**SOMMAIRE**

[Remerciements 3](#_Toc111842510)

[Compétences du référentiel couvertes par le projet 4](#_Toc111842511)

[Présentation du projet 6](#_Toc111842512)

[Contexte 6](#_Toc111842513)

[Conception graphique 6](#_Toc111842514)

[Charte graphique 6](#_Toc111842515)

[Maquettes 7](#_Toc111842516)

[Organisation du projet 9](#_Toc111842517)

[Architecture du projet 10](#_Toc111842518)

[Partie client (frontend) 10](#_Toc111842519)

[Partie serveur (backend) 11](#_Toc111842520)

[Schéma de l’application 12](#_Toc111842521)

[Principales fonctionnalités 12](#_Toc111842522)

[Création de compte et authentification 12](#_Toc111842523)

[Gestion de projet et ticket 13](#_Toc111842524)

[Spécification techniques 13](#_Toc111842525)

[Structure de la base de données 13](#_Toc111842526)

[Côté client 14](#_Toc111842527)

[Côté serveur 15](#_Toc111842528)

[Mise en place de la BDD 16](#_Toc111842529)

[Réalisations 19](#_Toc111842530)

[Gestion de l’authentification 19](#_Toc111842531)

[CRUD du projet 19](#_Toc111842532)

[Upload de fichiers 19](#_Toc111842533)

[Tests et déploiement 19](#_Toc111842534)

[Test 19](#_Toc111842535)

[Déploiement 19](#_Toc111842536)

[Conclusion 20](#_Toc111842537)

# Remerciements

# Compétences du référentiel couvertes par le projet

|  |  |  |
| --- | --- | --- |
| Activité types | Compétences professionnelles | Oui |
| Concevoir et développer des composants d’interface utilisateur et intégrant les recommandations de sécurité | Maquetter une application |  |
| Développer une interface utilisateur de type desktop |  |
| 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 et développer la persistance des données en intégrant les recommandations de sécurité | Concevoir une base de données |  |
| Mettre en place une base de données |  |
| Développer des composants dans le langage d’une base de données |  |
| Concevoir et développer une application multicouches repartie en intégrant les recommandations de sécurité | Collaborer à la gestion d’un projet informatique et à l’organisation de l’environnement de développement |  |
| Concevoir une application |  |
| Développer des composants métier |  |
| Construire une application organisée en couches |  |
| Développer une application mobile |  |
| Préparer et exécuter les plans de tests d’une application |  |
| Préparer et exécuter le déploiement d’une application |  |

# Présentation du projet

## Contexte

**« FREYGUES »** est une entreprise spécialisée dans la fabrication de frisbees 100% made in France et disponibles en plusieurs coloris. Cette entreprise a été créée par Aurélien LEYGUES qui est à la fois un joueur professionnel d’ultimate et aussi notre mentor à la **WildCodeSchool**.

L’entreprise est en pleine croissance et a pour objectif de devenir le premier fournisseur de frisbee en Europe d’ici la fin de l’année 2022.

Aujourd’hui **“FREYGUES”** a de plus en plus de collaborateurs et de nouveaux projets. Notre client recherche donc un outil pour mieux gérer ses projets et améliorer la productivité de ses équipes.

Pour répondre au besoin de notre client, on lui a proposé « **TaskApp**». Une application SaaS de gestion de projet qui va lui permettre de planifier et structurer ses projets d’une façon simple et intuitive.

## Conception graphique

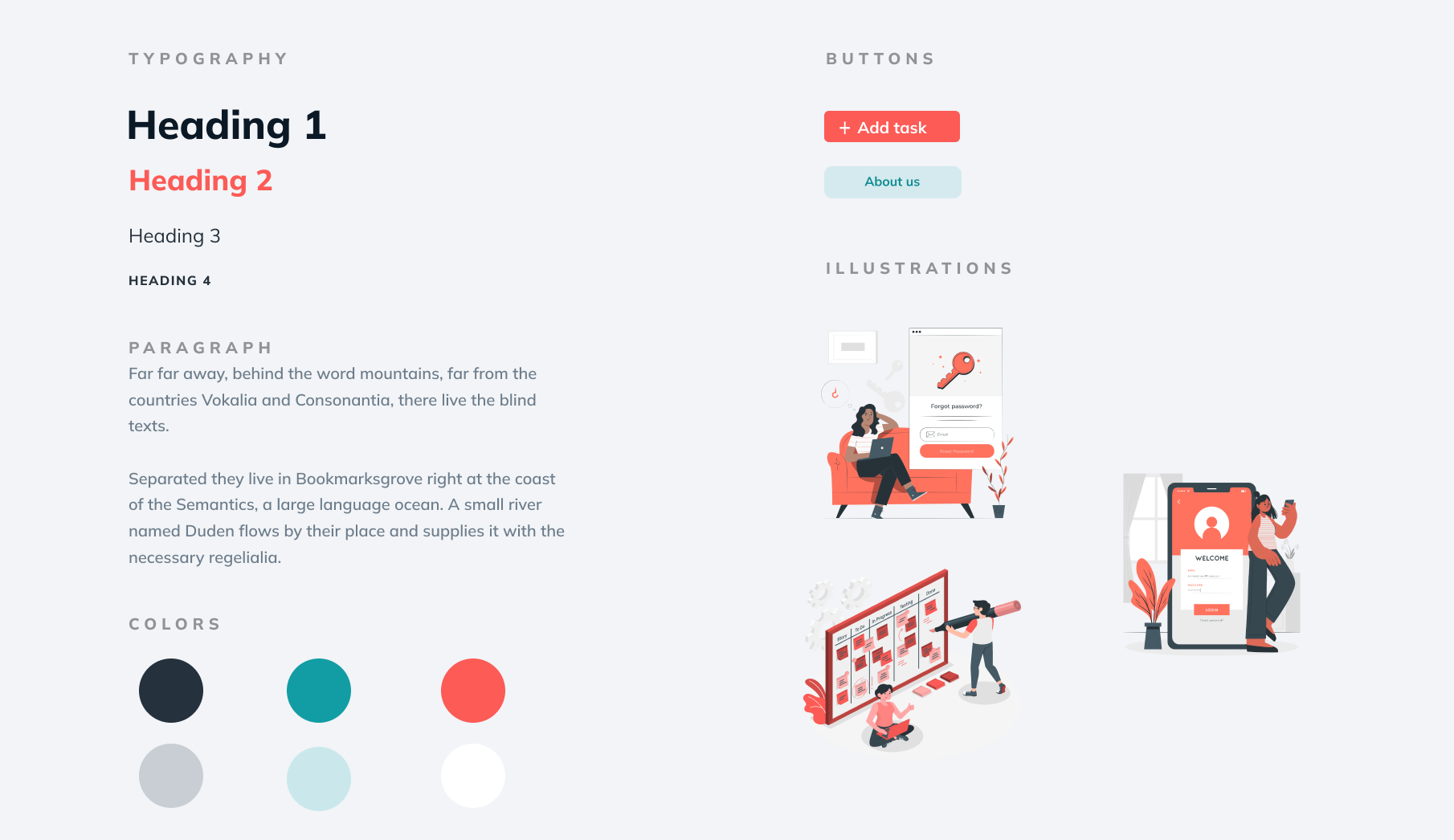
### Charte graphique

Notre client nous a donné une carte blanche pour le design et la charte graphique de son application. On a commencé d’abord par le choix de notre palette de couleurs.

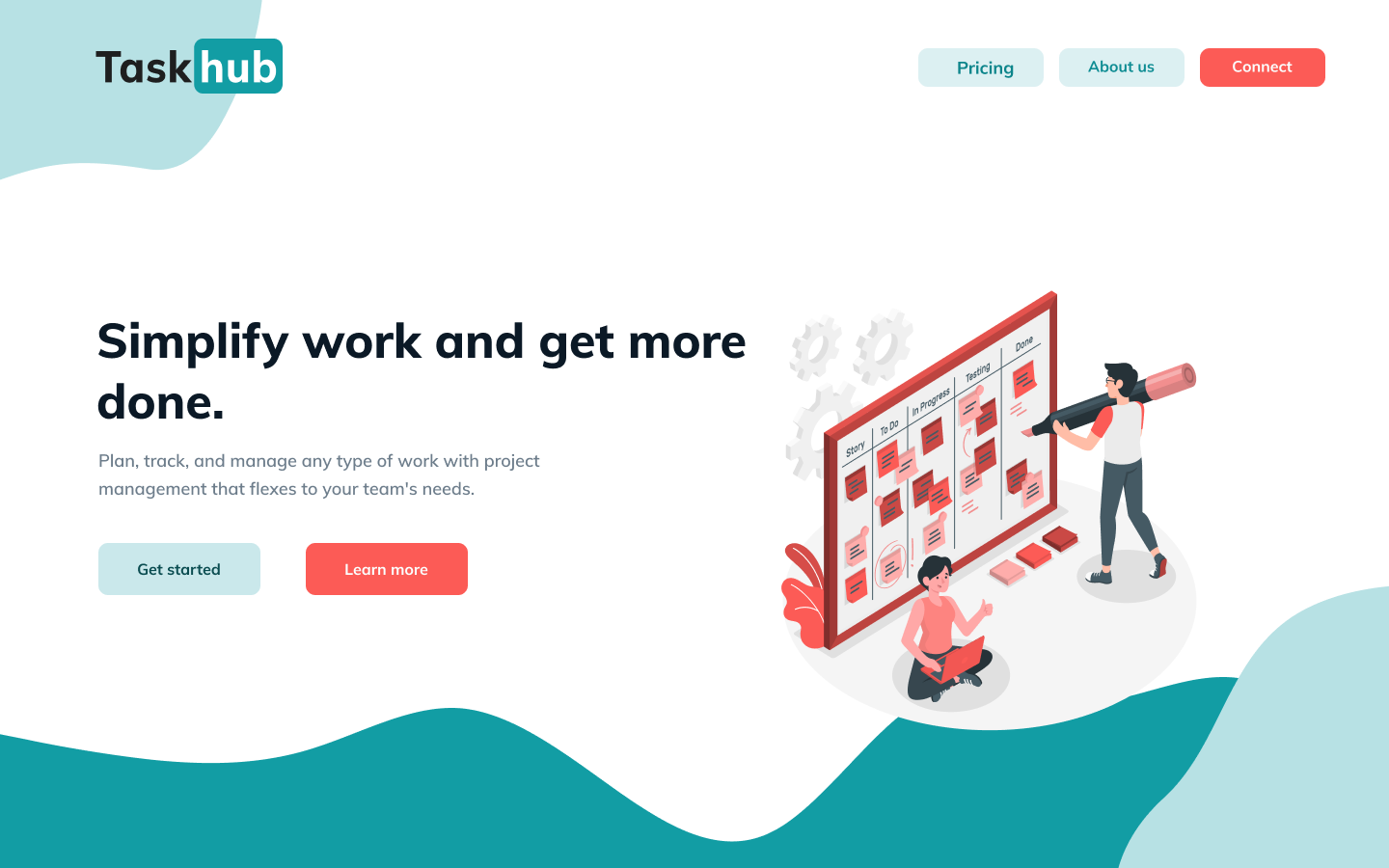
L’idée c’est de proposer des couleurs vives pour aider avec la concentration et un design simple et minimaliste pour faciliter l’utilisation de l’application.

On a choisi l’orange qui symbolise la créativité et l’optimisme avec le vert qui à son tour représente la stabilité et l’endurance, c’est des valeurs qui s’alignent parfaitement avec ceux de notre client.

On a utilisé Figma pour le design et la réalisation de maquettes, vous trouvez en bas la charte graphique présentée et validée par notre client.



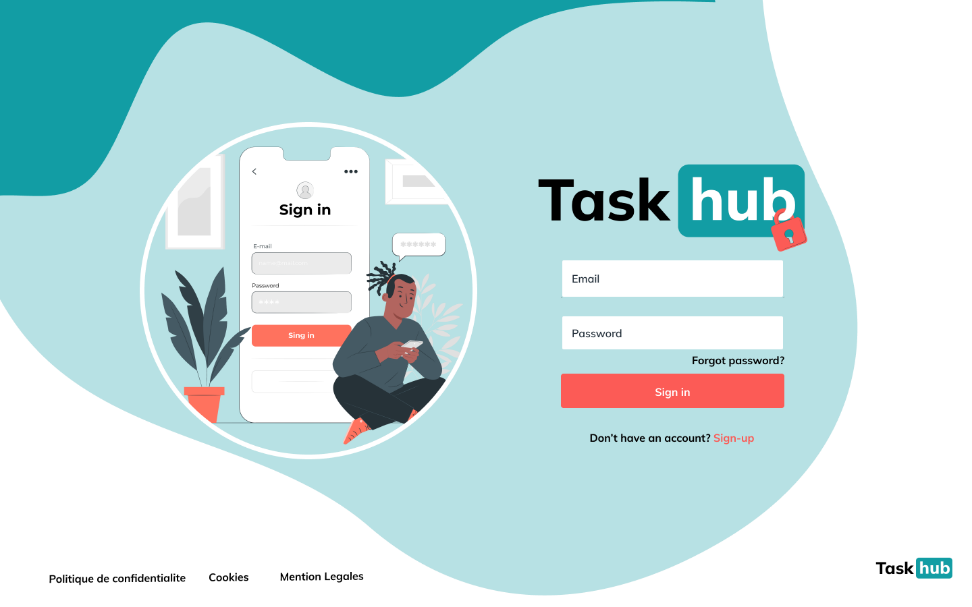
### Maquettes

L’utilisateur va en premier temps arriver sur la page d’accueil de l’application, pour après soit se connecter, soit créer un nouveau compte.

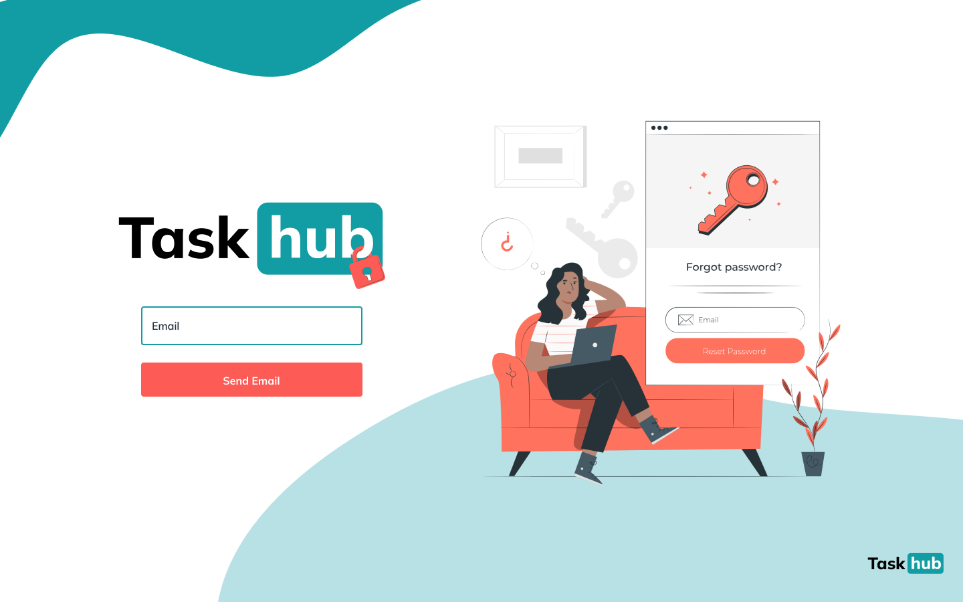
Voici également les écrans pour se connecter et créer un compte :



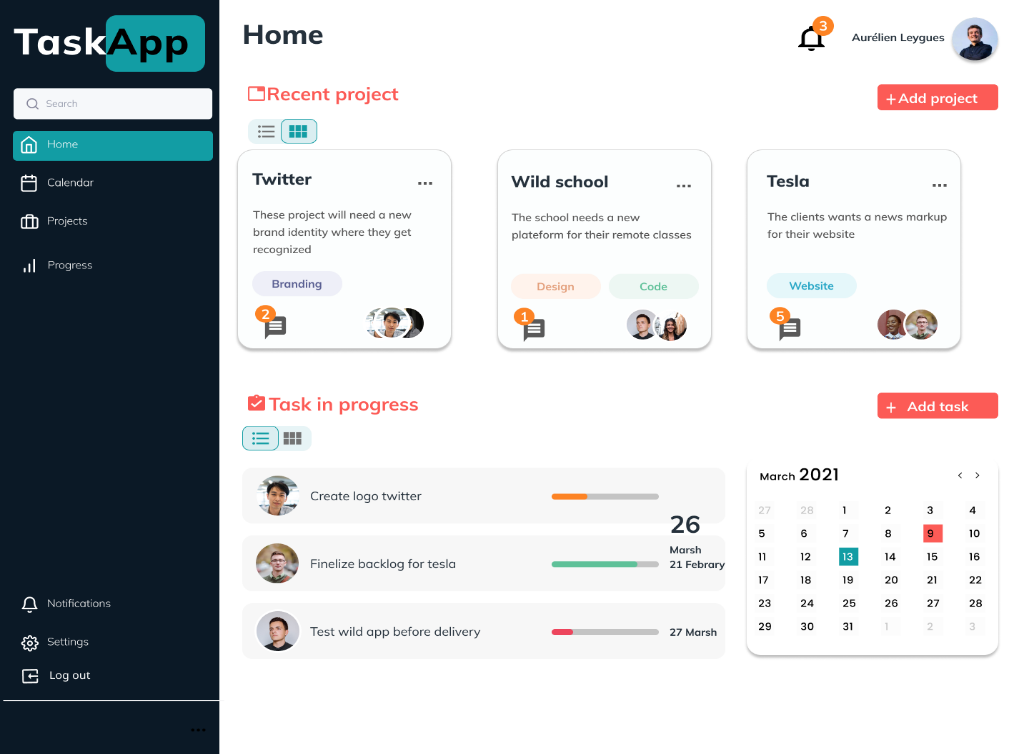
**Créer un compte**



**Se connecter**



**MDP oublié**

Voici également le Dashboard de l’utilisateur une fois connecté :

**Dashboard**

## Organisation du projet

Le projet est réalisé en binôme avec Clément Bacquet, également en formation à la Wild Code School.

Nous travaillerons suivant la méthode SCRUM avec des sprints d’une semaine et un backlog disponible sur GitHub Project et aussi OneNote.

Une **approche agile** consiste à donner de la visibilité en impliquant le client du début à la fin du projet et en adoptant un processus itératif et incrémental. Elle considère que le besoin ne peut être figé et propose au contraire de s’adapter aux changements de ce dernier. Mais pas sans un minimum de règles.

La méthode SCRUM telle qu’on a utilisé se décompose en plusieurs tâches :

* Nous avons défini les besoins avec le client.
* Nous avons écrit les **users stories**, ce sont des sous-ensembles de tâches qui forment chaque besoin du client et que l’on peut développer lors d’un sprint.
* Chaque sprint correspond à une ou deux semaines de développement.
* Chaque matin notre binôme de développeur se réunit pour un Daily (temps quotidien) de 15 minutes pour faire part des avancées, des difficultés rencontrées et du travail que l’on fera durant la journée.
* A la fin de chaque sprint nous faisons une rétrospective et nous informons le client des avancées pour savoir si cela lui convient et s’il désire des modifications.

Nous avons choisis de travailler en pair programming et de développer ensemble les fonctionnalités de notre application.

## Architecture du projet

La stack technique utilisé pour **« TaskApp »** a été choisi par l’équipe en fonction des besoins.

Notre application est réalisée avec le Framework ReactJs en Frontend, avec une API graphQL en backend sur un serveur NodeJs qui va interagir avec une base de donnée MySQL.

### Partie client (frontend)

* **ReactJs :**

On a utilisé le Framework ReactJs, Le but principal de cette bibliothèque est de faciliter la création d'application web monopage (SPA), via la création de composants dépendant d'un état et générant une page (ou portion) JSX à chaque changement d'état.

La bibliothèque se démarque de ses concurrents par sa flexibilité et ses performances, en travaillant avec un DOM virtuel et en ne mettant à jour le rendu dans le navigateur qu'en cas de nécessité.

* **TailwindCss :**

TailwindCss est un **Framework CSS complètement personnalisable. Contrairement aux autres Framework, Tailwind nous donne la possibilité de construire nos propres composants et donc avoir un** meilleur contrôle sur l'apparence des éléments.

* **Apollo:**

Apollo est une bibliothèque de gestion d'état complète pour les applications JavaScript. C'est un outil puissant car il peut être utilisé à la fois côté backend et frontend. Dans notre projet, nous allons l'utiliser sur les deux en créant d'abord un serveur Apollo GraphQL avec Node JS, puis en consommant les données côté client à l'aide de ReactJS.

### Partie serveur (backend)

* **NodeJs:**

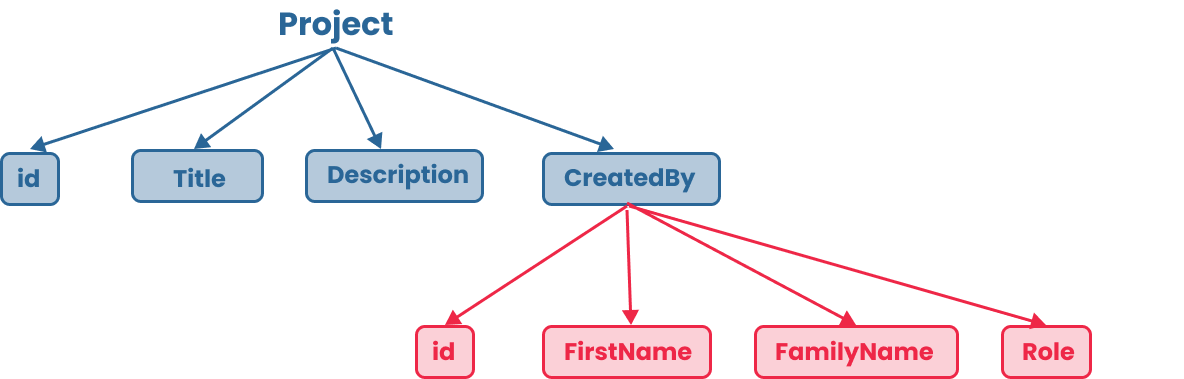
Est un environnement d’exécution comprenant tout ce qui est nécessaire pour exécuter un programme écrit en JavaScript.

* **GraphQl:**

GraphQL est, comme son nom l’indique (QL = Query Language), un langage de requêtes de données pour API, fortement typé, disposant d’un environnement d’exécution permettant de traiter ces requêtes, le tout basé sur un modèle client-serveur.

Avec GraphQL, le client effectuera un seul appel au serveur qui, lui, se chargera d’aller chercher toutes les ressources que la requête demande. Ce fonctionnement permet de demander exactement ce que l’on veut, sans récupérer trop de données (**over-fetching**) ou pas assez (**under-fetching**). Cela se traduit également par des appels plus légers et rapides entre le client et le serveur.

La réponse en sortie est au format JSON et représente un graphe d’objet, où les objets sont liés entre eux. Ces liens sont représentés par le graphe suivant :



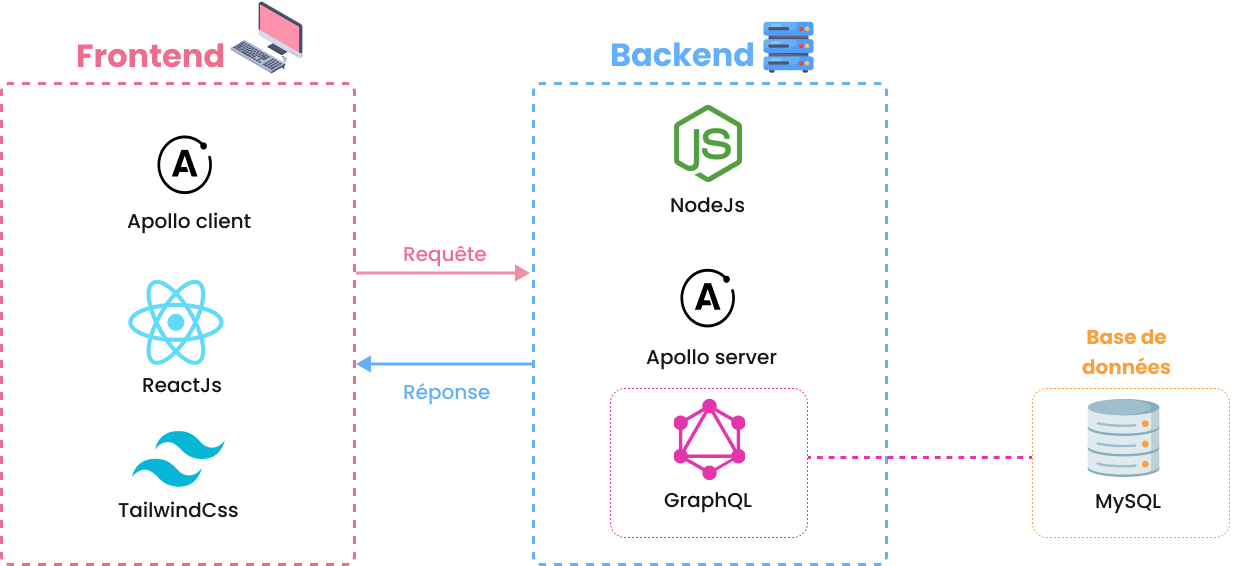
* **MySQL:**

Une base de données est une collection organisée d’informations structurées. Le

but est de pouvoir consulter, gérer ou mettre à jour ces données. Nous avons décidé pour ce projet d’utiliser la base de données apprise lors de la formation : MySQL. Il s’agit d’un système de gestion de base de données relationnelle (RDBMS) open source qui utilise le langage de requête structuré (SQL). Les éléments d'une base de données relationnelle sont organisés en un ensemble de tables comportant des lignes et des colonnes.

### Schéma de l’application

Ci-dessous un schéma récapitulant le fonctionnement de notre application:



# Principales fonctionnalités

Le but de « TaskApp » c’est de simplifier la gestion de projet et d’équipe pour notre client. A partir des user stories, nous avons déterminé les principales fonctionnalités attendues en nous appuyant sur la liste des Must Have.

## Création de compte et authentification

* L’utilisateur va pouvoir créer un nouveau compte en remplissant le formulaire d’inscription. Un mail de validation est envoyé pour confirmer son compte.
* Une fois le compte est confirmé, l’utilisateur est redirigé vers la page de connexion. Il saisit son identifiant et son mot de passe pour accéder à son espace personnel.
* L’utilisateur va pouvoir réinitialiser son mot de passe, un mail est envoyé pour

le rediriger vers le formulaire et changer son mot de passe.

## Gestion de projet et ticket

* Une fois arrivé sur son espace, l’utilisateur va pouvoir créer un nouveau projet, le modifier ou bien le supprimé.
* Pour chaque projet on peut :
* Mettre à jour son statuts
* Assigner des collaborateur
* Télécharger des documents et les afficher
* Pour chaque projet, on a la possibilité de créer des tâches, mettre à jour leur statut et les assigner à un collaborateur.

# Spécification techniques

## Structure de la base de données

Avant de pouvoir créer la base de données, il nous a fallu réfléchir à comment

architecturer ces données.

Avec le langage SQL, nous gérons des bases de données dites relationnelles. Les données sont stockées dans des tables, misent en relation les unes avec les autres.

Pour la modélisation nous avons utilisé la méthode Merise qui consiste en 4 étapes:

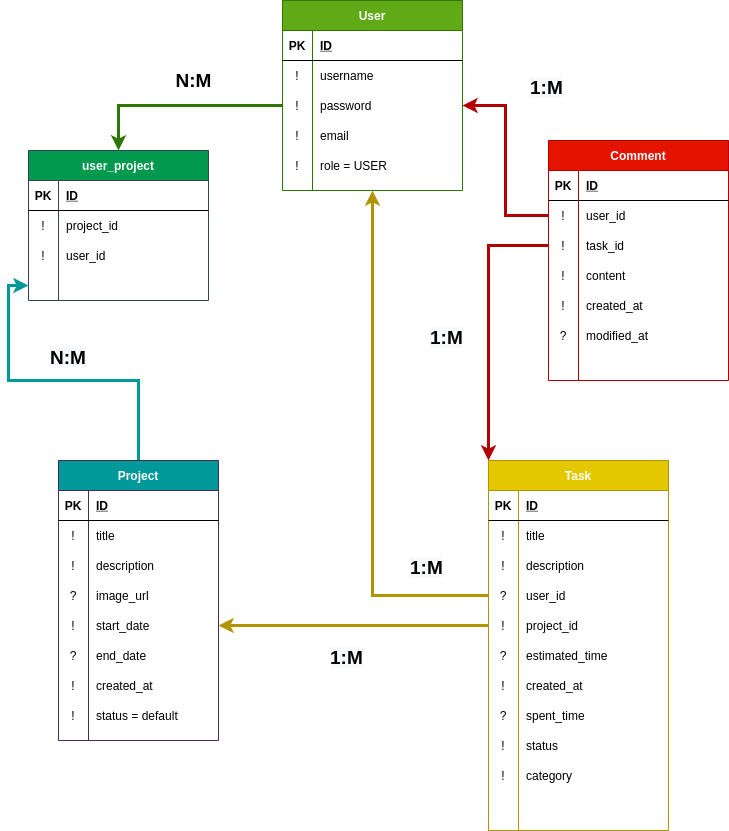
* Analyse du problème
* Modèle conceptuel des données (MCD)
* Modèle logique des données (MLD)
* Modèle physique

Prenons l’exemple de notre application :

* Un utilisateur peut créer plusieurs projets et un projet pourra avoir plusieurs tâches
* Chaque projet peut avoir plusieurs collaborateurs
* Une tâche a 1 seul projet et peut être assigner à un seul collaborateur

On définit les cardinalités et la relation entre nos tables. Un à plusieurs projets pour une à plusieurs tâches, il y aura des cardinalités de : Entité A (1 : n) Entité B (1 : n) et donc la relation sera many to many (N - M).

Voici donc le MPD qui correspond à l’architecture finale de la base de données :



## Côté client

« Create React App » va vous permettre de **générer un squelette de code** pour notre application. Il embarque un certain nombre d’**outils préconfigurés**, tels que Webpack, Babel et ESLint, afin de nous garantir la meilleure expérience de développement possible.

Pour manipuler « Create React App », nous allons avoir besoin d’un gestionnaire de paquet (package manager) directement dans le terminal. Ici, je vais utiliser **npm.**

Pour commencer, placez-vous dans le dossier où vous voulez créer votre projet, et lancer la commande suivante :

Une image contenant texte

Description générée automatiquement

Yaaaaay ! On a notre premier projet créé avec Create React App !

Une image contenant texte, moniteur, capture d’écran, noir

Description générée automatiquement

Dans le dossier **public**, on trouve le fichier **index.html** qui est notre point départ avec la div qui a pour class root qui permet de signifier a React ou est ce qu’il doit venir s’ancrer.

Le fichier **package.json** est certainement le fichier qui contient plusieurs informations sur notre application React notamment les scripts, les dépendances.

Le dossier **src** est l’endroit ou toute la magie de notre application va opérer, on y retrouve le fichier **index.js** qui est le fichier qui charge notre premier composant le composant App qui est ensuite greffer a la div avec la class root qui se trouve dans le fichier index.html au niveau du dossier public.

## Côté serveur

Pour créer notre API GraphQL dans un serveur express, on avoir besoin de :

* graphql : la bibliothèque JavaScript pour GraphQL
* express : un Framework web pour Node qui nous permet de créer des API et un serveur backend
* express-graphql : pour créer un serveur GraphQL pour les API
* ts-node : pour exécuter du code TypeScript dans Node
* typescript : pour compiler le code TypeScript en JavaScript
* @types/express : pour utiliser Express dans TypeScript
* nodemon : pour redémarrer le serveur lorsque des modifications sont apportées

### Mise en place de la BDD

Une image contenant texte, écran, capture d’écran

Description générée automatiquementOn commence par initier la base de données

Puis on définit notre modèle de donné , on prend l’exemple d’un projet:

Une image contenant texte

Description générée automatiquement

Pour fournir l’accès à nos données, on a utilisé TypeORM, ce dernier est un ORM écrit en TypeScript utilisant les dernières fonctionnalités du langage, on peut ainsi y définir nos entités via les décorateurs. et le typage.

Maintenant que nous avons un schéma GraphQL, il est temps de construire les résolveurs correspondants pour terminer l'opération de requête.

Un résolveur est une collection de fonctions qui permet de générer une réponse à partir d'une requête GraphQL.

Une image contenant texte, portable, écran, capture d’écran

Description générée automatiquement

En dernier il nous reste la création de notre serveur, pour cela on utilise **ApolloServer** du paquet apollo-server, c'est un constructeur qui reçoit un objet en argument. Et cet objet doit contenir le schéma et le résolveur pour pouvoir créer le serveur.

Une image contenant texte

Description générée automatiquement

Ici, nous passons en paramètre un objet qui contient le schéma et le résolveur à ApolloServer pour créer le serveur. Avec cela en place, nous avons maintenant un serveur fonctionnel avec lequel travailler.

Une image contenant texte

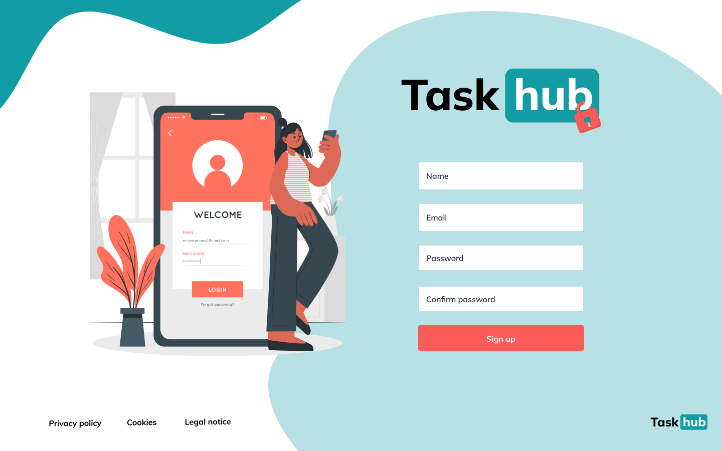
Description générée automatiquementOn peut envoyer des requêtes à l'aide de GraphQL Playground.

Ici, on a une simple requête GraphQL pour récupérer les données d’un seule projet. On voit rapidement la puissance de GraphQl, une seule requête nous permet de demander exactement ce que l’on veut, sans récupérer trop de données.

# Réalisations

## Gestion de l’authentification

### S’enregistrer



Pour une première utilisation de notre application, l’utilisateur va devoir créer un nouveau compte avec un email valide.

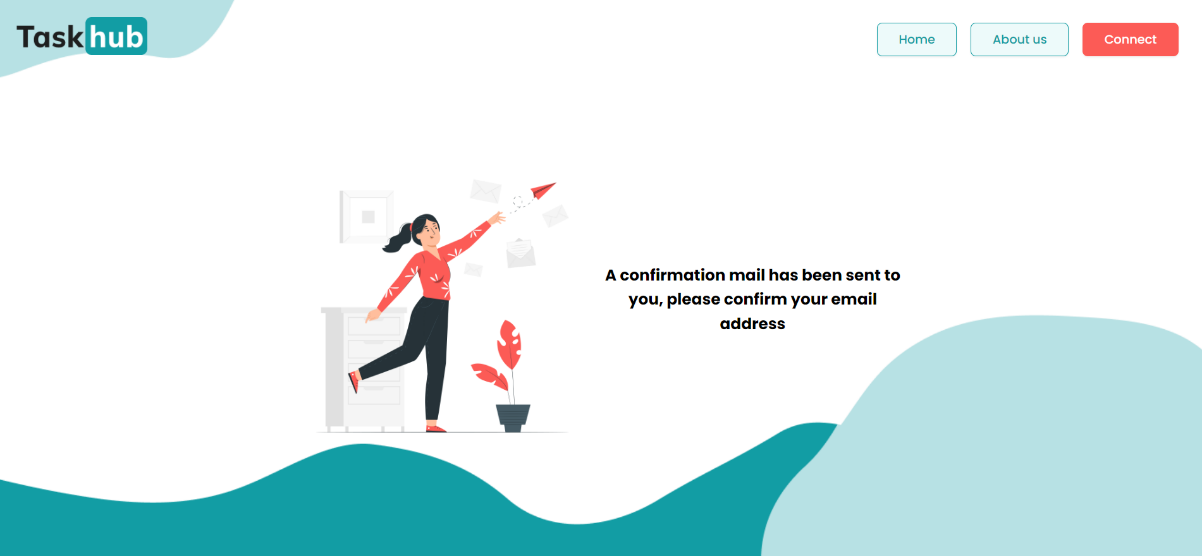
Coté serveur, on a le resolver signup qui est une mutation dans GraphQl.

Une image contenant texte

Description générée automatiquement

Pour bien sécuriser le mot de passe de notre utilisateur, on utiliser le package argon2 pour hacher le mdp avant de le stocker dans notre base de donnée.

A la suite on renvoi l’utilisateur vers une page de redirection.



Une image contenant texte

Description générée automatiquementA l’intérieur de notre signup resolver et avant de sauvegarder notre utilisateur, on envoie un mail de confirmation avec Nodemailer. Ce dernier est un module nodejs qui permet d'envoyer des emails. Dans ce contexte, nodemailer sera configuré pour envoyer des mails via le serveur Gmail.

Un lien est envoyé avec un **token** qui a été générer avec le package **jsonwebtoken**

Une image contenant texte

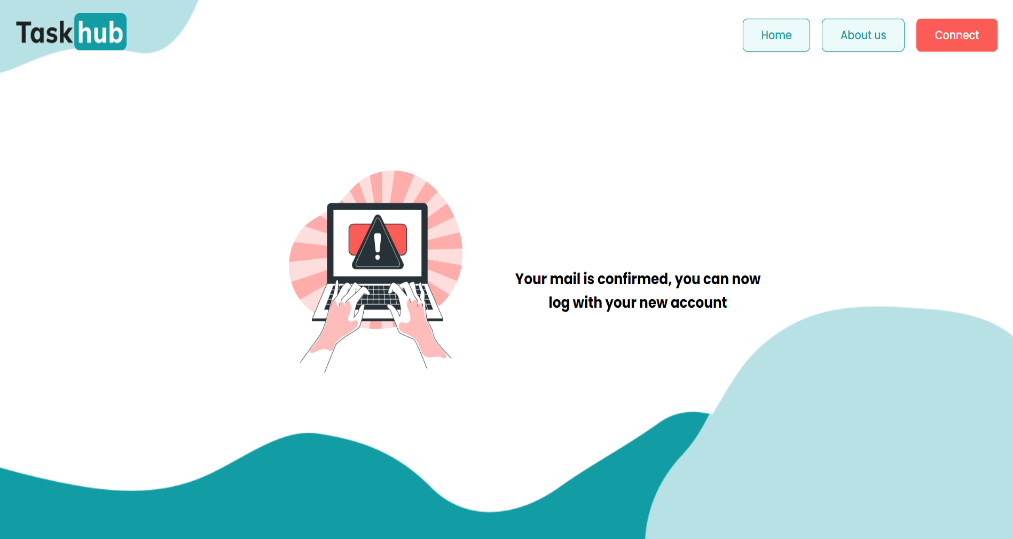
Description générée automatiquement

Une fois arriver sur ce lien coté client, la requête confirmUser est envoyée afin de passer son statuts confirmed à **true.**

Une image contenant texte

Description générée automatiquement

Coté client l’utilisateur se retrouve sur la page de redirection



Compte créer, mail confirmer maintenant notre utilisateur va pouvoir s’identifier et accéder à son Dashboard.

## Json Web Token

Maintenant on va s’intéresser un peu au fonctionnement de cette validation et surtout à l’utilisation d’un token comme le token JWT pour l’authentification.

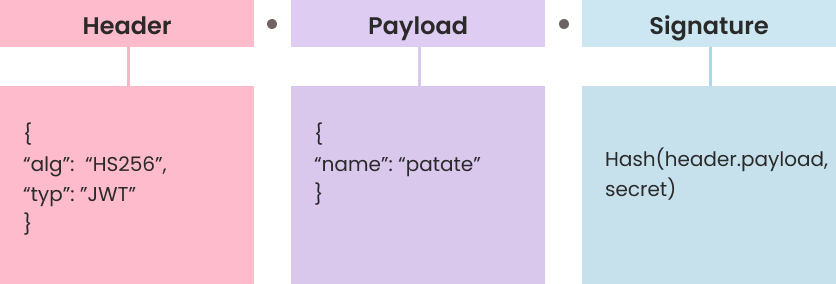
On entend toujours les deux mots authentification et autorisation quand on parle de sécurité, et c’est très important de faire la différence entre les deux.

L’**authentification**, c’est savoir qui appelle notre API. À partir d’un ensemble d’informations, il est possible de savoir de qui vient l’appel HTTP.

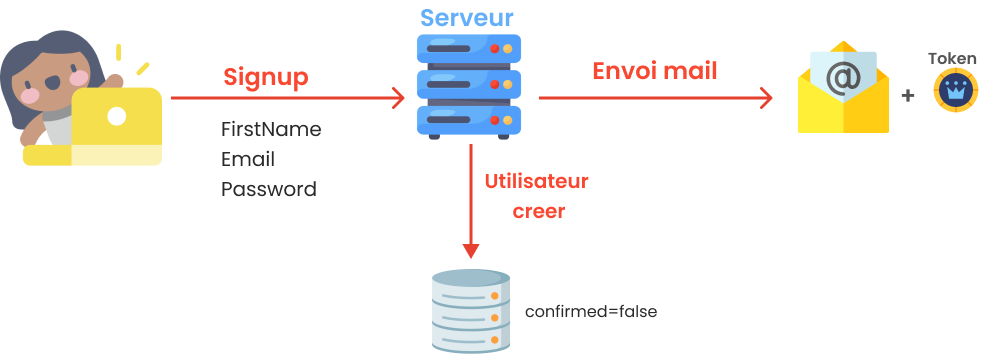
Une fois que la personne qui appelle votre API a été identifiée, l’**autorisation** consiste à savoir ce que cette personne a le droit de faire. Dans la mesure où chaque appel HTTP est traité indépendamment ,à chaque fois que vous recevrez un appel, il vous faudra décider si oui ou non l’appelant a le droit de consommer cette ressource en particulier.

J’ai regardé la documentation sur le site JWT.IO, afin de voir c’est quoi le Json Web Token, il s'agit d'une chaîne composée de 3 parties. Chaque partie séparée par un point.

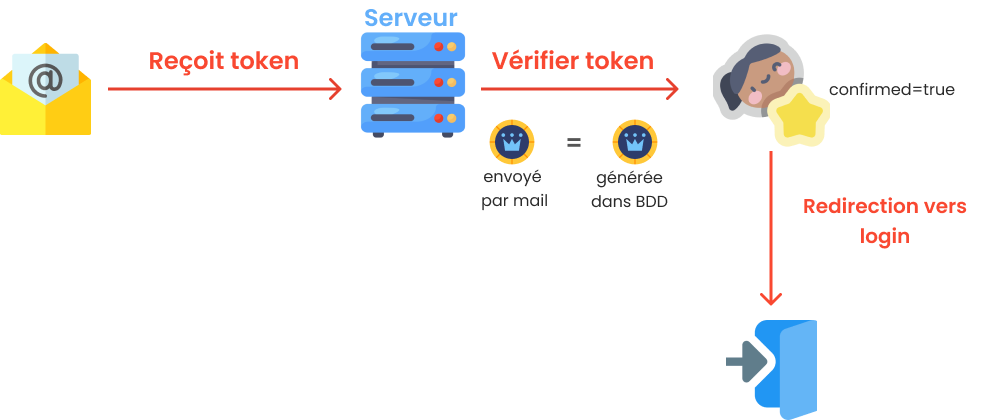
* La 1ère partie est l'en-tête : Header qui contient l’algorithme pour la partie signature.
* La 2ème partie est la charge utile : Payload, qui correspond au contenu du token.
* Et la 3ème partie est la signature, On utilise l’algorithme défini dans le header, on concatène la première clé avec la deuxième et on utilise une clé secrète et c’est avec cette clé secrète qu’on pourra générer la dernière chaîne de caractère.



Le schéma suivant résume notre processus d’inscription avec le token :



A la réception du mail de confirmation, on va effectuer la vérification du token pour lancer la confirmation du compte.



## Upload de fichiers

# Tests et déploiement

## Test

## Déploiement

# Conclusion