Mergify

Rapport de stage

Mergify

Août 2020 - Février 2021

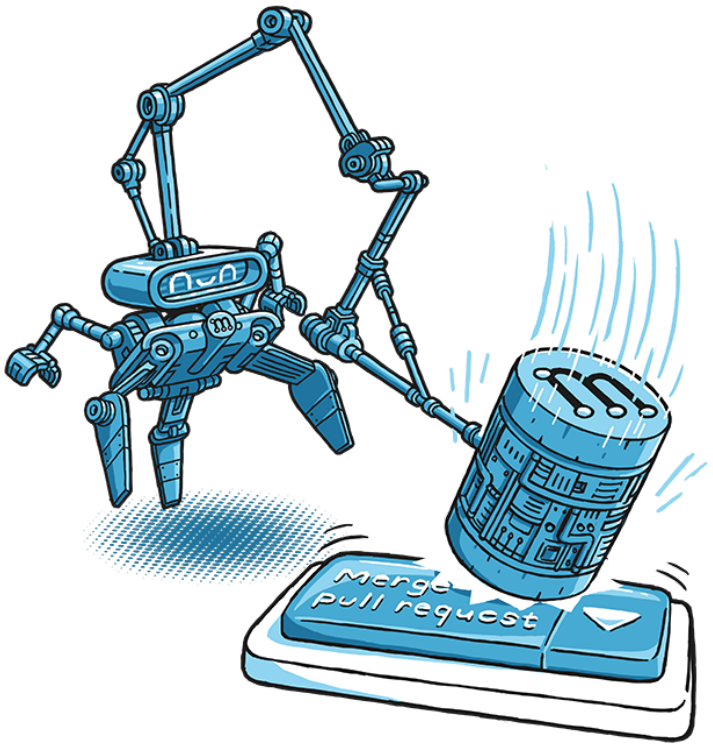

Table des matières

Remerciements

Julien

Mehdi

Je tiens vivement à remercier mon mentor de formation Erwan Keribin qui me suit depuis le début de mon parcours chez OpenClassrooms. Il a su me guider à travers l'ensemble des projets qui jalonnent ma formation, mais aussi me faire découvrir le métier de développeur en partageant ses expériences.

Contexte

Présentation

En juillet 2019, j'ai entamé le parcours de formation **Développeur d'Application Python** proposé par OpenClassrooms.

Cette formation s'inscrit dans le cadre de ma reconversion professionnelle. Dessinateur technique dans le domaine du bâtiment durant une dizaine d'années, puis formateur pour un logiciel de modélisation 3D, il est temps pour moi d'embrasser la carrière de développeurs.

L'entreprise

Mergify est une société fondée en 2018 par Julien Danjou et Mehdi Abaakouk. Ensemble, ils ont créé un outil d'automatisation autour des pull-requests sur GitHub.

Cet outil permet aux utilisateurs de prioriser et automatiquement merger, commenter, rebaser, mettre à jour, étiqueter, backporter, fermer, assigner les pull-requests.

Au cours de mon stage, Julien et Mehdi conservaient une activité salariée en parallèle de Mergify.

A la fin de mon stage, Mehdi était à mi-temps pour Mergify avec l'objectif de passer à plein temps dans le courant de l'année.

L'entreprise est domiciliée à Toulouse, cependant elle n'a pas de locaux officiels (Julien est sur Paris, Mehdi sur Toulouse). Par conséquent l'ensemble du stage s'est déroulé en "remote".

Motivation

La réalisation d'un stage dans le cadre de cette formation est une option. Pour le projet 12, l'étudiant a le choix entre :

- créer un outil de veille technologique;
- effectuer un stage d'une durée minimale de deux mois.

Je cherchais à faire un stage pour affirmer mes compétences et arriver sur le marché des développeurs avec plus de confiance.

L'opportunité chez Mergify s'est présentée grâce à Camille, une étudiante du parcours.

La pile technique de l'engine présentait tous les critères que je m'étais fixés : Python, Flask, PostgreSQL. La partie tableau de bord, en React, était pour moi l'opportunité de me confronter plus sérieusement à des problématiques de front-end.

Le dernier point qui m'a motivé à faire mon stage chez Mergify, l'engine est Open Source. Convaincu de l'importance des applications Open Source, l'opportunité se présentait à moi de voir comment une entreprise peut vivre d'un produit mis à disposition de ses clients.

Objectifs

Pour ce stage je me suis fixé trois objectifs :

1. confirmer ma préférence pour les projets orientés back-end;
2. découvrir la vie d'un projet d'envergure;
3. avoir une première expérience professionnelle, valorisable par la suite.

Problématique

Missions

Durant le stage plusieurs missions m'ont été confiées. Celle-ci peuvent être classées suivant trois catégories:

1. résolution de bug;
2. création de nouvelles fonctionnalités;
3. rédaction de documentation.

Mergify fonctionne avec deux dépôts :

1. un dépôt open source pour l'engine : https://github.com/Mergifyio/mergify-engine
2. un dépôt privée pour la partie tableau de bord : https://dashboard.mergify.io

Dans les parties suivantes, nous allons détailler trois projets en particulier. Deux d'entre eux concernent le tableau de bord, le dernier est un projet annexe.

Programme de parrainage

Le marketing de Mergify se base principalement sur le système de bouche à oreille. Il était donc important pour les fondateurs d'avoir un système de parrainage pour récompenser les clients au cœur de cette démarche.

Cahier des charges

Le cahier des charges était le suivant :

- chaque utilisateur à un code de parrainage unique qu'il peut partager avec son filleul;
- le parrain et le parrainé se voit offrir un siége gratuit pour une durée de un an;
- l'offre ne fonctionne que si le parrainé souscrit un abonnement payant.

Arborescence du projet

Ci-après une arborescence synthétique du tableau de bord :

```
root
├── emails
│   ├── dist
│   │   ├── ...
│   └── src
│     ├── ...
├── mergifyio                        # Back-end
│   ├── blueprints
│   │   └── api
│   ├── clients
│   ├── data
│   ├── middlewares
│   ├── models
│   ├── static -> ../src/static
│   └── tests
│     └── ...
├── ...
├── src                              # Front-end
│   ├── customers
│   ├── dashboard
│   │   ├── ...
│   │   └── referral
│   ├── ...
│   └── tests
│     └── ..
└── ...
```

Réalisation

Le projet est découpé par sous-projets réparties entre le back-end et le front-end du tableau de bord.

Back-End	Front-End
Modèle pour le parrainage	Page supplémentaire sur le tableau de bord
Code de parrainage unique	Formulaire pour récupérer son code parrainage
Endpoint "referral" pour le front-end	Formulaire pour entrer un code parrainage
Prise en compte pour la facturation	Liste des utilisateurs parrainés
	Afficher le nom du parrain et la date limite du parrainage

Modèle parrainage

Le modèle créé pour le parrainage est composé de trois colonnes :

1. l'ID GitHub du parrain;
2. l'ID GitHub du parrainé;
3. et la date du parrainage.

```
referee_github_id = sqlalchemy.Column(
    sqlalchemy.Integer,
    sqlalchemy.ForeignKey("github_account.id"),
    nullable=False, primary_key=True,
)

referrer_github_id = sqlalchemy.Column(
    sqlalchemy.Integer,
    sqlalchemy.ForeignKey("github_account.id"),
    nullable=False,
)

referred_at = sqlalchemy.Column(
    sqlalchemy.DateTime,
    nullable=False,
    server_default=sqlalchemy.func.now(),
)
```

Une contrainte est ajoutée entre les deux colonnes d'ID de façon à s'assurer qu'un utilisateur ne se parraine pas lui même :

```
__table_args__ = (
    sqlalchemy.CheckConstraint(
        "referrer_github_id != referee_github_id",
        name="check_referrer_not_referee",
    ),
 )
```

A celà s'ajoute aussi la création de deux méthodes de validation du parrainage :

- Est-ce que le parrainé à bien souscrit une offre payante ?
- Est-ce que les dates du parrainage sont valides (période de un an) ?

Code de parrainage

Le code de parrainage est ajouté comme propriété du modèle utilisateur :

```
@property
def referral_code(self) -> typing.Optional[str]:
    if self.login is None:
        return None

    login = self.login.lower()

    s = login + config.REFERRAL_SECRET + login

    return f"{login}-{sum(map(ord, s)) * 8529 % 1000000:06d}"
```

Cet algorithme fonctionne de la manière suivante :

1. création d'une chaîne de caractères composées du `login` de l'utilisateur, d'une clef secrète appelée `REFERRAL_SECRET` puis de nouveau du `login`;
2. avec `sum(map(ord, s))` on calcul la somme des numéros Unicode de chaque caractère que l'on multiplie par `8529` puis le reste de la division par `1000000` forme la partie numérique du code;
3. finalement on retourne une chaîne de caractères composées du `login` de l'utilisateur puis du nombre calculé à l'étape précédente. Soit un code de la forme suivante : `guillaumeoj-235608`.

Endpoint de parrainage

Un "endpoint" est ajouté pour permettre à la partie front-end d'ajouter un nouveau parrainage. Cet endpoint accepte uniquement la méthode `POST` et récupère un JSON de cette forme :

```
{
  "github_account_id": …, # ID GitHub du parrainé
  "referral_code": …,     # Code de parrainage
}
```

Avant d'ajouter le parrainage dans la base de données le code :

- vérifie que le JSON contient un ID de compte et un code de parrainage;
- récupère le login du parrain grâce au code de parrainage (voir Code de parrainage);
- vérifie que le login du parrain correspond à un compte existant;
- crée une instance et l'enregistre dans la base de données.

L'enregistrement dans la base de données peut retourner trois erreurs :

- l'utilisateur tente de se parrainer lui même;
- l'utilisateur a déjà un parrain;
- l'utilisateur utilise un code de parrainage erroné.

Prise en compte pour la facturation

Envoi d'e-mail par événement

Composant React

Bilan