Rapport de mi-mandat Green Flag

2024-11-13

Rapport Bettina-Sarah Janesch

Description d'un développement réalisé par l'étudiant

Je me suis occupé de la page de suggestions/Matching incluant la gestion des images et implémenter plusieurs librairies ensemble pour avoir le visuel similaire à une application de rencontres ainsi que la communication avec le backend et la base de données via le custom Hooks et des fonctions asynchrones génériques.

Le custom Hook useFetch est fait générique pour accepter n'importe quelle interface TypeScript. UseFetch possède un hook useEffect qui s'exécute automatiquement quand la composante React render. Une gestion des erreurs et états de loading est présente. Le useFetch dépend de la fonction asynchrone fetchData qui est aussi générique – celle-ci accepte n'importe quelle interface comme paramètre et se connecte au backend via une requête HTTP POST vers n'importe quelle route choisie. FetchData peut recevoir des JSON ou des images selon le cas d'usage. L'information retournée est propagée à useFetch qui est capable d'accepter n'importe quel type d'information.

UseFetch est appelé dans la page de Matching quand la page se charge dans le l'explorateur internet et retourne les suggestions (JSON) : une liste des personnes avec leurs informations ainsi que leurs clés de photo.

Ces clés ont été générées au début quand une personne veut ajouter une photo dans leur répertoire : la photo est envoyée au backend et le photo DAO (la base de données LMDB - clé valeur) se charge de transformer l'image dans un byte array et ensuite en bytes pour être stocké avec le nom du fichier encrypté comme clé. Ces clés ont été déposées dans la base de données Postgres associé au membre.

Les informations de chaque personne suggérée est donné au ProfileCard, une composante custom que j'ai créé avec plusieurs petites composantes à l'intérieur (incluant des boutons 'Icon' génériques), incluant aussi un PhotoCarousel: ici à l'intérieur, il reçoit tous les clés photo de chaque personne, et une par une, via fetchData, les photos sont cherchées de la base de données LMDB et ainsi le carrousel est rempli d'images.

Plus grand défi de cette réalisation

Toutes les composantes et les hooks/fonctions interagissent d'une manière complexe qui n'était pas toujours évidente. J'ai eu beaucoup d'erreurs de logique interne et ordre à résoudre. Un grand défi a été de transformer des photos envoyées du frontend dans un format acceptable pour le backend et le transformer pour être stocké. Cela étant, le chemin inverse de

retour vers frontend a été difficile aussi donc j'ai dû rajouter de fonctionnalité à mon fetchData pour qu'il puisse recevoir des images aussi : en le faisant, j'ai brisé la fonctionnalité d'accepter des JSON, j'ai donc souvent eu des erreurs en cascade, mais finalement je m'en suis sorti.

Où se trouve dans le code ce développement

Dossiers dans GreenFlag\dev:

frontend\src\pages\MatchingPage.tsx

frontend\src\components\ProfileCard.tsx

frontend\src\components\ profile_card_components\ tous les fichiers, surtout PhotoCarousel.tsx

frontend\src\api\fetchData.ts

frontend\src\api\useFetch.ts

backend\DAOs\ photo_lmdb_dao.py

backend\Managers\account_manager.py

Rapport Vincent Fournier

Description d'un développement réalisé par l'étudiant

J'ai fait l'implémentation du chat, ainsi que de son chat room. Le but de cette implémentation était de permettre à mes usagers de parcourir une liste de conversations qu'ils possèdent avec d'autres usagers et de pouvoir démarrer celle-ci en la sélectionnant.

J'ai conçu une page dans le frontend qui demande au backend de lui renvoyer à partir de son id une liste des chat rooms que l'usager possède. Avec cette liste, ma page affiche toutes les chat rooms avec la photo profil de l'autre usager, son nom et le dernier message qui a été envoyé dans la conversation avec sa date d'envoi. Lorsque l'usager clique sur un chat room, il est redirigé vers une autre page qui représente cette conversation. Dans celle-ci, une connexion par web socket est faite avec le backend afin d'envoyer instantanément les messages envoyés à l'autre usager.

Dans le backend, les messages sont gardés dans une DischargedList et lorsque cela fait 5 secondes qu'elle n'a pas reçu un message ou lorsqu'elle sa limite de message elle envoie tous les messages qu'elle contient dans la base de données. Les messages gardés dans la base de données seront utilisés par la suite lors de la connexion d'un des deux usagers à la discussion. J'ai dû ajouter une logique à l'affichage des messages afin de différencier ceux envoyé par un usager de ceux de l'autre.

Plus grand défi de cette réalisation

Le design du système de récupération des données a été le plus grand défi de cette partie du développement. La difficulté était de trouver à quel moment il était opportun de faire une requête au backend et à quel moment il était préférable de propager l'information de composante à composante. Certaines informations comme le prénom de l'autre usager avec l'usager actuel ou son image de profil sont utilisées à plusieurs places, mais malheureusement il s'est avéré impossible de les transmettre entre composantes dû à une redirection vers une autre page. Il m'a fallu faire une nouvelle requête au backend, ainsi que de nouvelles fonctions spécifiques à cette requête.

Où se trouve dans le code ce développement

Tous ces dossiers se trouvent dans Greenflag\dev

Fichiers principaux :
frontend\src\pages\ChatroomsPage.tsx
frontend\src\componets\Chat.tsx
backend\DAOs\chat_dao.py
backend\Managers\chatroom_manager.tsx
backend\Managers\chatroom_socket_manager.tsx
frontend\src\components\ChatroomItem.tsx

Fichiers liés :
frontend\src\pages\PrivateChatroomPage.tsx
frontend\src\components\chat_components*
frontend\src\components\chatroom_item_components*

Rapport équipe

Plus gros défi que vous avez eu jusqu'à maintenant

Le plus gros défi que nous avons rencontré jusqu'à maintenant a été de décortiquer en équipe l'ensemble des étapes pour qu'une suggestion fonctionne de sa création en passant par son transfert vers le frontend et son retour au backend jusqu'à la création d'un match.

Est-ce qu'on en envoie plusieurs à la fois et avec quelles informations ? L'implémentation de la logique du swipe dans le frontend : est-ce qu'on update la suggestion directement après le swipe où est-ce qu'on attend pour faciliter l'application du undo ? Quel est la meilleure façon de créer un match avec un trigger dans la base de données ou par le backend à l'aide de requête à la base de données ? Ceci fait partir des nombreuses questions que nous avons dû nous poser lors de l'implémentation des suggestions. On peut attribuer l'existence de ce défi rencontré par soit le manque de détails dans notre document de conception ou nos visions divergentes de comment nous pensions implémenter cette partie du projet. Essentiellement, il s'agit d'une logique très complexe donc les défis vécus associés à ça ont été attendus.

Plus gros défi que vous voyez venir pour l'atteinte de vos objectifs

La quantité de temps par rapport à la quantité de fonctionnalités attendu – nous allons peut-être avoir besoin de couper quelques features. Parmi les features que nous pensons devoir abandonner se trouve les modes jour/nuit, la localisation, la modification du mot-de-passe et la page des statistiques. Le feature de UserFactory semble être d'une ampleur importante et il nous reste certaines incertitudes sur comment l'implémenter. Ce dernier reste tout du moins très important afin de pouvoir faciliter les tests de notre application.

Estimation d'avancement total de développement du projet (incluant le sprint 1 de conception et de planification)

Nous estimons avoir fait environ 65% du travail jusqu'à maintenant. Au fil du temps, nous avons décidé de couper quelques fonctionnalités petites pour pouvoir atteindre les objectifs principaux. Si le temps le permet à la fin, nous allons les rajouter au besoin.
