HABIB Yassin
MANGINI Isabele
SONOT Adrien

2A SEOC

Foodgram

Documentation

GRENOBLE INP UGA Phelma

Sommaire

Foodgram

Foodgram est une application conviviale de partage de recettes faites maison avec des photos accompagnant une description ainsi que d'autres informations comme la catégorie, la date… Les utilisateurs peuvent notamment prendre en photo leur plat sur l'application directement, ou bien choisir la photo depuis leur galerie. Il est possible aussi de "liker" les recettes, et de retrouver cette liste de likes dans les infos utilisateurs afin de mettre de côté ses recettes préférées.

I) Introduction API Foodgram

Notre API utilise l'adresse suivante:
URL: https://foodgram.osc-fr1.scalingo.io/

Pour des tests en local, on peut utiliser le port 3000 sur la loopback (adresse 127.0.0.1). Pour faire les requêtes et donc tester nos fonctions du backend, on peut utiliser l'utilitaire Postman (REST). Sauf indication contraire, les requêtes utilisent toutes le format "x-www-form-urlencoded" avec les attributs passés directement dans le body pour les requêtes de type "POST". Le token utilisateur sera passé dans le header de la plupart des requêtes sous le champ "token".
Les requêtes qui aboutissent renvoient 200 pour la plupart, en cas d'échec il peut y avoir différents retours:

Grâce à Swagger AutoGen, nous avons pu généré une documentation de l'API d'une manière automatique, accessible via le lien:

https://foodgram.osc-fr1.scalingo.io/doc

II) Requêtes pour les utilisateurs

On peut manipuler la base de données des utilisateurs par le biais de requêtes pour ajouter, modifier ou supprimer des utilisateurs notamment. Pour l'authentification, on utilise un token, généré à l'aide d'une fonction de hachage.

On a notamment les requêtes permettant l'authentification:

Chemin	**POST**
/login	**Authentifie l'utilisateur avec l'email et mot de passe passés dans la requête sous les champs "email" et "password" Renvoie un objet json contenant le token**
/signup	**Permet d'inscrire l'utilisateur, doit contenir l'email, username et mot de passe sous les champs "email", "name" et "password" [T]**

[T] : un token utilisateur, passé dans le header sous le champ "token" est requis

Chemin	**GET**	**PUT**
/userinfo	**Renvoie les informations de l'utilisateur identifié par son token [T]**	**Non défini**
/edituserinfo	**Non défini**	**Modifie le nom de l'utilisateur passé dans la requête [T]**
/changepass	**Non défini**	**Modifie le mot de passe [T]**

III) Requêtes pour les posts

En ce qui concerne la base de données des posts, nous avons ajouté différentes requêtes.

Tout d'abord, nous avons l'ajout d'un nouveau post, qui nécessite la mise en place de deux requêtes, une première pour le post des champs textes nécessaire à la recette, et une deuxième pour le post de l'image, nous avons défini leurs routes respectives dans le fichier

“/Foodgram_adrien_sonot_isabele_mangini_silva_yassin_habib/backend/src/routes/post.js” comme suit :

Chemin	**POST**
/recipe	**Créé une recette avec ses informations passées dans le body de la requête, exemple body.title représente le titre [T]**
/picture	**Envoie l’image passée au format “form-data” sous le champ “file” [T]**

Pour /picture, le backend utilise les middlewares “multer” et “sharp” qui vont traiter les données de la requête et notamment redimensionner les images au format 800*500px puis les enregistrer dans le dossier src/pictures dans le backend. Une limitation de 10Mo a été choisie pour les photos envoyées sur le serveur, les photos dépassant cette taille ne seront pas postées.

Les autres méthodes (get, put, delete) ne sont pas définies, autrement dit on obtient un message “endpoint not found”.

Ensuite, nous avons créé deux requêtes pour avoir les posts des utilisateurs.
La première est pour afficher les posts de tous les utilisateurs dans le “Home Screen”, et la deuxième est pour n’afficher que ses propres posts dans le “UserInfo Screen”, leurs routes respectives sont définies par:

chemin	**GET**
/ownrecipes	**renvoie un objet json contenant la liste des recettes postées par l’utilisateur [T]**
/home	**renvoie un object json contenant toutes les recettes du site [T]**

De plus nous avons réalisé des requêtes concernant la manipulation des likes sur les posts, ils englobent l'ajout des likes, leur suppression ainsi qu'un get pour avoir les likes de l'utilisateur sur les posts.

Leurs routes sont spécifiées respectivement comme suit:

chemin	POST	GET
/like	**like la recette avec son id passée dans le body, ex: body.id = 5 [T]**	**Non défini**
/unlike	**comme like mais dislike la recette [T]**	**Non défini**
/likes	**Non défini**	**renvoie un objet json contenant la liste des likes de l'utilisateur [T]**

Et enfin nous avons établi une requête pour la suppression définitive d'un post, elle permet à un utilisateur de supprimer un post, seulement si il en est vraiment le propriétaire.

Sa définition de route est:

chemin	DELETE
/deleterecipe	**supprime la recette dont l'id est passée dans le body de la requête [T]**

III) Requêtes pour les administrateurs

chemin	GET	POST	DELETE
/user	**Non défini**	**Non défini**	**Supprime l'utilisateur dont l'id est passé en paramètre [T]**

/users	**renvoie un objet json contenant la liste des users [T]**	**Non défini**	**Non défini**
/admins	**pareil que users mais liste des admins [T]**	**Non défini**	**Non défini**
/admin	**Non défini**	**Donne le privilège d'admin à l'utilisateur dont l'id est passé en paramètre [T]**	**Supprime le privilège admin à l'user [T]**

IV) Installation

Le projet s'articule en deux parties: le frontend qui donne accès à l'interface et communique avec l'API, et le backend qui traite les requêtes du frontend et stocke les données. L'installation du backend est facultative si l'on ne souhaite pas modifier l'API de Foodgram.

Installation frontend:

- installer nvm
- nvm install v16 (pour installer Nodejs version 16)
- npm i pour installer les packages (si erreur de package ajouter --force en argument)
- npx expo start pour lancer le frontend (ajouter --tunnel si erreur)

Une fois que le client frontend est lancé on peut:

- utiliser la version web à l'aide de "w"
- utiliser la version android en scannant le qr code (à condition d'être sur le même réseau si on n'utilise pas le --tunnel)
- lancer un émulateur comme android studio en parallèle et appuyer sur "a"

Installation backend:

- npm install
- cd backend
- node
  > require('mandatoryenv').load()
  > require('./src/models/users').sync({force:true})
- nodemon start (le serveur va se relancer à chaque mise à jour du backend)

V) Tests automatisés:

Lors de la réalisation de notre application web/mobile, nous avons varié nos outils de test pour nous assurer de sa fiabilité et de sa robustesse. Pour tester les requêtes du backend, nous avons opté pour Supertest, qui est une bibliothèque de tests pour les applications Node.js et qui nous a permis de simuler les requêtes HTTP et de vérifier les réponses de manière automatisée, au lieu de les faire manuellement à l'aide de l'utilitaire Postman. Pour le frontend, nous avons opté pour Cypress, un outil de test de bout en bout pour les applications web modernes, qui nous a permis à son tour de faire des tests d'une manière automatique sur le Frontend de notre application Web/Mobile. Ces outils nous ont permis de nous assurer que notre application était fonctionnelle et répondait aux besoins de l'utilisateur.

Pour les tests cypress, on peut les lancer à l'aide des commandes suivantes:

- cd backend
- npx cypress install (si cypress n'est pas installé)
- npx cypress run

Pour les tests du backend, on fait:

- npm run test