Cassandra FORESTIER

RAPPORT

Projet : Cosplay-Maker

Titre professionnel : Concepteur Développeur d'Applications

Titre RNCP 31678

Année 2022-2023

Centre de formation : Le Bocal Academy à Nice

Référentiel de compétences

Activités types (Blocs de compétences)	Compétences professionnelles	Couvertes dans le projet
Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité	Maquetter une application	OUI
	Développer une interface utilisateur de type desktop	NON
	Développer des composants d'accès aux données	OUI
	Développer la partie front-end d'une interface utilisateur web	OUI
	Développer la partie back-end d'une interface utilisateur web	OUI
Concevoir et développer la persistance des données en intégrant les recommandations de sécurité	Concevoir une base de données	OUI
	Mettre en place une base de données	OUI
	Développer des composants dans le langage d'une base de données	OUI
Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité	Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement	OUI
	Concevoir une application	OUI
	Développer des composants métier	OUI
	Construire une application organisée en couches	OUI
	Développer une application mobile	OUI
	Préparer et exécuter les plans de tests d'une application	OUI
	Préparer et exécuter le déploiement d'une application	OUI

Partie I : Introduction

1. Resume

The aim of this project is to develop a web app for cosplayers to connect and share their creations. This app will let users show off their cosplays, save other people's cosplays, and get ideas for their own costumes. It will also help them stick to their budget and schedule. Users can add links and budget details for each cosplay item, which will make it easier for others to find what they need to start their own cosplays.

The app will make it quicker for beginners to learn how to create cosplays and provide support for experienced cosplayers too.

Cosplayers can use the app to organize events like photo shoots, cosplay meetups, or themed parties. They can also find and sign up for conventions. It will be a social platform where users can see who's attending these events.

2. Le cosplay en quelques mots

Le cosplay, contraction des mots anglais *'costume'* et *'play'*, consiste à porter / recréer un costume d'un personnage de fiction issu d'un manga, d'un dessin animé, d'un film, d'un jeu vidéo ou de tout autre support. Le cosplay est en constante croissance et rassemble une communauté passionnée à travers le monde.

Pratiquant personnellement le cosplay, je me rends compte que lorsque je commence un cosplay, je passe énormément de temps à chercher les tissus, les matériaux, les références en image, les perruques les plus fidèles …

3. Objectifs et enjeux

Ainsi, **l'objectif de ce projet** est de développer une application web visant à créer une communauté en ligne pour les passionnés de cosplay. Cette application permettra aux utilisateurs de partager leurs créations, de sauvegarder des cosplays d'autres utilisateurs, de trouver des idées d'inspiration pour leurs prochains costumes tout en respectant des budgets prédéfinis et un calendrier qui leur convient. Les liens de chaque élément du cosplay peuvent être ajoutés, ainsi que son budget : cela permet aux visiteurs de leur faciliter le travail de recherche pour débuter un cosplay.

Le processus de création d'un cosplay sera donc plus rapide à intégrer pour les débutants, mais facilitera également le travail des plus expérimentés en leur apportant un support adapté.

L'application permettra de réunir des cosplayeurs sur des évènements (création d'évènements tels que shooting, lieu de rendez-vous cosplay, bar à thème...) mais aussi de référencer les conventions afin que les utilisateurs puissent s'y inscrire. En outre, cela permettra d'apporter un côté social puisqu'il sera possible de voir les personnes inscrites à ces évènements.

4. Périmètre et limites du projet

Pour garantir la faisabilité du projet "Cosplay-Maker" et assurer une gestion efficace, il est nécessaire de préciser les aspects suivants :

- **Fonctionnalités de l'application :** Définir clairement les fonctionnalités principales de l'application, telles que la création de profils, le partage de photos et de tutoriels, la participation à des concours, etc. Il est important de limiter le nombre de fonctionnalités pour éviter un périmètre trop large qui pourrait compromettre la faisabilité du projet.

Voici donc la liste des fonctionnalités :

Création de compte : Les utilisateurs pourront créer un compte personnel pour accéder à toutes les fonctionnalités de l'application.

Gestion des cosplays : Les utilisateurs pourront créer, modifier et supprimer leurs cosplays. Ils pourront également rechercher des cosplays par nom de personnage ou par univers, mettre des cosplays en favoris et les supprimer de la liste. Ils pourront également mettre un cosplay en mode public ou privée.

Gestion des événements : Les organisateurs d'événements pourront créer des événements officiels, les modifier et les supprimer. Les cosplayeurs pourront indiquer leur participation à un événements (officiels ou non) en précisant le cosplay qu'ils porteront lors de l'événement. Les organisateurs pourront accéder à la liste des cosplayeurs participants à leur événement.

Fonctionnalités d'administration : Les administrateurs auront la possibilité de mettre à jour les tendances cosplay, de supprimer les comptes utilisateur, les cosplays et les événements qui ne respectent pas les standards de la communauté.

Autres fonctionnalités : L'application offrira également des fonctionnalités telles que la réinitialisation de mot de passe, la recherche de cosplayeurs par région, l'invitation des cosplayeurs à des événements, etc.

- **Plateforme cible :** Préciser les plateformes sur lesquelles l'application sera développée, par exemple, une application web optimisée pour les ordinateurs de bureau et les appareils mobiles.

Concernant la plateforme cible, j'ai fait le choix de développer une **application web** qui a été motivé par plusieurs raisons :

Accessibilité multiplateforme : Grâce à la mise en place de l'application Web, je pourrais plus facilement atteindre plusieurs types d'utilisateurs puisque l'application sera **responsive**, elle pourra être à la fois sur un PC fixe, un PC portable, une tablette ou encore sur un Smartphone.

Pas de téléchargement ni d'installation requis : il s'agit d'un avantage non négligeable. L'application est accessible partout et facilement. Sachant que pour utiliser l'application, il faut se créer un compte, l'expérience utilisateur est déjà impactée, c'est donc également un choix stratégique sur ce sujet. Dès que l'application Web fonctionne, je pourrais par contre, partir sur une application mobile.

Mise à jour et déploiement simplifiés : Avec une application web, les mises à jour et les déploiements se font du côté du serveur, ce qui signifie que les utilisateurs n'ont pas besoin de télécharger et d'installer manuellement les mises à jour. Les nouvelles fonctionnalités et les correctifs peuvent être déployés rapidement et efficacement pour tous les utilisateurs. Ainsi, je peux sortir une première version stable relativement rapidement, et faire des mises en production plus facilement et plus fréquemment.

Besoins utilisateurs précis : étant donné qu'il s'agit de cosplay, les utilisateurs vont pouvoir se baser sur leurs images de leur cosplay ou voir les références images d'autres cosplayeurs pour entamer la création d'un cosplay par exemple. Il est évidemment plus simple de voir les détails d'une image sur un écran de PC que sur un smartphone. C'est donc également un choix pertinent sur ce point précis du projet.

Connaissances techniques : Mes connaissances techniques se limitaient en début d'année au développement d'une application Web responsive, il était donc évident pour moi de me challenger sur ce sujet également et d'approfondir mes connaissances. J'ajouterai à terme une application mobile en React Native. Je pourrais utiliser mon API. Pour le côté front-end, passer du React au React Native sera plus facile.

- **Échelle de la communauté :** Déterminer la taille et la croissance attendue de la communauté. Cela permettra de définir les ressources nécessaires pour gérer et soutenir celle-ci, comme la modération des contenus et l'assistance aux utilisateurs.

La communauté cosplay ne cesse de grandir depuis les 10 dernières années. En 2020, l'industrie mondiale du cosplay était évaluée à 4,62 milliards de dollars. C'est selon Allied Market Research. Ils prévoient également qu'il atteindra 23 milliards de dollars d'ici 2030. Selon la recherche du mot "cosplay" sur Google, Google Trends nous indique bien une évolution de la popularité du cosplay. Il s'agit donc d'un marché qui émerge doucement, où bon nombre d'applications peuvent voir le jour.

- **Technologies utilisées :** Identifier les technologies nécessaires pour développer l'application, comme les langages de programmation, les FRAMEWORKS[1], les bases de données, etc. Il est important de s'assurer que les ressources et les compétences requises soient disponibles pour mettre implémenter ces fonctionnalités. Je détaillerai ce point dans la partie III.

- **Budget et ressources :** Établir un budget prévisionnel pour le développement de l'application, en prenant en compte les coûts liés : au développement, à l'hébergement, à la maintenance, à la promotion de la plateforme, aux ressources humaines nécessaires et enfin à l'expertise technique.

Pour finir, la collecte et le stockage des données personnelles des utilisateurs devront être conformes au règlement général sur la protection des données (RGPD) et nécessiteront une déclaration CNIL.
De plus, la mise à jour régulière des références des conventions sera nécessaire pour assurer l'exactitude des informations fournies aux utilisateurs. Enfin, la création d'un utilisateur "gérant convention" nécessitera une gestion spécifique pour permettre aux organisateurs de créer et de gérer leurs propres conventions sur la plateforme.

5. Présentation de l'équipe

Dans le cadre du projet "Cosplay-Maker", l'équipe est composée d'une seule personne, à savoir moi-même Cassandra Forestier.

Je suis responsable de toutes les étapes du projet, de la conception à la mise en œuvre, en passant par le développement front-end, back-end, la gestion de la base de données, la conception graphique, la mise en place de la

[1] Un framework est un ensemble structuré de bibliothèques, d'outils et de conventions qui facilite le développement d'applications en fournissant une structure de base et des fonctionnalités prédéfinies.

CI/CD[2], les opérations de développement (DEVOPS[3]), la sécurité du site, la veille technologique, la mise à jour du KANBAN[4], ainsi que le déploiement de l'application.

Bien que je sois seule sur ce projet, je suis prête à relever le défi en investissant le temps et les efforts nécessaires pour mener à bien cette initiative. Mon but étant que ce projet soit réellement mis en ligne, je me fixe donc un an pour sortir la première version de ce site.

[2] **Le CI/CD (Continuous Integration/Continuous Deployment)** est une pratique de développement logiciel visant à automatiser la mise en ligne du développement. Il combine l'intégration continue (CI), qui intègre régulièrement les modifications du code source et exécute des tests, avec le déploiement continu (CD), qui automatise le déploiement des applications après les tests réussis. Cela permet d'améliorer l'efficacité, la qualité et la rapidité de livraison des logiciels.

[3] **DevOps :** DevOps est un ensemble de pratiques qui **met l'accent sur la collaboration et la communication** entre les développeurs de logiciels et les professionnels des opérations informatiques, en **automatisant** le processus de livraison de logiciels et les changements d'infrastructure.

[4] Kanban est une méthode de gestion visuelle des tâches et des flux de travail. Les tâches à accomplir sont représentées sous forme de cartes et sont organisées en colonnes correspondant aux différentes étapes du processus. L'objectif est de visualiser clairement l'état d'avancement des tâches, d'identifier les éventuels goulots d'étranglement et d'améliorer la productivité en favorisant un flux de travail continu et équilibré. C'est une approche efficace pour gérer les projets et les tâches, que ce soit individuellement ou en équipe.

Partie II : Analyse des besoins

A. Analyse d'une application existante “Cosplanner”

L'application "Cosplanner" (application mobile : Cosplanner) offre plusieurs fonctionnalités intéressantes pour les cosplayeurs.

Elle permet la création de cosplays en ajoutant des éléments, des tâches, des images de référence et des informations détaillées. Ainsi, Cosplanner permet la gestion du budget en divisant le projet en 2 catégories : éléments achetés et éléments à fabriquer, avec la possibilité de suivre les dépenses associées. La gestion du temps alloué/disponible est également présente avec des pourcentages de progression pour chaque élément du cosplay, offrant une vision globale sur l'avancement du projet.

Cependant, malgré ces fonctionnalités intéressantes, il convient de noter que visuellement, l'application manque de modernité. Son interface pourrait bénéficier d'une mise à jour pour offrir une expérience utilisateur plus agréable et intuitive.

Certaines fonctionnalités manquent à l'appel, ce qui pourrait limiter son utilité pour les cosplayeurs. Par exemple, des fonctionnalités de partage avec d'autres cosplayeurs où la gestion des évènements peuvent réellement contribuer à la fidélisation des utilisateurs sur la plateforme.

De même, j'ai identifié plusieurs fonctionnalités manquantes à la plateforme telles que les références, les liens et les budgets associés aux différentes parties du cosplay. Via ces liens, les cosplayeurs indiqueront les plateformes externes permettant de se fournir en matière première, perruques, matériaux divers et variés, et de ne pas les “perdre”.

La pérennisation et l'établissement d'un lien solide entre le cosplay et sa source d'inspiration réelle sur Internet seraient favorisés. Ce point revêt une importance capitale car il simplifie la recherche des adeptes du cosplay. De plus, la possibilité de partager les méthodes de réalisation de cosplay constitue un avantage considérable. Cela permet non seulement d'échanger des astuces pratiques, mais aussi de bénéficier d'un énorme travail de référencement. Les utilisateurs de l'application peuvent ainsi trouver les meilleurs tissus, perruques, lentilles et autres éléments en se basant sur les cosplays d'autres utilisateurs.

En résumé, bien que l'application "Cosplanner" propose des fonctionnalités intéressantes pour les cosplayeurs, son aspect visuel qui manque de modernité et son manque de convivialité (user-friendly) pourraient être améliorés. De plus, l'ajout de fonctionnalités telles que le partage avec d'autres cosplayeurs, l'attachement de cosplays à des événements spécifiques et la gestion des budgets associés à des parties spécifiques du cosplay pourraient rendre l'application encore plus utile pour les utilisateurs.

Je rajoute à cela que l’application est disponible à la fois en version Web et en version mobile. Sur la version mobile, on peut constater plus de 100 000 téléchargements. Je l’avais téléchargé pour mon utilisation personnelle, mais j’avais été vite découragée par l’interface rendant compliquée la création d’un cosplay, mais aussi par le visuel.

B. Description des utilisateurs

Les utilisateurs de cette application seront :

- Les cosplayeurs.
- Les organisateurs d'évènement.
- Les administrateurs.

C. Cas d'utilisation

➢ <u>Pour un utilisateur cosplayeur :</u>

Un cosplayeur pourra faire les actions suivantes :

- Créer un compte, modifier les informations de ce compte et le supprimer.
- Créer des cosplays, les modifier et les supprimer.
- Créer des évènements non-officiels, les modifier, les supprimer.
- Préciser de manière non obligatoire le(s) cosplay(s) que l'utilisateur aura pendant l'événement.
- Se connecter, récupérer un mot de passe oublié, se déconnecter.
- Rechercher des cosplays par nom de personnage, ou par univers.
- Mettre en favoris des cosplays, mais aussi les supprimer de la liste.

➢ <u>Pour un utilisateur organisateur d'évènements :</u>

Un organisateur d'évènements pourra faire les actions suivantes :

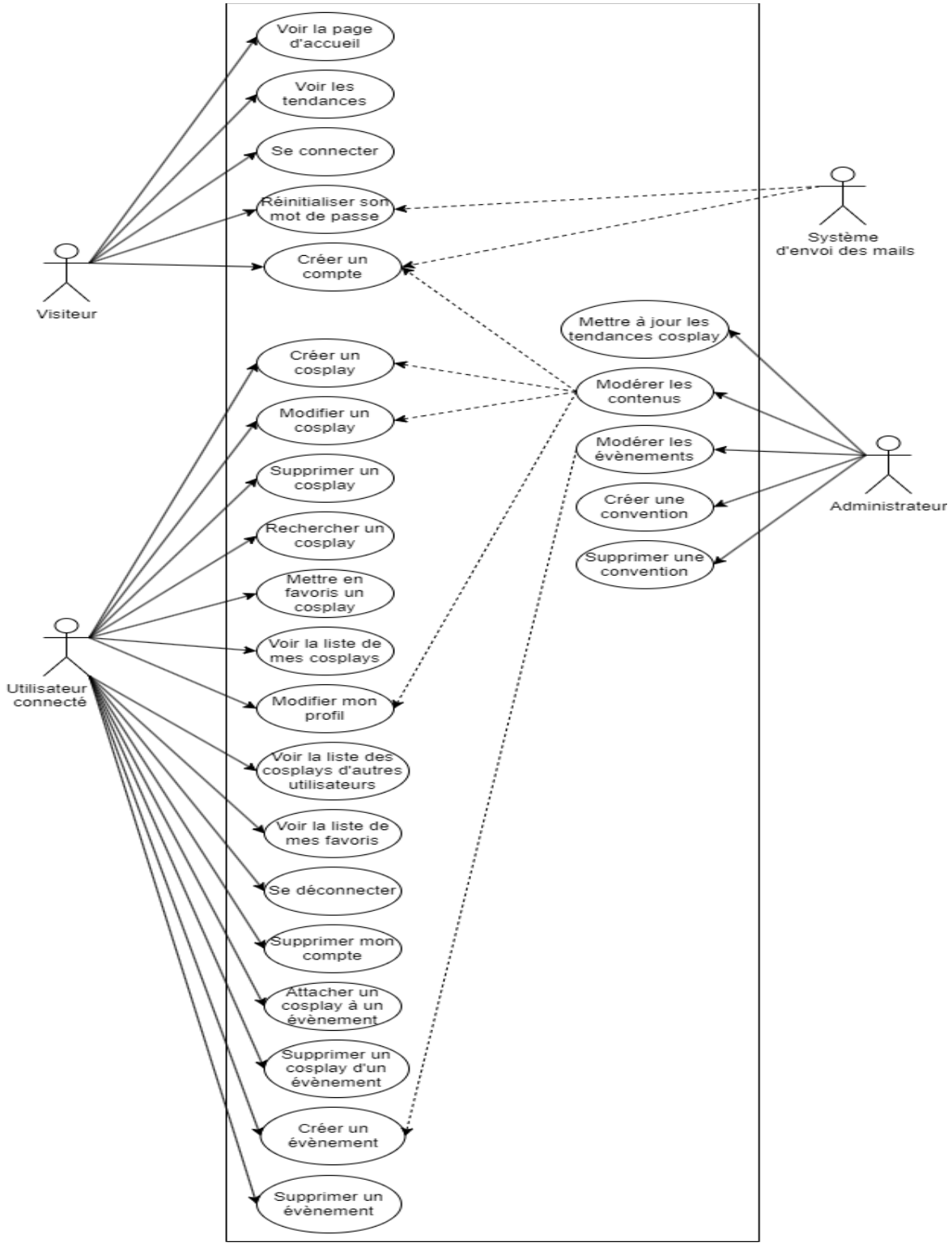
- Créer un évènement officiel, le modifier le supprimer.
- Créer son compte, le modifier, le supprimer.
- Accéder à la liste des cosplayeurs qui participent à l'évènement
- Accéder aux cosplayeurs de sa région et inviter un cosplayeur à un évènement.

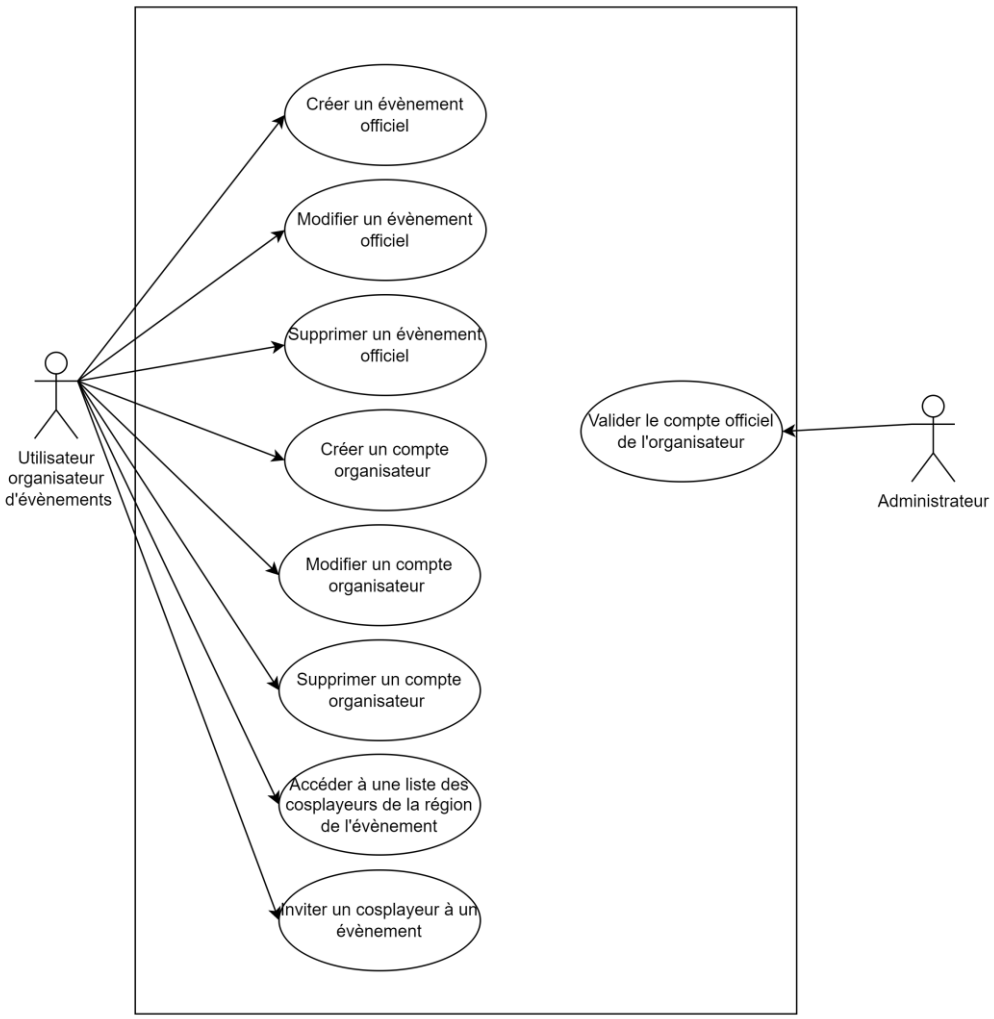
➢ <u>Pour un administrateur :</u>

Un administrateur pourra faire les actions suivantes :

- Mettre à jour les tendances cosplay.
- Supprimer un compte utilisateur s'il ne respecte pas les standards de la communauté.
- Supprimer un cosplay s'il ne respecte pas les standards de la communauté.
- Supprimer une image d'un cosplay si celle-ci ne respecte pas les standards de la communauté.
- Supprimer un évènement s'il ne respecte pas les standards de la communauté.
- Créer un évènement officiel, le modifier, le supprimer.

Selon les points ci-dessus, voici les diagrammes de cas d'utilisation de l'application Cosplay-Maker :





D. Analyse des fonctionnalités

Afin de répondre aux attentes de la communauté cosplay, j'ai pu identifier les besoins suivants :

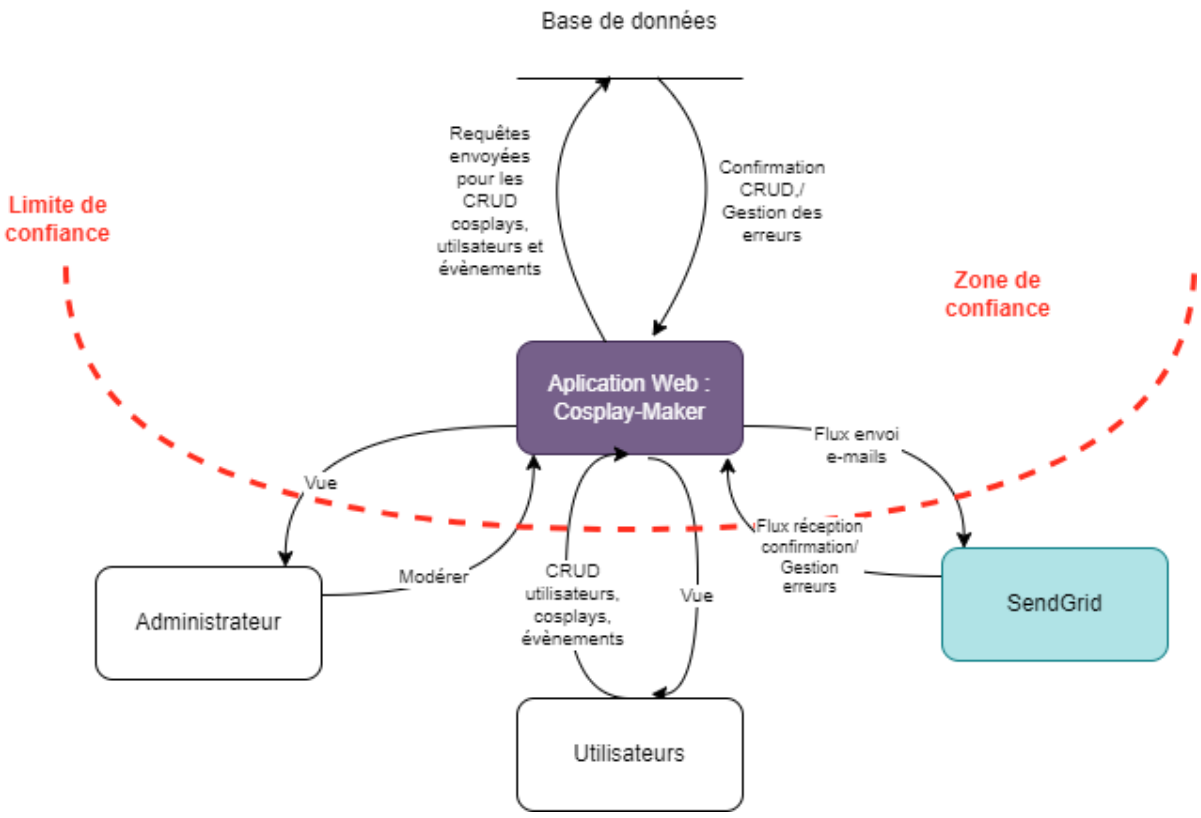
- Un système de profil utilisateur pour afficher les informations de contact, l'historique de cosplay et les créations passées de chaque utilisateur.
- Une page "découvrir", comme son nom l'indique, de parcourir de manière aléatoire les cosplays public disponible sur la plateforme.
- Une galerie de créations permettant aux utilisateurs de créer des cosplays tout en incluant le nom du personnage, son univers, la catégorie, la date de début et de fin, les éléments à acheter, les tâches à effectuer, le budget par élément et le budget total, ainsi que des images des pièces et du cosplay final.
- Un système de recherche de cosplay selon plusieurs critères (nom du personnage, univers) dans le but de trouver des idées de cosplays, accompagné de leurs liens vers des sites externes pour acquérir les matériaux nécessaires à leurs réalisations.
- Un calendrier de fabrication permettant de planifier de manière efficace la conception, les besoins et les achats.
- Un système de favoris permettant aux utilisateurs de sauvegarder les cosplays d'autres utilisateurs.
- Une carte interactive permettant de voir les cosplayeurs aux alentours et les évènements à venir proche de chez soi. Cette fonctionnalité reposera sur le stockage en base de données de localisation des utilisateurs et sera représentée par un cercle de proximité dans l'interface afin de ne pas divulguer la position exacte de la personne.

- Une gestion des évènements tels que les shootings ou les conventions, avec informations sur les dates, les lieux et le nombre de participants.
- Un système de tendances, comme son nom l'indique, mettra en avant les cosplays du moment sur le mois en cours. À savoir, les cosplays les plus réalisés sur le mois.
- Un système de partage sur les réseaux sociaux telles que Instagram ou Facebook.
- Une partie administration pour modérer le site, supprimer du contenu ou des comptes, gérer le carrousel des tendances et les annonces sur la page d'accueil, ainsi que des statistiques sur l'utilisation du site.

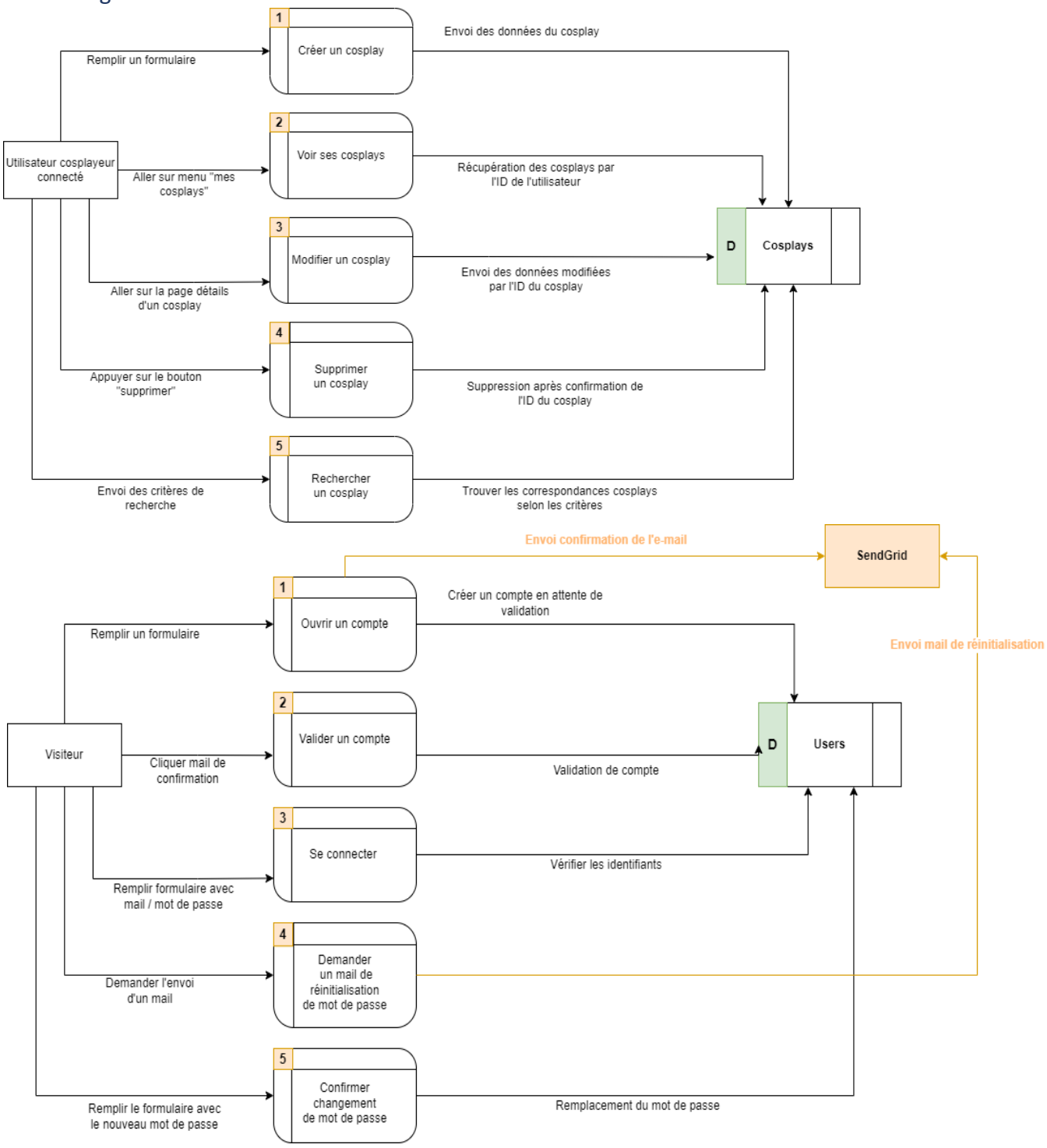
E. Diagramme de contexte - niveau 0

Pour mieux comprendre le fonctionnement et l'interaction des différentes fonctionnalités de notre application Cosplay-Maker, j'ai élaboré un diagramme de flux de données de niveau 0. Ce diagramme présente une vue d'ensemble des flux de données qui transitent au sein de l'application. Cela permet notamment d'illustrer les échanges entre les acteurs et les composants du système.

Voici donc le diagramme de contexte - niveau 0 :



F. Diagramme de flux de données - niveau 1

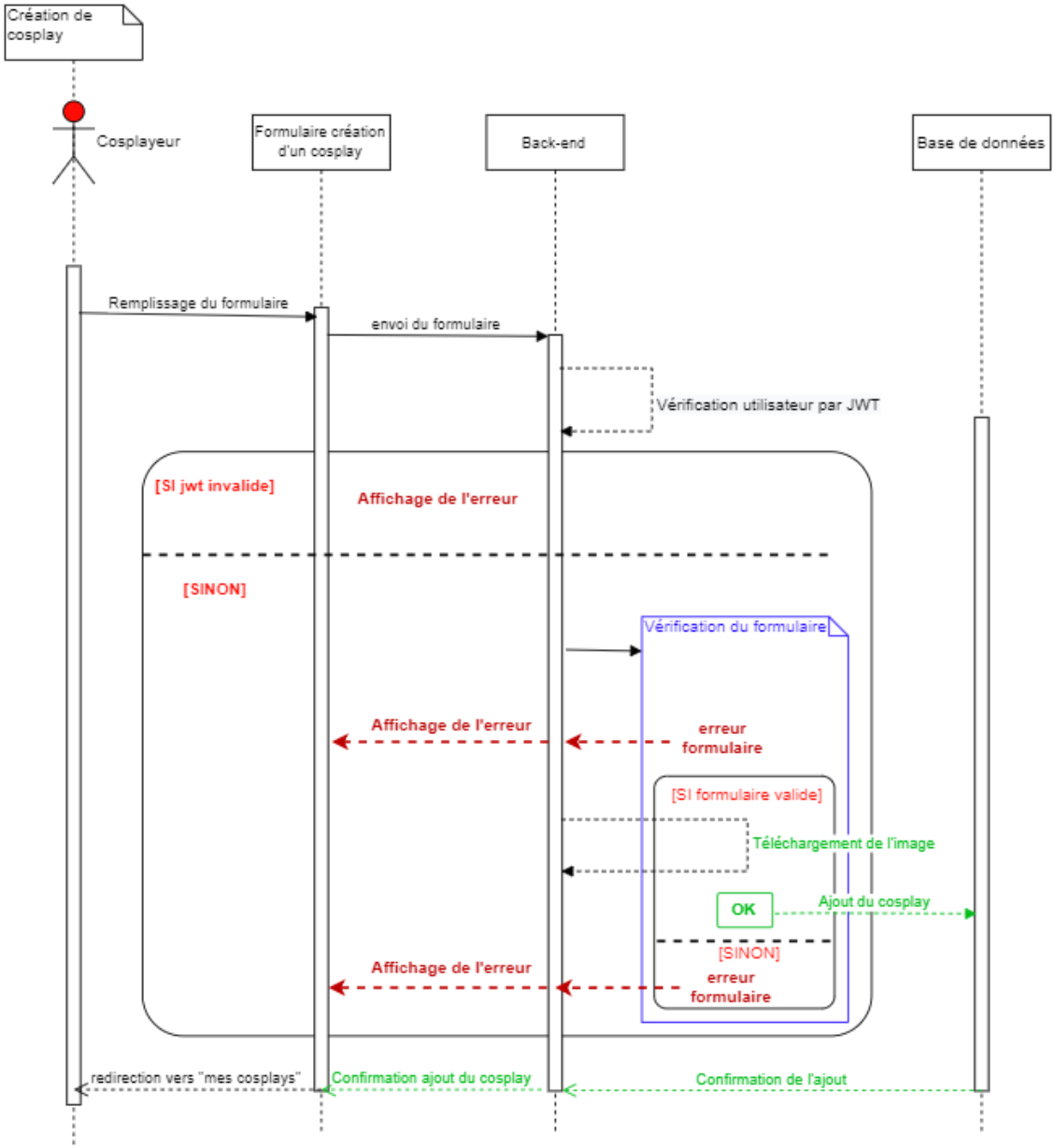


G. Diagramme général du site

Pour me permettre de mieux aborder les points cruciaux du projet et cerner les besoins sur chaque page, j'ai créé un diagramme général listant les pages et les accès sur chacune d'elles. Il permet d'avoir une vision globale du projet, mais aussi d'anticiper le découpage de l'interface en sous-composants réutilisables et le routage des différentes pages.

[Veuillez trouver en Annexe les différents diagrammes liés à cette partie]

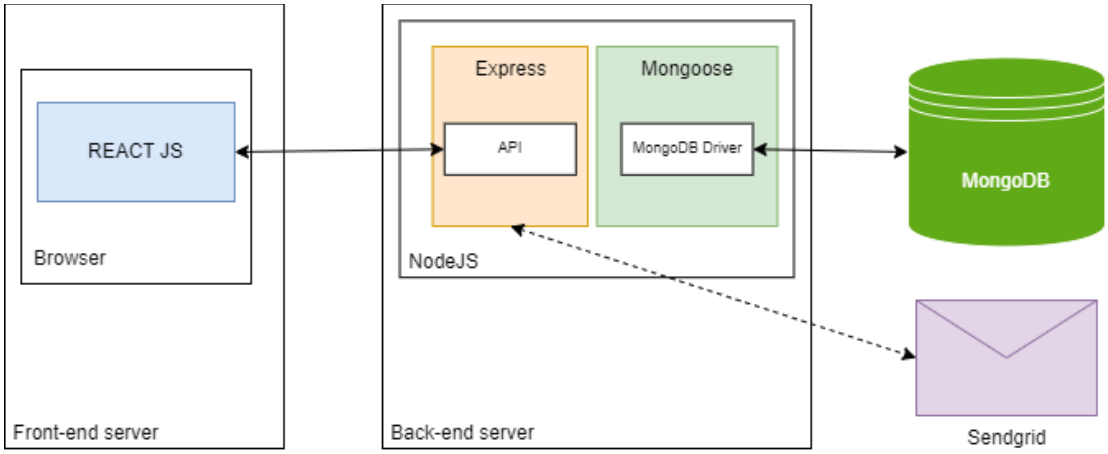
H. Diagramme de séquences



Partie III : Conception et architecture de l'application

A. Architecture technique

Voici l'architecture globale du projet Cosplay-Maker :



Architecture des dossiers du projet

Le découpage du code de mon projet Web suit une structure organisée et séparée en dossiers, avec une distinction entre le dossier "back" et le dossier "front". L'ensemble du code du projet est versionné dans un dépôt unique. (MONOREPO en anglais).

Dans le dossier "back"

C'est le dossier qui contient la partie back-end du projet.

- Le dossier **"features"** contient les différentes fonctionnalités de mon application. Chaque fonctionnalité est regroupée dans son propre dossier et comprend trois fichiers :
  - Le fichier du **".controller.js"** qui gère la logique métier, les actions liées à la fonctionnalité et l'interaction avec la base de données via l'ODM MONGOOSE[5].
  - Le fichier du **".model.js"** qui définit la structure des modèles de données Mongoose utilisées par la fonctionnalité.
  - Le fichier **".route.js"** qui spécifie les routes et les points d'entrée pour accéder à la fonctionnalité.
- Le dossier **"middlewares"** contient les middlewares qui s'appliquent aux routes de l'application. Les middlewares permettent d'exécuter des actions avant, pendant ou après le traitement des requêtes, tels que l'authentification, la gestion des erreurs, le téléchargement des images etc.
- Les dossiers **"utils"** et **"config"** contiennent des fichiers utilitaires et de configuration. Ces fichiers fournissent des fonctionnalités partagées et des paramètres de configuration pour simplifier le développement et la gestion de l'application.

Dans le dossier "front"

C'est le dossier qui contient la partie front-end du projet.

- Le dossier **"pages"** regroupe les différentes pages de l'application. Chaque page peut avoir sa propre fonctionnalité et est accompagnée de fichiers **".tsx"** pour la logique de présentation et de rendu, ainsi que de fichiers **".css"** pour les styles spécifiques à la page.
- Le dossier **"icons"** contient le fichier **"icon.tsx"**, qui regroupe les icônes SVG utilisées. Cela facilite l'utilisation et la gestion des icônes dans différents composants.
- Le dossier **"components"** contient les fichiers des composants réutilisables dans tout le projet.

[5] **Mongoose** est un ODM (Object-Document Mapping) pour MongoDB en Node.js, qui permet de simplifier et de faciliter l'interaction avec la base de données MongoDB. Il fournit une couche d'abstraction au-dessus de MongoDB et offre des fonctionnalités avancées telles que la définition de schémas, la validation des données, les requêtes avancées, les hooks et les méthodes personnalisées, facilitant ainsi la manipulation des données dans une application Node.js utilisant MongoDB.

- Le dossier **"services"** contient les fichiers qui effectuent les appels vers le back-end depuis le front-end. Ces fichiers facilitent les communications entre les deux parties.

En utilisant cette structure de dossiers et de fichiers, Cosplay-Maker bénéficie d'une organisation claire et modulaire, facilitant la maintenance, la collaboration future d'autres développeurs et le développement de nouvelles fonctionnalités.

Choix techniques

Pour ce projet, j'ai choisi la stack MERN[6], qui comprend les technologies suivantes :

1. Base de données : MongoDB

Il s'agit donc d'une base de données NoSQL[7] qui me permet de stocker plusieurs types de données :

- Les données utilisateurs
- Les données des cosplays
- Les données des évènements
- Les favoris

Mon choix s'est porté sur MongoDB pour plusieurs raisons.

La première raison est qu'il s'agit de la base de données que nous avons vu en cours théorique, cela me confère donc une bonne base de compréhension et d'adaptation pour ce projet.

De plus, mon choix s'est également tourné vers **le couple Disponibilité - Distribution** du THEOREME DE BREWER[8].

Ces deux propriétés (AP en anglais pour *Availability* and *Partition Tolerance*) permettent à la donnée d'être toujours disponible même en cas de panne partielle du système mais également que toute requête fournit un résultat correct, peu importe le nombre de serveurs.

Par défaut, sans modifier le fichier de configuration de mongoDB, on a :

- **Réplication :** Par défaut, MongoDB n'est pas configuré en tant que replica set. Cela signifie que la réplication et la haute disponibilité ne sont pas activées par défaut.
- **Partitionnement (sharding) :** Par défaut, MongoDB n'est pas configuré en tant que cluster de sharding. Le partitionnement des données n'est donc pas activé par défaut.
- **Cohérence :** Par défaut, MongoDB utilise un niveau de cohérence "quorum" pour les opérations de lecture et d'écriture, ce qui signifie qu'une majorité des membres du replica set doit confirmer l'opération pour qu'elle soit considérée comme réussie.

Cette configuration peut être changée selon les besoins, si par exemple, je veux une cohérence stricte pour des opérations complexes impliquant plusieurs documents, je pourrais utiliser les transactions dans MongoDB. Les

[6] La stack MERN permet de créer des applications web en utilisant MongoDB pour la base de données, Express.js pour le framework côté serveur, React pour la partie front-end, et Node.js pour exécuter le code JavaScript côté serveur.

[7] NoSQL, également connu sous le nom de "Not Only SQL" (pas seulement SQL), est une approche de stockage de données qui diffère du modèle relationnel traditionnel utilisé par les bases de données SQL. Le terme NoSQL fait référence à un large ensemble de technologies et de systèmes de gestion de bases de données qui sont conçus pour gérer efficacement des données volumineuses, variées et souvent non structurées. Les caractéristiques clés du NoSQL : Structure de données flexible, Évolutivité horizontale, Modèles de données variés, Haute disponibilité et tolérance aux pannes, Performances optimisées.

[8] **Théorème de Brewer :** aussi appelé théorème de CAP formalisé en 2000 par Éric A. Brewer qui précise les 3 propriétés fondamentales des bases de données : Consistency, Availability and Partition tolerance.

transactions garantissent l'atomicité, la cohérence, l'isolation et la durabilité (ACID[9]) des opérations, mais elles peuvent entraîner une perte de performance.

À noter que sur Cosplay-Maker, les actions utilisateurs provoqueront plus d'écritures que de lectures sur la base de données. Puisqu'il s'agit d'un projet que j'effectue seule, il peut arriver que j'ajoute au fur et à mesure des attributs sur une entité de ma base de données, par exemple, je n'avais pas pensé initialement à la catégorisation des cosplays (privée ou publique), pourtant, cela sera essentiel si la personne ne souhaite pas partager ces cosplays. Les schémas MongoDB sont flexibles et évolutifs par nature, ce qui me permettra d'adapter très rapidement la base de données à mon besoin.

2. Back-end : NodeJS

J'ai choisi NodeJS comme environnement d'exécution côté serveur pour mon application pour rester sur un environnement complet en JavaScript. En utilisant Node.js, je peux développer à la fois le code client et le code serveur en JavaScript, ce qui simplifie la gestion des données et des fonctionnalités de l'application.

En JavaScript, presque toutes les opérations sont non bloquantes (grâce au moteur d'exécution V8 mais aussi grâce à sa nature asynchrone). Il s'agit donc d'un choix pertinent pour ce projet, la seule limite pour ce type d'application serait les limites des ressources en CPU où la nature single-thread défavoriserait cela. Mais dans cette application, nous n'avons pas de calculs gourmands, pas de machine Learning etc...

3. Back-end : Express

Express.js est un Framework web pour Node.js. Je l'ai choisi pour créer l'API RESTful[10] de l'application en raison de sa simplicité et de sa flexibilité. Express.js facilite la définition des routes, la gestion des requêtes et des réponses, ainsi que l'intégration de middleware pour des fonctionnalités supplémentaires. La documentation est très facile à parcourir. De plus, tout comme MongoDB, il s'agit d'un Framework vu en cours, ce qui me facilitera son utilisation pour ce projet.

4. Front-end : React avec TypeScript

React.js est une librairie front. Je l'ai choisie pour construire l'interface utilisateur de mon application car il s'agit d'une librairie facile à prendre en main (notamment grâce aux cours de cette année), et elle est très largement utilisée et popularisée, ce qui me permettra de trouver facilement des ressources en ligne si nécessaire. React me permettra de découper l'interface en composants réutilisables.

TypeScript est un sur-ensemble de JavaScript qui ajoute des fonctionnalités de typage statique au langage. Il permet de détecter les erreurs potentielles de type pendant la phase de développement, ce qui facilite le débogage et améliore la maintenabilité du code. Les avantages de l'utilisation de TypeScript :

- **Typage statique :** TypeScript me permet de définir les types de données pour mes variables, mes paramètres de fonction, mes objets et mes retours de fonction. Cela permet de détecter les erreurs de type à la compilation plutôt qu'à l'exécution, ce qui conduit à un code plus robuste et moins sujet aux erreurs.
- **Autocomplétions :** VS Code autocomplète de manière précise et propose des suggestions contextuelles. Cela me permet de coder plus rapidement et de réduire les erreurs de syntaxe.

Voici un exemple ci-dessous d'un fichier de typage avec TypeScript pour la création d'un cosplay.

---

[9] ACID : Atomicity, Consistency, Isolation and Durability : Atomicity (Atomicité)
Atomicity (atomicité) : Les opérations d'une transaction sont traitées comme une unité indivisible.
Consistency (Cohérence) : La base de données passe d'un état valide à un autre état valide après l'exécution d'une transaction.
Isolation (Isolation) : Les transactions concurrentes sont exécutées de manière isolée les unes des autres.
Durability (Durabilité) : Les modifications effectuées par une transaction sont persistantes et survivent aux pannes du système.

[10] **API RESTful :** c'est une architecture logicielle qui permet la communication entre différentes applications via des services web. REST est un style d'architecture qui repose sur les principes du protocole HTTP (Hypertext Transfer Protocol). Cela nous permet donc d'effectuer les opérations standardisées : CREATE, READ, UPDATE, DELETE pour manipuler / mettre à jour les ressources.

create-cosplay.type.ts

front > src > pages > cosplay > cosplay-create > create-cosplay.type.ts > ...

```ts
Cassandra Forestier, 2 months ago | 1 author (Cassandra Forestier)
import { UploadChangeParam } from "antd/es/upload";

Cassandra Forestier, 2 months ago | 1 author (Cassandra Forestier)
export interface CreateCosplayFormValues {
  status: "En cours" | "Plan";
  character: string;
  universe: string;
  variant: string;
  category: string;
  budget: number;
  pictures: UploadChangeParam<string | Blob>;
}
```

Quelques librairies :

1. Masonry.js

Lors de la réalisation de la maquette du projet, j'avais en tête une mise en page avec une disposition en grille avec des éléments de taille variable pour dynamiser le contenu comme le fait Pinterest. Malheureusement, les techniques traditionnelles telles que Flexbox ou les colonnes CSS ne répondaient pas entièrement à mes besoins, car elles ne permettaient pas d'obtenir une disposition optimale des éléments. J'ai donc trouvé la librairie Masonry.js.

Masonry.js me permet d'organiser dynamiquement les éléments de la grille en fonction de leur taille. Une fois que le HTML est chargé, le script de Masonry calcule les hauteurs de tous les éléments et les replace en utilisant des coordonnées x/y avec la propriété CSS 'position:absolute'. J'ai grâce à cela une mise en page réactive, avec une utilisation optimale de l'espace disponible. Cela me permet de ne pas imposer une taille spécifique aux images que les personnes importeront sur le site.

2. Ant Design

Le choix d'Ant Design s'est essentiellement basé sur sa compatibilité forte avec React.

Ant Design a été développé en tant que bibliothèque de composants React. On a donc une intégration transparente avec React, permettant une utilisation et une personnalisation plus facile.

J'ai apprécié la clarté d'organisation de la documentation avec des exemples de code. De plus, Ant Design gagne en popularité auprès des développeurs, j'ai donc voulu adopter cette bibliothèque pour me familiariser avec celle-ci.

3. Morgan.js

Morgan.js est une bibliothèque JavaScript utilisée pour la gestion des journaux (logs) dans les applications web. Elle offre une solution pratique pour enregistrer les demandes HTTP et les réponses associées, ce qui permet de suivre et d'analyser l'activité du serveur de manière plus simple et rapide.

```
GET /api/validate-email/b971c2d26dcb608540715545d8e6cf2f16cbdab7 200 15.626 ms - 24
OPTIONS /api/login 204 0.798 ms - 0
POST /api/login 200 341.563 ms - 245
OPTIONS /api/cosplays/me 204 0.453 ms - 0
GET /api/cosplays/me 200 49.461 ms - 2
OPTIONS /api/cosplay/create 204 0.385 ms - 0
POST /api/cosplay/create 201 74.204 ms - 356
OPTIONS /api/cosplays/me 204 0.906 ms - 0
GET /api/cosplays/me 200 16.978 ms - 381
GET /uploads/image-file-1688975832475.jpg 200 7.686 ms - 228203
```

4. SendGrid

J’ai choisi SendGrid pour le système d’envoi de mails pour plusieurs raisons :

Évolutivité et fiabilité : SendGrid est une plateforme SaaS d'envoi d'e-mails plébiscité pour sa haute disponibilité et sa fiabilité. Chaque mois sendGrid envoi plus de 90 milliards d'emails à travers le monde avec un taux de disponibilité de 99.99%.

Système d'envoi transactionnel : Un système d'e-mail transactionnel est un mécanisme qui permet d'envoyer des e-mails dans le cadre d'interactions spécifiques et prévues par l'application. Ces e-mails sont généralement liés à des actions transactionnelles, telles que des confirmations d'inscription, des réinitialisations de mot de passe, des notifications de commande, des confirmations de paiement, etc. Le système d'e-mail transactionnel garantit que ces e-mails importants sont envoyés de manière fiable et en temps opportun.

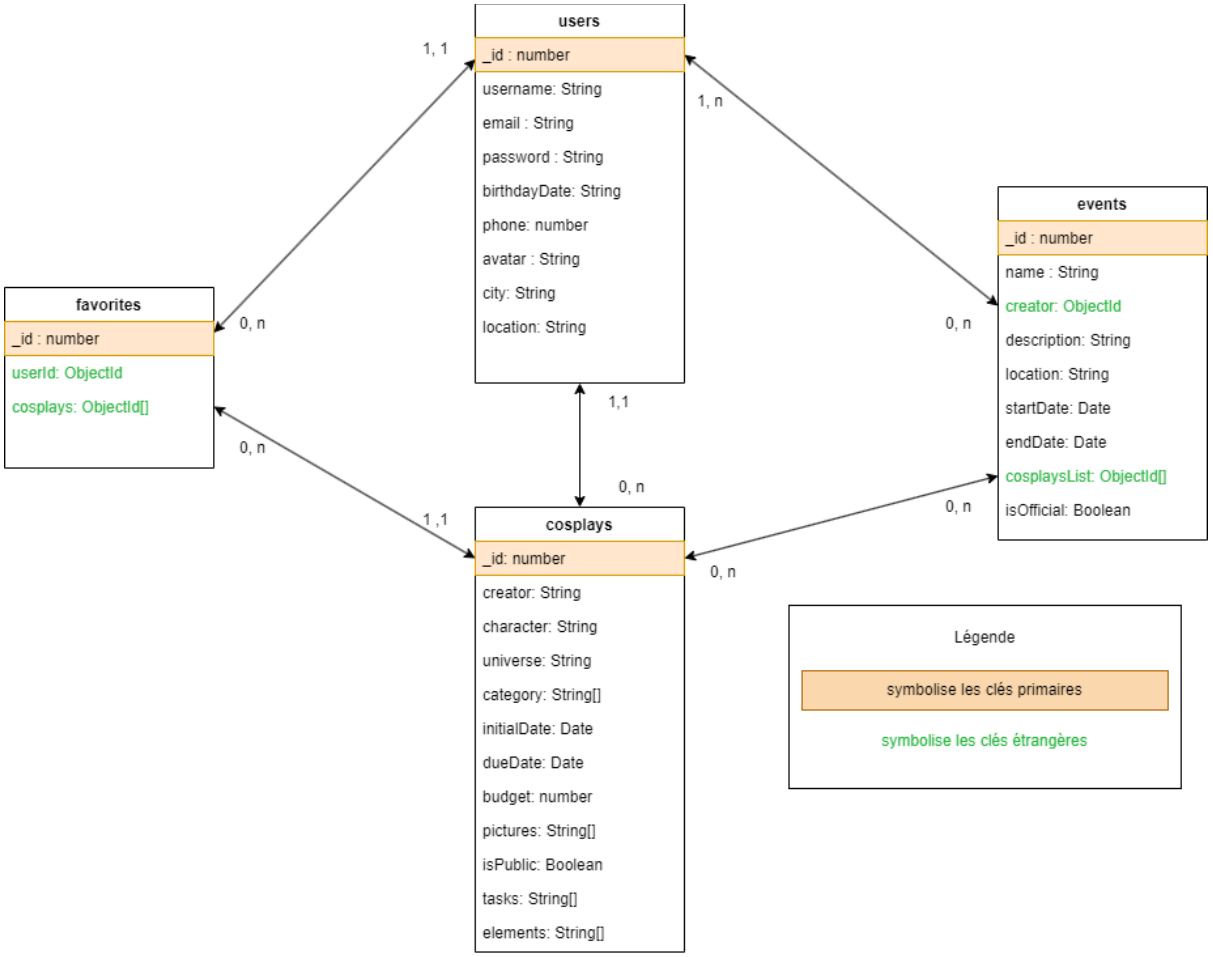
Facilité d'intégration : SendGrid fournit des bibliothèques et des API dans de nombreux langages de programmation courants, ce qui facilite l'intégration de ses fonctionnalités d'envoi d'e-mails dans différentes applications.

Gestion de la réputation d'expéditeur : SendGrid gère activement la réputation d'expéditeur pour garantir que les e-mails envoyés à partir de ses serveurs atteignent les boîtes de réception des destinataires. Cela peut inclure des fonctionnalités telles que le suivi des statistiques d'envoi, la gestion des listes de diffusion, la gestion des désabonnements et le filtrage des courriers indésirables. SendGrid travaille en étroite collaboration avec les fournisseurs de services de messagerie pour maintenir une bonne réputation d'expéditeur, ce qui peut améliorer les taux de délivrabilité des e-mails.

Offre gratuite : SendGrid dispose d'une offre gratuite à vie avec l'envoi de 100 e-mails par jour. Cette capacité d'envoi pour être augmenter par la suite en souscrivant à une offre premium.

B. Conception de la base de données

Voici le diagramme entité-relation de la base de données :



Dictionnaire de données

Suite à ce diagramme, j'ai pu apporter un peu plus de détails aux données que je veux dans la base de données. Pour des raisons de simplicité, voici les codes utilisés :

- Nature : soit **N** pour **Numérique, AN** pour **Alphanumérique** et Date pour les dates.
- Type : soit **C** pour **Calculé** ou **NC** pour **Non Calculé**

1. Entité : users

Propriété	Description	Nature	Type	Taille	Observation
_id	Identifiant unique de l'utilisateur	N	NC	12	Clé primaire, unique
username	Pseudo de l'utilisateur	AN	NC		Unique
city	Ville de l'utilisateur	AN	NC		
email	Adresse e-mail de l'utilisateur	AN	NC		Unique
password	Mot de passe de l'utilisateur	AN	NC		Unique, hashé
birthdayDate	Date de naissance de l'utilisateur	Date	NC		
phone	Téléphone de l'utilisateur	N	NC		
avatar	Image du profil de l'utilisateur	AN	NC		

2. Entité : cosplays

Propriété	Description	Nature	Type	Taille	Observation
_id	Identifiant unique du cosplay	N	NC	12	Unique, clé primaire
creator	Identifiant de l'utilisateur créateur du cosplay	N	NC		Clé étrangère vers l'entité users
character	Nom du personnage du cosplay	AN	NC		
universe	Univers auquel appartient le personnage	AN	NC		Tableau
category	Catégorie du cosplay (ex : manga, film, jeu vidéo)	AN	NC		Tableau
initialDate	Date de début de création du cosplay	Date	NC		
dueDate	Date de fin de création du cosplay	Date	NC		
budget	Budget total alloué au cosplay	N	NC		
pictures	Images de référence	AN	NC		Tableau
isPublic	Permet de catégoriser le cosplay en public ou privée	N	NC		Booléen
tasks	Tâches à réaliser pour ce cosplay (créer le sac, coudre le t-shirt …)	AN	NC		Tableau
elements	Eléments d'un cosplay (la perruque, la jupe ...)				Tableau

3. Entité : favorites

Propriété	Description	Nature	Type	Taille	Observation
_id	Identifiant unique du favori	N	NC	12	Unique, clé primaire
userId	Identifiant de l'utilisateur créateur du favori	N	NC		Clé étrangère vers l'entité users
cosplays	Un tableau des identifiants cosplays mis en favoris	N	NC		Clés étrangères vers l'entité cosplays

4. Entité : events

Propriété	Description	Nature	Type	Taille	Observation
_id	Identifiant unique de l'évènement	N	NC	12	Unique, clé primaire
name	Nom de l'évènement	AN	NC		
creator	Identifiant de l'utilisateur ayant créé l'évènement	N	NC		Clé étrangère vers l'entité users

description	Description de l'évènement	AN	NC		
location	Localisation de l'évènement sur une map	AN	NC		
startDate	Date du début de l'évènement	Date	NC		
endDate	Date de fin de l'évènement	Date	NC		
cosplaysList	Liste des identifiants des cosplays ajoutés par les utilisateurs pour cet évènement	N	NC		Tableau, clés étrangères de l'entité cosplays
isOfficial	Permet de catégoriser l'évènement en officiel ou non	N	NC		Booléen

C. Maquette et charte graphique

La maquette et le design de l'interface sont des aspects essentiels dans le développement d'une application web. Ils jouent un rôle crucial dans l'expérience utilisateur et dans l'attrait visuel de l'application. Pour mon projet, j'ai défini une charte graphique qui guide l'apparence visuelle de l'interface. Voici le tableau de la charte graphique :

Toutes ces couleurs ont été générées à l'aide de l'application Web Coolors, ce qui m'a permis de voir si le contraste était suffisant.

Pour implémenter cette charte graphique dans le code, j'utilise le composant "ThemeConfig" d'Ant Design. Les valeurs personnalisées du thème sont définies comme suit :

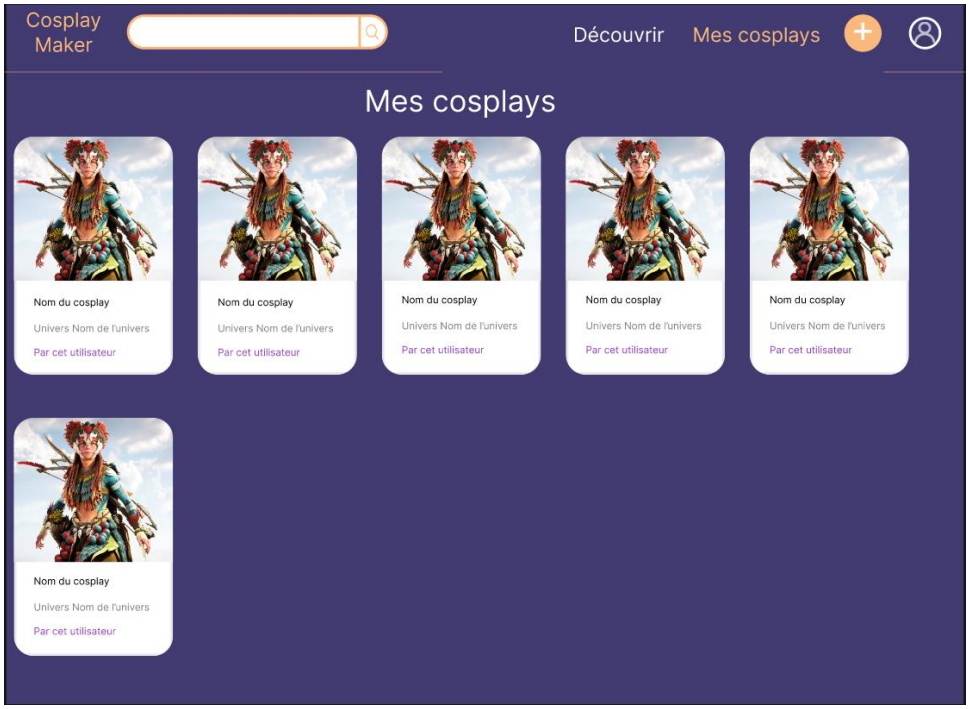
```
const themeCustomValues: ThemeConfig = {
  token: {
    colorPrimary: "#FBBA7B",
    wireframe: false,
    colorPrimaryBg: "#FBBA7B",
    colorPrimaryBgHover: "#FFF",
    borderRadius: 20,
  },
  algorithm: darkAlgorithm,
};
```

Concernant la maquette de l'application, elle a été conçue via l'application Web Figma.

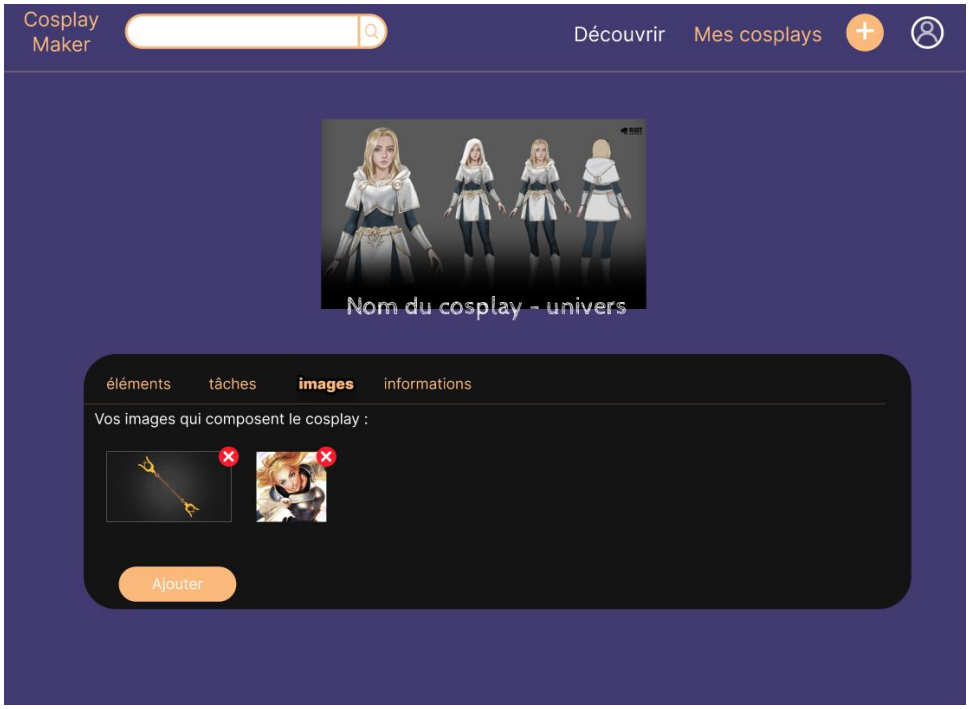
J'ai créé les principales pages du site, pour avoir une idée plus précise de mon besoin.

Avant cela, j'avais également regardé les composants fournis par Ant Design pour pouvoir faire une intégration plus facile de chacune des parties du site.

Voici pour exemple la page de la vision d'un utilisateur connecté sur la page de ses cosplays :



Page des détails d'un cosplay en mode édition pour un utilisateur connecté en version Web :



Veuillez trouver dans les annexes la suite de la maquette, comportant également la version mobile.

Partie IV : Réalisation de l'application

A. Outils utilisés

Voici la liste des outils utilisés sur mon projet Cosplay-Maker :

1. IDE : Visual Studio Code

L'un des choix les plus évidents pour ce projet est l'utilisation de VS Code. VS Code est l'un des IDE les plus populaires et l'un des plus faciles à prendre en main. Il était donc évident que je m'orienterai vers ce choix. De plus, tout au long de l'année, j'ai amélioré le paramétrage de VS Code selon mes goûts et mes besoins grâce aux nombreuses extensions mises à disposition.

Voici une liste non-exhaustive de mes extensions :

- **Prettier :** il s'agit d'un LINTER[11] qui permet de créer un paramétrage sur le projet.
- **Auto Close Tag** et **Auto Rename Tag :** permet de fermer automatiquement les balises HTML/XML, et d'automatiquement renommer les balises quand l'une des deux est modifiée.
- **Npm IntelliSense, sort-imports** et **Auto-Import :** trouve, analyse et fournit automatiquement des suggestions de complétion pour les imports, trier les imports lors de la sauvegarde du fichier, mais aussi l'auto-import des dépendances lors de l'écriture du code.
- **Community Material Theme :** permet d'avoir un thème sur VS Code.

J'ai également mis à la racine de mon projet un dossier **.vscode** où se trouve **tasks.json :**

[11] Un linter est un outil d'analyse statique du code source utilisé pour identifier les erreurs, les problèmes potentiels et les violations des conventions de codage. Il vérifie le code par rapport à un ensemble de règles prédéfinies ou personnalisées et signale les problèmes rencontrés. Ils peuvent détecter des problèmes tels que des variables non utilisées, des erreurs de syntaxe, des pratiques non recommandées, des problèmes de formatage, etc.

Ce fichier permet de configurer des tâches sur mon environnement de développement. Il définit trois tâches différentes qui peuvent être exécutées à partir de la ligne de commande (**CTRL + SHIFT + P)**.

- La première tâche, **"start-back"**, utilise la commande "npm run dev" pour exécuter le serveur back-end du projet.
- La deuxième tâche, **"start-front"**, utilise la commande "npm run start" pour exécuter le serveur front-end du projet.
- La troisième tâche, **"start-all"**, dépend des deux premières tâches et permet de les exécuter simultanément. Ces tâches sont configurées pour s'exécuter dans des répertoires spécifiques du projet en utilisant l'option "cwd" pour définir le répertoire de travail.

Cela me permet de lancer facilement mon projet en une seule commande.

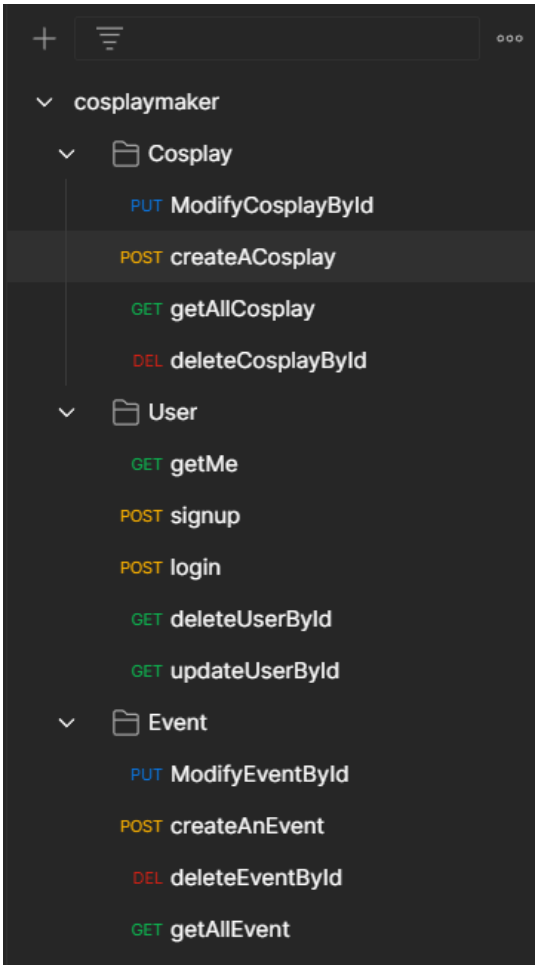
2. Postman

L'un des premiers outils que j'ai utilisé pour ce projet est Postman. En effet, j'ai commencé le projet par la mise en place back des CRUD[12] pour l'entité *Cosplay* et l'entité *User.*

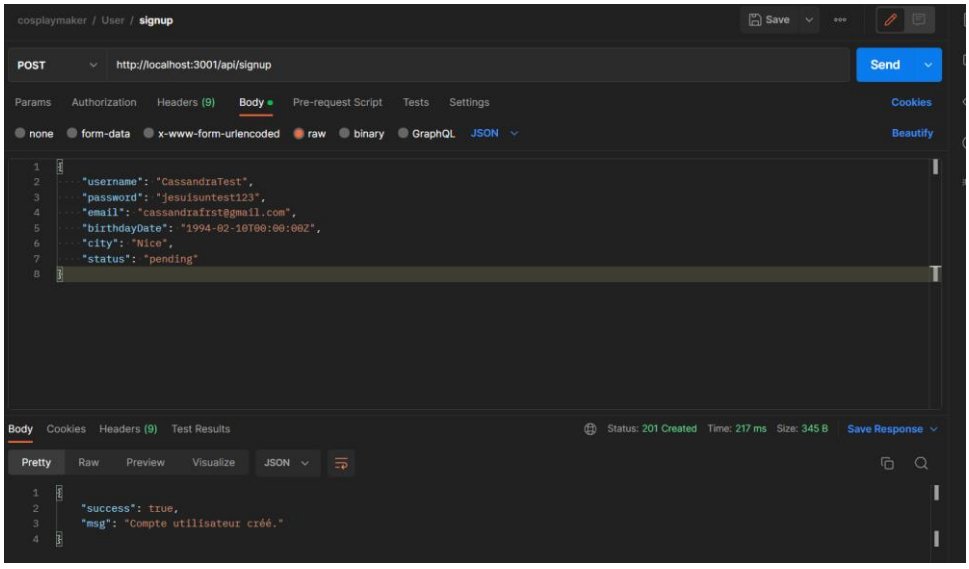
Postman permet donc l'envoi de requêtes HTTP (GET, POST, PUT et DELETE pour mon utilisation), on peut y définir les entêtes, les paramètres, le corps de la requête et même les authentifications nécessaires.

Cela me permettra à l'avenir, de pouvoir partager ce Workspace aux personnes qui s'ajouteront au projet.

[12] CRUD (Create, Read, Update, Delete) est un acronyme utilisé pour décrire les opérations de base effectuées sur les données d'une application, à savoir créer, lire, mettre à jour et supprimer des enregistrements.

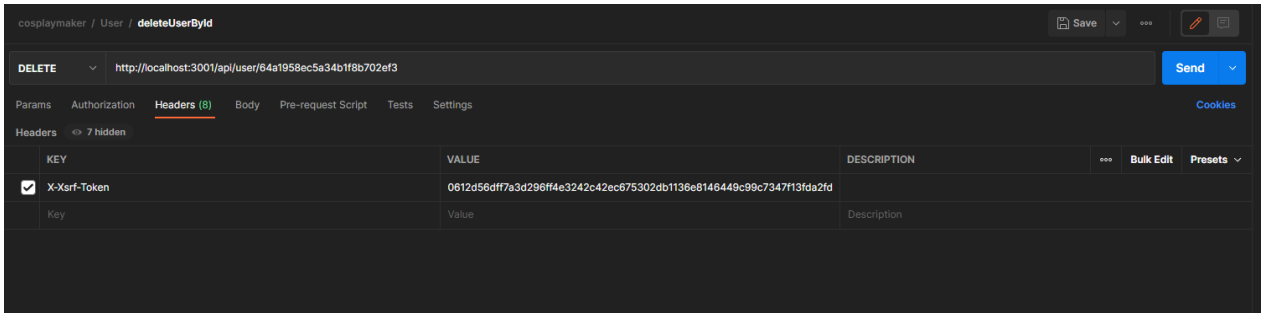


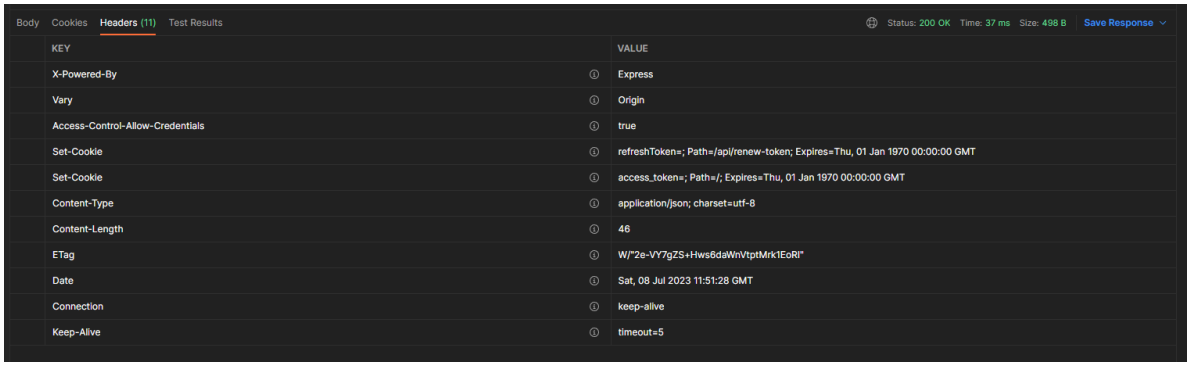
Voici un test pour la création d'un compte utilisateur :



Et voici la suppression de ce compte via Postman :

Ici, le **x-xsrf-token** doit être donné, mais également le cookie, qui doit être mis en place avant d'envoyer la requête.





3. MongoDB Compass

MongoDB Compass est une interface graphique qui permet d'interagir avec ma base de données en local. Il me permet de vérifier les schémas, vérifier si le CRUD est bien fonctionnel, mais aussi vérifier les hachages de données etc. C'est un outil qui facilite donc la vérification, notamment lors de mes tests manuels. Je peux également gérer mes utilisateurs, mes cosplayeurs directement depuis cette interface. Si j'ai besoin de faire des modifications en direct voire même des suppressions.

4. Docker Desktop

Docker desktop me permet de voir mon conteneur MongoDB lancé localement. Cela me permet de lancer automatiquement le conteneur à chaque fois que je lance mon projet.

5. GitHub

GitHub est une plateforme qui permet d'héberger son code et de le partager. Mon projet se trouve sur GitHub. Il me permet de :

- Versionner mon code
- Suivre les modifications que j'ai apporté au code, grâce à l'historique
- Définir un workflow de développement via GitHub Actions
- À l'avenir, partager mon code à d'autres développeurs qui se grefferont au projet.
- Mettre en place un Readme.md qui permet d'avoir une vue globale du projet, de ses dépendances, des outils nécessaires à installer ; mais également des technologies mises en place.

GitHub Actions :

- GitHub Actions est une fonctionnalité intégrée à la plateforme GitHub.

Elle me permet de lancer de lancer automatiquement une série d'action lorsqu'un "push" est effectué.

Voici le code mis en place :

```yaml
name: CI

on:
  push:
    branches:
      - "**"

jobs:
  build-and-test-back:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
        with:
          node-version: "18.15"
          cache: "npm"
          cache-dependency-path: |
            back/package-lock.json
            front/package-lock.json

      - name: Install dependencies ( back )
        working-directory: back
        run: |
          npm install
      Cassandra Forestier, 5 months ago • Create ci.yml …
      - name: Build and test ( back )
        working-directory: back
        run: |
          npm run test

      - name: Install dependencies  ( front )
        working-directory: front
        run: |
          npm install

      - name: Build and test ( front )
        working-directory: front
        run: |
          npm run test
```

Ce code représente un workflow CI[13] (Intégration Continue) configuré pour un référentiel GitHub. Lorsqu'un "push" est effectué sur n'importe quelle branche du référentiel, ce workflow est déclenché. Il contient plusieurs étapes qui seront exécutées sur une machine virtuelle Ubuntu.

Les étapes de ce workflow consistent à :

- Vérifier et récupérer les fichiers du référentiel.
- Configurer l'environnement Node.js en utilisant la version 18.15.
- Installer les dépendances du projet dans le dossier "back".
- Construire et tester le code du projet dans le dossier "back".
- Installer les dépendances du projet dans le dossier "front".
- Construire et tester le code du projet dans le dossier "front".

6. SendGrid

SendGrid est une plateforme de communication par e-mail basée sur le cloud. En utilisant SendGrid, je peux envoyer des e-mails transactionnels notamment pour des confirmations d'inscription et des réinitialisations de mot de passe actuellement. La plateforme offre également des fonctionnalités avancées telles que la personnalisation des e-mails, la création de modèles d'e-mails dynamiques, le suivi des statistiques d'envoi et la gestion des désabonnements. Pour le moment, le projet utilise que les deux premières fonctionnalités.

[13] L'intégration continue (CI) permet d'automatiser le processus de construction, de test et de déploiement d'une application. Avec le CI, chaque fois qu'un développeur effectue un commit ou une modification de code dans un référentiel partagé, un pipeline d'intégration continue est déclenché. Cela permet la détection précoce d'erreurs, améliore la qualité du code.

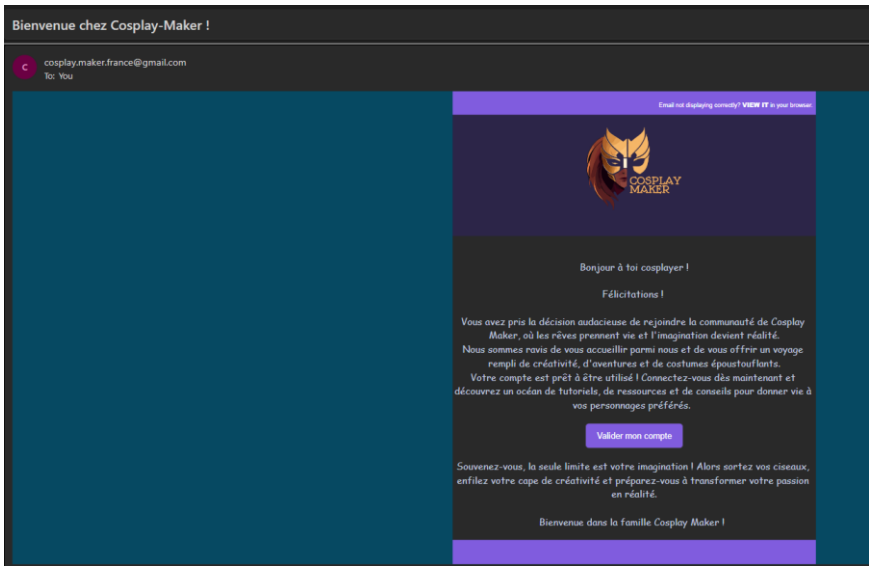
Je suis actuellement sur un compte sans abonnement, il m'offre donc 100 e-mails par jour, ce qui est largement suffisant pour le moment.

L'utilisation de l'API sendGrid V3 présente plusieurs avantages. Tout d'abord, cela simplifie l'intégration et l'envoi d'e-mails à partir de Cosplay-Maker, car je n'ai pas besoin de mettre en place et de gérer mon propre serveur SMTP. J'utilise simplement les informations d'identification fournies par SendGrid pour configurer mon application. De plus, ils fournissent un paquet npm **@sendgrid/mail** facilitant les requêtes vers leur API.

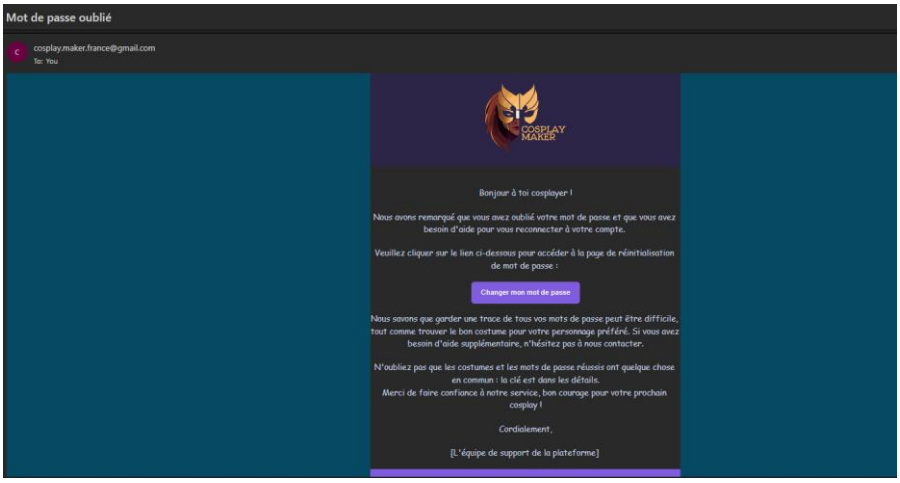
SendGrid se distingue par sa capacité à garantir une bonne délivrabilité des e-mails. La plateforme gère activement la réputation d'expéditeur pour s'assurer que les e-mails envoyés atteignent les boîtes de réception des destinataires et évitent les filtres anti-spam.

Voici donc mes deux modèles dynamiques reçus en test :

- **Création de compte :**



- **Mot de passe oublié :**



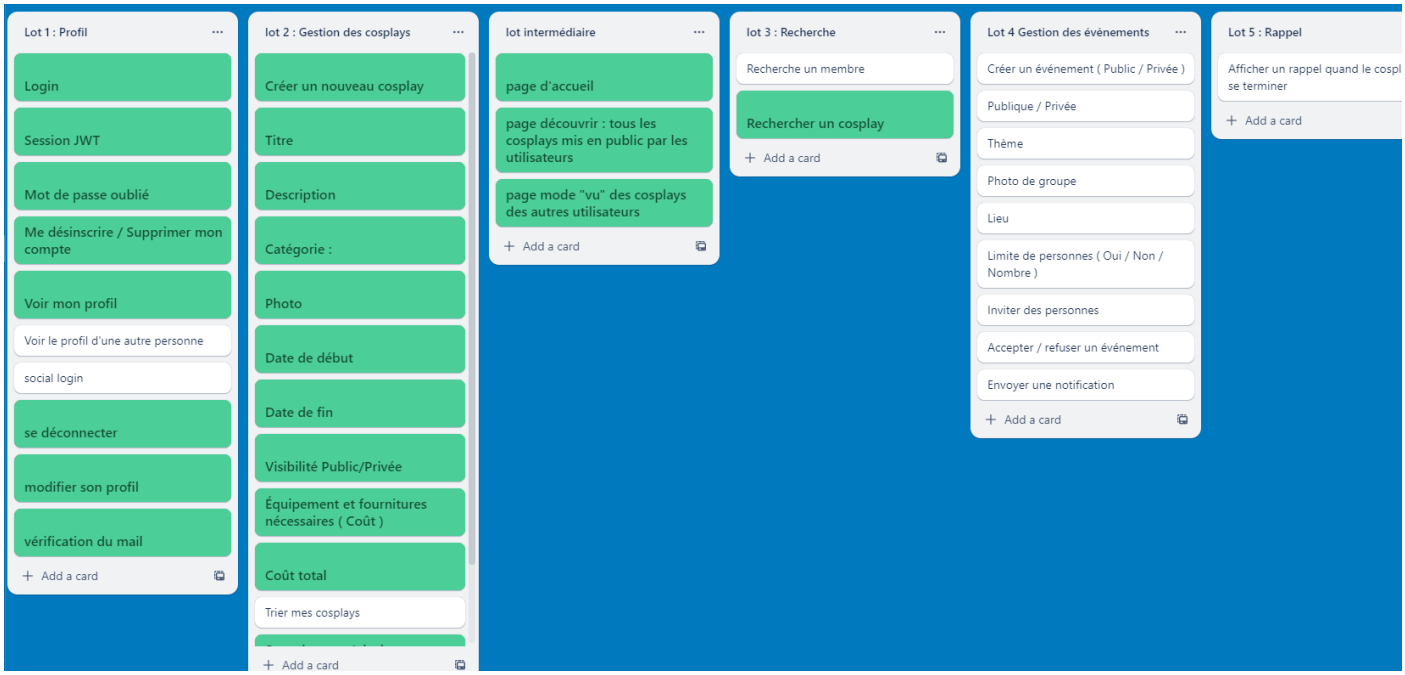
7. Trello

Trello est un outil de gestion de projet basé sur le concept de tableau Kanban. Je peux donc organiser et suivre mes tâches en cours, à faire ou archivées.

J'ai organisé mes tâches par lot. Chaque lot est donc par thème mais aussi un ordre de priorité.

Le **lot 1** est donc le premier lot que j'ai mis en place sur mon projet, puis le **lot 2** etc...

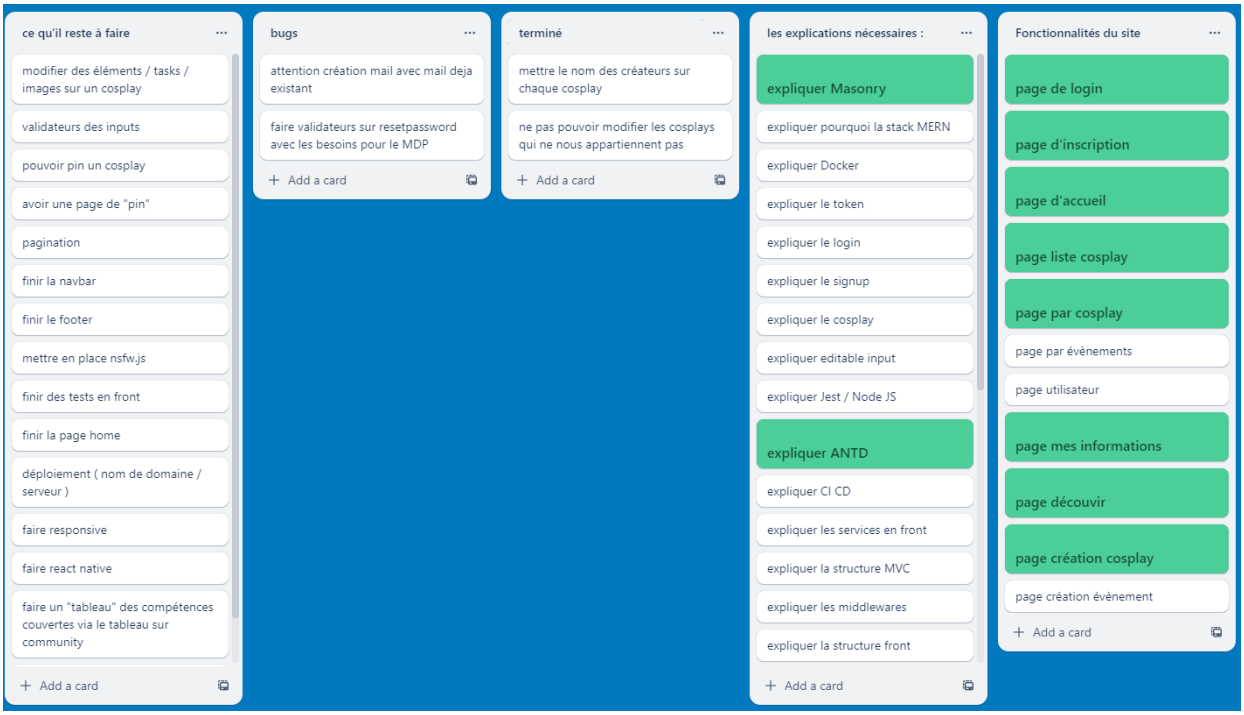
Une fois **une tâche terminée**, je la mets tout simplement **en vert**. Cela me permet de quantifier le besoin encore sur ces lots.



J'ai également d'autres colonnes plus précises notamment sur les micro-features **qu'il me reste à faire**, les **bugs** que j'ai pu détecter lors de mes tests manuels, la colonne **"terminé"** pour déplacer les cartes des deux premières colonnes dedans.

"Explications nécessaires" : c'est tout simplement pour ce rapport, et me permettre d'éviter les oublis.

"Fonctionnalités du site" : c'est les pages générales du projet en front qui ont été faites. Cela me permet également de voir les "grosses" parties manquantes sur le projet pour une visibilité générale.



A. Cas de recherche de la fonctionnalité "Recherche”

Problématique

La problématique consiste à déterminer la manière la plus efficace d'implémenter une recherche globale sur la collection "Cosplay" tout en répondant à plusieurs besoins spécifiques.

Tout d'abord, la recherche doit couvrir à la fois les champs "characters" et "universe", en utilisant une logique **“OR”** pour inclure les documents où l'un ou l'autre de ces champs correspond partiellement à la chaîne recherchée.

De plus, la recherche doit être insensible à la casse, afin de considérer indifféremment les majuscules et les minuscules dans les valeurs des champs. Cela permet d'obtenir des résultats plus complets en évitant les restrictions liées à la casse des caractères.

En outre, il est essentiel que l'implémentation soit évolutive. Cela signifie que la solution choisie doit être suffisamment flexible pour permettre l'extension de la recherche même lorsque de nouveaux champs sont introduits dans la collection "Cosplay" sans nécessiter de modifications majeures dans le code ou la structure de la base de données.

Méthodologie

Dans un premier temps, j'ai implémenté cette fonctionnalité en réalisant deux requêtes séparées à la base de données : une première requête pour obtenir les cosplays dont le "character" est semblable à la valeur recherchée puis une seconde requête pour obtenir les cosplays qui possède un "universe" semblable à la valeur recherchée.

First iteration

```
const cosplaysEqualsCharacter = await Cosplay.find({ "character" : { '$regex' : 'string', '$options' : 'i'}}.populate("creator")

const cosplaysEqualsUniverse = await Cosplay.find({ "universe" : { '$regex' : 'string', '$options' : 'i'}}.populate("creator")
```

La fonction "populate" permet de remplacer l'id du champ creator par l'utilisateur complet tel qu'il est stocké dans la collection users. En revanche elle se traduit par une requête supplémentaire vers la base de données.

Par la suite, j'ai cherché à optimiser d'avantage ces requêtes en utilisant l'opérateur "$or" de Mongo.

Second iteration

```
await Cosplay.find({
  $or: [
      { "character": { '$regex' : 'string', '$options' : 'i'}},
      { "universe" : { '$regex' : 'string', '$options' : 'i'}}
  ]
}).populate("creator");
```

Cela permet d'exécuter qu'une seule fois la fonction "populate" sur l'ensemble des résultats.

Par la suite, je me suis documenté afin de savoir si une alternative existait à cette implémentation et je suis tombée sur plusieurs articles mettant en avant l'agregagtion pipeline et la fonctionnalité full text search de mongoDB :

https://medium.com/shogun-team/how-moving-to-mongodb-text-search-solved-a-performance-bottleneck-in-shogun-fa762518b2e3

https://comsysto.wordpress.com/2013/06/26/mongo-fulltext-search-vs-regular-expressions/

Comme le souligne le paragraphe suivant, l'utilisation d'indexes de texte permet d'atteindre des performances très satisfaisantes. À l'inverse des expressions régulières, elle ne permet pas d'avoir une correspondance partielle sur un mot donnée mais uniquement une correspondance par mot dans une chaine de (plusieurs) mots. Dans notre cas cette limitation est acceptable pour les besoins du projet.

$regex

Regex doesn't take advantage of indexes, unless you are searching in beginning of string using ^ operator.
Regex allows you to search partial text. therefore . and so many other patterns.*
Regex doesn't support stop or noise words.

$text

text indexes in mongodb are really fast and should be preferred. However, MongoDB does not implement full featured text indexes. One main drawback is, it doesn't support partial match. e.g. if you are searching for cat, it will search only for cat and cats but not bobcat or caterpiller.

Bottom line is if you are looking to implement feature like RDBMS like operator, '$text' will not help you (at least in current implementations of MongoDB, but in future it may change).

L'un des gros avantages du pipeline d'agrégation est qu'il optimise les étapes de l'agrégation : *" One of the best features of the MongoDB Aggregation Pipeline is that it automatically reshapes the query to improve its performance"* https://www.mongodb.com/basics/aggregation-pipeline

Création d'un pipeline d'agrégation via l'outil Mongo Compass : https://www.mongodb.com/blog/post/quick-start-nodejs--mongodb--how-to-analyze-data-using-the-aggregation-framework

Pour aller plus loin dans l'optimisation du pipeline : https://www.mongodb.com/docs/manual/core/aggregation-pipeline-optimization/

Résolution du problème

Construction et appel au pipeline d'agrégation sur la collection Cosplay :

```
function searchCosplays(req, res) {
  // Si il n'y a pas de paramètre "q" dans la requête, on retourne tous les cosplays
  // car l'onglet "Découvrir" est sélectionné
  if (!req.query.q) return getAllCosplays(req, res);
  const query = req.query.q; // Récupère la valeur du paramètre "q" dans la requête

  Cosplay.aggregate(
    aggregatePopulateCreator({
      $match: {
        $text: {
          $search: query,
          $caseSensitive: false,
          $diacriticSensitive: false,
        },
      },
    })
  ).exec(function (err, data) {
    if (err) {
      console.log(err);
      return res.send(500).json({ msg: "cannot query cosplays" });
    }
    return res.status(200).json(data);
  });
}
```

J'ai créé une fonction utilitaire **aggregatePopulateCreator** pour construire plus facilement les étapes du pipeline et aussi afin factoriser le code, car elle sera utilisée à plusieurs endroits.

Cette fonction accepte un paramètre matchCriteria, qui est un objet utilisé pour effectuer une recherche dans une collection de données.

Voici une explication de chacune des étapes du pipeline :

- **$match:** Cette étape utilise l'opérateur $text pour effectuer une recherche textuelle dans les documents. La propriété query utilisée dans $text: { $search: query } n'est pas définie dans ce code. Il semble qu'il s'agisse d'une variable manquante. Elle devrait être définie auparavant ou passée en argument à la fonction.

- **$lookup**: Cette étape effectue une jointure avec la collection "users". Elle compare la valeur de la clé "creator" dans le document actuel avec la valeur de la clé "_id" dans la collection "users" et renvoie les correspondances trouvées dans un tableau nommé "creator". Cela permet de récupérer les informations sur le créateur lié à chaque document.
- **$unwind:** Cette étape décompose le tableau "creator" créé lors de l'étape précédente et génère un document distinct pour chaque élément du tableau. Cela permet de traiter chaque créateur individuellement plutôt que sous forme de tableau.
- **$project:** Cette étape spécifie les champs à inclure dans le résultat final. Les champs commençant par "creator." sont ceux qui ont été extraits de la collection "users" lors de l'étape de jointure.

En résumé, cette fonction **aggregatePopulateCreator** utilise les fonctionnalités d'agrégation de MongoDB pour effectuer une recherche textuelle sur la collection Cosplay, puis effectue une jointure avec la collection utilisateurs et enfin elle projette les champs à conserver dans le résultat final.

```
/**      Cassandra Forestier, 2 months ago • expand user infos inside cc
 * @param {Object} matchCriteria i.e  { $match: { creator: '123' } } }
 * @returns
 */
const aggregatePopulateCreator = (matchCriteria) => {
  return [
    matchCriteria,
    {
      $lookup: {
        from: "users",
        localField: "creator",
        foreignField: "_id",
        as: "creator",
      },
    },
    {
      $unwind: {
        path: "$creator",
        preserveNullAndEmptyArrays: true,
      },
    },
    {
      $project: {
        _id: 1,
        character: 1,
        universe: 1,
        initialDate: 1,
        dueDate: 1,
        budget: 1,
        category: 1,
        variant: 1,
        elements: 1,
        status: 1,
        pictures: 1,
        tasks: 1,
        "creator._id": 1,
        "creator.username": 1,
      },
    },
  ];
};

module.exports = aggregatePopulateCreator;
```

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const cosplaySchema = new Schema({
  character: String,
  universe: String,
  initialDate: Date,
  dueDate: Date,
  budget: Number,
  category: String,
  variant: String,
  elements: [String],
  //   {
  //     name: String,
  //     price: Number,
  //     link: String,
  //   },
  // ],
  status: {
    type: String,
    enum: ["In progress", "Planned", "Finished"],
    default: "Planned",
  },
  pictures: [String],
  creator: { ref: "User", type: mongoose.Schema.Types.ObjectId },
  public: Boolean,
  tasks: [String],
});

const Cosplay = mongoose.model("Cosplay", cosplaySchema);

// l'index du schema cosplay permet de faire des recherches textuelles sur les champs character et universe
// Par défaut, Mongo créé un index sur l'id, mais pas sur les champs textuels
// la création d'un index permet l'optimisation des requêtes car Mongo peut utiliser l'index pour accélérer les recherches
cosplaySchema.index({ character: "text", universe: "text" });
module.exports = Cosplay;
```

B. Description de l'implémentation

Le cosplay étant au cœur de mon projet, j'ai choisi d'expliquer de manière précise les opérations de CRUD cosplay pour vous montrer la façon dont je l'ai créé. A noter que son implémentation est venue après la création du CRUD de l'utilisateur.

➢ *Première étape : le back-end*

Pour commencer la feature du cosplay, j'ai implémenté le back-end. En effet, j'ai créé mes 3 fichiers : **cosplay.controller.js**, **cosplay.model.js** et **cosplay.route.js**.

cosplay.route.js

Ce fichier définit les routes d'API qui seront appelées par le front-end.

- Les routes sont créées à l'aide du module **express.Router()** qui permet de définir les routes de manière modulaire.
- Les middlewares et les contrôleurs nécessaires sont importés depuis les fichiers correspondants. Cette partie est donc celle qui a le plus évoluée au cours du développement.
- Les routes sont définies en utilisant différents verbes **HTTP (GET, POST, PUT, DELETE)** et les chemins d'URL correspondants.
- Chaque route est associée à un contrôleur qui gère la logique métier de l'action correspondante.
- Les middlewares *checkAuth*, *fileUpload*, et *referencesUpload* sont utilisés pour effectuer des vérifications d'authentification, le téléchargement des images et leur mise à jour.

Voici un résumé des routes :

1. **Création**

POST /cosplay/create : Cette route permet de créer un nouveau cosplay dans la base de données. Elle utilise le middleware checkAuth pour vérifier l'authentification de l'utilisateur et le middleware fileUpload pour gérer le téléchargement de fichiers.

2. **Mise à jour**

PUT /cosplay/:id : Cette route permet de mettre à jour un cosplay existant dans la base de données, identifié par l'id passé en paramètre de la route. Elle utilise les mêmes middlewares vus précédemment.

POST /update/:id : Cette route permet de mettre à jour un cosplay existant avec une ou plusieurs images, identifié par l'id passé en paramètre de la route. Elle utilise les mêmes middlewares vus précédemment.

3. **Lecture**

GET /cosplay : Cette route permet d'obtenir tous les cosplays de la base de données sans filtre. Elle utilise le middleware checkAuth.

GET /cosplays/me : Cette route permet de récupérer tous les cosplays de l'utilisateur authentifié. Elle utilise le middleware checkAuth.

GET /cosplay/:id : Cette route permet d'obtenir les détails d'un cosplay spécifique en utilisant son identifiant. Elle utilise le middleware checkAuth.

4. **Suppression**

DELETE /cosplay/:id : Cette route permet de supprimer un cosplay spécifique en utilisant son identifiant passé en paramètre de la route. Elle utilise le middleware checkAuth, mais aussi s'assure que l'utilisateur qui demande la suppression du cosplay est bien le créateur de celui-ci.

PUT /cosplay/:id/delete-picture : Cette route permet de supprimer une image spécifique associée à un cosplay en utilisant l'identifiant du cosplay et l'identifiant de l'image. Elle utilise le middleware checkAuth.

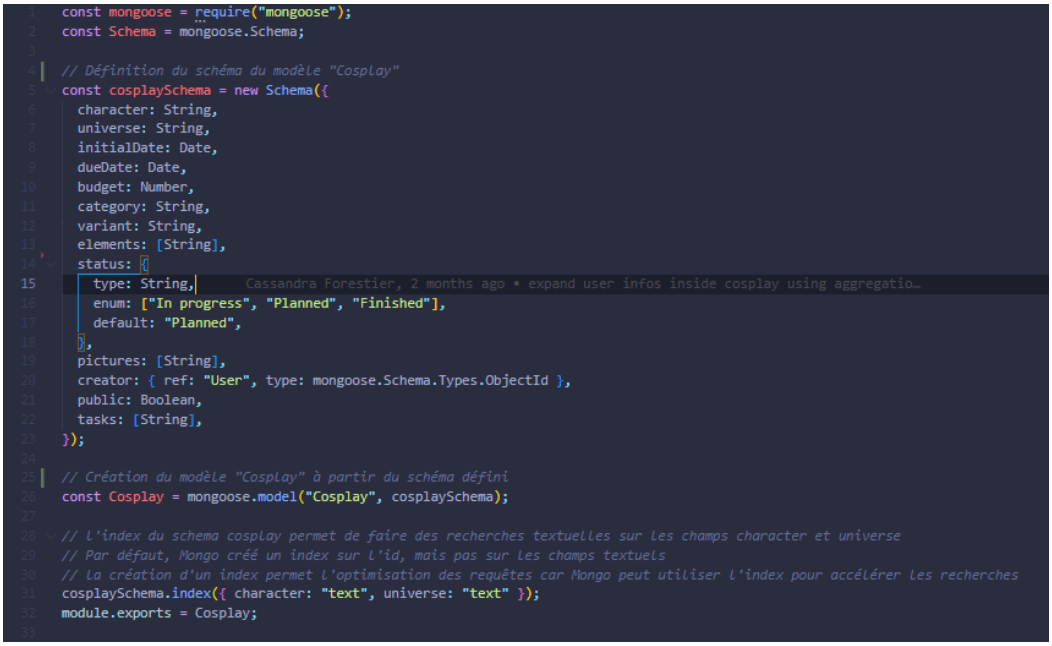
```
// CREATE
// Créer un cosplay dans la BDD
router.post("/cosplay/create", checkAuth, fileUpload, createACosplay);


// UPDATE
// Mettre à jour un cosplay
router.put("/cosplay/:id", checkAuth, fileUpload, updateACosplay);      You, 4 months ago

// Mettre à jour un cosplay avec une ou plusieurs images
router.post("/update/:id", checkAuth, referencesUpload, addImage);


// READ
// Obtenir tous les cosplays sans filtre
router.get("/cosplay", checkAuth, searchCosplays);

// Récupérer tous mes cosplays
router.get("/cosplays/me", checkAuth, getAllMyCosplays);

// Obtenir un cosplay selon son Id
router.get("/cosplay/:id", checkAuth, getCosplayById);


// DELETE
// Supprimer un cosplay selon son Id
router.delete("/cosplay/:id", checkAuth, deleteCosplayById);

// Supprimer une image d'un cosplay
router.put("/cosplay/:id/delete-picture", checkAuth, deleteImage);

module.exports = router;
```

cosplay.model.js :

cosplay.model.js est le fichier qui définit le schéma Mongoose et le modèle "Cosplay" utilisés par mon application pour représenter et stocker les informations liées aux cosplays.

Pour pouvoir créer ce schéma et définir ce qui sera stocké, je me suis basée sur la conception faites en amont. Ainsi j'ai ajouté :

- Le personnage
- L'univers
- Date de début et date de fin
- La catégorie
- Le variant
- Le statut
- Le budget
- Les images
- La référence au créateur du cosplay via le schéma “user”
- Les éléments du costume
- Les tâches à effectuer



```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

// Définition du schéma du modèle "Cosplay"
const cosplaySchema = new Schema({
  character: String,
  universe: String,
  initialDate: Date,
  dueDate: Date,
  budget: Number,
  category: String,
  variant: String,
  elements: [String],
  status: {
    type: String,      Cassandra Forestier, 2 months ago • expand user infos inside cosplay using aggregatio…
    enum: ["In progress", "Planned", "Finished"],
    default: "Planned",
  },
  pictures: [String],
  creator: { ref: "User", type: mongoose.Schema.Types.ObjectId },
  public: Boolean,
  tasks: [String],
});

// Création du modèle "Cosplay" à partir du schéma défini
const Cosplay = mongoose.model("Cosplay", cosplaySchema);

// l'index du schema cosplay permet de faire des recherches textuelles sur les champs character et universe
// Par défaut, Mongo créé un index sur l'id, mais pas sur les champs textuels
// la création d'un index permet l'optimisation des requêtes car Mongo peut utiliser l'index pour accélérer les recherches
cosplaySchema.index({ character: "text", universe: "text" });
module.exports = Cosplay;
```

L'indexation des champs "character" et "universe" est effectuée pour optimiser les recherches textuelles dans la base de données comme vu précédemment dans le cas de recherche. (Mais ce bout de code a été ajouté bien après).

cosplay.controller.js

```
function createACosplay(req, res) {
  if (!req.body["cosplay-data"]) {
    return res.status(400).json({ msg: "La requête n'est pas valide" });
  }
  // Récupération des données nécessaires depuis la requête
  const creatorName = req.user.username;
  const creator = req.user._id;
  const pictures = req.files["image-file"].map((file) => {
    return file.filename;
  });
  // Récupération des données du cosplay depuis le corps de la requête
  const {
    character,
    universe,
    initialDate,
    dueDate,
    budget,
    category,
    elements,
    status,
    variant,
  } = JSON.parse(req.body["cosplay-data"]);
  // Création d'une nouvelle instance du modèle "Cosplay"
  const cosplay = new Cosplay({
    variant,
    character,
    universe,
    initialDate,
    dueDate,
    budget,
    category,
    elements,
    status,
    pictures,
    creator,
    creatorName,
  });
  // Enregistrement du cosplay dans la base de données
  cosplay.save((err, savedCosplay) => {
    if (err) {
      console.error(err);
      res.status(500).json({
        error: "Une erreur est survenue pendant la création du cosplay",
      });
    }
    res.status(201).send(savedCosplay);
  });
}
```

La fonction **createACosplay** est un contrôleur qui gère la création d'un nouveau cosplay dans la base de données.

- La première partie de la fonction vérifie si la propriété **cosplay-data** existe dans le corps de la requête. Si elle n'est pas présente, la fonction renvoie une réponse d'erreur avec un statut **400** (Bad Request).
- Ensuite, la fonction extrait les informations nécessaires de la requête :
  - **creatorName :** le nom de l'utilisateur créateur du cosplay, récupéré à partir de **req.user.username**.
  - **creator :** l'identifiant de l'utilisateur créateur du cosplay, récupéré à partir de **req.user._id.** Cette propriété provient du middleware “checkAuth” qui l’ajoute lorsque l’utilisateur est authentifié.
  - **pictures :** un tableau contenant les noms des fichiers d'images téléchargés pour le cosplay. Les noms des fichiers sont extraits à partir de **req.files["image-file"]**. Cette propriété provient du middleware “fileUpload” qui l’ajoute lorsque des fichiers sont attachés à la requête.
  - Les données du cosplay sont extraites du **corps de la requête** en utilisant **JSON.parse** pour convertir la chaîne de caractères JSON **req.body["cosplay-data"]** en un objet JavaScript. Les différentes propriétés sont extraites individuellement (telles que character, universe, initialDate, etc.).
- Une nouvelle instance du modèle "Cosplay" est créée en utilisant les données extraites et les valeurs récupérées précédemment.

- La méthode **save** est appelée sur l'instance du modèle "Cosplay" pour enregistrer le cosplay dans la base de données. En cas d'erreur, une réponse d'erreur avec un statut **500 (Internal Server Error)** est renvoyée. Sinon, une réponse avec un statut **201 (Created)** est renvoyée, contenant les données du cosplay enregistré.

Les autres contrôleurs sont basés sur la même logique. Je me permets donc de vous laisser voir en annexes quelques contrôleurs supplémentaires. Voir les annexes

Middleware checkAuth

Comme vous avez pu le voir ci-dessus, j'utilise sur chaque route du cosplay le middleware "checkAuth".

Voici le code :

```
async function checkAuth(req, res, next) {
  try {
    const { cookies, headers } = req;
    /* On vérifie que le JWT est présent dans les cookies de la requête */
    if (!cookies || !cookies.access_token) {
      return res.status(401).json({ message: "Missing token in cookie" });
    }

    const accessToken = cookies.access_token;

    /* On vérifie que le token CSRF est présent dans les en-têtes de la requête */
    if (!headers || !headers["x-xsrf-token"]) {
      return res.status(401).json({ message: "Missing XSRF token in headers" });
    }

    const xsrfToken = headers["x-xsrf-token"];

    /* On vérifie et décode le JWT à l'aide du secret et de l'algorithme utilisé pour le générer */
    const decodedToken = jwt.verify(accessToken, secret, {
      algorithms: algorithm,
    });

    /* On vérifie que le token CSRF correspond à celui présent dans le JWT  */
    if (xsrfToken !== decodedToken.xsrfToken) {
      return res.status(401).json({ message: "Bad xsrf token" });
    }

    /* On vérifie que l'utilisateur existe bien dans notre base de données */
    const userId = decodedToken.sub;
    const user = await User.findOne({ _id: userId });

    if (!user) {
      return res.status(401).json({ message: `User ${userId} not exists` });
    }

    /* On passe l'utilisateur dans notre requête afin que celui-ci soit disponible pour les prochains middlewares */
    req.user = user;

    /* On appelle le prochain middleware */
    return next();
  } catch (err) {
    console.error(err);
    return res.status(500).json({ message: "Internal error" });
  }
}
```

Ce middleware **checkAuth** vérifie l'authentification de l'utilisateur en vérifiant la présence d'un JWT (JSON Web Token) dans les cookies de la requête, le token CSRF dans les en-têtes, et la correspondance entre le token CSRF et celui présent dans le JWT décodé. Il vérifie également si l'utilisateur existe dans la base de données et ajoute l'utilisateur à la requête pour les prochains middlewares. Voici les étapes :

1. Vérifie la présence du JWT et du token CSRF dans les cookies et les en-têtes de la requête.
2. Décode le JWT à l'aide du secret et de l'algorithme spécifiés.
3. Vérifie la correspondance entre le token CSRF dans les en-têtes et celui présent dans le JWT décodé.
4. Recherche et récupère l'utilisateur correspondant à l'identifiant du JWT dans la base de données.
5. Ajoute l'utilisateur à la requête (req.user).
6. Appelle le prochain middleware.
7. En cas d'erreur, renvoie une réponse d'erreur avec un statut 500 (Internal Server Error).

Ce middleware permet donc de vérifier l'authentification de l'utilisateur avant de lui permettre d'accéder aux ressources protégées de l'application.

Middlewares fileUpload, avatarUpload et referencesUpload

Il était également intéressant de vous parler du middleware de téléchargement des fichiers.

Voici le code :

```
const multer = require("multer");
const path = require("path");

const storage = multer.diskStorage({
  // Permet de donner la destination des fichiers
  destination: (req, file, callback) => {
    callback(null, path.join(__dirname, "../uploads/"));
  },

  // Permet de formatter le nom de fichier
  filename: (req, file, callback) => {
    callback(
      null,
      file.fieldname + "-" + Date.now() + path.extname(file.originalname)
    );
  },
  // Permet de filtrer les fichiers pour n'accepter que les photos
  filefilter: (req, file, callback) => {
    const ext = path.extname(file.originalname);
    if (ext !== ".png" && ext !== ".jpeg" && ext !== ".jpg") {
      return callback(new Error("Seulement les images sont autorisées."));
    }
    callback(null, true);
  },
});

const upload = multer({ storage: storage });
const fileUpload = upload.fields([
  {
    name: "image-file",
    maxCount: 1,
  },
]);
const avatarUpload = upload.fields([
  {
    name: "avatar",
    maxCount: 1,
  },
]);
const referencesUpload = upload.fields([
  {
    name: "pictures",
    maxCount: 1,
  },
]);

module.exports = { fileUpload, avatarUpload, referencesUpload };
```

Ce middleware utilise le module **multer** pour gérer le téléchargement de fichiers. Il configure le stockage des fichiers téléchargés, le formatage des noms de fichiers et le filtrage des types de fichiers acceptés.

Voici les étapes :

1. Configure le stockage des fichiers téléchargés en spécifiant la destination et le formatage des noms de fichiers.
2. Filtre les fichiers téléchargés pour n'accepter que les images au format PNG, JPEG et JPG.
3. Crée une instance de multer avec la configuration de stockage spécifiée. Les fichiers téléchargés sont déposés dans le dossier “upload” à la racine du projet.
4. Crée des middlewares pour gérer le téléchargement de fichiers dans des champs de formulaire spécifiques.
5. Exporte les middlewares pour les rendre disponibles dans d'autres parties de l'application.

Toutes ces parties ont été testées via Postman pour vérifier leur bon fonctionnement. Une fois cette étape validée, je suis passée sur la partie front-end.

➢ *Seconde étape : le front-end*

a. React Router Dom

La première chose qui a été mise en place est le React Router DOM. Cette bibliothèque me permet de faire la gestion des routes dans mon application. Elle permet de créer des routes et de naviguer entre les différentes pages qui seront implémentées en utilisant des composants React.

On peut voir que le composant App contient un **ConfigProvider, i**l repose sur le Context API de react pour rendre ce thème disponible à tous les composants enfant de ConfigProvider.
Ce composant est fourni par Ant Design pour définir notamment la localisation globale du site, le thème personnalisé pour le site en entier et même un message de validation globale à tous les formulaires du site via la prop “form”.

La première route correspond à l'accueil de l'application et utilise le composant **Layout** pour l'afficher.

Chaque route est composée d’un path et d’un composant à afficher lorsque l’URL visitée correspond à ce chemin.

On peut d’ailleurs remarquer que, dans certaines routes, j’ai des chemins (path) avec des paramètres dynamiques comme dans **“cosplayDetails/:id/view”**. Cet id pourra être récupéré dans le composant correspondant.

La dernière route est une route spéciale. En effet, il s’agit d’une route par défaut. Elle est définie avec un chemin “*”, qui fait correspondre toutes les URL qui n'ont pas été capturées par d'autres routes définies précédemment.

Cela permet une redirection vers le composant Home si l’URL est inconnue dans le React Router.

```
return (
  <ConfigProvider locale={frFR} theme={themeCustomValues} form={{ validateMessages }}>
    <Routes>
      <Route path="/" element={<Layout isLogged={isLogged} />}>
        <Route index element={<Home />} />
        <Route path="cosplay" element={<CreateCosplay />} />
        <Route
          path="showCosplays"
          element={<ShowAllCosplay onlyMyCosplays={true} />}
        />
        <Route path="searchResults" element={<SearchCosplay />} />
        <Route
          path="allCosplays"
          element={<ShowAllCosplay onlyMyCosplays={false} />}
        />
        <Route path="profil" element={<User />} />
        <Route path="login" element={<Login />} />
        <Route path="signup" element={<Signup />} />
        <Route path="cosplayDetails/:id/edit" element={<CosplayDetails />} />
        <Route
          path="cosplayDetails/:id/view"
          element={<CosplayDetailsView />}
        />
        <Route path="forgotPassword" element={<ForgotPassword />} />
        <Route path="passwordReset/:token" element={<PasswordReset />} />
        <Route
          path="confirmAccount/:token"
          element={<UserConfirmAccount />}
        />
        <Route path="*" element={<Home />} />
      </Route>
    </Routes>
  </ConfigProvider>
);
```

Ensuite, le composant App ci-dessus est injecté dans le **BrowserRouter** : C'est un composant qui enveloppe toute l'application et fournit le contexte nécessaire pour gérer la navigation basée sur l'URL. Il utilise l'API de l'objet **history** du navigateur pour synchroniser l'URL avec les routes définies.

```
const root = ReactDOM.createRoot(
  document.getElementById("root") as HTMLElement
);
root.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
);
```

b. Le layout

Le layout (mise en page) de Cosplay Maker correspond l'organisation visuelle et structurelle. Il s'agit de la disposition des différents éléments de l'interface utilisateur tels que la barre de navigation, le pied de page et les zones de contenu. Ce layout sera donc chargé lors de la visite par défaut de l'utilisateur (le path "/"). Elle chargera donc les composants par défaut : La barre du menu publique ou privée, l'Outlet, puis le pied de page. J'ai donc rendu le pieds de page et le menu globale à toutes les pages qui seront chargées quel que soit l'URL visitée.

c. Outlet

Outlet est un composant fournit par React Router DOM. C'est un composant spécial qui sert de point d'entrée pour le rendu des composants enfants en fonction des routes empruntées par l'utilisateur. Il permet de spécifier l'emplacement où les composants spécifiques à chaque route seront rendus dans le layout.

d. Exemple de la route de création d'un cosplay

Chaque page sera donc un composant fonctionnel de React qui possède une logique qui lui est propre.

Ici, la page de création de cosplay s'affiche lorsque la route "/cosplay" est visitée.

e. Le fichier cosplay.service.ts

J'ai centralisé les requêtes vers l'API liées au cosplay dans ce fichier. (De même pour user.service.ts pour tout ce qui est lié aux utilisateurs).

J'ai créé une instance personnalisée d'Axios qui possède un intercepteur afin de gérer l'expiration du token d'accès et son rafraîchissement. Je vous invite à aller voir le diagramme d'activité concernant ce point dans les annexes.

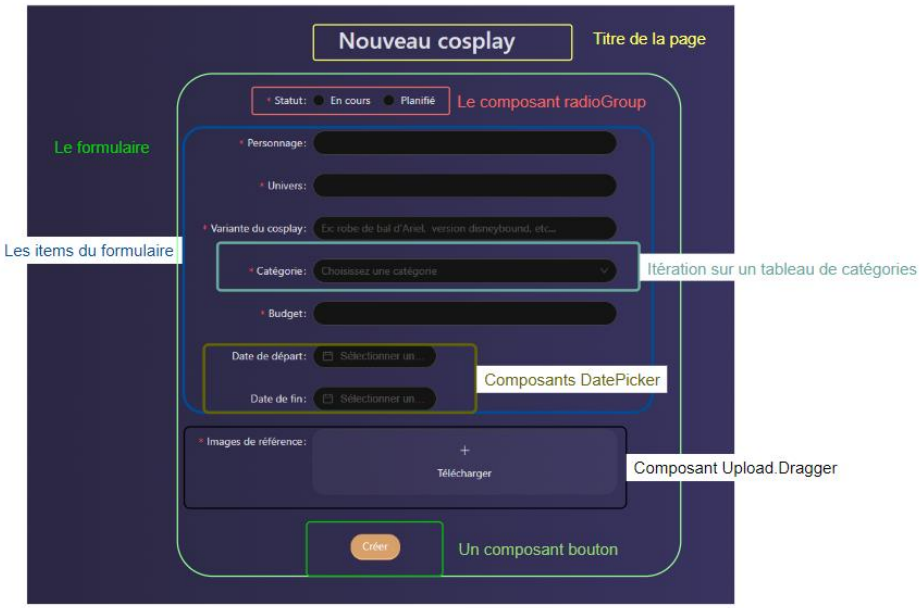
```
export class CosplayService {
  async createCosplay(formData: FormData) {
    await axiosApiInstance.post(`${DOMAIN_URL}/api/cosplay/create`, formData, {
      withCredentials: true,
      headers: prepareHeaders(true),
    });
  }
```

Les services sont donc des classes qui contiennent des méthodes asynchrones.

Les services, de manière générale, me permettent d'avoir un code modulaire et d'éviter la duplication de code.

Voici l'appel de la classe **cosplayService** dans le composant **createCosplay**. On l'instancie donc une fois dans le composant, ce qui permet de ne pas utiliser de la mémoire inutilement.

```tsx
const cosplayService = new CosplayService();

const CreateCosplay: React.FC = () => {
  const navigate = useNavigate();
  const [form] = Form.useForm();

  const onFinish = async (values: CreateCosplayFormValues) => {
    // FormData est un autre type de format qui permet d'envoyer notamment du binaire
    const formData = new FormData();

    // append permet d'ajouter la propriété image-file à nos datas pour récupérer l'image
    formData.append("image-file", values.pictures.file);
    const { pictures, ...cosplayData } = values;

    // append ici permet d'ajouter en plus la propriété cosplay-data pour récupérer les autres données
    formData.append("cosplay-data", JSON.stringify(cosplayData));

    await cosplayService.createCosplay(formData);
    navigate("/showCosplays");
  };
```



Voici ci-dessus une analyse visuelle des composants présents sur la page de création d'un cosplay. La plupart des composants sont des composants mis à disposition par Ant Design.

Déroulement de l'action de création du cosplay :

Une fois que l'utilisateur a rempli le formulaire, j'appelle **cosplayservice.createcosplay** afin d'envoyer les données au back, en cas de succès l'utilisateur est redirigé vers la page "mes cosplays" au moyen de la fonction **navigate** de React Router.

Pour certains champs du formulaire j'ai ajouté la règle Rules *required : true*, pour rendre la saisie de ces champs obligatoire et afficher un message de validation en cas de non-respect de la règle.

```tsx
<Form.Item name="budget" label="Budget" rules={[{ required: true }]}>
  <Input type="number" />
</Form.Item>
```

Une fois le formulaire remplit, les données sont envoyées via un objet **formData** au back-end.

f. L'instance Personnalisée d'Axios

L'intercepteur est une fonctionnalité puissante qui permet de traiter les réponses des requêtes avant qu'elles ne soient traitées par l'application. Ici, cela me permet de gérer les erreurs de réponse, en particulier les erreurs HTTP avec un code de statut 401, où l'utilisateur n'est pas autorisé à accéder à la ressource demandée.

1. L'intercepteur tente de rafraîchir le jeton d'accès de l'utilisateur via UserService.refreshToken().
2. Les nouveaux jetons d'accès sont stockés dans le cookie, et l'en-tête "x-xsrf-token" est mis à jour pour les futures requêtes.
3. La requête d'origine est relancée avec les nouvelles informations d'authentification pour éviter l'accès non autorisé.
4. Si le rafraîchissement échoue ou si l'erreur n'est pas liée à l'authentification, l'erreur est rejetée pour un traitement approprié.

Cette approche améliore l'expérience utilisateur en maintenant l'authentification sans nouvelles connexions manuelles après l'expiration du jeton. Elle permet également une meilleure sécurité en réduisant la fenêtre d'exposition du jeton d'accès.

De plus, il s'agit d'un site où les utilisateurs peuvent être connectés durant des heures (car par exemple, il fait ses recherches sur le côté, ainsi on a une prévention contre la perte de données en cours de saisies).

```
Cassandra Forestier, 3 months ago | 1 author (Cassandra Forestier)
import axios from "axios";
import { UserService } from "./services/user.service";

const axiosApiInstance = axios.create();

axiosApiInstance.interceptors.response.use(
  function (response) {
    return response;
  },
  async function (error) {
    const originalRequest = error.config;
    if (error.response.status === 401 && !originalRequest._retry) {
      originalRequest._retry = true;
      try {
        const res = await UserService.refreshToken();
        const xsrfToken = res.xsrfToken;
        const expireDate = new Date();
        expireDate.setMilliseconds(
          expireDate.getMilliseconds() + res.accessTokenExpiresIn
        );
        localStorage.setItem("accessTokenExpiresIn", expireDate.toString());
        localStorage.setItem("xsrfToken", xsrfToken);
        originalRequest.headers["x-xsrf-token"] = xsrfToken;
      } catch (err) {
        window.location.href = "/login?disconnected=true";
        return Promise.reject(err);
      }
      return axiosApiInstance(originalRequest);
    }
    return Promise.reject(error);
  }
);

export default axiosApiInstance;
```

g. Exemple d'un composant réutilisable

Voici un exemple de composant réutilisable. Je l'ai mis en place après m'être rendue compte que je dupliquais beaucoup de codes dans les tableaux d'édition du cosplay, mais aussi dans le formulaire de mise à jour des informations générales de l'utilisateur.

Ainsi, le même composant peut être utilisé à plusieurs endroits de l'application sans avoir à créer des versions distinctes pour chaque page. Cela améliore la maintenabilité du code, car les mises à jour et les corrections apportées à ce composant se répercutent automatiquement sur toutes les pages où il est utilisé.

La modularité du code est également mise en avant. Il est autonome et je pourrais à l'avenir le mettre dans d'autres parties de l'application sans affecter son fonctionnement global.

J'ai ainsi pu voir comment bien profiter d'un principe fondamental de React.

```
export const EditableField = (props: EditableFieldProps): JSX.Element => {
  const {
    type,
    placeholder,
    defaultValue,
    onSubmit,
    name,
    validationMessage,
    isRequired = true,
  } = props;

  const [isEditing, setIsEditing] = useState(false);

  return isEditing ? (
    <Form
      initialValues={{ [name]: defaultValue }}      Cassandra Forestier, 2 months
      onFinish={(values) => {
        onSubmit(name, values[name]);
        setIsEditing(false);
      }}
      layout="inline"
    >
      <Form.Item
        rules={[{ required: isRequired, message: validationMessage }]}
        name={name}
      >
        <Input
          placeholder={placeholder}
          defaultValue={defaultValue}
          autoFocus
        />
      </Form.Item>
      <Form.Item>
        <Button htmlType="submit">Enregistrer</Button>
        <Button onClick={() => setIsEditing(false)}>Annuler</Button>
      </Form.Item>
    </Form>
  ) : (
    <>
      <div
        style={{ cursor: "pointer", display: "flex", justifyContent: "center" }}
        onClick={() => setIsEditing(true)}
      >
        <Text>{defaultValue}</Text>
        <PencilSvg />
```

h. Un composant "gardien"

Il s'agit d'un composant React appelé **ProtectedRoute**, qui est utilisé pour gérer les routes et les protéger. Le composant prend deux propriétés en entrée : **isLogged**, un booléen indiquant si l'utilisateur est connecté ou non, et **children**, qui est un élément JSX représentant le contenu à afficher lorsque l'utilisateur est authentifié.

Le rôle principal de ce composant est de protéger certaines pages de l'application en vérifiant si l'utilisateur est authentifié. Si l'utilisateur n'est pas connecté, le composant redirige automatiquement vers la page de connexion en utilisant **Navigate** de React Router, en remplaçant l'entrée d'historique pour empêcher l'utilisateur de revenir en arrière après la connexion.

En cas de connexion valide, le composant renvoie simplement le contenu spécifié par la propriété **children**, ce qui permet à l'utilisateur d'accéder à la page protégée.

```tsx
interface ProtectedRouteProps {
  isLogged: boolean;
  children: JSX.Element;
}

const ProtectedRoute = ({ isLogged, children }: ProtectedRouteProps) => {

  if (!isLogged) {
    return <Navigate to="/login" replace />;
  }

  return children;
};

export default ProtectedRoute;
```

C. Tests et validation

1. Tests

Concernant les tests sur le projet, je les ai fait après avoir implémenter les actions des CRUD dans le back-end.

J'ai fait le choix de Jest pour implémenter les tests que ce soit pour le front-end ou le back-end.

Jest est particulièrement adapté pour les tests React (support des composants React, simulateurs d'événements et les outils de rendu virtuel). De plus, je peux également l'utiliser pour mes tests back-end, ce qui a facilité grandement le développement des tests unitaires et d'intégration du site de manière globale.

Pour le moment, les tests sont sur le back-end.

- La première chose importante est donc la configuration dans le fichier **jest.config.js**

```js
module.exports = {
  collectCoverage: true,
  coverageDirectory: "coverage",
  coverageProvider: "v8",
  testEnvironment: "node",
  testPathIgnorePatterns: ["__tests__/config/*", "__tests__/test-utils.js"],
  collectCoverageFrom: ["app.js", "features/**/*.js", "middlewares/**/*.js"],
};
```

Ce code configure les options de couverture de test avec Jest, y compris la collecte des statistiques de couverture, le répertoire de couverture, le fournisseur de couverture, l'environnement de test, les fichiers à ignorer lors des tests et les fichiers à inclure dans la collecte de couverture.

- Ensuite, j'ai également dû faire la configuration du fichier **database.js.** Ce fichier permet la configuration des fonctions pour établir une connexion à une base de données MongoDB en mémoire pour les tests, fermer la connexion et effacer les données de la base de données entre les tests.

```
const mongoose = require("mongoose");
const { MongoMemoryServer } = require("mongodb-memory-server");

let mongoServer;

const connect = async () => {
  mongoServer = await MongoMemoryServer.create();
  await mongoose.connect(mongoServer.getUri());
};

const close = async () => {
  await mongoose.connection.dropDatabase();
  await mongoose.connection.close();
  await mongoServer.stop();
};

const clear = async () => {
  const collections = mongoose.connection.collections;
  for (const key in collections) {
    await collections[key].deleteMany({});
  }
};
module.exports = { connect, close, clear };
```

- La dernière configuration qui est survenue après l'implémentation des tests, m'a permis de supprimer de la duplication de code. Il s'agit donc du fichier **test-utils.js.**

```
module.exports = async function login(agent) {
  await agent
    .post("/api/signup")
    .send({
      city: "Paris",
      username: "test",
      email: "test@test.fr",
      password: "testtest",
      birthdayDate: "2020-01-01T00:00:00Z",
    })
    .set("Accept", "application/json");
  const res = await agent
    .post("/api/login")
    .send({
      email: "test@test.fr",
      password: "testtest",
    })
    .set("Accept", "application/json");
  const cookies = res.header["set-cookie"];
  const xsrfToken = res.body.xsrfToken;
  const user = await agent
    .get("/api/me")
    .set("Accept", "application/json")
    .set("Cookie", cookies)
    .set("x-xsrf-token", xsrfToken);
  const uid = user.body._id;

  return {
    uid,
    cookies,
    xsrfToken,
  };
};
```

Je définis une fonction asynchrone "login" qui effectue les étapes nécessaires pour s'inscrire en tant qu'utilisateur, se connecter, récupérer les cookies et le token XSRF, puis renvoie ces informations pour les utiliser ultérieurement dans les tests. La fonction utilise l'objet "agent" qui est une instance de **supertest**, un module permettant de tester les API HTTP, pour effectuer des requêtes POST et GET aux endpoints /api/signup, /api/login et /api/me afin de réaliser les différentes étapes du processus de connexion et récupérer les informations nécessaires.

- Voici un cas de test : la création d'un cosplay

Tout d'abord, je fais appel à mes configurations :

```
const request = require("supertest");
const app = require("../app");
const db = require("./config/database");
const login = require("./test-utils");
const agent = request.agent(app);
let cookies = [];
let xsrfToken = "";
let uid = "";

beforeAll(async () => {
  // connexion à la base de données
  await db.connect();
  // création d'un utilisateur + connexion pou
  const data = await login(agent);
  cookies = data.cookies;
  xsrfToken = data.xsrfToken;
  uid = data.uid;
});

afterAll(async () => {
  await db.clear();
  await db.close();
});
```

Ce code configure les HOOKS[14] *beforeAll* et *afterAll* pour établir une connexion à la base de données, effectuer une création d'utilisateur et une connexion pour récupérer les cookies de session et le jeton d'authentification XSRF, puis nettoyer la base de données après les tests.

Ce code est un ensemble de tests unitaires pour la route **POST "/api/cosplay/create"**. Ces tests vérifient différents scénarios :

[14] **Un hook**, fait référence à une fonctionnalité fournie par les frameworks ou les bibliothèques qui permet d'exécuter du code à des moments spécifiques lors du cycle de vie d'une application ou d'un test.

```
// Test de la route POST /api/cosplay/create
describe("create a cosplay", () => {
  describe("Without authentication token", () => {
    it("should return 401", async () => {
      const res = await agent
        .post("/api/cosplay/create")
        .send({})
        .set("Accept", "application/json");
      expect(res.status).toEqual(401);
      expect(res.body).toEqual({ message: "Missing token in cookie" });
    });
  });
```

- Le premier scénario teste l'appel à la route sans jeton d'authentification. Il s'attend à recevoir une réponse avec le statut **401 (non autorisé)** et un message indiquant "Missing token in cookie".

```
  describe("With authentication token but no cosplay data", () => {
    it("should return 400", async () => {
      const res = await agent
        .post("/api/cosplay/create")
        .send({})
        .set("Accept", "application/json")
        .set("Cookie", cookies)
        .set("x-xsrf-token", xsrfToken);
      expect(res.status).toEqual(400);
      expect(res.body).toEqual({ msg: "La requête n'est pas valide" });
    });
  });
```

- Le deuxième scénario teste l'appel à la route avec un jeton d'authentification valide mais sans données de cosplay. Il s'attend à recevoir une réponse avec le statut **400 (mauvaise requête)** et un message indiquant "La requête n'est pas valide".

```
describe("With valid cosplay and token", () => {
  it("should return 201", async () => {
    const cosplay = {
      variant: "test",
      character: "test",
      universe: "test",
      initialDate: "1111-11-10T23:50:39.000Z",
      dueDate: "1111-11-10T23:50:39.000Z",
      budget: 10,
      category: "test",
      elements: [],
      tasks: [],
      status: "Planned",
    };
    const res = await agent
      .post("/api/cosplay/create")
      .type("form")
      .field("cosplay-data", JSON.stringify(cosplay))
      .attach("image-file", __dirname + "/assets/Cosplay.png")
      .set("Accept", "form-data")
      .set("Cookie", cookies)
      .set("x-xsrf-token", xsrfToken);
    expect(res.status).toEqual(201);
    expect(res.body).toEqual({
      __v: 0,
      _id: expect.any(String),
      ...cosplay,
      pictures: [expect.any(String)],
      creator: expect.any(String),
    });
  });
});
```

- Le troisième scénario teste l'appel à la route avec des données de cosplay valides et un jeton d'authentification valide. Il envoie une demande avec les détails du cosplay, une image en pièce jointe et les en-têtes appropriés. Il s'attend à recevoir une réponse avec le statut **201 (créé avec succès)** et un corps de réponse contenant les détails du cosplay créé, y compris un identifiant généré, les détails du cosplay lui-même, une liste d'images (dans ce cas, une seule image) et l'identifiant du créateur.

Pour lancer les tests, on utilise “npm run test” ce qui nous donne la couverture de notre back-end :

File	% Stmts	% Branch	% Funcs	% Lines
All files	59.4	59.78	50	59.4
back	77.61	16.66	100	77.61
app.js	77.61	16.66	100	77.61
back/features/auth	100	100	100	100
refreshToken.model.js	100	100	100	100
back/features/cosplay	67.02	70.37	70	67.02
cosplay.controller.js	51.19	69.23	66.66	51.19
cosplay.model.js	100	100	100	100
cosplay.query-helper.js	100	100	100	100
cosplay.route.js	100	100	100	100
back/features/event	50.35	100	0	50.35
event.controller.js	16.66	100	0	16.66
event.model.js	100	100	100	100
event.route.js	100	100	100	100
back/features/user	60.19	53.84	50	60.19
user.controller.js	51.83	53.57	46.15	51.83
user.model.js	87.69	54.54	100	87.69
user.route.js	100	100	100	100
back/middlewares	45.92	70	55.55	45.92
auth.js	38.02	60	33.33	38.02
sgMail.js	42.3	33.33	50	42.3
validator.js	74	100	75	74

On remarque ici que la moyenne de couverture des tests du back-end s'élève à un peu plus de 57%. L'objectif à terme sera de tendre vers les 80%.

2. *Validation*

La validation permet de vérifier et valider des données entrées par les utilisateurs dans un formulaire ou une interface web. Ainsi, je m'assure que les données respectent les critères de format, de type et de cohérence attendus. Pour cela, j'ai choisi Express validator .

Cette bibliothèque offre différentes fonctionnalités, notamment la validation des champs obligatoires, la vérification du format des données (comme les adresses email, les URLs, les numéros de téléphone, etc.), la vérification des longueurs minimales et maximales, la validation des nombres etc.

```
const { body, validationResult } = require("express-validator");
function userValidationRules(req, res, next) {
  return validate([
    body("city").isString(),
    body("username").isString(),
    body("email").isEmail(),
    body("password").isLength({ min: 8 }),
    body("birthdayDate").isISO8601().notEmpty(),
  ])(req, res, next);
}

function logginValidationRules(req, res, next) {
  return validate([
    body("email").isEmail(),
    body("password").isLength({ min: 8 }),
  ])(req, res, next);
}

const validate = (validations) => {
  return async (req, res, next) => {
    await Promise.all(validations.map((validation) => validation.run(req)));

    const errors = validationResult(req);
    if (errors.isEmpty()) {
      return next();
    }

    res.status(400).json({ errors: errors.array() });
  };
};

module.exports = {
  userValidationRules,
  validate,
};
```

La fonction **userValidationRules** définit les règles de validation pour les données utilisateur telles que la ville, le nom d'utilisateur, l'e-mail, le mot de passe et la date de naissance. Ces règles vérifient que les champs sont de type chaîne, que l'e-mail est valide, que le mot de passe a une longueur minimale de 8 caractères, et que la date de naissance est une date au format ISO8601 non vide.

La fonction **logginValidationRules** définit les règles de validation pour la connexion, où seuls l'e-mail et le mot de passe sont vérifiés.

La fonction **validate** est un middleware qui exécute les validations définies pour une route spécifique. Elle utilise la méthode run de chaque validation pour exécuter la validation sur les données de la requête. Ensuite, elle vérifie s'il y a des erreurs de validation à l'aide de **validationResult**. Si aucune erreur n'est trouvée, la fonction **next**() est appelée pour passer à l'étape suivante du traitement de la requête. Sinon, une réponse JSON avec les erreurs de validation est renvoyée avec un code de **statut 400**.

Ainsi j'ajoute sur les routes concernées les middlewares de validation.

```
/* PUBLIC ROUTES */
router.post("/signup", userValidationRules, signupUser, sgMail);
router.post("/login", logginValidationRules, loginUser);
```

D. Sécurité

Pour la sécurité, je me suis essentiellement basée sur la méthode STRIDE qui permet d'analyser et évaluer les menaces potentielles qui pourraient affecter un système ou une application. Elle permet d'identifier les différentes catégories de menaces et de prendre des mesures de sécurité appropriées pour les contrer. On catégorise donc avec cette méthode 6 menaces.

1. Usurpation d'identité (Spoofing) :

L'usurpation d'identité est le fait de s'authentifier à la place d'une personne et d'accéder "légitimement" à ses autorisations et privilèges.

Pour cette catégorie, j'ai mis plusieurs choses en place :

- Un middleware **"checkAuth"** permettant de vérifier sur chaque route du back-end si l'utilisateur qui demande les informations est bien la bonne personne.
- Utilisation d'un jeton X-CSRF, celui-ci est renvoyé au moyen d'un cookie qui est paramétré en HTTP Only. Il n'est donc pas accessible via du code JavaScript.

2. Falsification (Tampering) :

Le sabotage ou modification de données, les attaques XSS sont indirectement des exemples de modifications non autorisées. L'idée générale est donc de filtrer les entrées utilisateurs.

Pour cette catégorie, j'ai mis plusieurs choses en place :

- Utilisation de la librairie **React** qui échappe automatiquement les caractères spéciaux.
- Non utilisation de "**dangerouslySetInnerHTML**" qui ouvrirait des failles XSS non gérés par React
- Non utilisation de **a.href** dynamique. Ce qui limite les remplacements non maîtrisés dans le href et ainsi n'ouvre pas la possibilité de modifier des données.

3. Refus d'assumer (Répudiation) :

Pour cette partie-là, aucune sécurité n'a été mise en place. En effet, je n'ai aucune action/transaction que l'utilisateur pourrait refuser d'assumer. Il n'y a pas d'utilisation de transaction bancaire, ou d'envoi de messages à des tiers. Cette faille est donc relativement restreinte sur mon application.

Le middleware "checkAuth" permet notamment de vérifier sur chaque route l'authentification correcte de l'utilisateur.

À termes, je mettrai en place une archive des requêtes dites "sensibles" (notamment pour les flux d'images) lors de la création d'un cosplay, la modification d'un cosplay, l'ajout d'un avatar sur le compte et également sa modification dans la page des informations de l'utilisateur. Cela permettra de protéger les utilisateurs des actions malveillantes. Et ainsi, pouvoir administrer les profils problématiques sur le site.

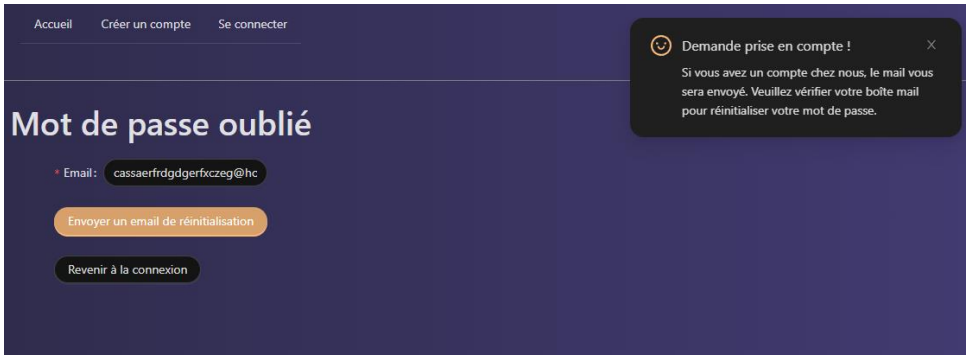
4. Divulgation d'informations (Information disclosure) :

Accès non autorisé à des informations sensibles, ce qui peut inclure des informations sur le système ou la fuite d'informations sur des utilisateurs.

J'ai veillé à ne pas divulguer d'informations dans les erreurs que l'on transmet à l'utilisateur, mais aussi dans certaines notifications.

Par exemple :

- L'envoi du mail pour le mot de passe oublié génère une notification :



On ne donne ici aucune information qui permet de savoir si le mail existe ou non dans la base de données.

- Lors de la connexion, lors de l'erreur dans les champs de données :

L'erreur est générique, nous ne savons donc pas s'il s'agit du mail et / ou du mot de passe qui n'est pas bon. Cela permet d'éviter le "brute-force[15]".

5. Déni de service (Denial of service) :

L'envoi massif de requêtes (attaque DDOS[16]) au système ou à l'application qui ne supporte pas la montée en charge est une attaque courante. Pour cela, avec la mise en place de **Scalingo** (plateforme de déploiement d'applications) qui

[15] Brute-force est un type d'attaque qui permet de se connecter en testant un à un les mails / les mots de passe sur un site jusqu'à ce que la combinaison soit bonne.

[16] **DDOS : Distributed Denial-of-Service Attack :** vise à rendre inaccessible un serveur afin de provoquer une panne ou un fonctionnement fortement dégradé du service.

permet de gérer les pics de trafic. Mais également **CloudFlare** (service de sécurité) qui protège de ce genre d'attaques grâce au réseau de distribution de contenu (CDN[17]) et à son filtrage de trafic.

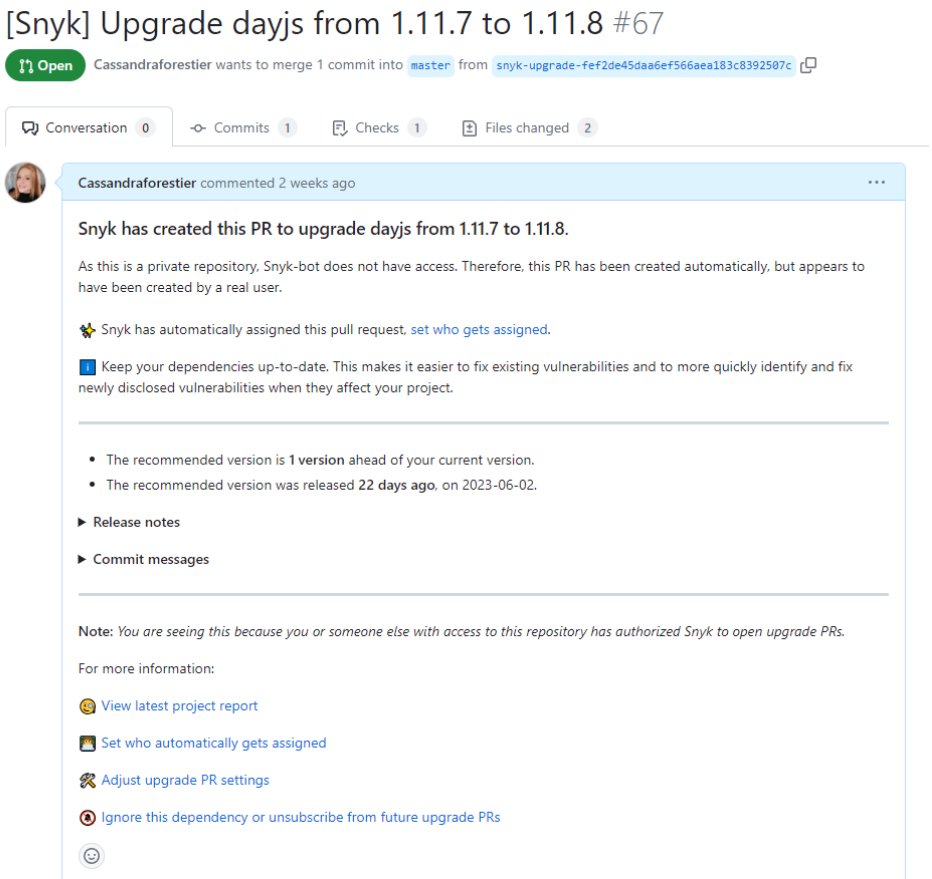
6. **Elévation des privilèges (Elevation of privileges) :**

Il s'agit d'obtenir des privilèges supérieurs à ceux qui était initialement prévu par l'application. Pour ce point-là, une validation des données entrantes provenant des utilisateurs a été réaliser. En effet, je valide que chaque utilisateur ne puisse manipuler que les données qui lui appartient. Cela est donc mis en place sur les routes back via un middleware. Dans le futurs, lorsque les rôles seront implémentés (administrateur, organisateur d'événement, etc...), je m'assurerai du bon cloisonnement des actions par rôle utilisateur.

Utilisation de Snyk

Snyk est un outil permettant d'analyser la sécurité des dépendances dans les projets. Il permet de détecter les vulnérabilités connues dans les bibliothèques tierces utilisées dans un projet. S'il détecte qu'une des bibliothèques tierces a été mise à jour, il créé automatiquement un pull request sur GitHub avec toutes les informations nécessaires.

Voici un exemple :



Cela permet donc une analyse automatisée de la composition des logiciels tiers sur le projet et cela me permet donc de mettre en place le SCA[18].

[17] CDN : Content Delivery Network en anglais : est une infrastructure de serveurs située à différents emplacements géographiques à travers le monde. L'objectif principal d'un CDN est d'optimiser la livraison de contenu sur Internet en rapprochant les ressources des utilisateurs finaux.

[18] SCA (Software Composition Analysis) : L'analyse des compositions logicielles est une méthode qui vise à identifier et à évaluer les vulnérabilités présentes dans les composants logiciels tiers utilisés dans une application. Cela inclut l'identification des dépendances, l' évaluation des licences, et la détection des vulnérabilités connues dans les bibliothèques et les frameworks utilisés.

Snyk fonctionne en intégrant le processus d'analyse de sécurité directement dans le pipeline de développement. Je l'ai donc intégré sur mon répertoire GitHub. Snyk analyse donc continuellement la sécurité, notamment lorsqu'une mise à jour du code est faite.

Une fois cette analyse terminée, Snyk génère un rapport détaillé sur les vulnérabilités détectées y compris des informations sur les risques associés, les recommandations de correctif et les mesures à prendre pour atténuer les vulnérabilités. Il s'agit donc ici de la technique d'analyse SAST [19] mise en place.

Voici un exemple :

Partie V : Conclusion

A. Bilan du projet

Le projet "Cosplay-Maker" a été une expérience enrichissante et stimulante. Malgré les contraintes d'un projet réalisé dans un cadre scolaire (travail le soir/les week-ends) et seule, j'ai réussi à concevoir et développer une application web fonctionnelle avec un back, un front, une base de données, un pipeline, faire des tests et un minimum de sécurité mis en place. Cela m'a permis de mettre en pratique les connaissances acquises tout au long de l'année dans un projet concret.

La résolution de problèmes complexes a été une facette importante de mon rôle de développeur web au sein du projet. J'ai amélioré ma capacité à analyser les problèmes, à rechercher et mettre en place une solution appropriée. Ma curiosité m'a permis de me former constamment, notamment en lisant beaucoup d'articles sur énormément de sujets différents, à m'abonner à des comptes spécialisés sur des sujets techniques ce qui me permet de me tenir informée, et de faire de la veille technologique.

Globalement, sur 6 mois de développement, je suis satisfaite de là où j'ai pu amener le projet, et le nombre d'implémentations mises en place depuis. J'ai hâte de pouvoir continuer ce projet et de le mettre en ligne.

B. Perspectives d'évolution

Actuellement, je n'ai pas pu implémenter toutes les features annoncées lors de l'analyse des besoins.

La première évolution serait donc d'ajouter :

- Sur la page "découvrir", ajouter le caractère aléatoire de l'affichage des cosplays. En effet, actuellement l'affichage des cosplay se fait par ordre de création
- Le calendrier de fabrication pour la planification de la conception, besoins et achats du cosplay
- Le système de favoris permettant aux utilisateurs de sauvegarder les cosplays qui les intéressent

[19] SAST : Les tests statiques de sécurité des applications sont une méthode d'analyse statique du code source de l'application pour identifier les vulnérabilités potentielles. Cette technique examine le code sans le faire s'exécuter, en recherchant des problèmes de sécurité tels que des failles de sécurité, des erreurs dans le code, des problèmes de validation des entrées, etc.

- Avoir la possibilité de voir un profil utilisateur avec ses cosplays attachés.
- Ajouter le mode public/privée du cosplay
- La gestion des évènements non-officiels.
- Le système de tendances des cosplays sur le mois en cours
- Un système de partage sur les réseaux sociaux, créer son compte avec un réseau social
- Une application mobile
- L'administration des comptes officiels des utilisateurs organisateurs d'évènements
- La carte interactive avec les cosplayeurs et les évènements
- Ajouter un middleware de vérification des photos pour une modération plus adéquate (nsfw.js par exemple)
- Finir les tests et les validateurs
- Un bouton "signaler" sur un compte utilisateur ou sur un cosplay pour signaler des comportements non appropriés
- Ajouter des liens d'affiliation avec les marques dans le milieu du cosplay

Il serait également intéressant de prévoir des partenariats avec des conventions et des marques liées au cosplay afin d'offrir des avantages exclusifs aux utilisateurs de l'application. Par exemple, des réductions sur les achats de matériaux de cosplay ou des invitations spéciales à des événements autour du cosplay.

Annexes

Diagramme général : cas utilisateur connecté (en tant que cosplayeur)

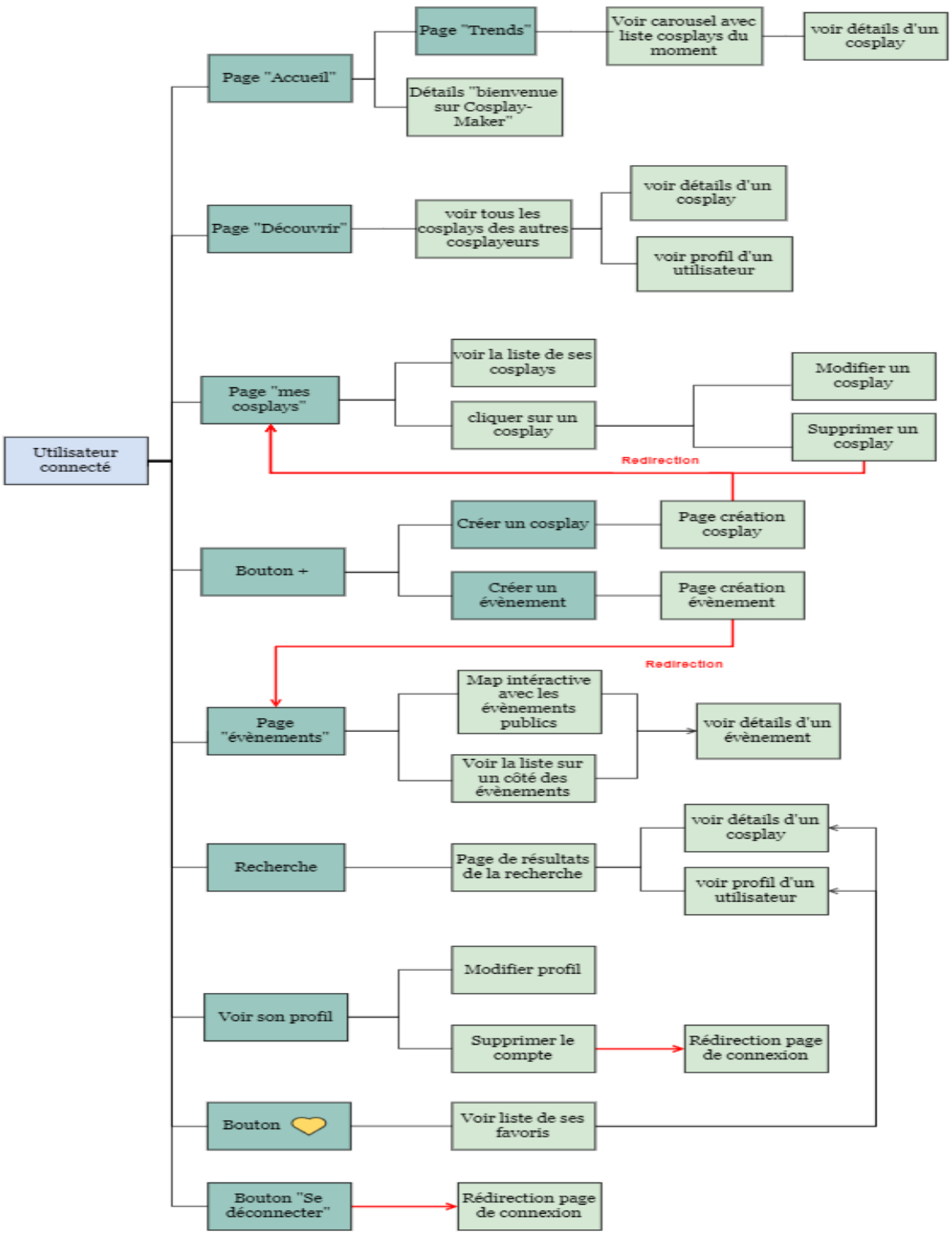


Diagramme général : cas utilisateur non connecté

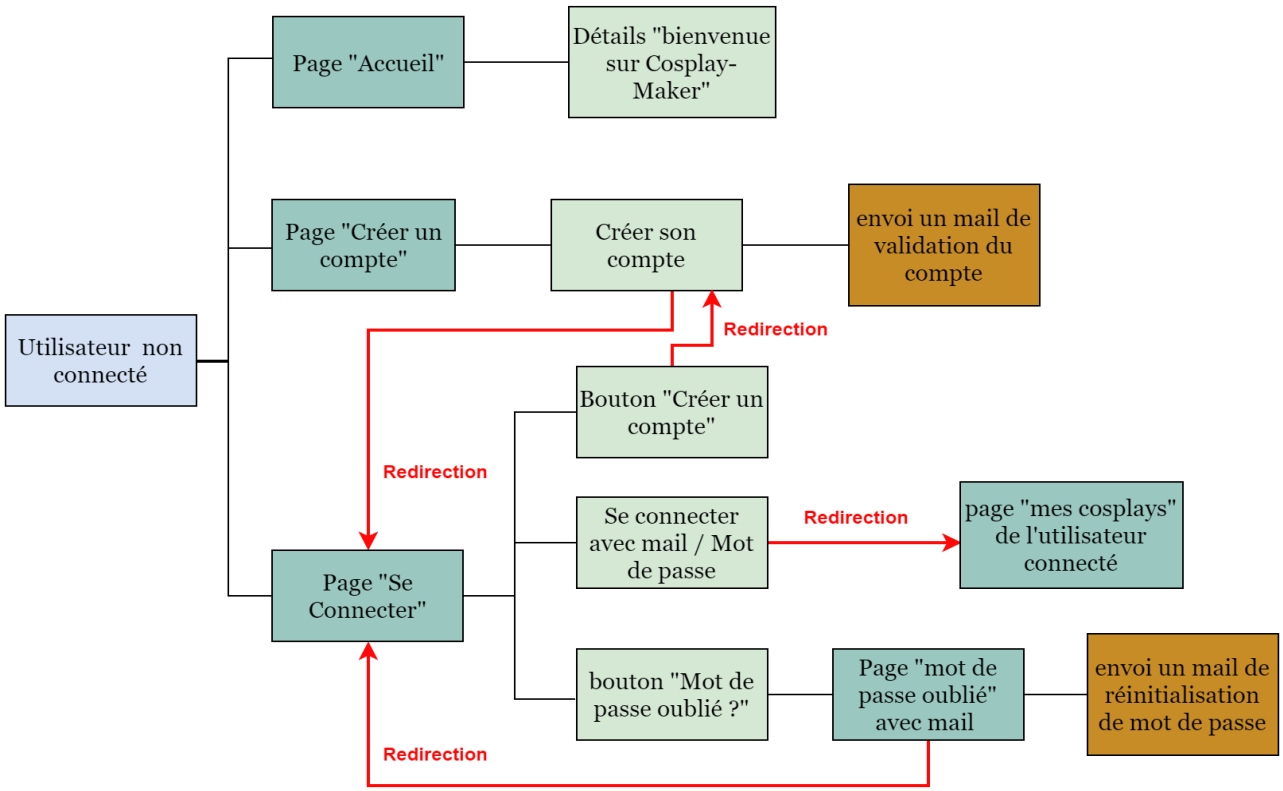


Diagramme général : cas utilisateur connecté (en tant qu'organisateur évènements)

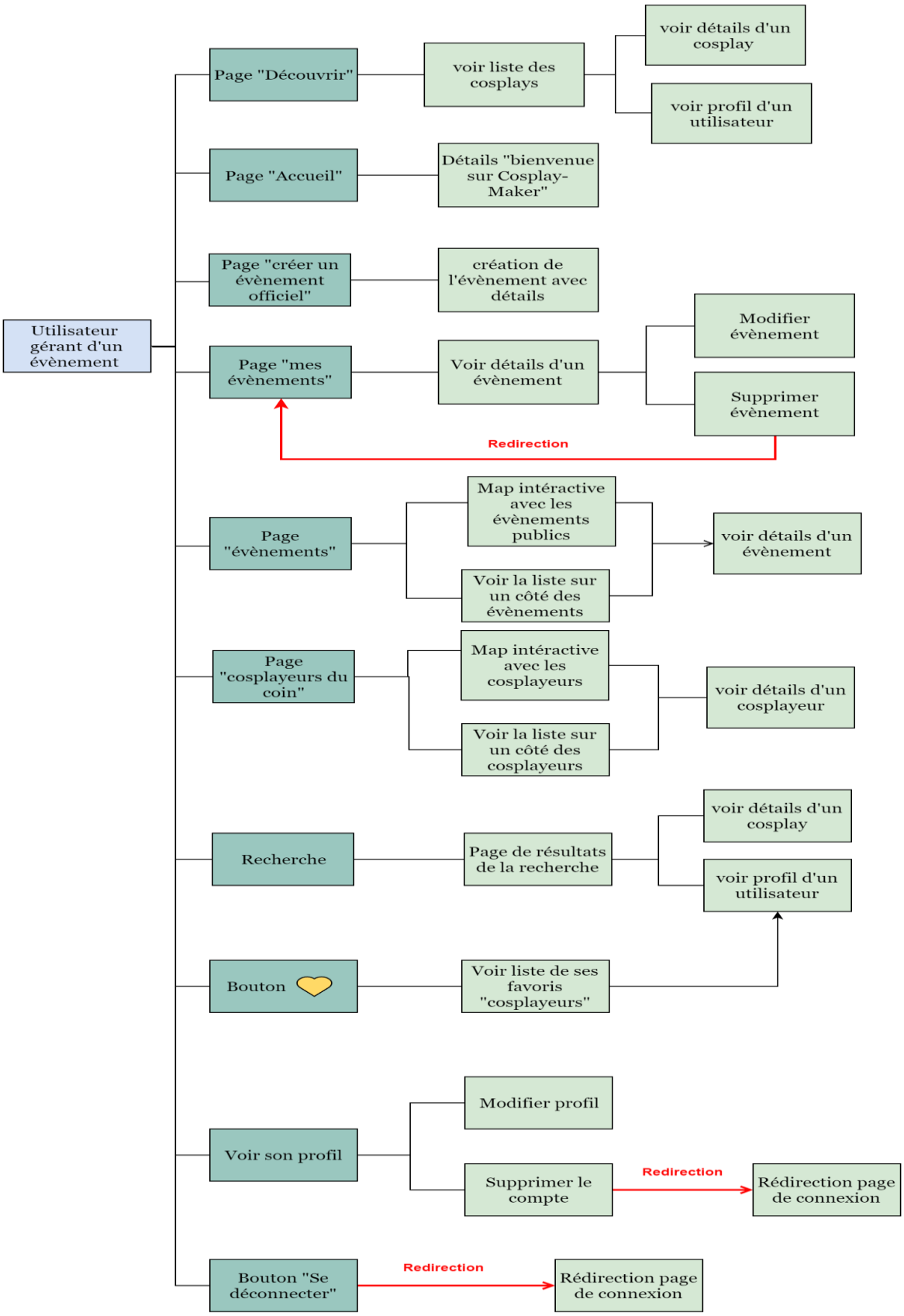
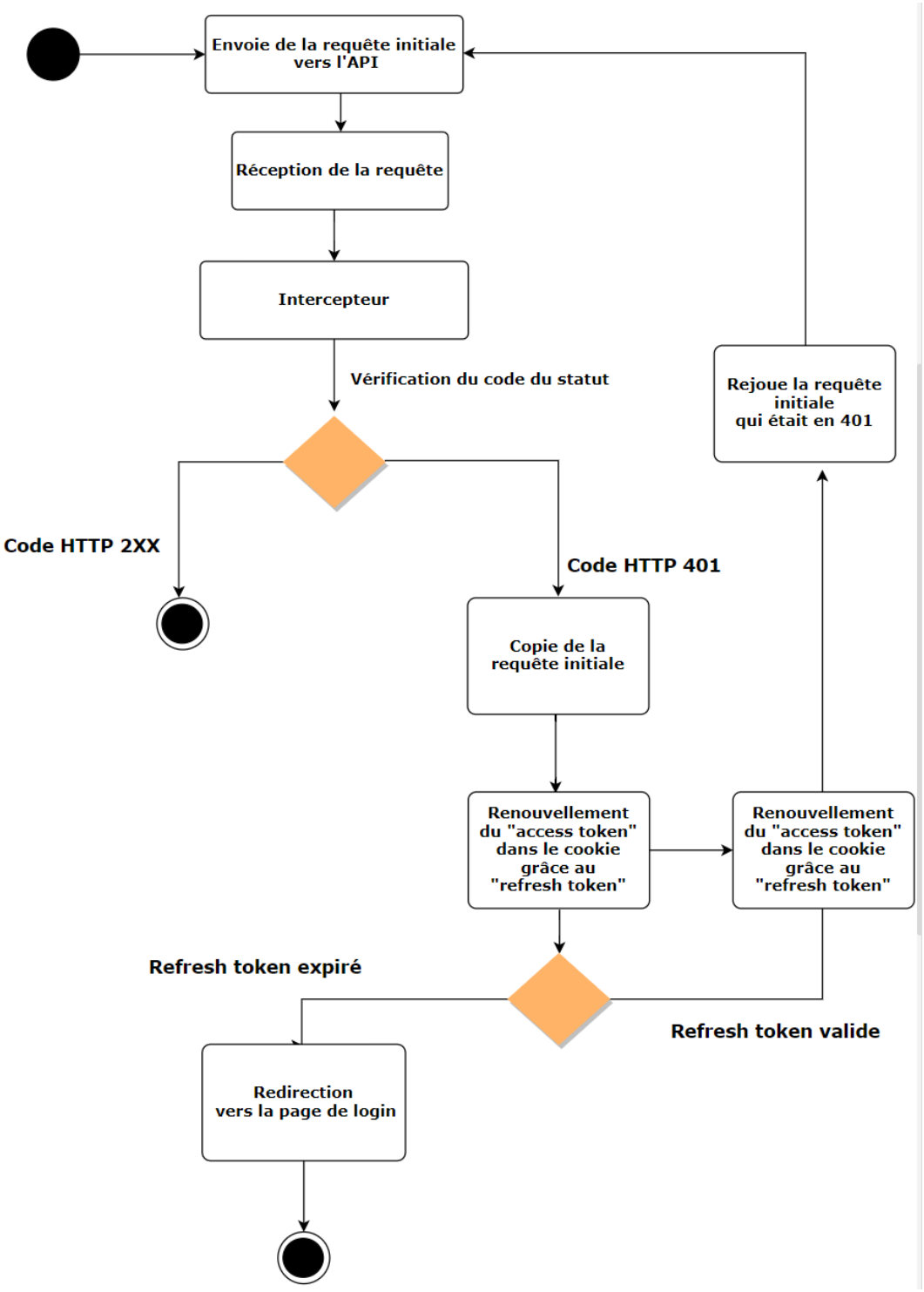


Diagramme d’activité : l’intercepteur d’Axios



Codes supplémentaires du CRUD cosplay en back-end

- Fonction pour obtenir tous ses propres cosplays :

```
function getAllMyCosplays(req, res) {
  const userId = req.user._id;

  Cosplay.aggregate(
    aggregatePopulateCreator({
      $match: { creator: userId },
    })
  ).exec(function (err, data) {
    if (err) {
      console.log(err);
      return res.send(500).json({ msg: "La requête n'est pas valide" });
    }
    return res.status(200).json(data);
  });
}
```

- Fonction pour mettre à jour un cosplay selon l'id de l'utilisateur :

```
function updateACosplay(req, res) {
  const id = req.params["id"];
  const creator = req.user._id;

  const update = req.body;
  update.creator = creator;

  Cosplay.findOneAndUpdate({ _id: id }, update, { returnDocument: "after" })
    .then((cosplay) => {
      if (!cosplay) {
        return res.status(404).json({ msg: "Cosplay non trouvé" });
      }
      return res.status(200).json(cosplay);
    })
    .catch((err) => {
      if (err) {
        console.error(err);
      }
      res.status(500).json({
        error: "Une erreur est survenue pendant la recherche du cosplay",
      });
    });
}
```

- Fonction pour supprimer un cosplay selon l'id de l'utilisateur :

```
function deleteCosplayById(req, res) {
  const id = req.params["id"];
  const creator = req.user._id;

  Cosplay.findOne({ _id: id }, (err, cosplay) => {
    if (!cosplay) return res.status(404).json({ msg: "Cosplay non trouvé" });
    if (!cosplay.creator.equals(creator)) {
      return res
        .status(403)
        .json({ msg: "Vous n'êtes pas le créateur de ce cosplay" });
    }
    Cosplay.deleteOne({ _id: id }, (err, cosplayToDelete) => {
      if (!cosplayToDelete)
        return res.status(404).json({ msg: "Cosplay non trouvé" });
      if (err) {
        console.error(err);
        res.status(500).json({
          error: "Une erreur est survenue pendant la recherche du cosplay",
        });
      } else {
        res.status(200).json({ msg: "Cosplay supprimé" });
      }
    });
  });
}
```

Maquettes Figma

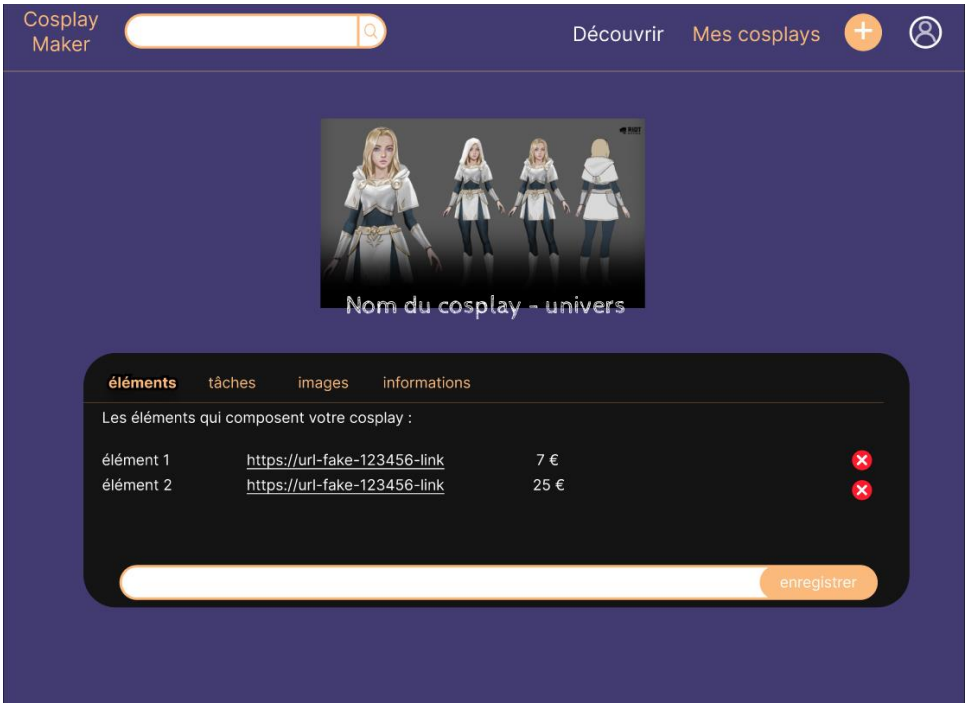
I. Page “mes cosplays” - utilisateur connecté - Version mobile

Cette version sera également similaire pour la page “Découvrir” en version mobile.

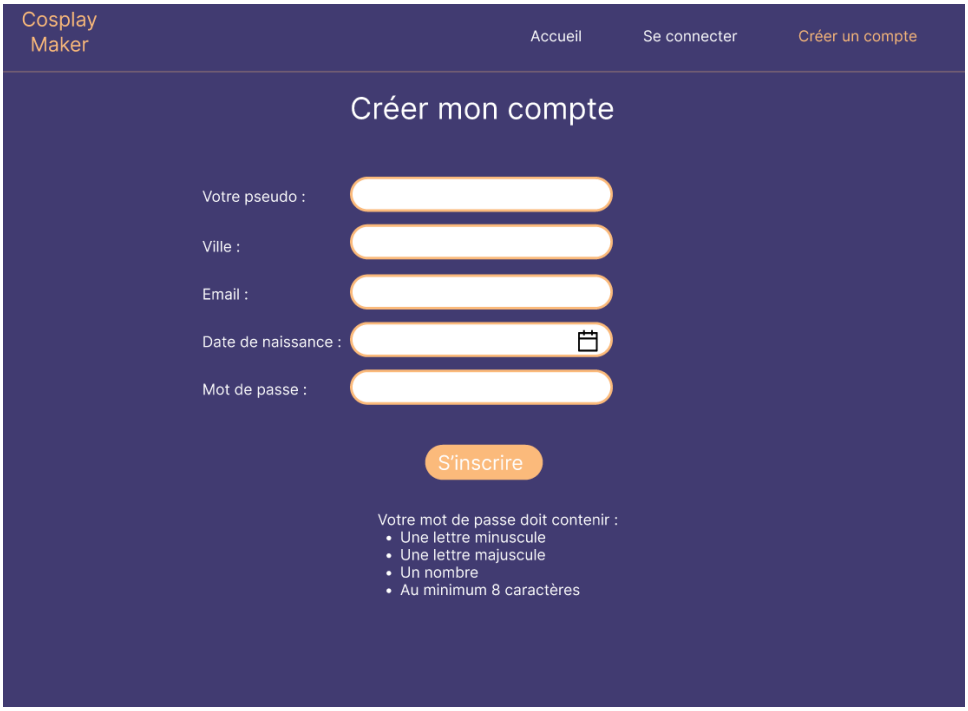
II. Page “Se connecter” - utilisateur non connecté - Version mobile

III. Page “Créer mon compte” - utilisateur non connecté - Version mobile

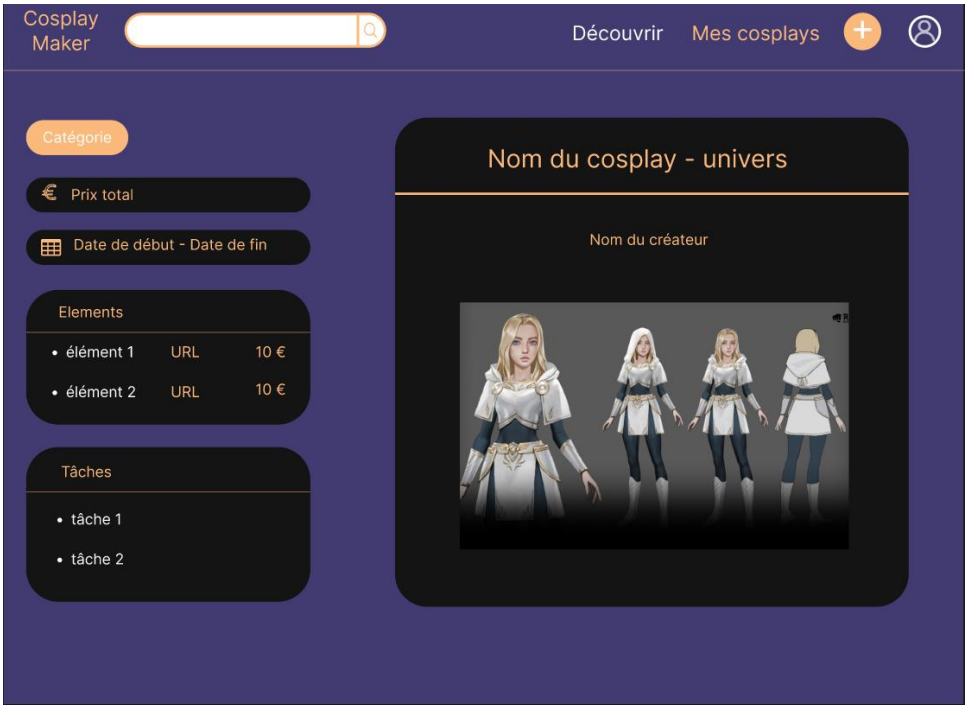
IV. Page “détails d’un cosplay” en mode édition - utilisateur connecté - Version Web



V. Page “Créer mon compte” - utilisateur non connecté - Version Web



VI. Page “détails d’un cosplay” en mode “vue sans édition” - utilisateur connecté - Version Web



VII. Page “Découvrir” - utilisateur connecté - Version Web