre les tables. Dans notre base de données, les colonnes «userId» et «channelId» dans la table «participants» et «participantId» dans la table «conversations» sont des clés étrangères.

# Contraintes Spécifiques liées aux relations

Nous avons décidé de conserver les contraintes en cascade lors de la modification des clefs primaires et nous nous assurons que les suppressions des messages se font également en cascade si nécessaire.

## **participants\_ibfk\_1 :**

Action ON DELETE : CASCADE  
Action ON UPDATE : CASCADE  
Colonne : userId  
Table : users  
Colonne : id

## **conversations\_ibfk\_1 :**

Action ON DELETE : RESTRICT  
Action ON UPDATE : RESTRICT  
Colonne : participantId  
Table : participants  
Colonne : id

## **participants\_ibfk\_2 :**

Action ON DELETE : CASCADE  
Action ON UPDATE : CASCADE  
Colonne : channelId  
Table : channels  
Colonne : id

## Utilisation du champ «**deletedAt**»

A cela s'ajoute le champ '**deletedAt**' qui, s'il est défini comme NON NULL, permet de filtrer les enregistrements de la base de données afin qu'ils ne soient pas forcément récupérés par les requêtes dans le cas de l'usage de notre ORM. L'ajout de ce champ nous offre une plus grande flexibilité dans la gestion de la suppression des comptes utilisateurs, en utilisant des conditions logiques. Cette approche nous permet de conserver différentes possibilités lors de la suppression de messages par un modérateur ou lorsque qu'un participant quitte un groupe

## Normalisation et choix de structuration du MLD

Lors de la conception de notre modèle logique de base de données (MLD), nous avons appliqué les principes de la normalisation 3NF pour réduire les redondances et les dépendances fonctionnelles. La relation entre les tables «Channel» et «User» a été normalisée en introduisant la table intermédiaire «Participant». Cette approche nous permet de séparer les informations spécifiques aux participants d'un canal, telles que leur rôle, dans une table distincte. Ainsi, chaque attribut de la table «Participant» dépend uniquement de la clé primaire de cette table, évitant ainsi les dépendances fonctionnelles indésirables.

Notre choix de normaliser la relation entre les tables «Channel», «User» et «Participant» en respectant les principes de la **3NF** nous permet de réduire les redondances, les anomalies et d'optimiser les performances de notre système d'information.

Une approche clé dans notre processus de conception a été la collaboration étroite avec l'équipe de maquettage. Le zoning et les wireframes ont été élaborés simultanément à la réflexion sur le MCD et le MLD. Cette approche a validé la conformité entre les équipes, assurant une cohérence fonctionnelle et interactions entre les entités.

Le zoning a joué un rôle crucial en fournissant des indications claires sur le comportement attendu de la base de données et des entités. Il a permis d'affiner notre modèle et de répondre efficacement aux besoins des utilisateurs. En combinant une réflexion approfondie sur la base de données avec le maquettage de l'application, nous développons une solution répondant aux besoins des utilisateurs tout en assurant une gestion efficace des données.



## Maquettage de l'application

Dans ce chapitre, nous explorerons le maquettage de l'application, soulignant son importance pour la visualisation et la validation des fonctionnalités du système. Nous mettrons en évidence le lien étroit entre le maquettage et la conception de la base de données, montrant comment ces deux aspects se complètent pour créer un système performant, convivial et cohérent.

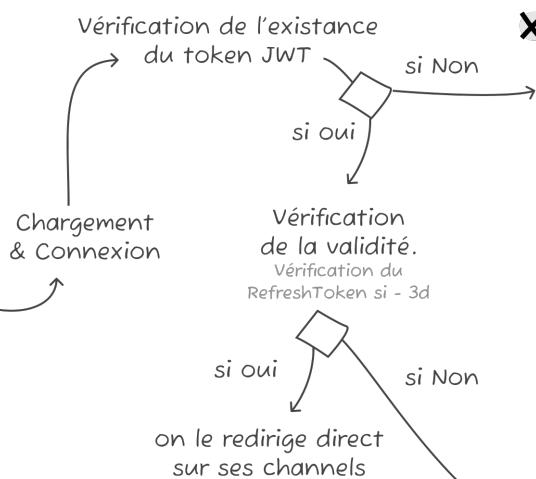
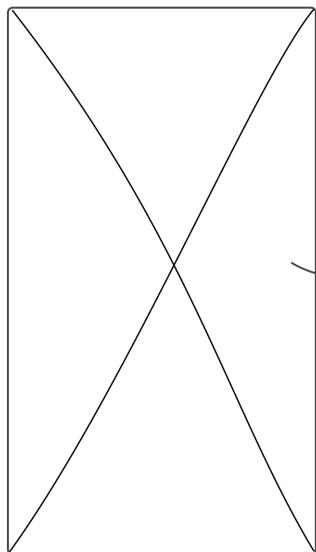
### Un support intuitif qui se révèle être un formidable outil technique

Le zoning a joué un rôle clé dans notre démarche de conception. A l'aide de tableaux blancs nous avons testé la structure de différent écrans, en cherchant à trouver un équilibre entre élégance et faisabilité technique. Tout en veillant à maintenir l'intégrité des données, nous nous sommes posé des questions sur la navigation, la composition des différents écrans et l'ordre dans lequel ils devaient apparaître.

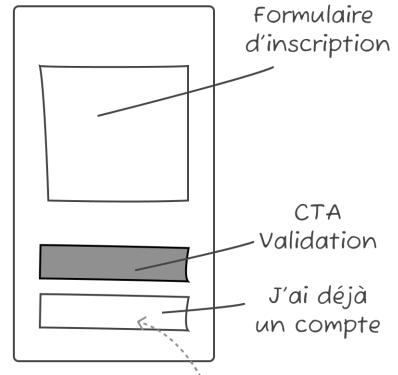
L'objectif était de créer une expérience utilisateur fluide et cohérente, en alignant le maquettage avec la réflexion sur la base de données. Toujours dans un accord de résilience et de minimalité, nous voulions assurer à la fois la rapidité d'inscription à travers un Onboarding extrêmement simple et à la fois de s'assurer que les informations étaient suffisantes.

# utilisation du Token JWT comme acteur limitant les points de friction

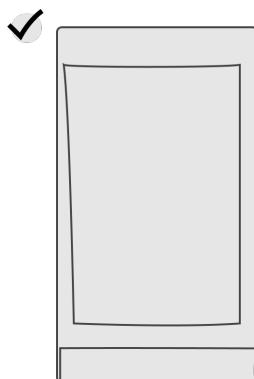
## Splash Screen



## ✗ inscription



## channel Screen



Au-delà de la simple redirection vers le formulaire d'inscription, nous utilisons le JWT pour vérifier si l'utilisateur a déjà utilisé notre application. Même si le token n'est pas valide, nous pouvons déterminer s'il provient de notre système. Dans ce cas, nous le redirigeons vers le formulaire de connexion.

## ✗ connexion

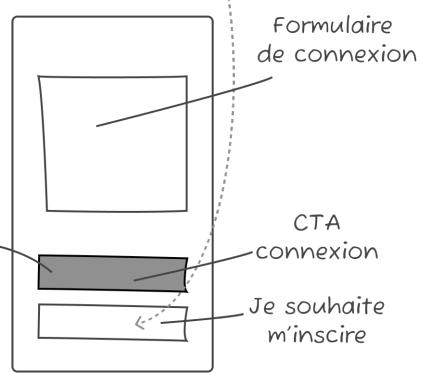
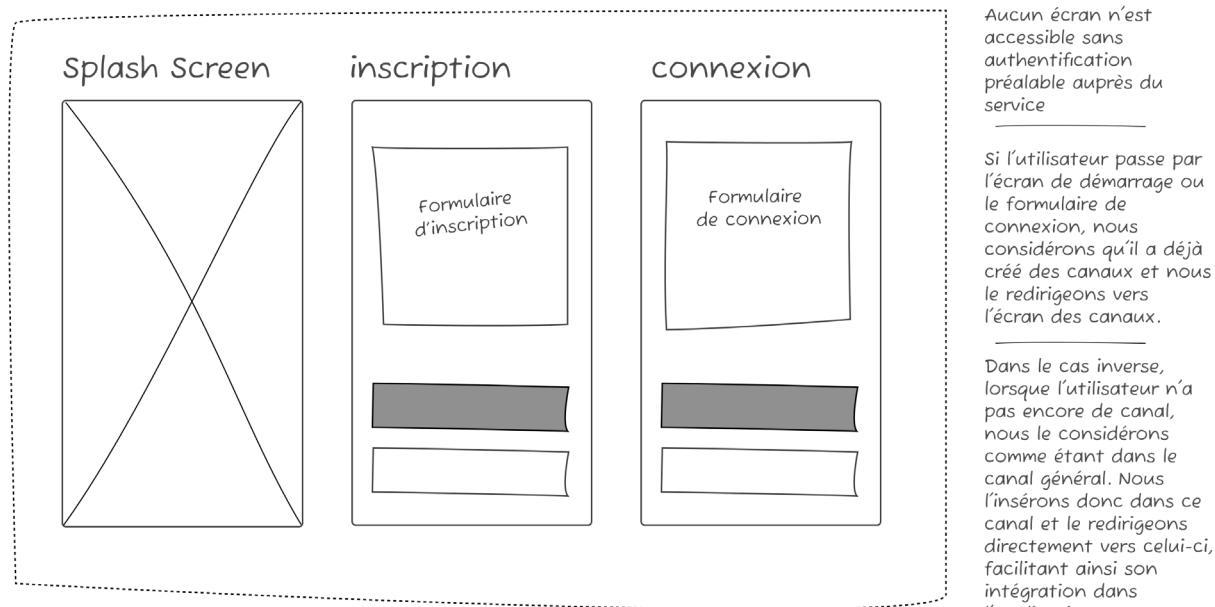


Illustration donnant un exemple de réflexion dans la première phase de maquettage

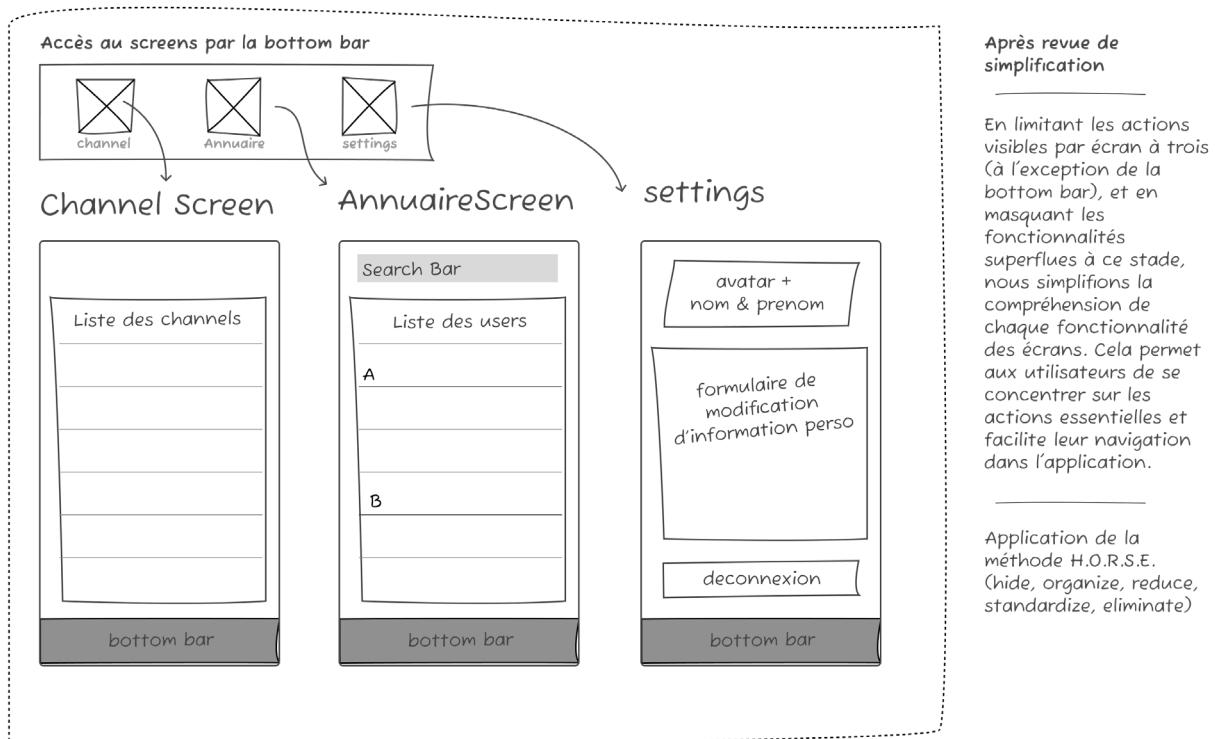
L'illustration présentée reflète bien la réflexion approfondie menée lors de la conception de notre outil. Elle met en évidence la nécessité d'intégrer un middleware pour garantir une authentification simplifiée grâce à un token JWT. En plus de la présence du token, nous identifions deux fonctionnalités clés à développer : la gestion de l'expiration du token et sa validation. Cela nous indique également la nécessité d'un store côté front pour stocker les deux tokens. Le premier token, valide pendant une journée, et le second, le refresh token, qui assure une validité de trois jours et permet de mettre à jour le premier de manière sécurisée. Cette approche assure ainsi une expérience utilisateur fluide en optimisant l'authentification tout en préservant la confidentialité des données.

## Réflexion sur l'enchainement des stack

### stack de sécurité à franchir



Une fois authentifier, on reste dans la simplicité



Dans cet exemple, nous pouvons observer que l'organisation des stack (empilement des écrans) en fonction des étapes de l'utilisateur nous permet de définir la hiérarchie dans notre application mobile. En utilisant React Native, la création d'une barre de navigation inférieure (Bottom Bar) est déterminée par la manière dont nous déclarons les stacks correspondants aux différentes familles d'écrans. Cela nous indique donc les mesures à prendre lors de la création de l'architecture pour garantir la cohérence du parcours utilisateur.

En structurant nos stacks de manière réfléchie, nous pouvons assurer une navigation fluide et intuitive, en suivant les étapes et les besoins de l'utilisateur. Cela permet d'optimiser l'expérience utilisateur et de rendre l'application plus conviviale.

Il est important de prendre en compte cette organisation lors de la conception de l'architecture, en définissant clairement les étapes et les interactions entre les écrans, ainsi que la manière dont ils seront accessibles depuis la bottom bar. Cela garantit une cohérence dans le parcours utilisateur et facilite la compréhension et l'utilisation de l'application.

## La méthode H.O.R.S.E.

La méthode H.O.R.S.E. (hide, organize, reduce, standardize, eliminate) est une approche utilisée pour optimiser l'expérience utilisateur et simplifier les interfaces. Elle se compose de cinq étapes :

1. **Hide** (cacher) : Masquer les fonctionnalités non essentielles ou avancées par défaut, afin de réduire la complexité visuelle et cognitive de l'interface.
2. **Organize** (organiser) : Organiser les éléments de l'interface de manière logique et cohérente, en regroupant les fonctionnalités similaires et en créant une structure claire.
3. **Reduce** (réduire) : Réduire le nombre d'éléments et de choix visibles à l'écran, en ne montrant que les options les plus pertinentes et en évitant les redondances.
4. **Standardize** (normaliser) : Adopter des conventions de design cohérentes et des éléments d'interface standardisés, tels que des icônes, des boutons et des menus, pour faciliter la compréhension et l'utilisation de l'application.
5. **Eliminate** (éliminer) : Éliminer les fonctionnalités, les étapes ou les actions qui ne sont pas indispensables, afin de simplifier le flux de l'interface et d'éviter les éléments superflus.

En utilisant ainsi cette méthode, nous avons pu être en mesure d'optimiser les interfaces en réduisant leur complexité, en améliorant la clarté et en facilitant l'utilisation pour les utilisateurs.

## Structure du “channelStack” - Approche classique de l'organisation de l'information

Dans la prochaine illustration, nous adoptons une approche classique de l'organisation de l'information en forme d'entonnoir. Au fur et à mesure de notre progression dans le parcours, les détails apparaissent, ce qui correspond à l'approche de la méthode H.O.R.S.E mentionnée précédemment. Nous essayons de rester fidèles au dicton célèbre «KISS» (Keep It Simple, Stupid) en cachant et en organisant les éléments de manière simple.

## Channel Stack - étape de consultation

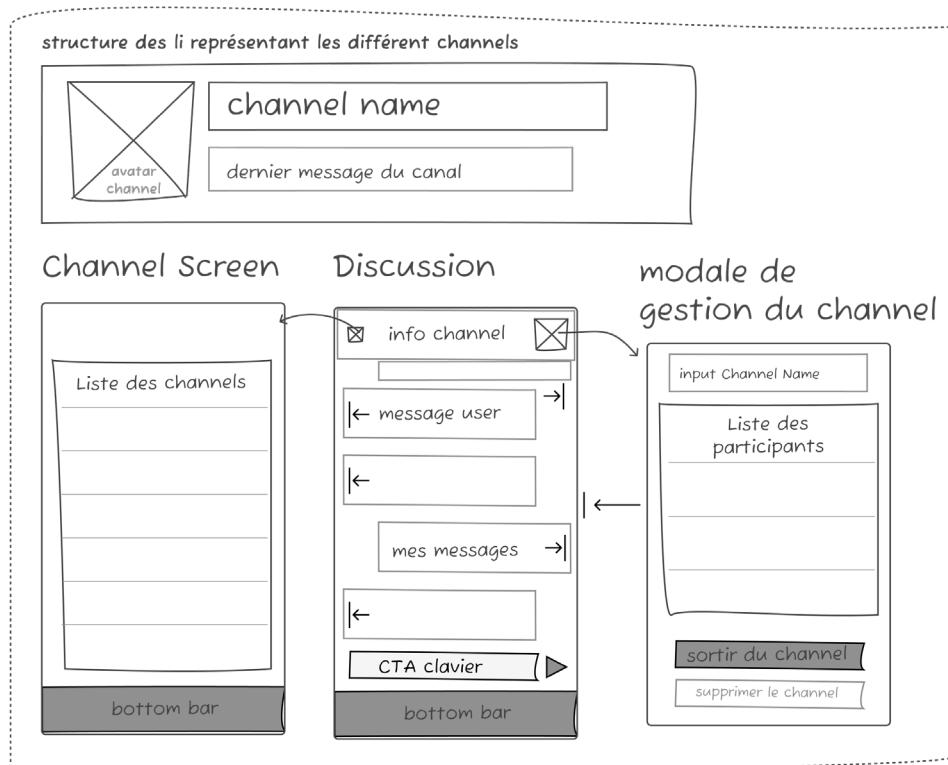


Illustration du zoning pour la stack channel

Dans la première étape, l'utilisateur consulte les canaux. Nous affichons simplement une liste des canaux auxquels il est abonné, avec le dernier message envoyé. Ce dernier message peut sembler anodin, mais il permet à l'utilisateur de suivre rapidement l'avancement des discussions ou de se préparer cognitivement à poursuivre la discussion en se rappelant où il s'était arrêté.

Dans la deuxième étape, sur le deuxième écran, nous affichons les différents messages, avec les messages de l'utilisateur à droite et ceux des autres membres à gauche. Un champ de saisie lui permet d'ouvrir le clavier virtuel pour répondre aux demandes. Nous laissons également la bottom bar disponible pour faciliter la navigation de l'utilisateur. En haut, nous affichons des informations sur le canal dans la barre de commande. À gauche de la barre de commande, l'utilisateur peut sortir de l'écran de discussion et revenir au canal. À droite, une icône invite à accéder aux réglages, utilisant les notions d'affordance. Lorsque l'on clique dessus, nous arrivons à la dernière étape, qui est la modification des informations du canal.

Le nom du canal est mis en avant en haut et est séparé du reste du formulaire, car nous supposons que cet usage sera le plus fréquent. En dessous, nous mettons à disposition la liste des participants à ce canal. L'idée est que dans une prochaine itération, l'utilisateur puisse changer les rôles prédéfinis ou supprimer un membre en quelques clics. Il y a également un bouton pour quitter le groupe, disponible pour tout le monde, et un bouton pour supprimer le groupe, disponible uniquement pour l'utilisateur ayant le rôle SUPER.

étape  
d'approfondissement de  
fonctionnalité du  
channel Stack

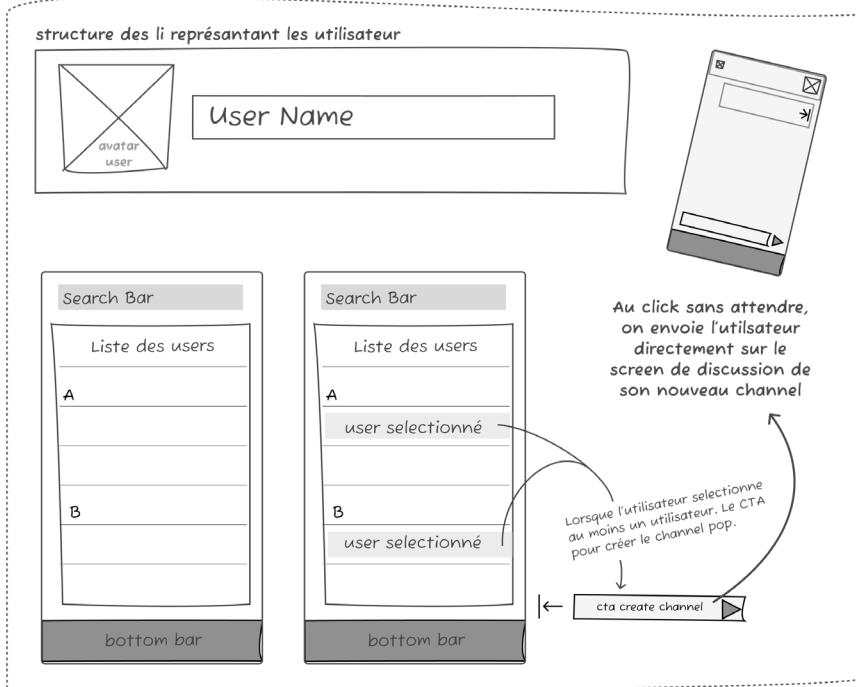
Dans l'écran des channels, la première liste affiche uniquement les canaux auxquels nous sommes abonnés. Le dernier message visible dans chaque channel permet d'avoir un aperçu rapide de l'avancée des discussions. Une fois dans un channel, nous avons la possibilité de participer aux conversations.

En haut de l'écran, des actions plus avancées sont disponibles si nécessaire. Cela peut inclure des fonctionnalités supplémentaires telles que la recherche, les filtres ou d'autres options spécifiques au channel. Ces actions complémentaires offrent une flexibilité supplémentaire pour les utilisateurs souhaitant effectuer des tâches spécifiques ou accéder à des fonctionnalités avancées dans le contexte du channel en cours.

Cette organisation des éléments dans l'écran du channel permet de simplifier l'interface en affichant les informations essentielles en premier et en offrant des options supplémentaires selon les besoins et les actions que l'utilisateur souhaite effectuer.

# L'informatique au service de l'expérience utilisateur

## Annuaire stack - étape de création d'un canal



### Simplification de l'étape de création du channem

Dans l'annuaire, nous avons simplifié les étapes de création du channel. L'utilisateur est invité à sélectionner un ou plusieurs utilisateurs, ce qui fait apparaître directement un bouton Call To Action en bas à droite de l'écran.

Lorsque l'utilisateur clique sur ce bouton, la création du channel est lancée automatiquement. Une fois que la base de données confirme la création réussie, l'utilisateur est redirigé vers l'écran du channel où la discussion est déjà en cours. Un message d'introduction est automatiquement envoyé pour inaugurer la création du canal.

À ce moment-là, l'utilisateur a la possibilité de renommer la discussion si nécessaire. Dans le cas où la discussion est un dialogue entre deux utilisateurs, le nom du channel est attribué d'après le nom de l'utilisateur. Cette approche facilite la création rapide des channels tout en permettant une personnalisation ultérieure si souhaité.

Dans la stack de l'annuaire, nous avons adopté une approche visant à simplifier au maximum l'étape de création d'un nouveau canal. Tout d'abord, aucune information n'est requise de la part de l'utilisateur. Il recherche simplement d'autres membres de l'application et les sélectionne d'un simple clic. Lorsque le tableau contient au moins un utilisateur sélectionné, le bouton de création du canal apparaît. En cliquant dessus, le processus de création est lancé.

Nous créons le nouveau canal en concaténant les prénoms des utilisateurs, en commençant par celui à l'initiative de la création du canal. Ce titre est conçu pour être temporaire, car nous limitons le nom du canal au nom du créateur et à deux autres utilisateurs. Une fois la création du canal établie, un déclencheur insère la liste des participants et un second insère le premier message qui sera envoyé au nom du créateur. À ce moment-là, une réponse est renvoyée avec les informations nécessaires agrégées dans le front-end, permettant ainsi une redirection automatique vers ce canal.

Cette étape montre à quel point les décisions entre les UI/UX designers doivent être prises en collaboration avec les concepteurs et les gestionnaires de la base de données. Dans le cadre de cette réflexion, le ticket comprenait également un diagramme d'activité simple qui séquençait de manière précise les étapes côté front-end et back-end afin de faciliter le développement.

## DIAGRAMME D'ACTIVITE - CREATION DE CHANNEL

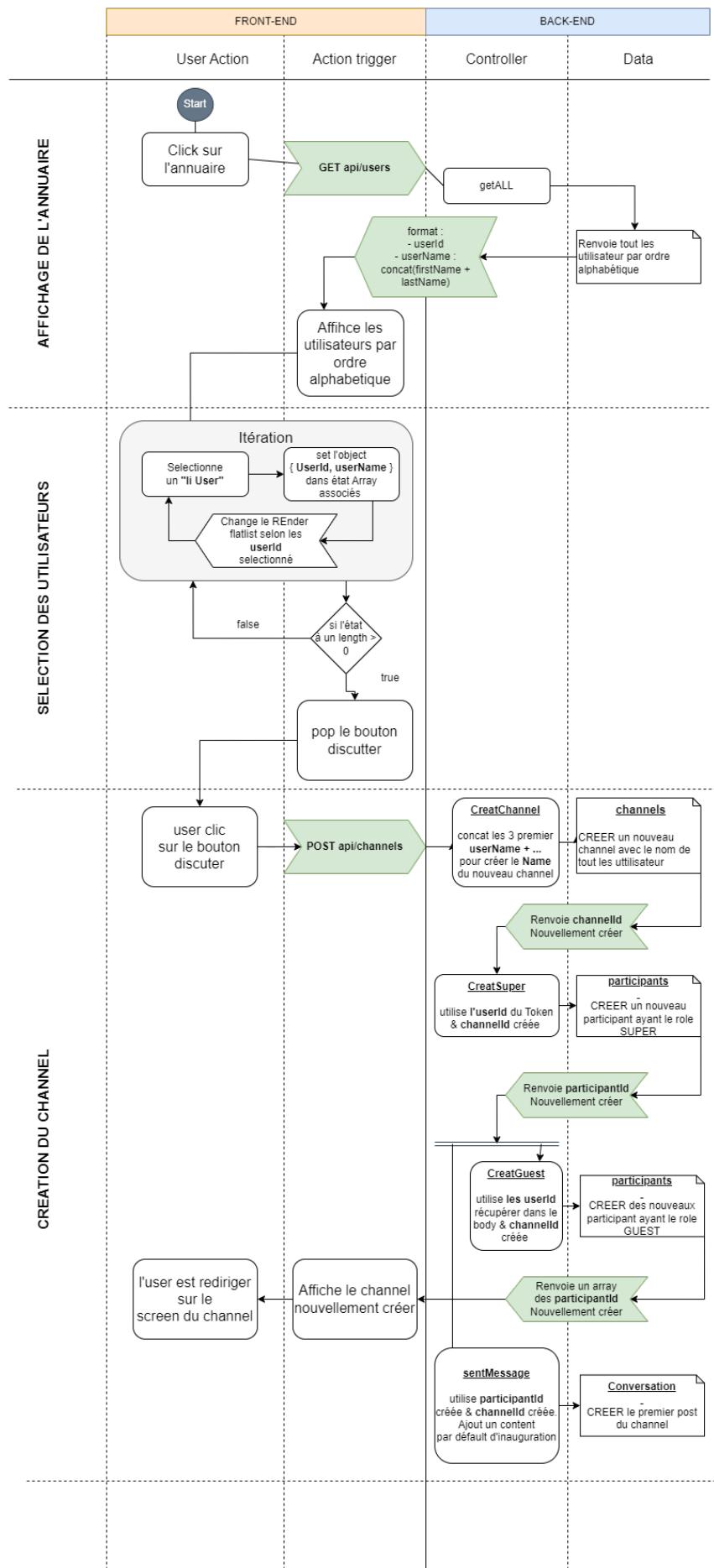
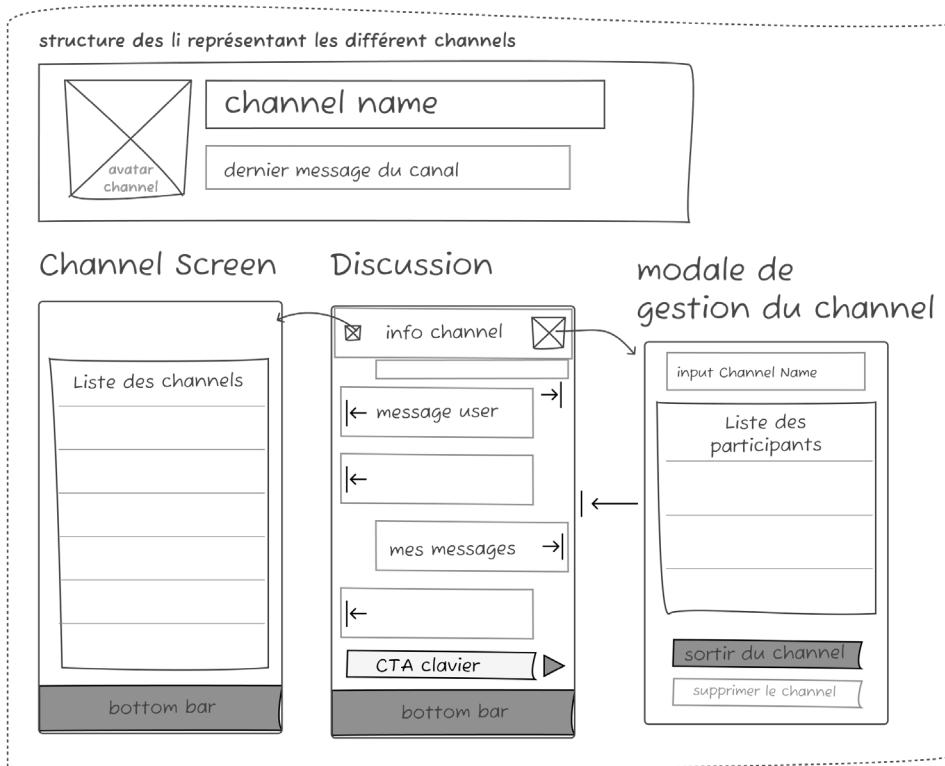


Diagramme d'activité résumant le comportement front et back de la création d'un nouveau channel

# A la lisière de notre périmètre de conception

## Channel Stack - étape de consultation



Nous pouvons constater dans cette illustration que certains éléments de notre zoning laissent présager de nouvelles fonctionnalités à venir. Et en effet, c'est le cas. Par exemple, nous voyons ici et là des avatars, même si notre POC ne les intègre pas encore. Au-delà de la question de la faisabilité dans les délais impartis, il aurait tout à fait été possible de les mettre en place. Cela démontre l'importance de hiérarchiser nos priorités en fonction du travail à fournir. Consacrer seulement deux ou trois heures à la mise en place de cet élément n'aurait pas été très utile. Bien que cela aurait permis une personnalisation de l'application, cela restait néanmoins un «nice to have».

D'autres exemples se situent dans cette fourchette de temps. Typiquement, dans la conception de la base de données, nous avons déjà défini une colonne pour gérer les rôles des utilisateurs, mais elle ne nous sert actuellement qu'à déterminer si l'utilisateur a le rôle de super-utilisateur et est autorisé à supprimer un canal. L'idée est d'ouvrir de nouveaux chantiers afin d'aborder cette réflexion ultérieurement. Cela ne signifie pas que nous renonçons à cette fonctionnalité, bien au contraire.

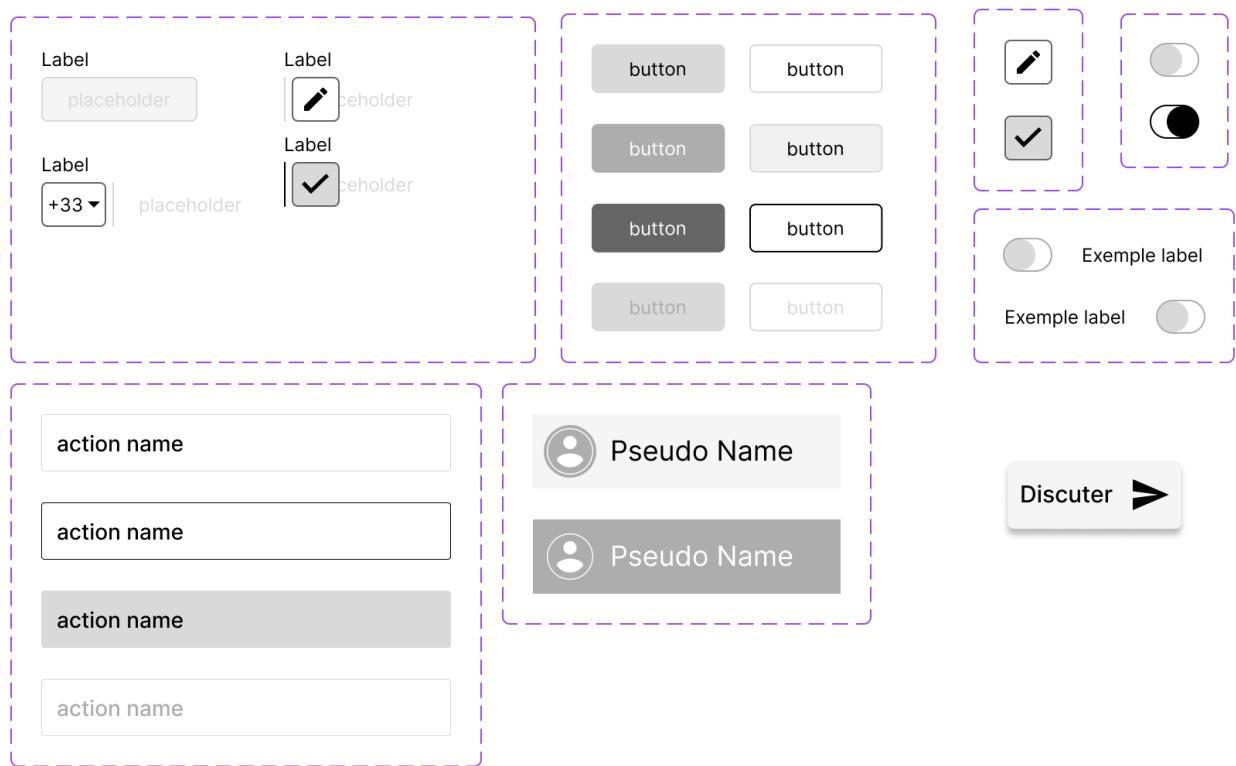
En intégrant le maquettage, l'organisation réfléchie des stacks et l'optimisation de l'interface, nous avons créé un support intuitif et technique pour une expérience utilisateur fluide. La prochaine étape sera la création du wireframe à partir du zoning, en continuant à hiérarchiser nos priorités pour offrir une expérience optimale.

# Le wireframe | Concevoir c'est aussi s'aménagé du temps

Dans ce chapitre, nous explorons l'utilisation des fonctionnalités avancées de Figma pour la création de systèmes de design complexes, en mettant l'accent sur le principe d'atomicité. Nous examinerons également comment la représentation de la structure moléculaire dans le code peut être utilisée en corrélation avec des librairies modulaires telles que React ou React Native. Notre approche de conception modulaire a permis d'économiser du temps dans la création de composants et de faciliter la multiplication des écrans. Cette méthode a déjà fait ses preuves en réduisant de trois jours le processus de maquettage, et elle continuera de s'avérer bénéfique lors de l'ajout de nouvelles interfaces ou de nouveaux écrans à notre application.

## Utilisation des fonctionnalités avancées de Figma

Grâce à ses fonctionnalités avancées, nous avons pu créer facilement un design système complexe, bénéficiant d'outils puissants pour la création et la manipulation des composants. En adoptant le principe de composants réutilisables, nous nous rapprochons d'une cohérence de conception proche des librairies ou frameworks modulaires. Pour accélérer l'élaboration de notre maquette, nous avons pris la décision, en collaboration avec Amhed Magassouba, de commencer le projet directement avec une page dédiée au design système.



Screen shot de la frame form system et utilisation des composant dans Figma lors du wierFrame

Dans l'exemple suivant, l'image illustre une partie de notre design système, où les composants sont organisés dans des cadres (Frames). Ces cadres nous permettent de regrouper les éléments selon leurs fonctionnalités clés. Dès la phase de wireframe, nous avons structuré les composants en créant plusieurs variantes pour définir leurs différents états. En adoptant cette approche et en programmant les états initiaux du design système, nous avons facilité les étapes ultérieures d'application du style.

Dans le wireframe, nous avons également inclus les actions, allant au-delà de la simple logique du parcours utilisateur. Il était important pour nous de prendre en compte les différents comportements des éléments. Par exemple, dans la section «Setting Stack» de notre zoning, certains comportements des éléments étaient étroitement liés à l'expérience que nous souhaitions offrir à nos utilisateurs.

En structurant notre design système de cette manière, nous avons pu rationaliser notre processus de conception et garantir une cohérence visuelle et fonctionnelle tout au long du développement de l'application.

### Structure et composition du composant avec les attribut booléens

The screenshot shows the structure and composition of a button component. On the left, a tree view lists variants for the 'button' component, including primary, disabled, primary, onclick, primary, over, primary, simple, secondary, disabled, secondary, onclick, secondary, over, and secondary, simple. To the right, there are two 'Edit variant property' dialogs. The first dialog is for 'state' and shows values: simple, over, onclick, and disabled. The second dialog is for 'type' and shows values: primary, secondary, and secondary.

### Exemple de paramétrage d'action du prototype sur un composant

The screenshot shows the configuration of interaction details for a button component. It displays three panels: 'Interaction details' for 'While hovering' (set to 'primary over'), a central 'button' component preview, and 'Interaction details' for 'Mouse leave' (set to 'primary simple'). The preview shows a purple 'While hovering' state with a mouse cursor icon.

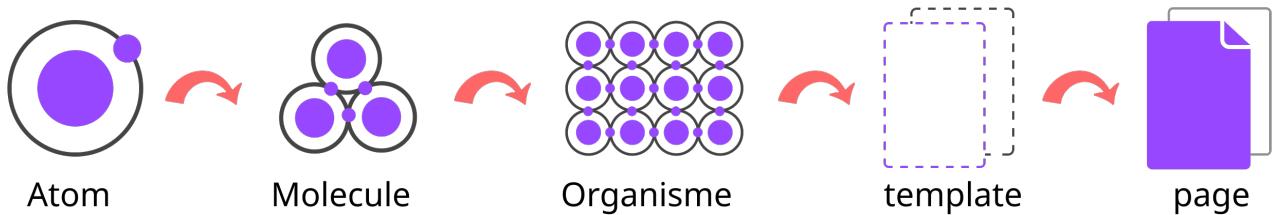
Screen shot de paramétrage d'un composant button type dans le wireframe de l'application sent

## Une maquette adaptée pour une approche modulaire

Lors de la conception du wireframe, nous avons exploré en profondeur le concept d'atomicité, où les composants sont imbriqués pour former des éléments plus complexes, rappelant la structure moléculaire. Un exemple concret est celui de notre composant «input text», qui comprend plusieurs sous-composants tels que le label, le texte interne et le message d'erreur. Chacun de ces sous-composants possède ses propres états, mais ensemble, ils forment un élément de base. Ce dernier peut ensuite être surchargé pour construire des éléments plus complexes.

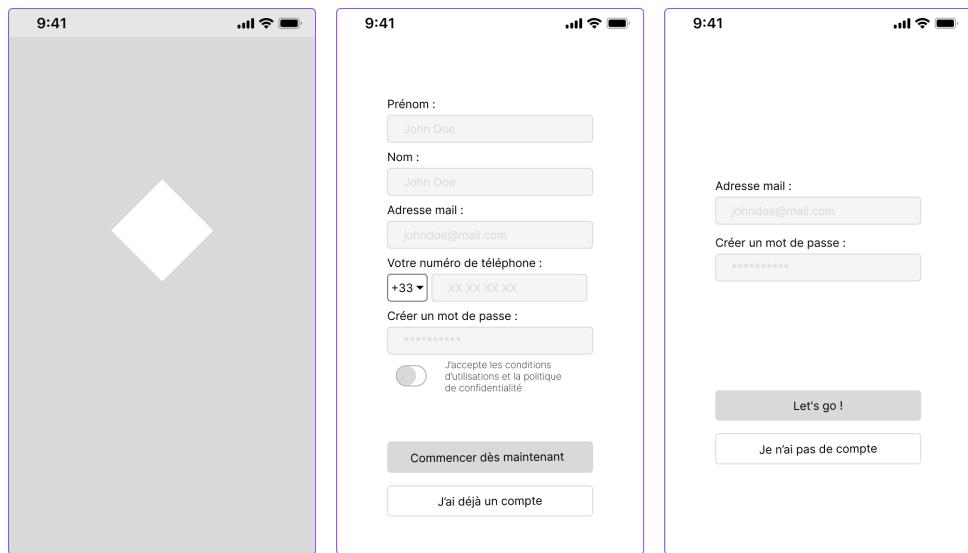
L'un des avantages des composants à état dans Figma est qu'ils ne peuvent être restructurés qu'à partir du composant d'origine. Cela signifie que lors de la conception du design d'interfaces, nous devons anticiper tous les états possibles dès le départ. Cette approche est similaire à celle des composants en React, qui doivent prendre en compte à l'avance tous les paramètres de leur utilisation. Ainsi, nous constatons que la maquette offre également une réflexion approfondie sur l'utilisation des composants pour les développeurs.

En adoptant cette approche d'atomicité, nous avons pu créer une bibliothèque de composants cohérente et modulaire, où chaque élément peut être réutilisé dans différents contextes. Cela facilite grandement le travail des développeurs, car ils disposent d'éléments prêts à l'emploi avec des états préconfigurés, leur permettant de se concentrer sur l'implémentation fonctionnelle de l'interface sans avoir à se soucier des détails de conception. De plus, cette approche favorise la maintenabilité et l'évolutivité du design système, car les modifications apportées à un composant se répercutent automatiquement sur tous les éléments qui l'utilisent, garantissant ainsi la cohérence de l'interface utilisateur.

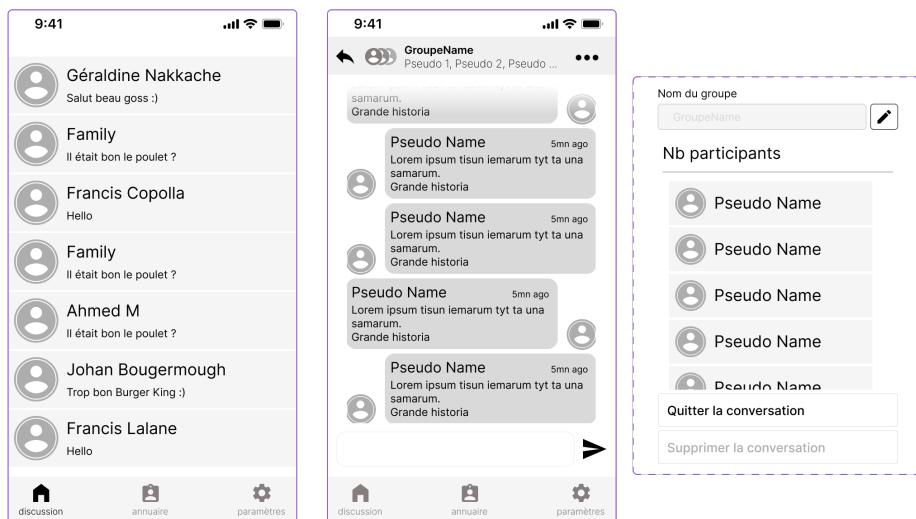


Grâce à notre design système simple et modulaire, nous avons pu gagner du temps significatif lors du processus de maquettage, permettant une itération plus rapide et une multiplication aisée des écrans. Cette méthode de conception sera d'autant plus bénéfique lors de l'intégration de nouvelles interfaces ou de nouveaux écrans dans notre application, offrant une base solide et extensible.

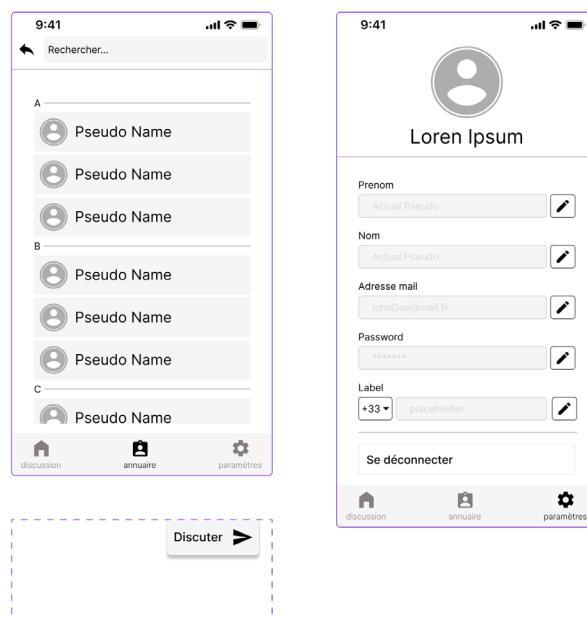
# Présentation du wireframe en détail



wireframe - Stack d'authentification



wireframe - ChannelStack

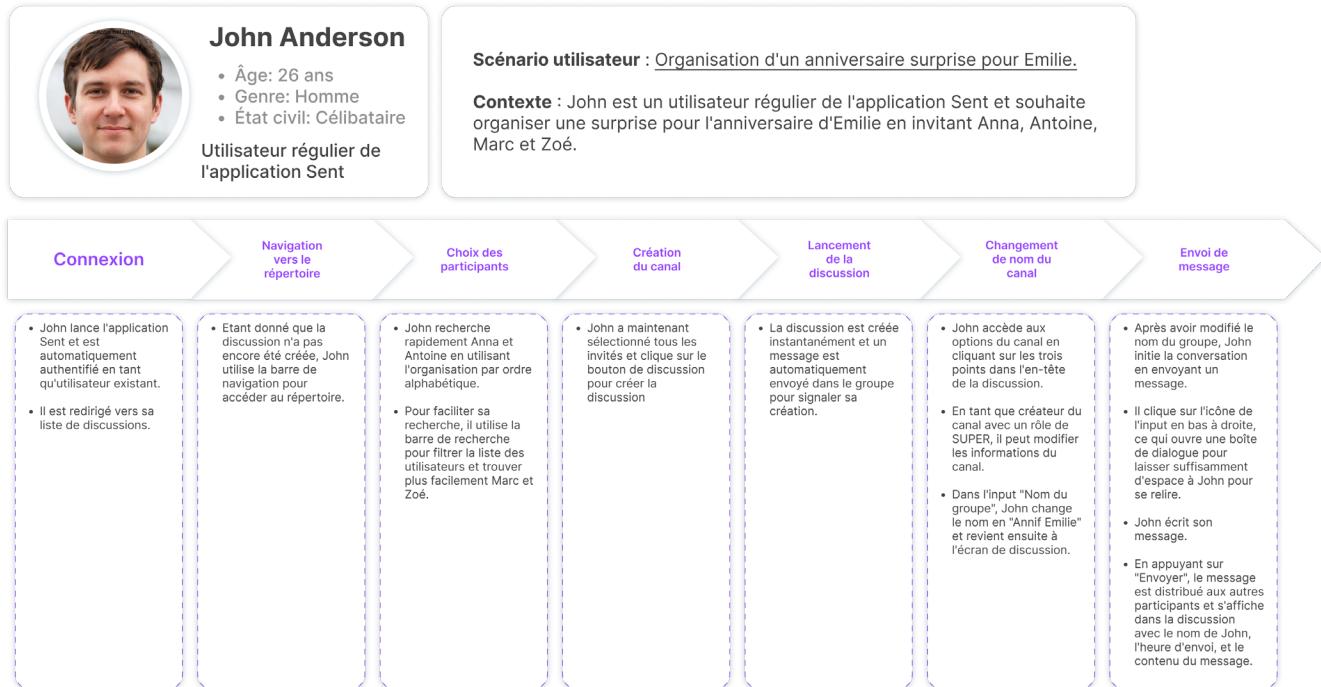


wireframe - DirectoryStack & SettingStack

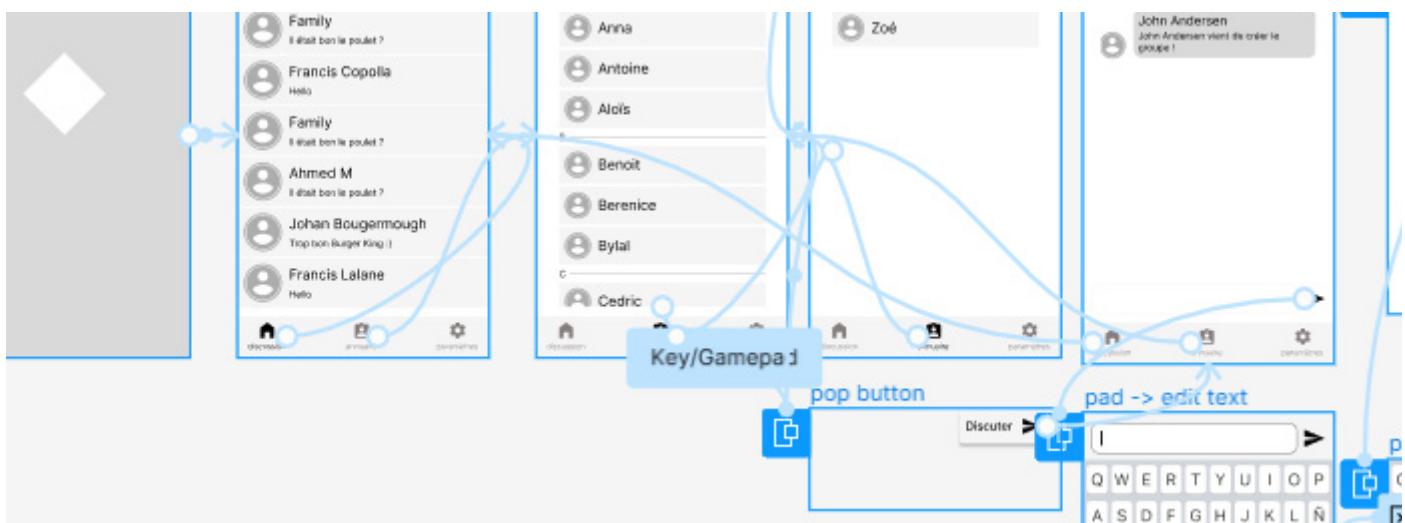
# Gestion des Parcours utilisateur dynamique avec Figma

Nous avons décidé d'intégrer les actions de l'utilisateur dès le stade du wireframe afin de simuler et de tester le parcours utilisateur dès le début du processus de conception. Cela nous permet de détecter les éventuels points d'amélioration et de trouver la meilleure approche pour rendre le parcours utilisateur plus fluide et simplifier les interactions.

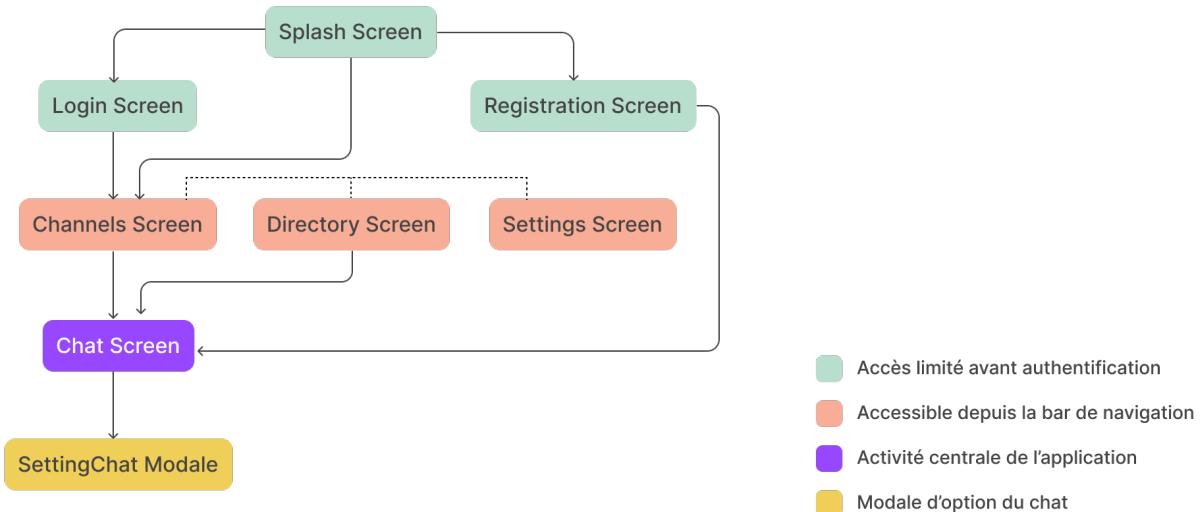
## Exemple de scénario pouvant être utilisé pour faciliter le travail des UX Designers



## Résultat obtenu lors du wireframe



## Enchainement des screens



## Maquette haute fidélité

Une fois que l'équipe a validé le wireframe et les actions liées à l'expérience utilisateur, nous avons pu passer à la réalisation de la maquette haute fidélité. Pour ce faire, nous nous sommes appuyés sur un ensemble d'éléments graphiques préalablement établis en début de projet. Nous avons ensuite défini une pseudo charte graphique, qui correspondait à nos intentions esthétiques, et l'avons directement incorporée à notre maquette Figma.

Local styles

Text styles

Ag h1 - 24/Auto

Ag h2 - 12/Auto

Ag h3 - 18/Auto

Ag h4 - 16/Auto

Ag h5 - 16/Auto

Ag default - 13/Auto

Color styles

- Primary Color
- secondary
- purple
- green
- red

SimpleWhite

backgroundLight → Transparency

backgroundcolor

place holde color

Effect styles

- inner shadow
- SimpleContent → backgroundBlur
- Input → BackdropBlur

Loren Ipsum

Loren Ipsum

Loren Ipsum

Loren Ipsum

Loren Ipsum

Loren ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis

logotype

sentr

BackGround picture

SplashScreen | illustration exemple

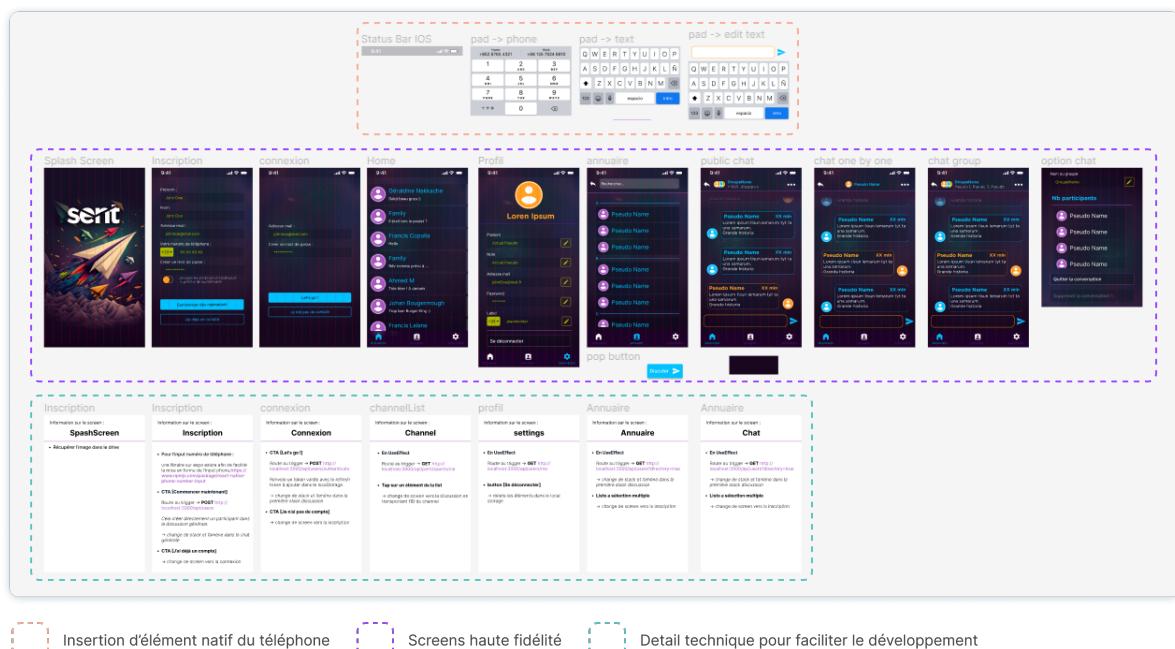
This screenshot displays a user interface for a UI design tool. On the left, there's a sidebar with sections for 'Local styles', 'Text styles', 'Color styles', 'Effect styles', and a preview area for text ('Loren Ipsum'). Below these are lists of font sizes and colors. The main workspace contains several dashed-line boxes showing different styling examples: a grid of colored circles, a row of grayscale circles, and a checkered pattern. To the right are three preview panels: 'logotype' showing a large stylized word 'sentr', 'BackGround picture' showing a dark gradient background with placeholder text, and 'SplashScreen | illustration exemple' showing a vibrant illustration of a paper airplane launching over a mountain range.

## *Présentation des éléments de styles principaux*

Une fois les styles appliqués, la maquette prend vie et s'adapte rapidement. Grâce à l'auto Layout, tous les écrans sont flexibles et s'ajustent aux différentes tailles de smartphones. Cela garantit une expérience cohérente et optimale. L'auto Layout gère efficacement les variations de contenu et de mise en page, évitant les problèmes de débordement. Cette approche nous a permis de concevoir des maquettes représentatives de l'expérience utilisateur sur différents appareils. En utilisant l'auto Layout et en adoptant une approche responsive, nous avons facilité l'intégration des composants lors du développement.

## Considérations d'interopérabilité des comportements natifs pour une intégration fluide

Nous avons utilisé des templates d'éléments natifs du téléphone pour garantir le comportement approprié lors de l'ouverture des panneaux, affinant ainsi la position des composants en fonction de leurs réactions. Bien que React Native soit une librairie censée offrir des composants natifs multiplateformes, nous savions que les comportements natifs des différents systèmes d'exploitation pouvaient parfois différer. En assurant le positionnement des composants selon les comportements attendus, nous facilitons leur intégration lors du développement. De plus, nous avons inclus des indications sur les comportements attendus pour faciliter la prise en main de l'interface lors des itérations du projet. Cette approche optimise les interactions et fournit des informations claires pour guider le développement.



Insertion d'élément natif du téléphone

Screens haute fidélité

Detail technique pour faciliter le développement



## Premier Sprint | Développement de l'API

Une fois la maquette haute fidélité approuvée, nous avons entamé notre premier sprint dédié à la mise en place du back-end de notre application. Comme nous l'avons mentionné précédemment, ce sprint a été divisé en deux phases distinctes. La première phase, «Full Squad», a réuni l'ensemble de l'équipe pour accélérer le développement des fonctionnalités classiques telles que les opérations CRUD pour chaque entité, ainsi que l'établissement des routes métier essentielles. Dans la seconde phase, une équipe restreinte de deux personnes a pris en charge les tests, le contrôle qualité et les ajustements finaux du back-end.

### Outil est stack utilisé



**Visual studio code** | éditeur de code



**Node.js** | environnement d'exécution

- Gestion des requêtes HTTP.
- Routage et middleware simplifiés.
- Programmation asynchrone optimisée.

ex

### **Node js** | infrastructure d'applications Web

- Minimalité de l'infrastructure
- Architecture simplifié
- Intégration de middlewares simplifiée



### **Sequelize** | Framework Object-Realation mapping

- Gestion relationnelle simplifiée
- Portabilité MySQL
- Simplification des interactions avec la BDD



### **Thunder Client** | extension client API Rest

- Test des routes en local
- Légèreté et simplicité d'utilisation



### **BCrypt** | système de hachage des données



### **JSON WEB TOKEN** | librairie de gestion de l'authentification par token



### **GitHub Project** | interface de gestion du Projet

- Établissement de délais et de jalons
- Partage du code sur répertoire commun
- Peer Review lors des merge
- Intrication des issues vs branches
- Interface de projet sous forme de Kaban
- Détail des issues en MarkDown



### **Git** | Système de contrôle de versionning

# Mise en place de conventions pour une approche plus cohérente

La mise en place simultanée de 4 développeurs sur une même application impose une rigueur nécessaire afin d'assurer un bon fonctionnement. Bien que l'approche d'une communication osmotique offre déjà un appui non négligeable au bon déroulement des sprints, il était nécessaire pour nous de définir des règles et des conventions de bonne conduite afin d'assurer à tous une bonne compréhension et une bonne lisibilité du code. Ces règles encadrent les conventions de nommage, de commit, l'organisation et l'architecture des dossiers, offrant à chacun un cadre de travail optimal.

## Convention de nommage des variables :

- Utiliser le Camel Case, ex : nomDeLaVariable
- Si la variable a un concept de «avant» ou «après», utiliser un suffixe dans le temps présent ou passé, comme dans «ControlAdd» ou «ControlAdded»
- Les variables doivent être nommées en anglais
- Éviter les noms d'une seule lettre. Être explicite dans la dénomination
- Utiliser les PascalCase seulement lors de constructors ou de classes

## Convention de gestion des constantes :

- Mettre une constante en uppercase uniquement si elle (1) est exportée, (2) est une constante (elle ne peut pas être réaffectée), et (3) le programmeur peut être sûr qu'elle (et ses propriétés imbriquées) ne changera jamais.
- Favoriser la multiplication des constantes plutôt que la mutabilité des variables
- Utiliser le destructuring pour une utilisation optimale des variables
- Favoriser l'usage du spread operator lors d'une réaffectation partielle ou complète d'une variable

## Configuration du linter et du code formateur :

- - Utiliser Prettier code formateur pour garantir une gestion cohérente des retours à la ligne et des espaces
- - Adopter la nomenclature «airbnb-base» pour les projets Node.js et React

## Convention de commit :

- Le message du commit doit définir le QUOI et non le COMMENT, ex :  
*bad > Ajout d'un attribut supplémentaire dans foo01 pour gérer le cas X*  
*good > update foo01*
- Préfixer chaque commit selon la [Convention Commit 1.0.0](#)
- Le commit doit de préférence avoir un nombre de caractère inférieur à 50 caractère
- Une issue, une branche. La branche doit porter le nom de l'issue en question

## **Convention de nommage des routes :**

- Suivre au mieux les principes RESTful
- Nommage cohérent des ressources : Des URIs représentatives pour chaque entité, par exemple «/users» pour les utilisateurs.
- Méthodes HTTP appropriées : Utiliser GET, POST, PUT, DELETE pour les actions correspondantes.
- Utilisez des conventions de dénomination des ressources et un format URI cohérents pour un minimum d'ambiguïté et une lisibilité et une maintenance maximales, ex :  
`/device-management/managed-devices/{id}/scripts/{id}`
- Paramètres dans les routes : Utiliser des paramètres explicites pour filtrer, ex : «/users?role=admin».

## **Convention de nommage des fonctions :**

- Utiliser des noms de fonctions en anglais
- Les noms de fonctions doivent être explicites quant à l'action qu'elles effectuent
- Préciser les propriétés attendues en tant que paramètres de la fonction
- Si possible chaque fonction doit comporter un JsDoc définissant son rôle, ses attributs et la valeur de retour attendue en respectant les conventions associées.

## **Convention de nommage des fonctions :**

- Respecter au maximum les principe singleton
- Respecter la Convention du suffixe de dossier : le nom du fichier se termine par le nom du dossier dans lequel il est rangé
- Nom de fichier en Camel Case

## **Convention de nommage des fonctions :**

- Suivre la convention HTTP des status code
- Le Messages auto-descriptifs doit être présente pour chaque requête et explicite
- Les ressource sont retourné en JSON

# **Mise en place d'express et de l'architecture dossier Back-End**

A la suite des réflexions préliminaires à l'édition du logiciel, nous avons pu établir les bases d'un cadre de travail adapté à nos besoins. Notre back-end est composé d'un ensemble de dossiers permettant une organisation structurée des fichiers, axée sur la séparation des responsabilités. Pour garantir une organisation claire et une gestion efficace des dépendances, nous avons opté pour l'injection de dépendances, qui offre une modularité et une évolutivité accrues à notre back-end.

Cette approche de séparation des responsabilités favorise la réutilisabilité des modules et facilite les modifications et les ajouts ultérieurs. De plus, elle facilite la collaboration entre les développeurs, car chacun peut se concentrer sur sa partie spécifique du code sans interférer avec les autres.

# Architecture dossier de notre Back-End



Présentation de l'architecture et des éléments clefs

Nous avons architecturé notre côté serveur en suivant les principes des API REST, en répartissant les tâches selon un modèle multicouches. Pour faciliter l'utilisation de Node.js, nous avons installé la bibliothèque Express.js à l'aide de notre gestionnaire de dépendances NPM (Node Package Manager). Cela nous a permis de mettre en place rapidement une architecture et un environnement favorables à l'implémentation des principes REST en utilisant le CLI (Command Line Interface) express-generator.

Le déroulement des appels se conforme à la structure classique :

1. Le router gère les routes et redirige les appels vers les contrôleurs correspondants.
2. Le middleware de confirmation vérifie les autorisations d'accès avant de laisser passer la requête vers le contrôleur.
3. Le contrôleur traite la logique métier, interagit avec le modèle et prépare la réponse à renvoyer.
4. Le modèle gère l'interaction avec la base de données et effectue les opérations CRUD.
5. Le contrôleur renvoie la réponse en JSON après avoir obtenu les données du modèle.

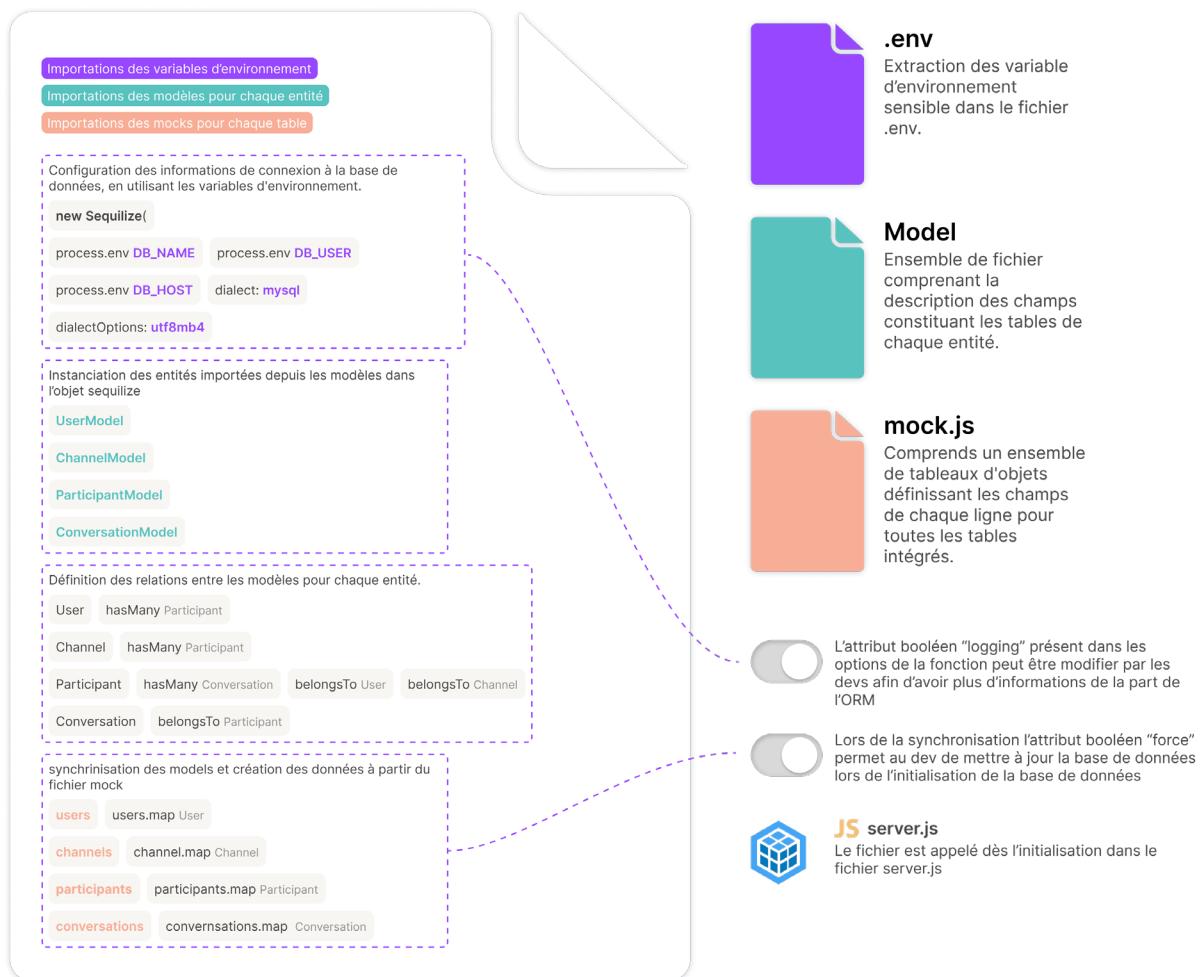
# Mise en place de l'ORM et des Mocks

La mise en place de l'ORM (Object-Relational Mapping) et des mocks a été cruciale pour notre backend. Nous avons utilisé Sequelize comme ORM pour interagir avec notre base de données MySQL. Une fois la connexion établie, nous avons mis en place les relations entre les modèles en utilisant les associations de Sequelize favorisant ainsi l'intégrité et la cohérence des données.

L'utilisation de mocks a également été bénéfique pour tester les fonctionnalités sensibles telles que les opérations de mise à jour et de suppression. En isolant ces tests, nous avons pu vérifier la robustesse de nos routes sans perturber le travail des autres développeurs.

Cela nous a permis de bénéficier d'une base de données cohérente, d'une meilleure réutilisabilité des modules et d'une collaboration fluide entre les membres de l'équipe de développement.

## Présentation simplifiée de l'installation de sequelize dans notre back-end



Représentation simplifiée de la structure d'instanciation de la base de donnée à l'aide de Sequelize

Nous avons créé des modèles de classe qui répondent aux principes de la Programmation Orientée Objet (POO). Ces modèles de classe ont été utilisés pour définir les entités de notre application, en les représentant comme des objets JavaScript.

```

/*****/
/* CONNEXION A LA BASE DE DONNEES */
/*****/

let sequelize = new Sequelize(
  process.env.DB_NAME,
  process.env.DB_USER,
  process.env.DB_PASSWORD,

  {
    host: process.env.DB_HOST,
    dialect: "mysql",
    dialectOptions: {
      charset: "utf8mb4",
    },
    logging: false,
  }
);

/***** *****/
/* MIS EN PLACE DES RELATIONS */
/*****/



const db = {};
db.sequelize = sequelize;
db.User = Users(sequelize);
db.Channel = Channels(sequelize);
db.Participant = Participants(sequelize);
db.Conversation = Conversations(sequelize);

```

*Extrait de code du fichier config/db.js pour l'instanciation de la base de données avec Sequelize*

## Usage des mocks

L'utilisation des mocks nous a permis de garantir une base de données cohérente pendant le développement. En manipulant les fonctionnalités de Sequelize, nous avons pu contrôler les mises à jour des données selon nos besoins. Cette approche a permis à tous les développeurs d'accéder simultanément à des données cohérentes et de travailler sur les opérations d'update et de delete tout en assurant un intégrité d'état de la base de donnée.

```

const salt = bcrypt.genSaltSync(10);
const password = bcrypt.hashSync("Test1234", salt);

const users = [
  {
    id: 1,
    firstname: "Johan",
    lastname: "Bouguermouh",
    phone: "0606060606",
    role: "USER",
    email: "johan@mail.fr",
    password,
  }, ...
]
```

*Extrait de code du fichier mock.js création des objets nécessaire à l'hydratation des champs Users*

L'utilisation de mocks nous permet de définir de manière précise le rôle de chaque entité et de tester le comportement des fonctionnalités dans différentes situations. Par exemple, en définissant le rôle des participants sur un canal spécifique, nous pouvons observer comment l'utilisateur avec l'ID 1 réagit. Cela nous permet de développer et de tester des fonctionnalités spécifiques à cet utilisateur en utilisant sa session.

```
const participants = [
  //général
  {
    id: 1,
    userId: 1,
    channelId: 1,
    role: "SUPER",
  },
  {
    id: 2,
    userId: 2,
    channelId: 1,
  },...]
```

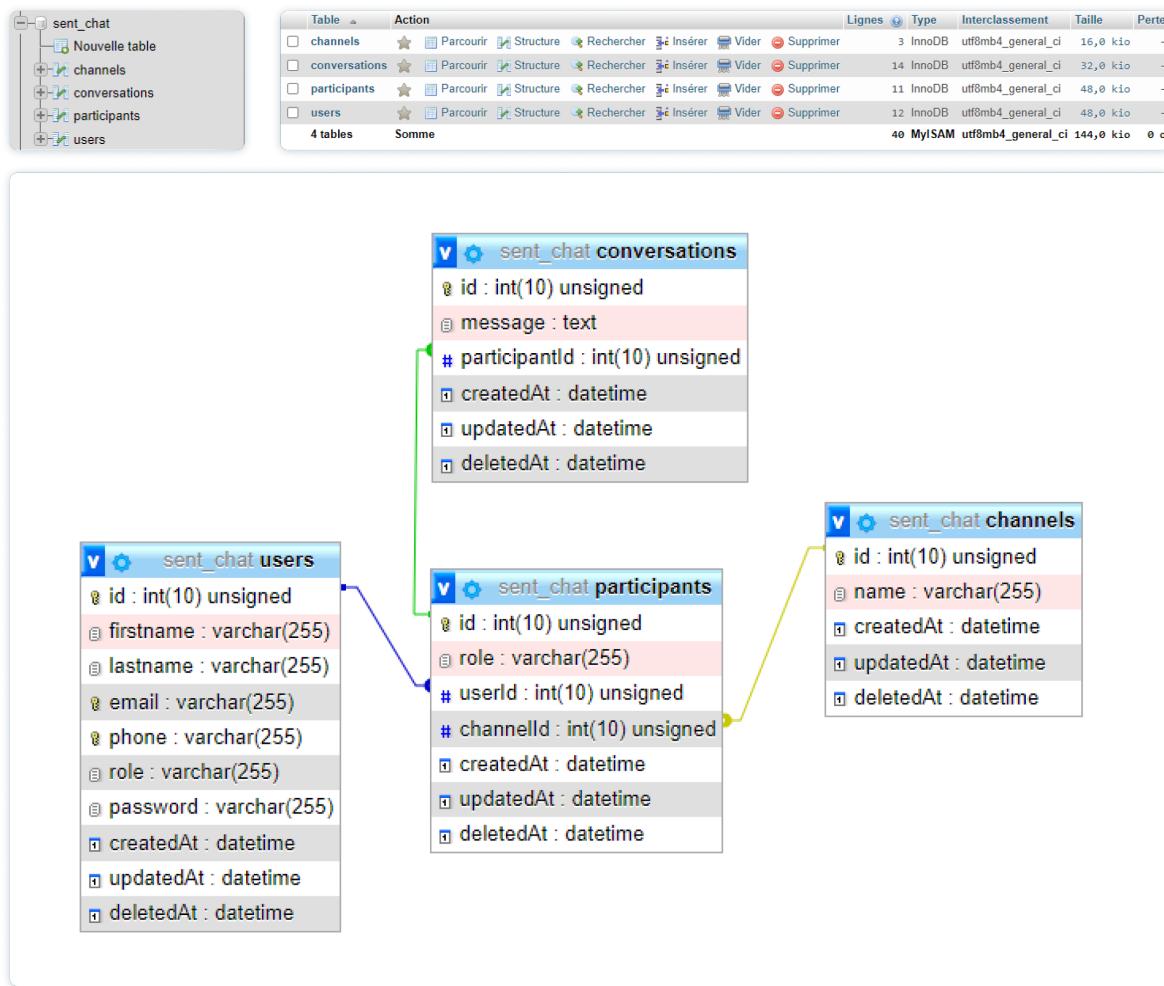
*Extrait de code représentant des participants et leurs rôle sur le mock du canal générale*

## Une fois les mocks et l'orm mise en place

Une fois les mocks, les modèles et l'ORM mis en place, nous sommes prêts à effectuer nos tests de conformité. Après avoir structuré notre appel dans le fichier server.js et créé une base de données locale nommée «sent\_chat», lorsque nous lançons la commande npm start dans le dossier «back», nous devrions voir apparaître dans la console :

```
Connection has been established successfully...
Synchronizing models with database...
Server is running on port 3000 : http://localhost:3000
Database is connected...|
```

Après une brève vérification de notre serveur, nous constatons le succès de l'installation. Le serveur est en cours d'exécution, ce qui signifie que notre application est accessible et fonctionnelle. Nous pouvons maintenant procéder à des tests plus approfondis pour confirmer le bon fonctionnement de toutes les fonctionnalités.



## L'élaboration des tickets et formalisation de la demande

La définition précise de la demande à l'intérieur de chaque ticket s'est révélée essentielle pour plusieurs raisons. Tout d'abord, cela permettait à chaque développeur de comprendre rapidement les objectifs et les spécifications de la tâche à accomplir. Cela évitait les malentendus et les interprétations différentes, ce qui aurait pu entraîner des erreurs ou des retards.

De plus, en formalisant la demande dans le ticket, nous pouvions mieux répartir le travail et éviter les chevauchements. Chaque développeur pouvait se concentrer sur sa partie spécifique de l'application sans empiéter sur le travail des autres. Cela permettait également de maintenir une certaine cohérence dans l'approche et la mise en œuvre des fonctionnalités et ainsi gagné en productivité.

Chaque ticket de l'issue était structuré de la manière suivante, en prenant comme exemple le Ticket User Service. Pour illustrer mon propos, je vais détailler uniquement la route de création d'un utilisateur. Comme mentionné précédemment, lors de la création d'un nouvel utilisateur, nous avons utilisé un trigger pour insérer une relation entre l'utilisateur et le canal général. Ainsi, dès son intégration, l'utilisateur fait déjà partie d'un canal. Dans ce contexte, la réponse renvoie l'`participantId` de l'utilisateur.

## Détail d'une partie de l'issue Back- End | Channels Service

### Channels Service

Route	Méthode HTTP	Params URL	Authentification	Action
/channels/:channelId	PUT	:channelId	true	Modifier le nom du canal
/channels/:channelId	DELETE	:channelId	true	Supprimer le canal
/channels	POST		true	Créer un nouveau canal
/channels/:channelId	GET	:channelId	true	Récupérer les informations du canal avec l'ID spécifié

▼ Modifier le nom du canal

#### information sur la route

Route	Méthode HTTP	Params URL	Authentification	Action
/channels/:channelId	PUT	:channelId	true	Modifier le nom du canal

- REQUEST BODY

```
{  
  "name": "string"  
}
```

▼ Supprimer le canal

#### Information sur la route

Route	Méthode HTTP	Params URL	Authentification	Action
/channels/:channelId	DELETE	:channelId	true	Supprimer le canal

- TOKEN REQUIRED
- request body : vide
- DELETE en cascade sur conversation et participants

► Créer un nouveau canal

► Récupérer les informations du canal avec l'ID spécifié

En résumé, nous envoyons les informations suivantes :

- Les détails de l'utilisateur pour alimenter le store.
- Le token et le refreshToken à stocker dans le local storage.
- Les informations nécessaires à son intégration dans le canal général vers lequel il sera redirigé.

L'ensemble de ces routes a ensuite été regroupé dans un document Markdown et rendu disponible aux développeurs sur le projet GitHub ainsi que sur Notion. Cela a facilité la consultation de cette ressource pendant le processus de développement. Le principe était le suivant : après la Merge Request du développeur ayant résolu l'issue, la documentation était mise à jour si des changements étaient nécessaires. Ainsi, la documentation pouvait être régulièrement mise à jour lors de nouvelles itérations, améliorations ou corrections nécessaires. Cette approche nous a permis d'avoir une vision claire des attentes lors du développement du front-end.

# Développement des routes nécessaires

La mise en place de l'ORM et la répartition du travail ont favorisé un développement rapide de chaque route. Le travail a été divisé par entité, avec des issues dédiées à chaque entité et ses routes correspondantes. Cette approche a permis de créer une documentation claire pour chaque développeur et d'assurer une compréhension globale du travail effectué.

Grâce à cette organisation préalable, nous avons pu mettre en place les 35 routes nécessaires au bon fonctionnement de l'application en seulement 3 jours. Les merge requests ont permis à chaque développeur de prendre connaissance du travail des autres et de signaler d'éventuelles anomalies dans le code. Cela a également facilité la cohérence des données ainsi que la coordination entre les deux développeurs s'occupant d'optimisé et factorisé le code à la fin du sprint.

## Intégration des middleWare gestion du JSON Web Token

L'intégration des middlewares est une étape essentielle dans la mise en place de notre application. Parmi ces middlewares, nous accordons une attention particulière à l'intégration du token JWT. Ce token joue un rôle crucial dans le contrôle d'accès aux différentes routes de notre application. Le middleware extrait le token de l'en-tête de la requête, le décide et le vérifie à l'aide de la clé secrète utilisée lors de la signature. S'il s'avère valide, l'accès à la route est autorisé, sinon une erreur d'authentification est renvoyée.

## Intégration des middleWare gestion du JSON Web Token

Nous utilisons le Token JWT pour authentifier les utilisateurs de notre service. Bien qu'il ne garantisse pas la prévention du vol du token lui-même, il nous permet de vérifier l'authenticité de l'utilisateur à l'origine de la requête. Nous avons deux tokens JWT, chacun avec une clé de chiffrement différente de 256 bits. Le premier token a une durée de validité d'un jour et contient la date de création, l'ID de l'utilisateur et son rôle. Le deuxième, appelé refreshToken, a une durée de validité de trois jours et permet uniquement de rafraîchir le premier token s'il n'a pas expiré. Le premier token nous assure que l'utilisateur a les droits d'accès appropriés, tandis que le deuxième permet la mise à jour si nécessaire.

Pour ce faire nous avons utilisé la librairie jsonwebtoken 9.0.0.

Nous avons tout d'abord stocké les variables sensibles liées aux tokens JWT dans notre fichier .env, qui est exclu de Git afin de prévenir tout accès non autorisé à ces informations confidentielles. Celui-ci comprend la clef de chiffrement secrète ainsi que le temps d'expiration des tokens.

```
JWT_KEY="FF9693CA4367C32BE14B6276EEFBE"
JWT_EXPIRES_IN="1d"
JWT_REFRESH_KEY="F2DCC743F6CEBE83C2A258259D21D"
JWT_REFRESH_EXPIRES_IN="3d"
```

À l'aide de la fonction «sign» de la librairie, nous créons et incluons les deux tokens JWT dans notre réponse, garantissant ainsi leur génération sécurisée et leur utilisation dans notre application. Ces deux tokens seront ensuite stockés dans le LocalStorage du client, assurant ainsi la persistance de sa session côté client.

```
// Création du token
const token = jwt.sign(
  { id: user.id, role: user.role },
  process.env.JWT_KEY,
  {
    expiresIn: process.env.JWT_EXPIRES_IN,
  }
);

//Creation du refresh token
const refreshToken = jwt.sign(
  { id: user.id, role: user.role },
  process.env.JWT_REFRESH_KEY,
  { expiresIn: process.env.JWT_REFRESH_EXPIRES_IN }
);
```



Il convient de noter que le stockage des tokens dans le LocalStorage présente des risques de sécurité, car ils peuvent être vulnérables aux attaques XSS (Cross-Site Scripting). Il est préférable d'utiliser des mécanismes de stockage plus sécurisés tels que les cookies sécurisés avec des attributs appropriés pour la gestion des sessions.

Pour cela il existe des librairies adaptées à l'usage de React Native telles que react-native-cookies. Cette librairie offre la fonctionnalité httpOnly: boolean; que nous pourrions mettre à true afin de s'assurer que le JXT n'apparaisse pas. Une autre partie de la sécurité serait côté client en s'assurant que l'en-tête comprend bien les cookie en mettant credentials : true

Dans notre architecture, nous avons défini trois fonctions fondamentales pour l'utilisation du token JWT :

- **CheckToken** : Cette fonction est responsable de la vérification de la validité du token. Elle s'assure que le token n'a pas été altéré et qu'il n'a pas expiré. Si le token est invalide, l'accès à la route est refusé.
- **CheckIsAdmin** : Cette fonction est réservée aux routes accessibles uniquement aux administrateurs. Elle vérifie si l'utilisateur qui fait la requête possède les autorisations nécessaires pour accéder à la route. Si l'utilisateur n'est pas un administrateur, l'accès est refusé.
- **refreshToken** : Cette fonction permet de rafraîchir le token en fonction de la validité du refreshToken. Lorsque le token expire, le refreshToken peut être utilisé pour obtenir un nouveau token valide sans nécessiter une nouvelle connexion. Cela permet une expérience utilisateur fluide et sécurisée.

```

back > src > middlewares > tokenMiddleware.js > ...
...
1 import dotenv from "dotenv";
2 import jwt from "jsonwebtoken";
3 import DB from "../config/db.js";
4
5 const User = DB.User;
6
7 dotenv.config();
8
9 > const checkToken = (req, res, next) => { ...
10
11 };
12
13 > const checkIsAdmin = (req, res, next) => { ...
14 };
15
16 > const refreshToken = async (req, res, next) => { ...
17 };
18
19 };
20
21
22 export { checkToken, checkIsAdmin, refreshToken };
23
24

```

En intégrant ces fonctions dans nos routes, nous assurons un contrôle d'accès sécurisé et granulaire, en autorisant uniquement les utilisateurs authentifiés et autorisés à accéder aux ressources appropriées. Cela garantit la confidentialité et l'intégrité des données, ainsi que la protection contre les accès non autorisés.

## Explication du fonctionnement du JWT

### Token Encodé

eyJhbGciOiJIUzI1NilsInR5cCI6IkpXVCJ9.eyJzdWliOilxMjM0  
 NTY3ODkwliwibmFtZSI6IkpvaG4gRG9IliwiaWF0IjoxNTE2Mj  
 M5MDlyfQ.SfIKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_ad  
 Qssw5c

### Token Décodé

#### Header

- "alg" : algorithme de chiffrement utilisé pour signer le token
- "typ" : type du token, qui est généralement "JWT"

Exemple :

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

#### Payload

contient les données de l'utilisateur ou les informations supplémentaires

exemple :

```
{
  "sub": "1234567890",
  "id": "John Doe",
  "role": "ADMIN",
  "iat": 1516239022
}
```

#### Signature

- Le header et le payload sont encodés en base64 et concaténés
- la clé secrète est utilisée pour signer le résultat avec un algorithme de signature comme HMACSHA256.

# Usage approfondie de Sequelize

L'utilisation de Sequelize présente plusieurs avantages significatifs. Tout d'abord, il offre une couche d'abstraction qui permet une meilleure visibilité et une facilité de compréhension du code grâce à des noms de fonctions explicites. De plus, Sequelize facilite la gestion des transactions, ce qui permet d'effectuer des opérations complexes de manière simplifiée et sécurisée. En outre, Sequelize fournit des fonctionnalités supplémentaires pour renforcer la sécurité des données, notamment la validation des entrées et la prévention des attaques par injection SQL.



Présentation d'une partie de code servant de démonstration à l'usage de Sequelize. Cette fonction permet de retourné l'ensemble des participants d'un canal ainsi que leurs nom et prénom.



Pour des requêtes plus complexes, Sequelize offre la possibilité d'exprimer directement notre requête en SQL. Cependant, il est important de noter que cette approche peut présenter des vulnérabilités potentielles. En effet, la CVE Détails répertorie [12 problèmes de sécurité connus liés à cette bibliothèque](#) (tous corrigés à ce jour). Par conséquent, il est essentiel de faire preuve de vigilance lors de l'utilisation de cette fonctionnalité et de prendre les mesures nécessaires pour sécuriser correctement les requêtes SQL. Notamment en s'assurant de bien échapper les caractères spéciaux afin d'éviter les injections SQL.

## Exemple de requête spécifique en SQL

```
SELECT
    co.createdAt AS sendAt,
    c.id AS channelId,
    c.name AS channelName,
    CONCAT(u.firstname, ' ', u.lastname, ' : ', co.message) AS LastMessage
FROM conversations AS co
INNER JOIN participants AS p1 ON p1.id = co.participantId
INNER JOIN users AS u ON u.id = p1.userId
INNER JOIN channels AS c ON c.id = p1.channelId
WHERE co.createdAt IN (
    SELECT MAX(co2.createdAt)
    FROM conversations AS co2
    INNER JOIN participants AS p2 ON p2.id = co2.participantId
    WHERE p2.channelId IN (
        (SELECT pA.channelId
        FROM participants AS pA
        JOIN channels AS cA ON cA.id = pA.channelId
        JOIN users AS uA ON uA.id = pA.userId
        WHERE pA.userId = ${id}
        AND pA.deletedAt IS NULL)
        GROUP BY p2.channelId)
    ORDER BY co.createdAt DESC;
```

### Requête Principale

Récupère les informations des conversations, des participants, des utilisateurs et des channels associés, en filtrant les conversations en fonction des valeurs rentrées par la SubRequest A.

### SubRequest A

Extrait la date de création maximale parmi les conversations liées aux participants ayant les channelId présents dans le retour de la SubRequest B suivante. Ces dates maximales groupé par channels seront ensuite utilisées dans la requête principale pour filtrer les conversations en fonction de cette valeur.

### SubRequest B

Extrait les channelId des participants associés à un utilisateur spécifique qui n'ont pas été supprimés. Ces channelId seront ensuite utilisés dans la subrequest A pour filtrer les conversations en fonction de ces valeurs.

Extrait de code d'une requête SQL dans une fonction query de Sequelize.

Nous pouvons observer que l'utilisation de requêtes personnalisées nous permet de plus de flexibilité dans l'exécution des requêtes. Cependant, cette approche présente des risques potentiels en termes de sécurité. Bien que cela améliore la lisibilité, il peut y avoir des problèmes de logique. Dans cet exemple, les messages sont sélectionnés en fonction de la valeur createdAt (DATETIME), ce qui peut poser des problèmes dans des cas très rares de retour en arrière. Il serait donc préférable d'identifier les ressources par leur ID. Cependant, ce problème reste mineur.

# Créer des tests unitaires associés aux routes

L'exemple suivant illustre une partie du test unitaire associé à la route «/participants/me» qui permet de consulter l'état de tous les canaux auxquels l'utilisateur est abonné. Dans cet exemple, nous testons le retour de l'utilisateur ayant l'identifiant 5 en utilisant un token valide généré pour ce test spécifique.

```
const axios = require("axios");
const jwt = require("jsonwebtoken");
require("dotenv").config();

const token = jwt.sign({ id: 5, role: "GUEST" }, process.env.JWT_KEY, {
  expiresIn: process.env.JWT_EXPIRES_IN,
});

test("Vérifie si les données sont au bon format", async () => {
  const response = await axios.get(
    "http://localhost:3000/api/participants/me",
    {
      headers: {
        Authorization: `Bearer ${token}`,
      },
    }
  );

  const channels = response.data.data;
  // Vérifiez que les données sont bien un tableau
  expect(Array.isArray(channels)).toBe(true);

  // Vérifiez chaque canal dans les données
  channels.forEach((channel) => {
    expect(channel.sendAt).toBeDefined();
    expect(channel.sendAt).toMatch(
      /\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}\.\d{3}Z/
    );
    expect(typeof channel.channelId).toBe("number");
    expect(typeof channel.channelName).toBe("string");
    expect(typeof channel.LastMessage).toBe("string");
  });
});
```

Extrait de code d'un test unitaire servant à s'assurer de l'intégrité de la route participants/me

Bien que l'application du workflow en TDD (Test Driven Development) soit une excellente approche pour s'assurer de la solidité et de l'intégrité des fonctions inhérentes au projet, nous avons préféré limiter le nombre de tests unitaires effectués afin d'améliorer la productivité à court terme. Cependant, nous nous sommes employés à couvrir des fonctions sensibles avec un panel de tests.

## Retour des différents tests exécutés pour cette requête :

```
> app-mobile-chat@1.0.0 test
> jest

PASS  test/participantMe.test.js
  ✓ Récupération des canaux avec un token invalide (151 ms)
  ✓ Récupération des canaux de l'utilisateur connecté (23 ms)
  ✓ Vérifie si les données sont au bon (10 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.999 s, estimated 1 s
Ran all test suites.

PS C:\Users\bougu\sent-chat\back> []
```

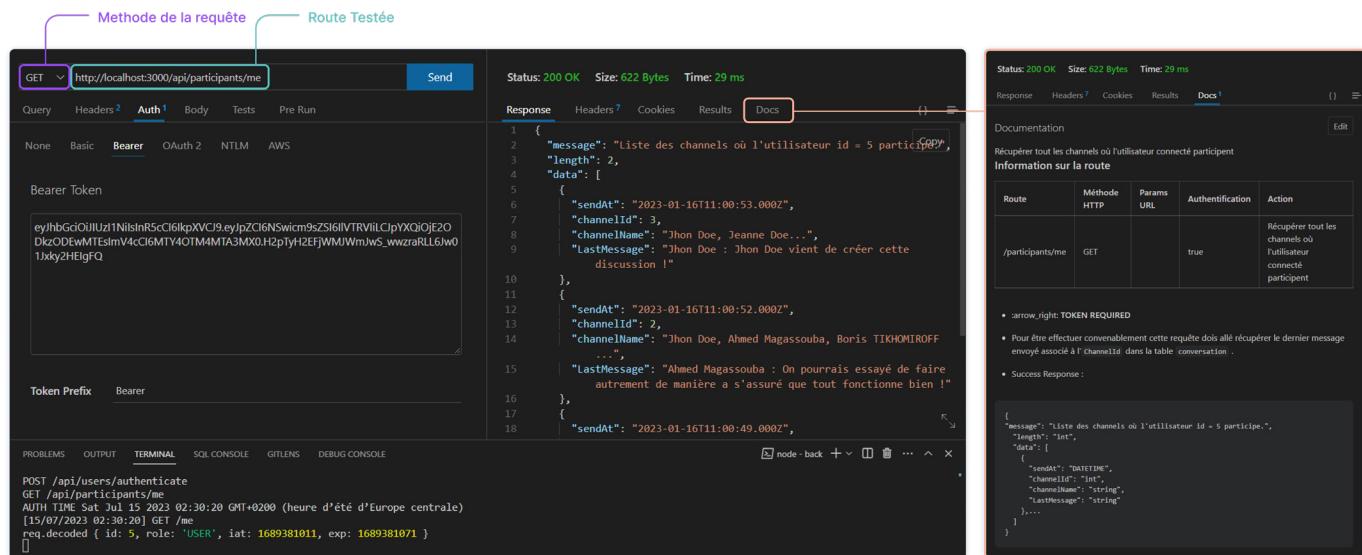
Screenshot de la console lors du passage des tests

# Test Fonctionnel des routes avec Thunder Client

Nous avons vu préalablement que chaque route avait été documenté dans les issues pour garantir la cohérence des données et faciliter le développement côté client prévu la semaine suivante. Pour assurer la conformité des demandes, nous avons effectué des tests d'API simultanément afin d'affiner les résultats des ressources attendues.

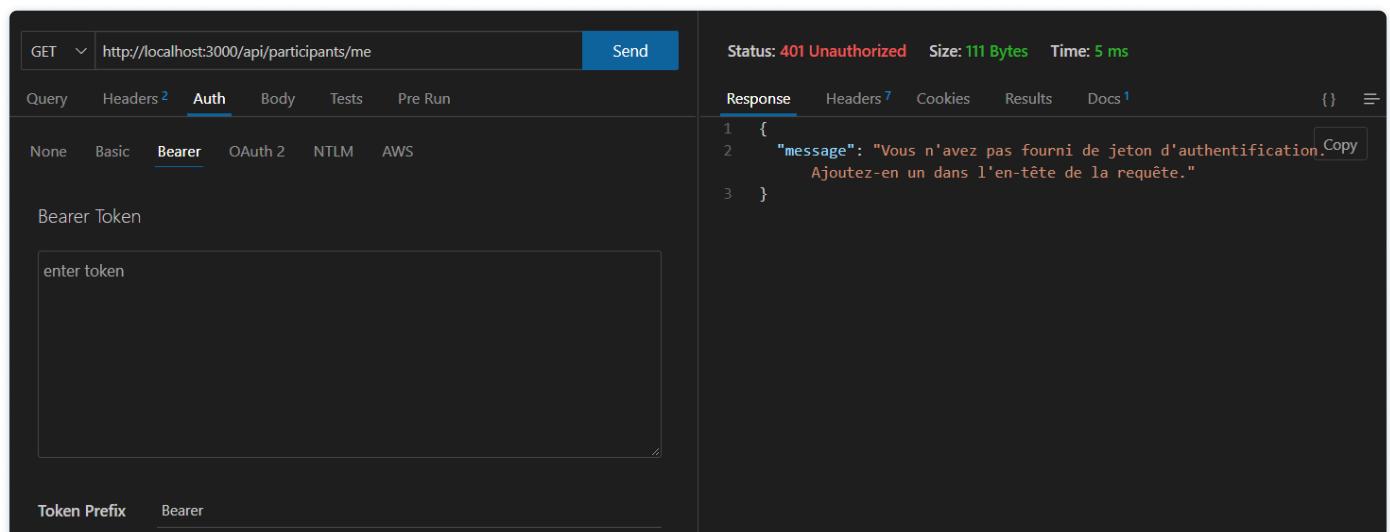
Thunder Client est une extension pour Visual Studio Code qui permet d'effectuer des requêtes HTTP directement depuis l'éditeur. Cet outil facilite le test et le débogage des API en fournissant une interface conviviale pour envoyer des requêtes, visualiser les réponses et inspecter les en-têtes et les données. Il offre des fonctionnalités avancées telles que la gestion des environnements, l'enregistrement de requêtes et la génération de scripts automatisés, ce qui en fait un outil précieux pour les développeurs lors du développement et du débogage des API.

## Exemple de retour en cas de succès



Screen shot de l'interface proposée par Thunder Client

## Screen shot de l'interface proposée par Thunder Client



Thunder Client nous a offert la possibilité de créer des collections liées à notre application, ce qui facilite le suivi des appels de test et permet un partage facile entre les membres de l'équipe. Grâce à cette fonctionnalité, nous avons pu organiser et gérer efficacement nos tests API, améliorant ainsi la collaboration et la productivité.

The screenshot shows the Thunder Client interface. On the left, there's a sidebar with a tree view of API endpoints under 'sent\_chat'. The main area shows a POST request to 'http://localhost:3000/api/users/authenticate'. The 'Body' tab is selected, showing a JSON payload:

```

1 {
2   "email": "jeanne@mail.fr",
3   "password": "Test1234"
4 }

```

The response tab shows the following details:

- Status: 200 OK
- Size: 695 Bytes
- Time: 83 ms
- Headers: 7
- Cookies: 0
- Results: 1
- Docs: 1

The 'Authentifier un Utilisateur' section contains the route information:

Route	Méthode HTTP	Params URL	Authentification	Action
/users/authenticate	POST		false	Authentifier un utilisateur

Below the route table, there are two bullet points:

- route : http://localhost:3000/api/users/authenticate
- Body attendu

The 'Success response' section shows the JSON response body:

```

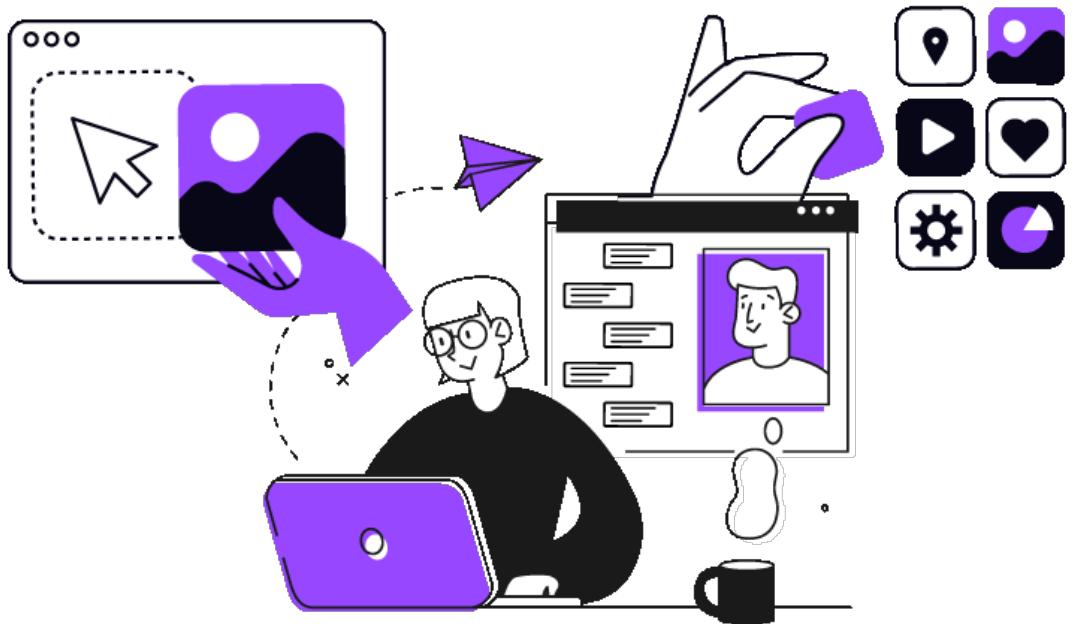
{
  "message": "L'utilisateur a été connecté avec succès",
  "data": {
    "id": 1,
    "firstname": "string",
    "lastname": "string",
    "email": "string",
    "phone": "string",
    "role": "string",
    "createdAt": "dateTime",
    "updatedAt": "dateTime",
    "deletedAt": null
  },
  "token": "/^Bearer\\s[A-Za-z0-9-_]+\\.[A-Za-z0-9-_]+\\.[A-Za-z0-9-_]*$/",
  "refreshToken": ""
}

```

## Un Premier Sprint Réussi grâce à l'Optimisation des Processus de Développement

Au terme de ce premier sprint, nous pouvons affirmer que celui-ci a été couronné de succès, compte tenu du temps imparti. Bien que notre approche initiale reposait principalement sur l'estimation de la quantité de travail à réaliser, il est maintenant évident que nos estimations étaient précises. La mise en détail des tickets a certes demandé du temps, mais s'est avérée être un outil puissant pour guider notre production.

Grâce à cette approche, nous avons pu permettre à deux de nos collaborateurs, Boris TIKHOMIROFF et Cyril Porez, de se concentrer en avance sur l'élaboration du front-end. Leur objectif était de rechercher des solutions adaptées pour l'architecture front-end, tout en veillant à une transmission fluide des informations pertinentes et à une documentation complète concernant leurs choix. De notre côté, Ahmed Magassouba et moi-même avons passé en revue chaque route afin de garantir leur bon fonctionnement dans diverses conditions.



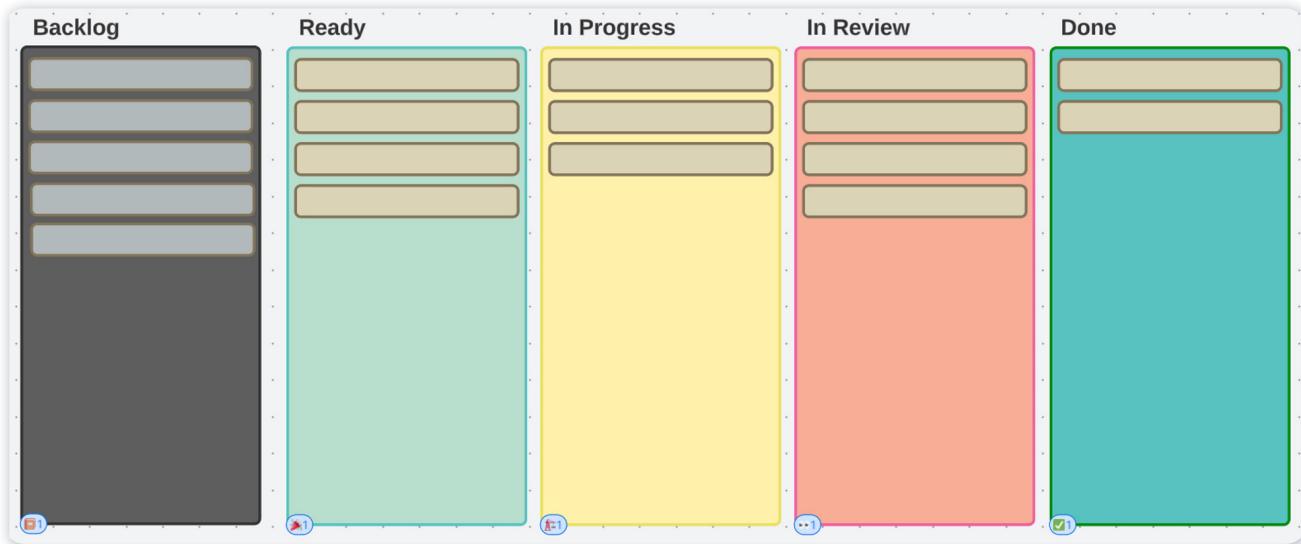
## Deuxième Sprint | Développement du Front-End

Dans ce chapitre, nous allons plonger dans le déroulement et l'organisation du sprint de la deuxième semaine, entièrement dédié à la mise en place de l'architecture front-end et au développement des écrans essentiels. Nous explorerons en détail l'architecture dossier, l'utilisation d'Expo, les contraintes liées à son utilisation, ainsi que l'intégration de Redux pour la création d'un store dans le contexte d'une application en pleine croissance. De plus, nous aborderons la sélection de composants spécifiques et de bibliothèques connexes pour la gestion des entrées utilisateur. Tout en passant en revue les différentes bibliothèques que nous avons mises en place, l'utilisation de nos composants modulaires et la conception de l'architecture des piles (stacks).

Nous approfondirons notamment l'utilisation de board selon la méthode kaban, la gestion des ticket à travers les Userr Storie, l'organisation des label pour une évalution sensible de la vélocité ainsi que les Jalon comme marqueur de production.

# Organisation de notre git-Hub Project

## Organisation de notre board Kaban



### Backlog

Le Backlog représente l'ensemble des issues définies lors de la cérémonie de planification. Ces issues sont directement découpées à partir des User Stories, et elles définissent le périmètre de la tâche à exécuter. Elles comprennent également des consignes annexes pour clarifier le rôle du développeur lors de la réalisation de la tâche. Le Backlog est rigoureusement suivi et géré tout au long du processus de développement.

### Ready

Toutes les issues marquées comme «Ready» sont des issues estimées comme étant prêtes à être prises en charge rapidement. Elles ont souvent déjà été partagées au sein de l'équipe et ont fait l'objet d'une discussion approfondie pour clarifier les exigences et les attentes. Lorsqu'une issue est marquée comme prête, elle est disponible pour être assignée à un membre de l'équipe afin d'être traitée dans les plus brefs délais selon les priorités établies.

### In Progress

Lorsqu'une issue est marquée comme «In Review», cela indique qu'une merge request est en attente et que la revue par les pairs est nécessaire pour valider l'issue. Dans notre contexte où nous n'avons pas de testeur dédié, cela implique également que chaque développeur doit prendre l'initiative de tester la branche localement afin de détecter d'éventuels problèmes et de les remonter. Cela nécessite une collaboration et une responsabilité collective de la part de tous les membres de l'équipe.

### Done

Lorsqu'une issue est marquée comme «Done», cela signifie que la branche associée à l'issue a été fusionnée avec succès dans notre branche principale. Aucun effet indésirable n'a été observé et l'issue est considérée comme étant réalisée avec succès. Cela représente une étape importante dans le processus de développement et souligne l'achèvement et la réussite de l'issue.

# Utilisation des Labels

Les labels sur GitHub sont des étiquettes utilisées pour classer les tâches ou les demandes de fonctionnalités dans un projet. Ils permettent de catégoriser les tâches en fonction de leur statut, de leur priorité ou de leur type. Les utilisateurs peuvent créer des labels personnalisés en fonction des besoins de leur projet et les appliquer aux tâches ou aux demandes de fonctionnalités pour les organiser de manière efficace. Les labels permettent de filtrer les tâches et de les regrouper selon des critères spécifiques facilitant ainsi la gestion de projet.

## **Les différents type de label utilisé dans notre méthodes :**

### **Tutors**

Le Labels tutors définis une issue comme étant un parentes a un ensemble de features à apportées. Il est souvent liées a un milestones.

### **documentation**

Le label documentations défini une issue destinées seulement offrir un ensemble de données étant utile aux développeurs. Elle rassemble des éléments centraux permettant a ces derniers d'y apposé des grande modification ou des changement majeurs devant être consulté par les autres

### **question**

Le label question est dédié au discussions de groupes sur un sujet précis. Les sujets peuvent êtres, l'utilisations d'une bibliothèque, l'implémentation d'une nouvelle fonctionnalité ou encore la révision d'étapes organisée. Elle servent a concilié la volonté de chacun sur une vision commune de manière proactive.

### **Back-End**

Issues en relations avec le dossier back

### **Front-End**

Issue en relation avec le dossier front

### **Feature**

L'issue est associé a une amélioration

### **Bug**

Lier a un fix

### **Duplicate**

L'issue est un doublon et a été délaissé au profit d'une autre.  
Lors de l'utilisation de duplicate on lie la nouvelle issue a celle-ci

### **Help Wanted**

L'issue est associé a une amélioration L'issue est en attente d'informations supplémentaire

# Evaluation des priorités

Lors de la mise de l'issue en «Ready», il est essentiel de comprendre l'importance de chaque niveau de priorité pour une meilleure planification et exécution du travail. Voici une explication de l'importance de chaque niveau :

## **Urgent**

Les issues marquées comme urgentes nécessitent une attention immédiate et doivent être traitées en priorité absolue. Elles peuvent inclure des problèmes critiques ou des blocages majeurs qui doivent être résolus rapidement pour éviter tout impact négatif sur le projet ou les utilisateurs.

## **High**

Les issues avec une priorité élevée ont un impact significatif sur le projet et doivent être traitées avec une attention particulière. Elles peuvent inclure des fonctionnalités essentielles, des améliorations importantes ou des corrections de bugs qui doivent être résolus rapidement pour maintenir la qualité et la progression du projet.

## **Medium**

Les issues de priorité moyenne ont une importance modérée et doivent être traitées dans un délai raisonnable. Elles peuvent inclure des fonctionnalités ou des améliorations qui ne sont pas critiques mais qui apportent une valeur ajoutée au projet.

## **Low**

Les issues de priorité basse ont une importance moindre et peuvent être traitées avec une priorité plus faible. Elles peuvent inclure des demandes de fonctionnalités mineures, des améliorations cosmétiques ou des tâches de maintenance qui peuvent être réalisées ultérieurement sans impacter de manière significative le projet.

# Estimation de la vitesse

A la récupération d'une issue nous évaluons la vitesse de la tâche.

## **XLARGE**

L'issue est complexe et risque de prendre tout le temps imparti du sprint pour être résolue, ou nécessite un temps supplémentaire pour sa résolution.

## **Large**

L'issue est de taille importante mais réalisable dans les délais du sprint.

## **Medium**

L'issue est de taille moyenne et ne présente pas de difficultés majeures, pouvant être réalisée dans un temps raisonnable.

## **Small**

L'issue est de petite taille et peut être terminée en moins de 24 heures.

## **Tiny**

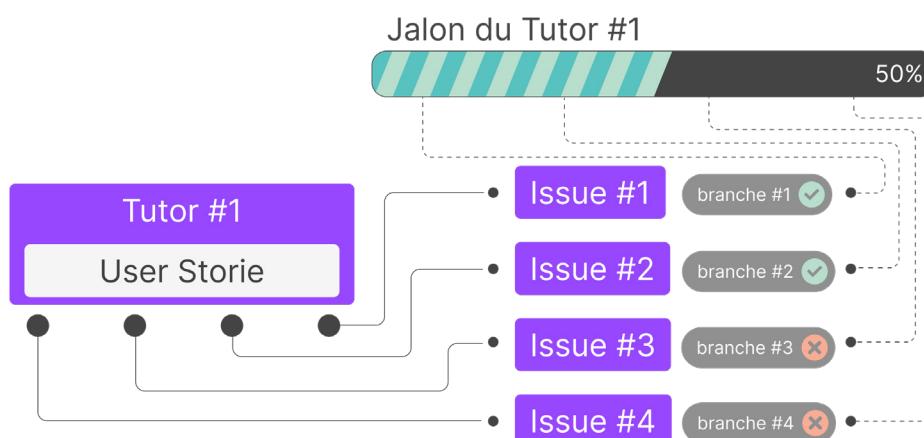
L'issue est de taille très petite et peut être rapidement résolue.

# Mise en place du Backlog pour le développement front-End

Afin de faire face à l'ampleur du développement restant et de respecter les délais impartis, nous avons adopté une approche rigoureuse pour la planification et la gestion du projet. Dans un premier temps, nous avons segmenté les différentes problématiques métier en User Stories, permettant ainsi de définir le travail à accomplir du point de vue de l'utilisateur. Ces User Stories ont ensuite été divisées en tickets liés à des jalons spécifiques.

Nous avons utilisé des tickets appelés «**Tutor**» qui comprenaient une description détaillée de l'User Story correspondante. Cela permettait aux collaborateurs de consulter ces tickets afin de maintenir une vision globale de l'application. Chaque ticket Tuteur était directement lié à un jalon portant son nom, et était associé à des tickets représentant les différentes étapes à réaliser dans ce même jalon. Cette approche nous a permis de suivre l'avancement de chaque Tuteur de manière précise et de quantifier et identifier les différentes difficultés rencontrées.

## Exemple de l'usage d'un jalon dans notre organisation



Lors des premières cérémonies et de la distribution des tickets, chaque collaborateur s'est vu attribuer des missions avec pour objectif d'évaluer leur \*\*complexité\*\* et leur \*\*vitesse\*\*. Cela a permis d'approfondir la compréhension des demandes et d'aborder la réflexion sur les outils nécessaires. De plus, des encarts présents sur les maquettes ont permis de prendre en compte la complexité des différents composants.

Les missions ont été réparties de manière à ce que les composants utilisés dans différents écrans soient rapidement disponibles pour les autres collaborateurs, favorisant ainsi une meilleure collaboration et un flux de travail efficace.

# Outils utilisé



## Expo Go | Outil communautaire

- Environnement d'exécution adapté
- Test rapide sur Mobile
- SDK adapté à l'usage



## React Native | Framework Front-End Cross-Platform destinée l'édition d'application Mobile

- Transcompilassions du code Javascript en composant Natif
- Programmation modulaire à partir de la librairie React
- Simplification de la gestion d'environnement



## Axios | Client HTTP

- Gestion des requêt asynchone
- Gestion des retour d'erreurs
- Gestion des entêtes HTTP



## Redux | Gestion du State Management selon les principes Flux

- Gestion centralisée de l'état
- Prévisibilité de l'état
- Simplification de la gestion d'état



## Formik | Assistants de formulaire et standardisation des entrées

- Gestion d'erreur spécifiques aux inputs
- Gestion de schéma de validation
- Gestion des Regex



## Yup | Constructeur de schéma pour l'analyse et la validation des valeurs d'exécution

- Gestion d'erreur spécifiques aux inputs
- Gestion de schéma de validation
- Gestion des Regex



## React Native Async Storage | Librairie d'un système de stockage asynchrone, non chiffré, persistant, de type clé-valeur pour React Native.

- Gestion de stockage local de donnée

# Architecture générale du dossier Front-end

L'usage de React Native nous a permis d'adopter une approche modulaire basée sur les principes du développement de composants. Les fichiers de composants sont organisés selon une structure modulaire, favorisant la réutilisabilité et la maintenabilité du code.

Le dossier Expo joue un rôle essentiel en offrant des fonctionnalités et des outils spécifiques à l'environnement Expo, notamment la gestion du fichier manifeste qui contient des informations cruciales pour une expérience utilisateur cohérente. Parallèlement, l'utilisation de modules Node permet de bénéficier de bibliothèques tierces et d'accélérer le développement en réutilisant du code existant, tout en suivant les meilleures pratiques de gestion des dépendances.



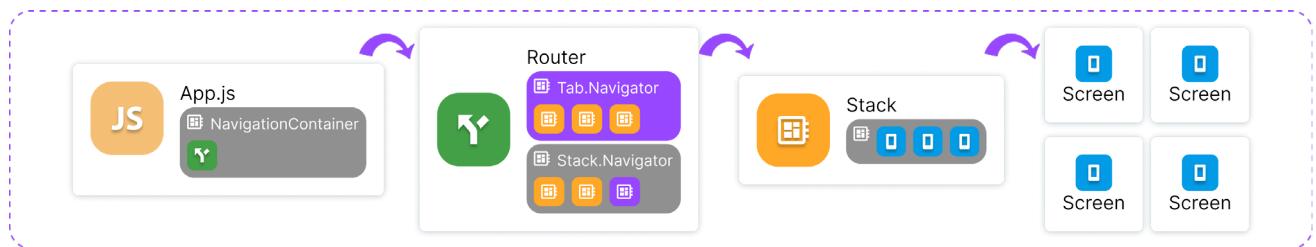
## Développement du Front-end

### Organisation des Stacks et empilement des screens

Au cours de la semaine précédente, nous avons réussi à répondre à notre besoin en construisant une architecture de dossiers adaptée, en appliquant le concept d'architecture basée sur le «Routing» afin d'avoir une approche plus concrète de l'acheminement. Cette approche, largement utilisée dans le développement web, a été adaptée à notre application mobile en React Native afin de faciliter la compréhension de l'empilement des «Stacks» côté client.

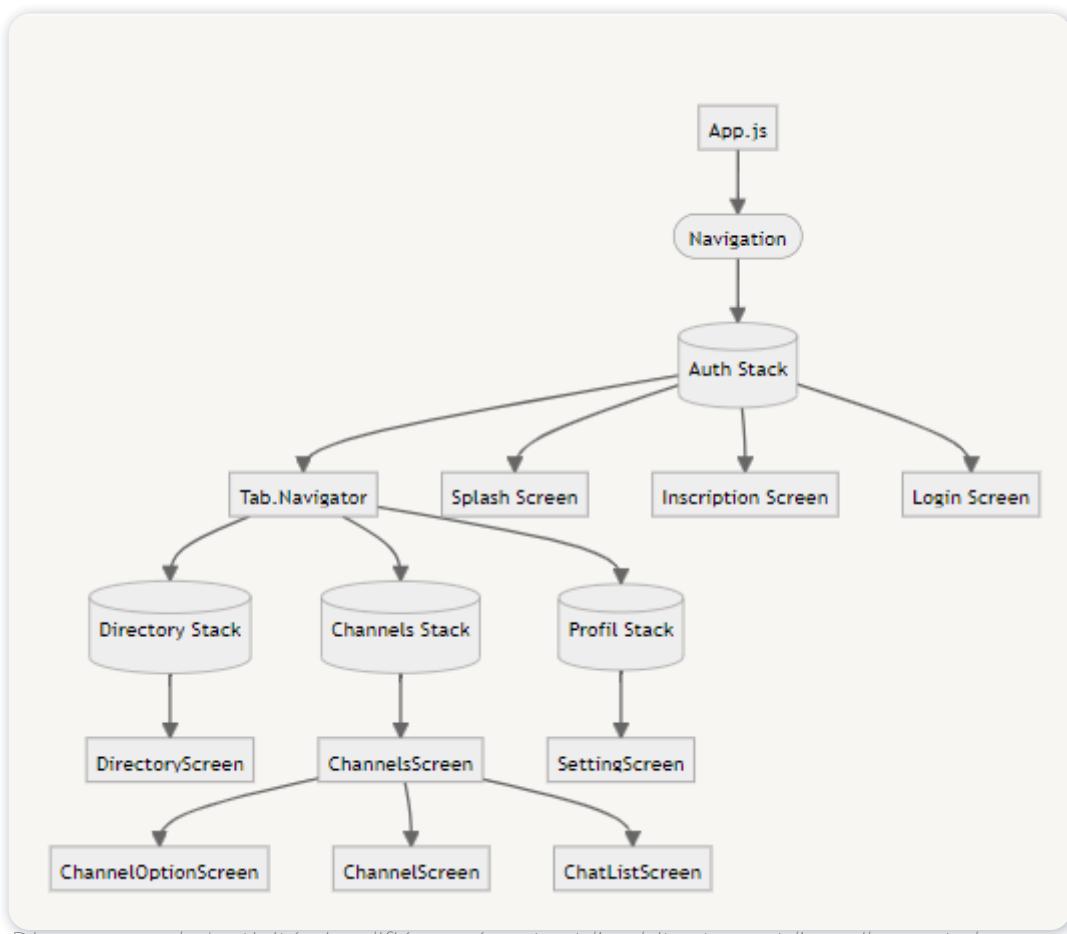
Contrairement à l'approche conventionnelle, nous avons choisi de décomposer les étapes des appels en adoptant une approche inspirée des pratiques du back-end et de la séparation des responsabilités. Ainsi, nous avons créé un dossier nommé «Routes» qui joue un rôle clé dans l'organisation cohérente de notre application. Le fichier principal «App.js» fait appel à ce dossier pour déterminer la distribution et l'ordonnancement des différentes piles. Chaque pile est construite et appelée dans un fichier dédié, où nous définissons nos Stack Navigator qui intègrent les différents screens.

## Ordonnancement et distribution des différentes piles



Cette approche a permis à tous les collaborateurs de comprendre la subtilité entre les routes dans une application web classique et l'empilement des écrans dans une application React Native. Dans une application web classique, les routes sont utilisées pour définir les différentes pages accessibles via des URLs distinctes, tandis que dans une application React Native, l'empilement des écrans est utilisé pour gérer la navigation entre les différentes vues de l'application.

## Diagramme d'empilement des stacks récupérée de l'issue en question



## **Organisation et mise en place de Redux**

En connaissant l'utilisation du «Context» dans React, nous avons fait le choix d'approfondir nos connaissances techniques en optant pour Redux. Ce choix était motivé par notre volonté de développer une application qui puisse évoluer et grandir avec le temps, en bénéficiant d'une gestion d'état centralisée et structurée. Bien que le Context offre une solution plus légère et plus simple, Redux nous permet d'explorer des concepts avancés tels que les actions, les reducers

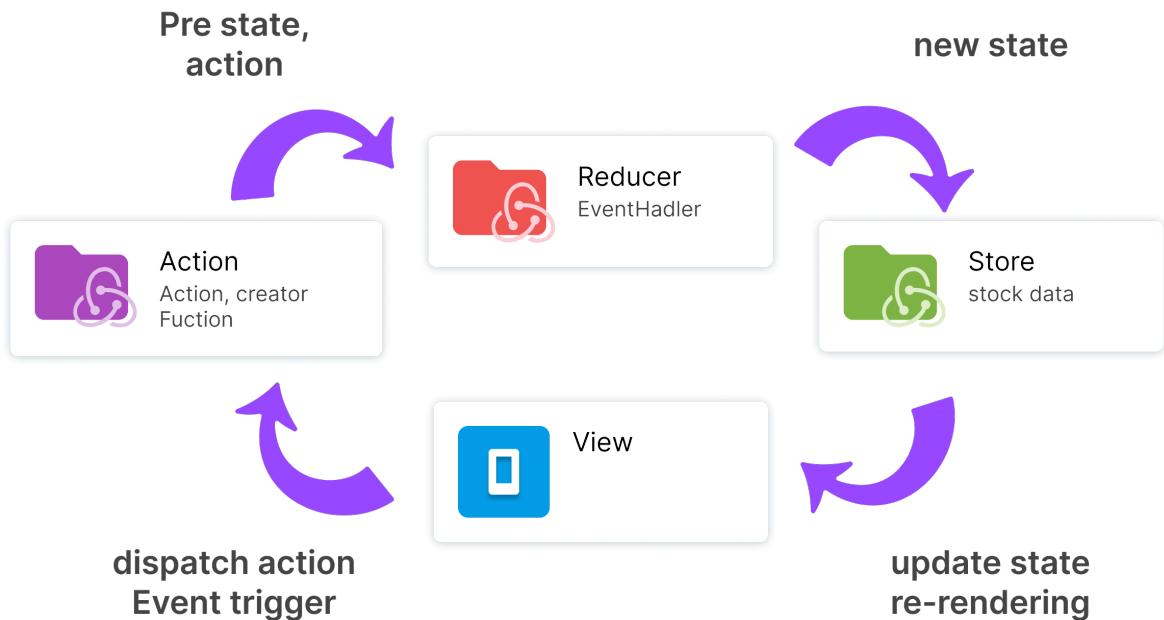
et l'écosystème riche qui l'accompagne

La première chose à savoir à propos de Redux, est qu'il nous permet de stocker un état général de l'application dans un objet appelé store accessible par l'ensemble des composants de l'application. Cet objet est en read-only ce qui signifie que la manipulation des states stockés au sein du store ne sont pas fait directement, ils sont trigger par une Action.

Redux apporte le concept de state management (gestion d'état) au sein de l'application.

Sans Redux, il faudrait rendre les données dépendantes des composants et les faire passer par différents composants jusqu'à l'endroit où elles sont nécessaires, ce qui nous amènerait à faire du props drilling, qui dans certains cas n'est pas recommandé.

### **Détail du fonctionnement de Redux**



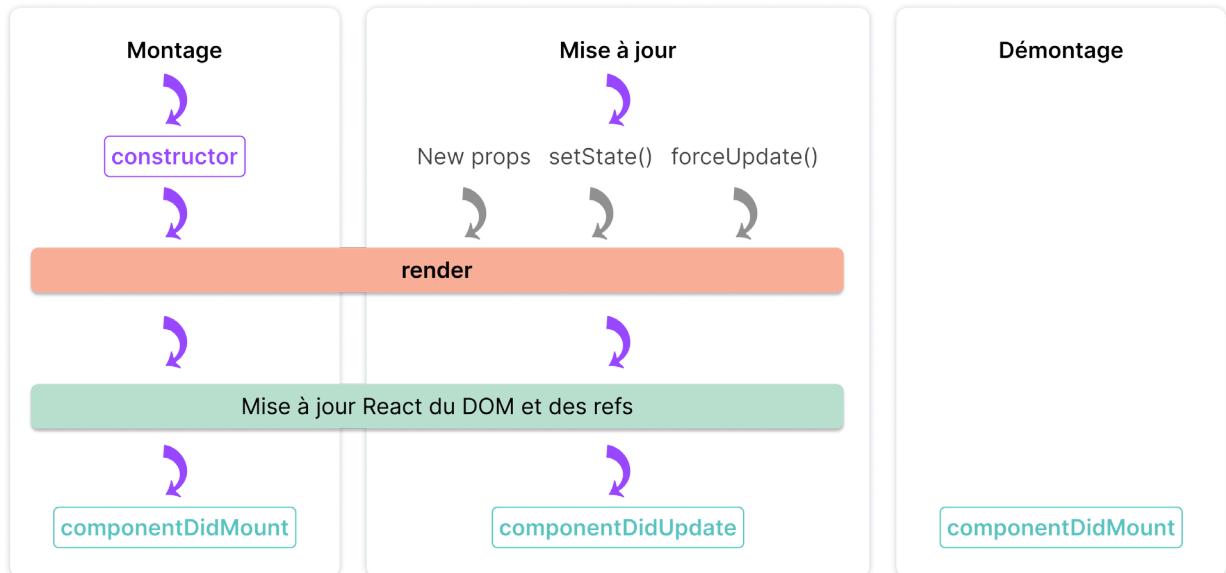
## **Exemple d'utilisation de composant avec React Native**

Dans le chapitre sur la conception du wireframe, nous avons exploré le principe d'atomicité et ses avantages lors de l'utilisation de bibliothèques ou de frameworks tels que React ou React Native. Ce chapitre nous permettra d'approfondir l'utilisation de ce principe à travers des exemples simples et représentatifs, mettant en évidence comment il facilite la modularité, la réutilisabilité et la maintenabilité du code.

Nous allons commencer ce chapitre en expliquant brièvement le fonctionnement de React, en mettant l'accent sur les concepts clés tels que le cycle de vie des composants et les Props, qui sont manipulés à l'aide des Hooks. Nous aborderons également la distinction entre les composants «stateless» (dumb) et «stateful» (smart), ainsi que l'application du concept de «molécule» à travers un exemple concret.

# Gestion de cycle de vie dans un composant React

Les cycles de vie des composants dans React sont des étapes clés qui contrôlent la création, la mise à jour et la destruction d'un composant. Ils permettent d'exécuter du code spécifique à chaque phase. Avec les méthodes de classe ou les Hooks, on peut gérer ces cycles de vie de manière déclarative, assurant ainsi un fonctionnement optimal des composants.



En React Native, il n'y a pas à proprement parler la manipulation du Document Object Model (DOM). Mais il y a un homologue Virtual DOM. Ce dernier fait une mise à jour sélective (diffing) lors d'une modification de composant, ce qui améliore les performances de l'application et évite de re-render l'ensemble de l'application.

## Utilisation des états et des hooks en React

En React, deux approches sont utilisées pour déclarer les propriétés d'un composant. La première, orientée objet, consiste à les déclarer dans le constructeur et à utiliser des méthodes pour les modifier. L'autre approche, plus fonctionnelle, utilise le hook `useState()` qui permet de déclarer le nom, l'état et le setter d'une propriété directement à partir d'une fonction. Cette approche simplifie la gestion de l'état dans les composants fonctionnels.

### Usage des hooks

```
const [ propriété , setter ] = useState( valeur );
```

En React, `useState` est un hook qui permet de gérer l'état dans un composant fonctionnel. Lorsque vous appelez `useState` avec une valeur initiale, il retourne un tableau contenant deux éléments :

1. La première valeur dans le tableau est la valeur de l'état elle-même (par exemple, `count` dans notre exemple).

2. La deuxième valeur est une fonction qui permet de mettre à jour l'état (par exemple, `setCount` dans notre exemple).

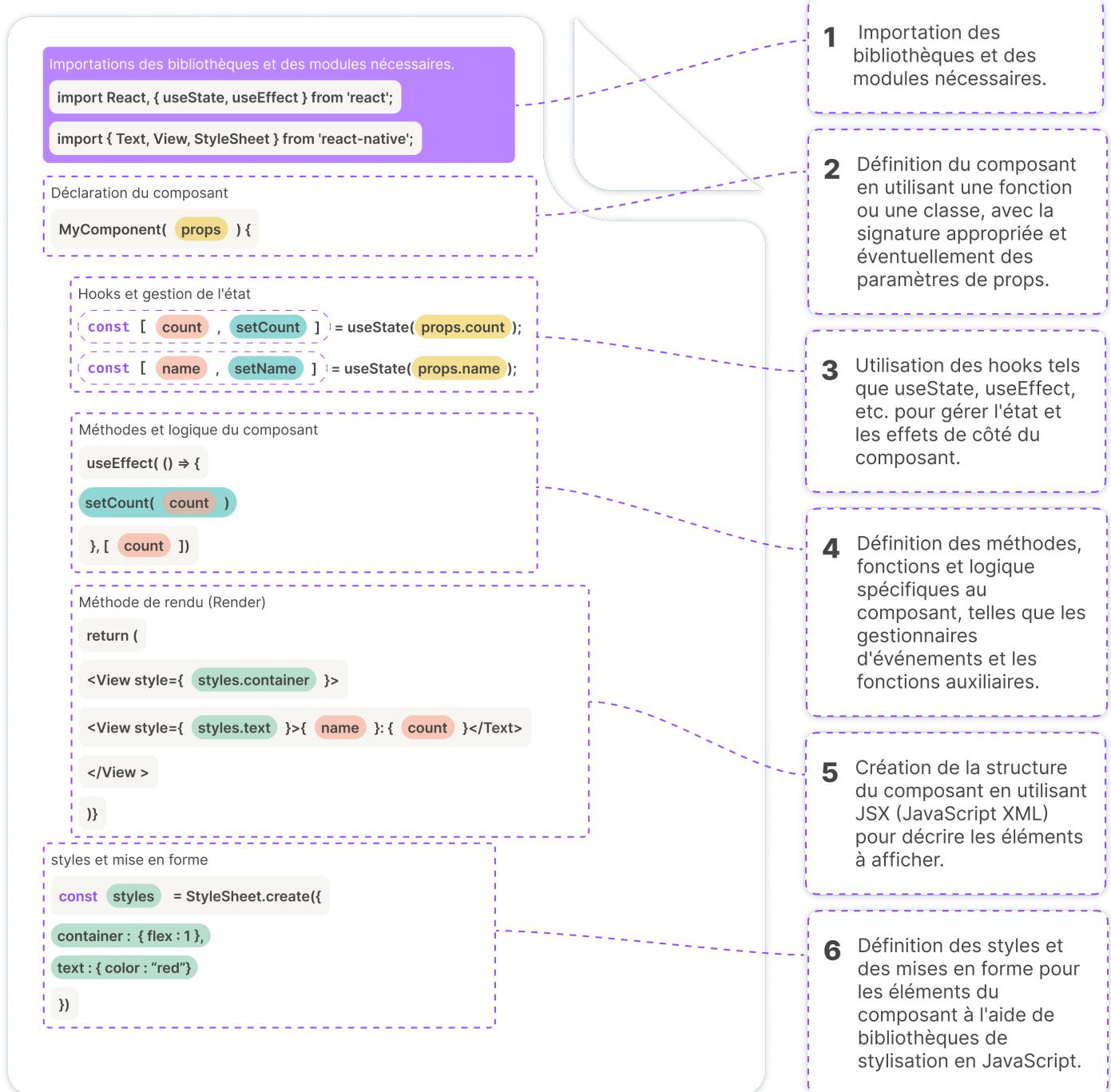
En utilisant la déstructuration (le **destructuring assignment**), nous attribuons les valeurs du tableau retourné par `useState` à des variables distinctes (`count` et `setCount` dans notre exemple).

Lorsque nous utilisons `count` dans notre composant, nous accédons à la valeur actuelle de l'état. Pour modifier cette valeur, nous appelons simplement **setCount** avec la nouvelle valeur que nous souhaitons lui assigner. Lorsque **setCount** est appelé, React se charge de mettre à jour l'état et de re-render le composant avec la nouvelle valeur.

```
//Orienté objet
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: this.props.initialCount
    };
  }
}

//orienté fonctionnelle
function MyComponent(props) {
  const [count, setCount] = useState(props.initialCount);
```

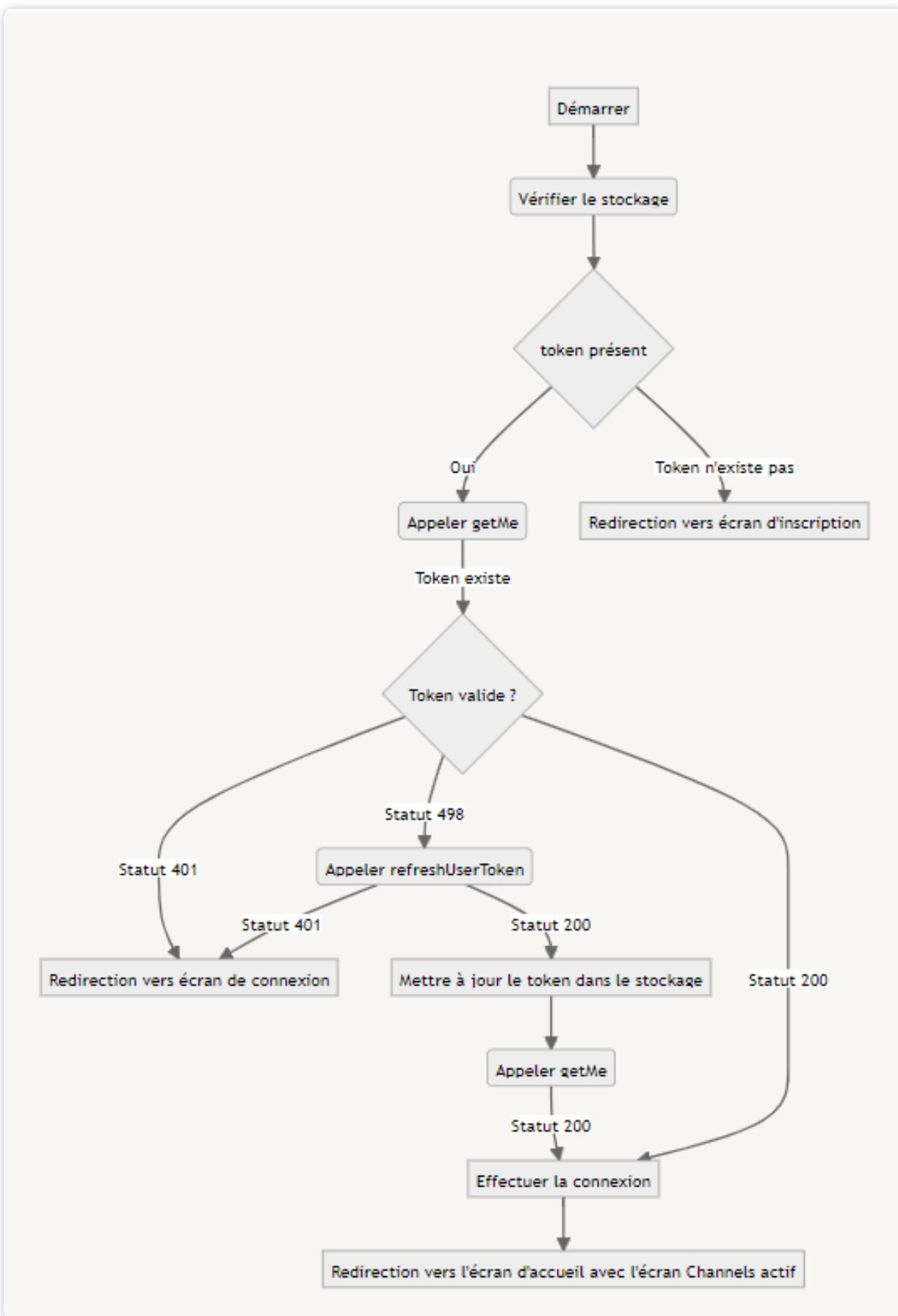
# Présentation de la structure basique d'un composant en React



## Exemple de Jeu d'essaie sur la fonctionnalité d'authentification automatique

Nous avons effectué des tests approfondis sur la fonctionnalité d'authentification de l'application, en simulant différentes situations pour garantir sa fiabilité et sa sécurité. Afin d'illustrer un exemple de code fonctionnel, prenons l'exemple de l'authentification de l'utilisateur, comme décrit dans la spécification du zoning, qui met en avant «*l'utilisation du Token JWT pour limiter les points de friction*». La fonction suivante est appelée après le chargement dans l'écran de démarrage, une fois que l'utilisateur a lancé l'application et que la connexion à la base de données a été établie. Nous vérifions d'abord la présence du Token JWT dans le LocalStorage, puis nous redirigeons l'utilisateur vers l'écran approprié en fonction de sa présence et des circonstances.

## Diagramme d'activité présentant la logique attendue dans le déroulement de la fonction



## Scénarios de test

Numéro	Description de l'entrée	Description du scénario	Résultat attendu
1	Données en stockage : { "auth": { "token": "TOKEN_VALIDE", "refreshToken": "REFRESH_TOKEN" } }	Le <u>token</u> est valide et existant en stockage.	L'utilisateur est redirigé vers l'écran d'accueil avec l'écran "Channels" actif.
2	Données en stockage : { "auth": { "token": "TOKEN_EXPIRE", "refreshToken": "REFRESH_TOKEN_VALIDIDE" } }	Le <u>token</u> existant a expiré, mais le <u>refresh token</u> est valide.	L'utilisateur est redirigé vers l'écran d'accueil avec l'écran "Channels" actif.
3	Données en stockage : { "auth": { "token": "TOKEN_EXPIRE", "refreshToken": "REFRESH_TOKEN_EXPIRE" } }	Le <u>token</u> existant a expiré et le <u>refresh token</u> a également expiré.	L'utilisateur est redirigé vers l'écran de connexion.
4	Aucune donnée en stockage	Aucun <u>token</u> en stockage.	L'utilisateur est redirigé vers l'écran d'inscription.

## Présentation de la fonction codé

```
/*
 * Check si le token existe en storage et redirige selon l'état
 */
const authRedirection = () => {

  getItemStorage("auth").then(async (asyncStorageData) => {
    // 1. Check if a token already exist in storage
    if (asyncStorageData?.token) {
      // If token exist
      const userData = await getMe();
      const { status } = userData;

      // 2. Refresh Token
      if (status === 498) {
        // 498 = Token expired
        const { status: codeStatus, data: dataRefresh } =
          await refreshUserToken(asyncStorageData.refreshToken);

        // After refresh token, check if the new token is valid
        switch (codeStatus) {
          case 401: // 401 = Refresh token expired
            navigation.navigate("LoginScreen"); // Navigate to Login
            break;
          case 200: // 200 = Refresh token is valid

            updateToken(dataRefresh.token);
            const updatedUser = await getMe();
            const { data } = updatedUser.data;
            dispatch(
              login(data, datarefresh.token, asyncStorageData.refreshToken)
            );
            navigation.navigate("Home", {
              // Navigate to Home
              screen: "Channels",
            });
            break;
          default:
            dispatch(login(email, password));
            navigation.navigate("Home", {
              // Navigate to Home
              screen: "Channels",
            });
        }
      } else if (status === 200) {
        // 200 = Token is valid
        dispatch(
          login(
            userData.data?.data,
            asyncStorageData.token,
            asyncStorageData.refreshToken
          )
        );
        navigation.navigate("Home", {
          screen: "Channels",
        });
      } else {
        navigation.navigate("LoginScreen");
      }
    } else {
      // If no token in storage
      navigation.navigate("RegisterScreen"); // Navigate to Register
    });
  });
};
```

## Retour du Jeu d'essaie dans les trois scénario



Connexion à la base de données  
Ajout de Bearer Token en cas de présence



Retour de reponse du BackEnd

### ScreenShot du retour console en mode Dev | Scenario 1

```
iOS Bundling complete 57ms
iOS Running app on iPhone
BASE_URL : http://10.10.2.158:3000/api/
-- USER AUTHENTIFIED --
Scenario 1 : Le token est valide et existant en stockage.
etape 1 : token présent ? true
etape 2 : retour du getMe => 200
userData Object {
  "data": Object {
    "createdAt": "2023-07-12T14:05:49.000Z",
    "deletedAt": null,
    "email": "johan@mail.fr",
    "firstname": "Johan",
    "id": 4,
    "lastname": "Bouguermouh",
    "phone": "0606060608",
    "role": "ADMIN",
    "updatedAt": "2023-07-12T14:05:49.000Z",
  },
  "message": "L'utilisateur a été trouvé avec succès",
}
etape 3 : token valide => userData, token, refresh on dispatch for login
etape 4 : navigate to Home, screen Channels
[]
```

### ScreenShot du retour console en mode Dev | Scenario 2

```
iOS Bundling complete 57ms
iOS Running app on iPhone
BASE_URL : http://10.10.2.158:3000/api/
-- USER AUTHENTIFIED --
Scenario 2 : Le token existant a expiré, mais le refresh token est valide.
etape 1 : token présent ? true
ROUTE http://10.10.2.158:3000/api/users/me
ERR REPONSE DATA => Token expiré...
ERR_STATUS=> 498
etape 2 : retour du getMe => 498
etape 3 : retour du status refreshUserToken en cas de status 498 => 200
etape 4 : switch case status code 200 => dispatch login
-- USER HAS AUTHENTIFIED --
etape 5 : navigate to Home, screen Channels
[]
```

### ScreenShot du retour console en mode Dev | Scenario 3

```
iOS Bundling complete 65ms
iOS Running app on iPhone
BASE_URL : http://10.10.2.158:3000/api/
-- USER AUTHENTIFIED --
Scenario 3 : Le token existant a expiré et le refresh token a également expiré.
etape 1 : token présent ? true
ROUTE http://10.10.2.158:3000/api/users/me
ERR_REPONSE DATA => Token expiré...
ERR_STATUS=> 498
etape 2 : retour du getMe => 498
etape 3 : retour du status refreshUserToken en cas de status 498 => 401
etape 4 : switch case status code 401 => navigate to LoginScreen
[]
```

### ScreenShot du retour console en mode Dev | Scenario 4

```
iOS Bundling complete 54ms
iOS Running app on iPhone
BASE_URL : http://10.10.2.158:3000/api/
token not found
Scenario 4 : Aucune donnée token dans le local storage
etape 1 : token présent ? false
etape 2 : token non présent => navigate to RegisterScreen
[]
```

## Pourquoi code error 498 ?

Lors de notre réflexion, nous avons été confrontés à un dilemme concernant la gestion des tokens. Nous avons constaté qu'il existait différentes approches pour traiter les différentes situations liées au token : lorsque le token est valide (code 200), lorsque le token est invalide ou que l'utilisateur n'a pas les droits nécessaires (code 403), et lorsque le token est absent (code 401). Cependant, il nous manquait un code d'erreur spécifique pour indiquer que le token avait expiré. Après des recherches approfondies, nous avons constaté qu'il n'y avait pas de code d'erreur officiellement défini pour cette situation. Il était courant de voir que le code d'erreur 4nn était utilisé, mais cela dépendait des conventions propres à chaque entreprise. Finalement, nous avons découvert, sur différents sites tels que <https://http.dev/498>, que le code d'erreur 498 était souvent associé à ce cas d'utilisation spécifique.

Nous avons donc décidé de créer une issue spécifique pour les modifications du backend en priorité. Une fois la branche créée, le développeur a immédiatement effectué les modifications nécessaires. Après la fusion de la branche, j'ai pu terminer mes tests fonctionnels pour m'assurer que tout fonctionnait correctement.