# Architecture de l'application

## AnimeExplorer

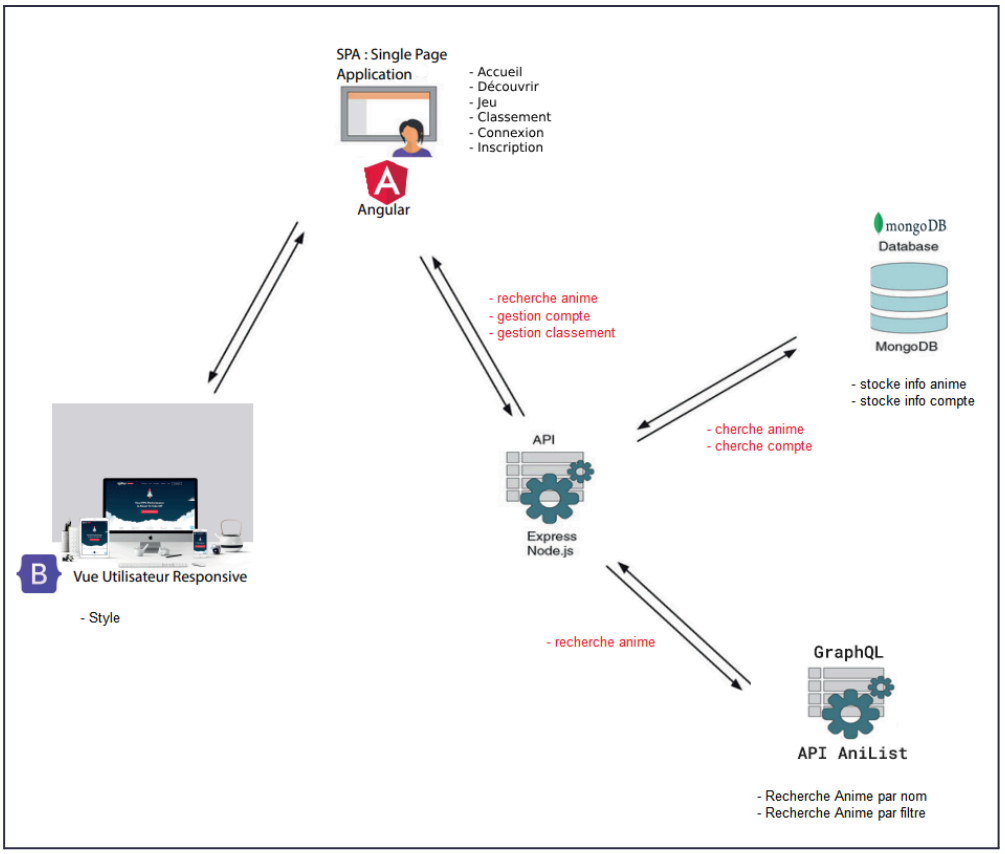
Yamis MANFALOTI
Cody THEARD
Nathan RACCOUARD

## I. Fonctionnalités et limites

- Afin de donner du contexte au choix fait dans l'architecture, il me paraît important d'expliquer les fonctionnalités que nous devions implémenter ainsi que les limites auxquelles nous avons fait face.
- Dans notre application, nous retrouvons 3 grandes fonctionnalités principales. La première étant celle qui permet de découvrir des animés, la seconde est celle qui correspond au jeu interactif et enfin, la dernière est celle de la gestion des membres dont découle aussi le classement.
- La partie découverte nécessite principalement de récupérer des données, les triées selon les demandes de l'utilisateur puis les afficher dans un format simple. La partie de jeu interactif reprend une sous-fonctionnalité de la partie découverte qui est la recherche par nom d'animé. Mais cette partie inclut aussi une logique de jeu afin de pouvoir divertir le joueur tout en lui apprenant des informations sur les animés. Enfin, la partie gestion de membres correspond à l'inscription et la connexion des utilisateurs sur l'application ainsi que la gestion des points et du classement des joueurs qui est lié aux performances sur le jeu interactif.
- Sachant que notre application est principalement basée sur l'interaction avec les données sur les animés, nous avons besoin d'un accès constant et répété à ces données. Cependant, l'API proposée par AniList nous limite à 90 requêtes par minute. Nous avons donc dû faire certains choix d'architecture pour prendre cela en compte.

# II. Architecture globale

- Schéma global :




- Comme illustré par le schéma ci-dessus, l'architecture de notre application peut être divisée en 5 grands composants.
- On retrouve tout d'abord l'utilisation de Bootstrap qui permet de gérer la partie affichage et mise en page de l'application avec ce framework qui simplifie les modifications au niveau du CSS et permet surtout une bonne réactivité en fonction de l'appareil utilisé.
- Dans le cas de ce document, nous classifierons le framework Angular dans la partie Front car nous l'avons principalement utilisé comme tel, bien que nous avons réalisé quelques fonctions de logique pour l'animation du site ou dans la logique du jeu interactif.
- La première partie du Back est le serveur Node.js qui tourne afin de récupérer les demandes d'informations demandées par le Front, la logique du jeu ou d'interagir avec la base de données.
- La seconde partie est celle de la base de données mongoDB qui permet de stocker les informations des animés et celle des membres.
- La dernière correspond à l'API d'AniList qui permet d'obtenir les différentes informations qui sont demandées par l'application.

# III. Partie Front - Angular

- Vu qu'il existe un document sur la description du design Front bootstrap, nous parlerons ici seulement de la partie Angular.
- Dans cette partie, nous détaillerons donc les 8 pages qui sont reliées entre elles par un système de routage, les 8 composants utilisés dans l'application ainsi que les 4 services qui font tout tourner.
- Parmis ces pages on retrouve :
    - Page d'accueil
    - Page découvrir
    - Page jeu
    - Page classement
    - Page connexion
    - Page inscription
    - Page connecté
    - Page not-found
- Les composants sont utilisés dans différentes pages :
    - Pour la page découvrir :
        - filtre
        - info-animes
        - search-bar
        - vignette
    - Pour la page jeu :
        - jeu-victoire
        - search-zone
    - La page classement dispose de son composant ranking.
    - Enfin, le composant nav-bar se retrouve sur chaque page du site.
- Pour les services, on dispose du service :
    - auth qui gère la partie authentification des membres
    - back-end qui communique via des requêtes au serveur Node.js
    - search qui s'occupe de toute la partie des requêtes au back-end pour recherche un animé
    - ranking qui gère les requêtes au back pour la gestion du classement

# IV. Partie Back - Serveur Node.js

- Concernant tout ce qu'il ce passe en Back-End sur le serveur Node.js, on se retrouve avec 6 gestions de requête en Post et 2 en Get.
- Pour les requêtes en Post :
    - "/searchAnimeByName" correspond à la recherche d'animé par nom avec un système de recherche flou ( Fuzzy Matching )

    - “/searchAnimeByFilter” pour la recherche d’animé par filtre
    - “/ranking” pour la demande du classement ainsi que le classement personne si l’utilisateur est connecté
    - “/emailDisponible” pour vérifier l’email
    - “ /signup” pour l’inscription
    - “/login” pour la connexion
- Pour les requêtes en Get :
    - “/getTopAnimesList” pour obtenir la liste des animés populaires affichés par défaut à l'ouverture de la page découvrir
    - “/askAnimeATrouver” pour l’animé à deviner dans le jeu interactif

## V. Partie Back - MongoDB & Api Anilist

- Comme nous l’avons cité dans les limites, l’API d'anilist nous limite à 90 requêtes par minute. Pour répondre à cela, on décide de stocker beaucoup d’informations d’animés sur notre base de données et de faire des requêtes sur la base de données au lieu de directement à l'API.
- Pour cela, nous avons rempli la base de données avec les informations de :
    - 1000 animés les plus populaires.
    - 10 animés par année disponible en choix de filtre ( de 2024 à 1971 )
    - 10 animés par statut de diffusion en choix de filtre ( 4 * 10 animés )
- Dans l’état actuel de notre application, l’API d'AniList a juste servi à remplir la base de données au départ. Cependant, on avait prévu comme fonctionnalité de pouvoir laisser le choix à l’utilisateur d'obtenir plus de résultats s’il n’était pas satisfait de sa recherche. Dans ce cas, on aurait fait une requête à l’API d’AniList afin d’utiliser leur système de recherche qui dispose d’une plus grande base de données que la nôtre et des meilleures performances.