Fichier API et Dashboard

*Lien URL :*

Dashboard (Streamlit) : [Streamlit (gregoryburiez-streamlit-p7-streamlit-x5sxlx.streamlit.app)](https://gregoryburiez-streamlit-p7-streamlit-x5sxlx.streamlit.app/)

API : [Test de modèle Flask (buriez-flaskp7.herokuapp.com)](https://buriez-flaskp7.herokuapp.com/)

GITHUB : [GregoryBuriez (Buriez Grégory) (github.com)](https://github.com/GregoryBuriez)

1. *API sous Flask*

Dans le cadre de la mise en place de notre API, nous avons mis en place notre projet sur 2 accès.

Le premier accès est la page d’accueil qui nous permet d’indiquer l’identifiant de notre client puis de tester notre modèle. Cette page va provoquer une prédiction des données disponible dans la phase de modélisation de notre projet.

Le second accès est tout simplement la page qui indique le résultat de notre prédiction. Cette dernière va présenter deux possibilités, à savoir :

* Si le client est présent dans notre jeu de données, nous allons pouvoir obtenir une prédiction de données.
* Si le client n’est pas présent dans notre jeu de données, cela va nous indiquer une erreur et nous recommander de retourner vers la page d’accueil.

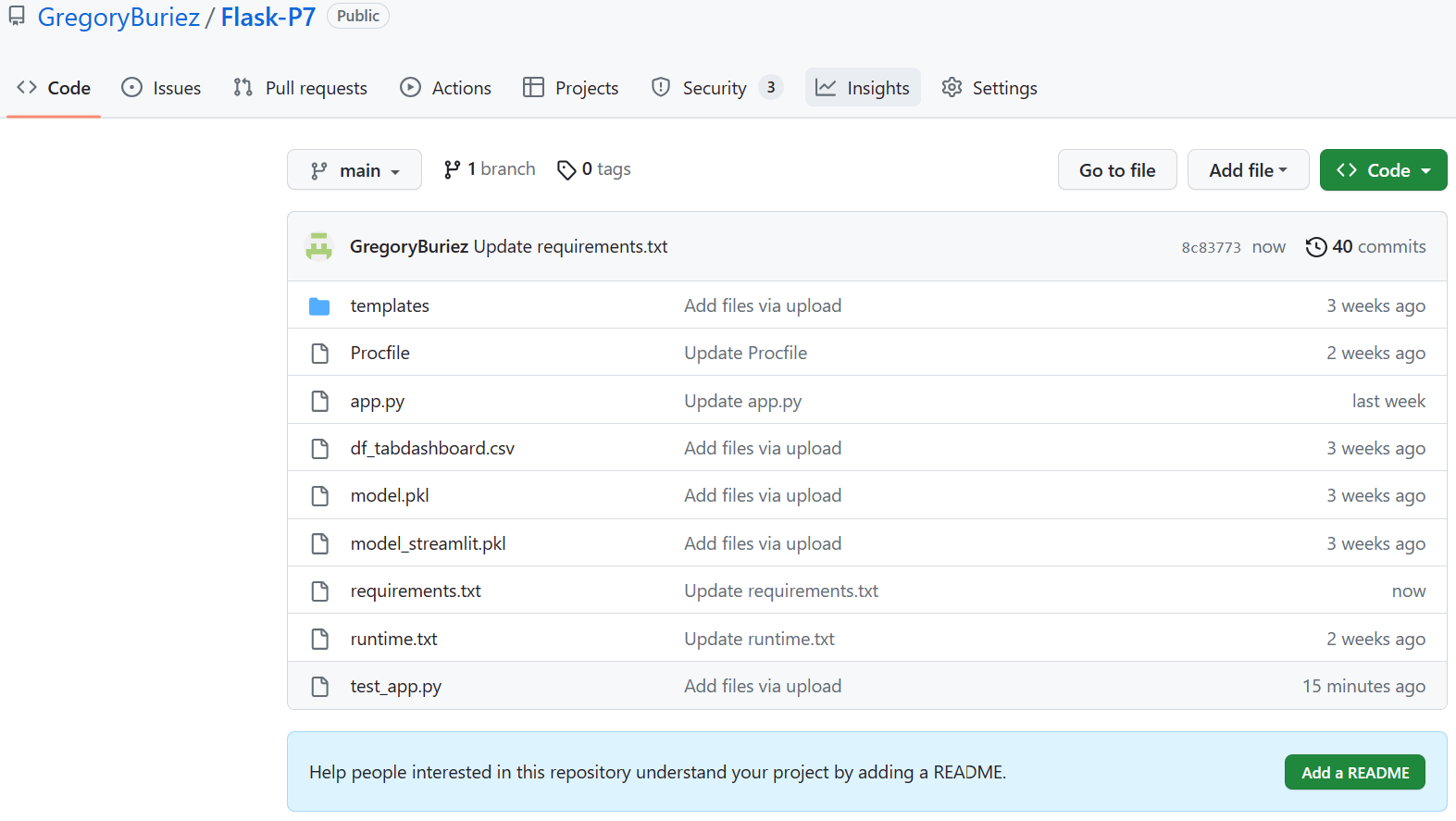
De plus, il nous a été demandé d’effectuer un test unitaire via la librairie Pytest, l’idée étant de tester la phase de déploiement et si l’ensemble des étapes fonctionnent correctement. Nous avons affiché le résultat de cette phase de test.

Afin de pouvoir déployer notre projet API, nous avons pris la décision d’utiliser Heroku. Certains fichiers sont essentiels pour le déploiement de notre projet :

* Le code sous format python,
* Le requirement.txt (qui représente toutes les librairies),
* Notre modèle (mis en place dans l’étape de modélisation),
* Notre jeu de données (panel de 100 clients),
* Le fichier procfile (ce dernier va permettre de créer le lien entre notre Github et Heroku avec la bibliothèque gunicorn)
* La réalisation du test unitaire
* Un dossier templates (qui contient notre page d’accueil et notre page de prédiction),
* Un environnement virtuel.

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement



1. Dashboard (Streamlit)

Afin de procéder à la mise en place de notre tableau de bord, nous allons prendre la décision de séparer notre code en plusieurs étapes bien distincts.

La première étape correspond à l’appel de l’ensemble des données (fichiers, modèles, librairies) mais aussi à la création de notre barre de recherche. Cette dernière sera notamment essentielle pour l’ensemble de notre projet car l’interactivité se fera à partir de cet accès.

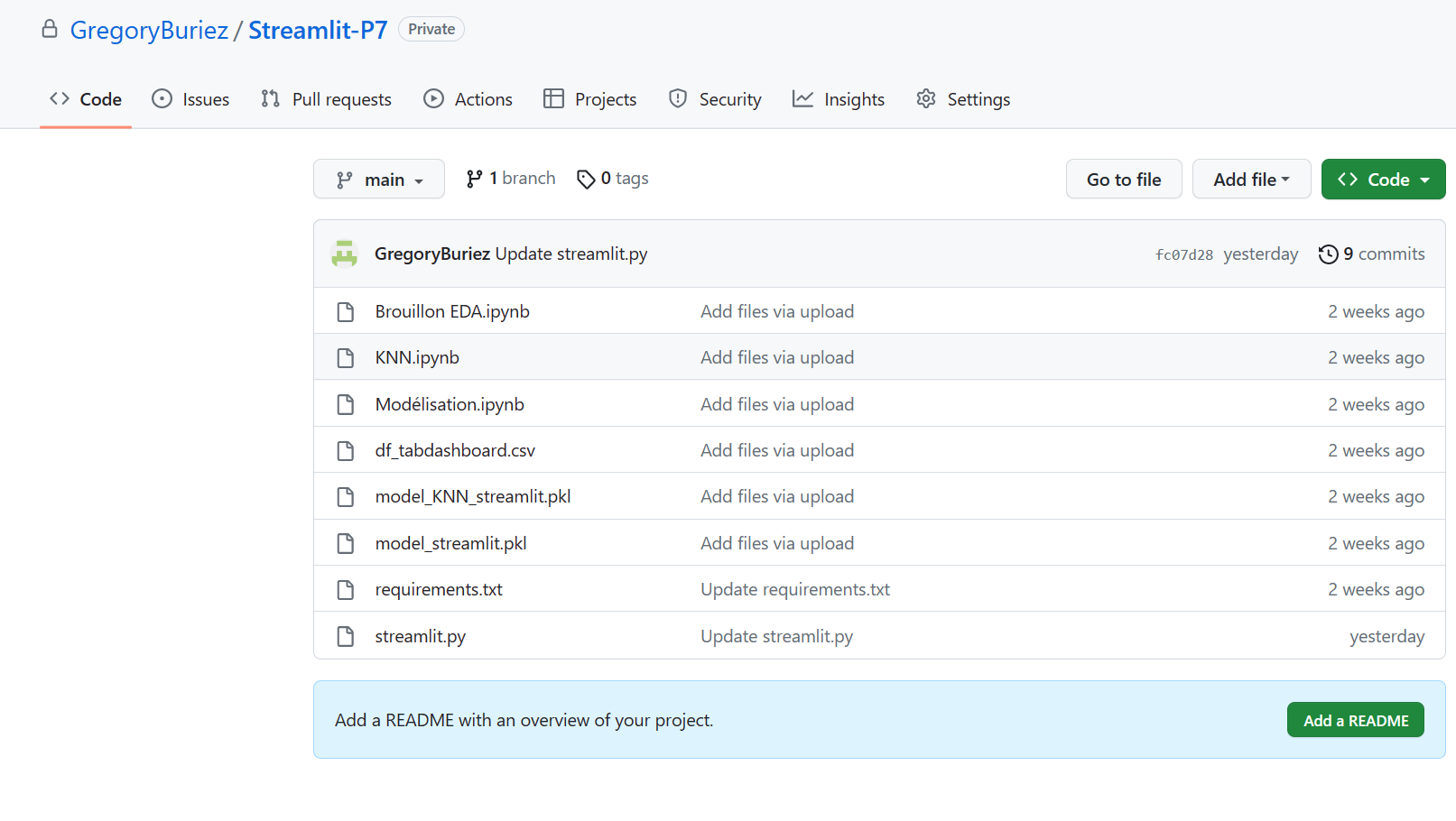
La seconde étape est l’affichage de plusieurs variables que nous avons sélectionné mais aussi l’obtention d’un taux d’endettement du client. Cela nous permet simplement d’avoir connaissance des informations du client.

L’étape suivante consiste à un appel de notre API afin d’obtenir la prédiction du client. Cette dernière se fera à partir du lien URL de notre API (il est donc essentiel de déployer cette dernière). A partir de ce résultat, nous allons afficher le résultat, nous avons pris l’initiative de l’encadrer pour bien appuyer sur le résultat obtenu.

Ensuite, nous allons procéder à un affichage de deux graphiques interactifs à partir de notre barre de recherche. Nous avons pris l’initiative d’afficher le rapport des crédits et celui de l’endettement.

Enfin, la dernière étape sera l’utilisation de notre modèle du KNN pour obtenir le client le plus proche en termes d’informations avec le client que nous avons sélectionné également dans la première étape. Notre décision est d’afficher uniquement un seul client dans notre dashboard sur un panel de 100 clients.

Dans le cadre du déploiement de ce projet directement sur le cloud de Streamlit, nous avons utilisé un ensemble de fichier présent dans notre dossier Github.



1. Informations supplémentaires

Dans le cadre de ces deux projets, nous avons utilisé le logiciel VSCode que ce soit pour pour l’étape du codage mais également la création de l’environnement virtuel.

Pour procéder au déploiement, nous avons utilisé GITHUB afin de pouvoir créer un répertoire pour Streamlit et Flask.

Enfin, l’utilisation du terminal sous Windows mais également le logiciel GIT ont permis d’envoyer l’ensemble de nos fichiers en direction de nos deux répertoires distincts.