Détecteur d’émotions

Projet final

Chef d’œuvre (Évaluation E1)

Frédéric CARRICART

Formation “**Développeur Data / IA**” 2019 / 2021

Sommaire

1. Problématique

Les habitudes de consommation changent. Le commerce en ligne est en pleine expansion, poussé par les avancées et la démocratisation des outils numérique et grandement accéléré par la pandémie mondiale et les bouleversements qu'elle impose dans nos modes de vie.
Et même quand nous continuons à nous rendre chez nos commerçants préférés, bien souvent nous nous sommes renseigné auparavant sur les produits que nous souhaitons acheter.

Afin de fixer notre choix sans pouvoir réellement voir les produits, les toucher où les sentir, le moyen le plus précieux dont nous disposons, est l'avis des autres utilisateurs ayant achetés le même produit. Il peut s'avérer fastidieux de passer en revue tous ces commentaires, il y en a parfois plusieurs centaines, parfois dans différentes langues. Une analyse des émotions de ces opinions pourrait nous permettre de nous faire un avis le plus éclairé possible.

Notre application, propose de récupérer les avis des consommateurs. En lui fournissant l'adresse URL, elle se charge d'aller récupérer l'ensemble des commentaires laissés sur la page de ce produit, puis en analyse les émotions émergentes à l'aide d'un algorithme d'intelligence artificielle.

2. Analyse des besoins

L'utilisateur type de notre application est un consommateur cherchant à se forger un avis sur un produit ou un service.

➢ Parcours utilisateur

Lorsqu'un utilisateur souhaite connaître les émotions qu'un produit a pu susciter chez les consommateurs ayant fait part de leurs expériences, il se connecte sur l'application en ayant copié dans son "presse papier" l'adresse de la page des commentaires. Après l'avoir collée dans le champ adéquat, une barre de chargement s'affiche pour indiquer que le traitement de la demande est en cours, puis, il est redirigé vers la page de résultat. L'utilisateur peut à tout moment se créer un compte afin de garder en archive ses requêtes et leurs résultats.

➢ Spécifications fonctionnelles

Pour couvrir l'ensemble du périmètre fonctionnel du besoin exprimé, l'application devra au moins être constituée de :

- Une interface permettant de soumettre sa demande et d'en consulter les résultats.
- Un algorithme capable de détecter les émotions sous-jacentes d'un texte.
- Une base de données stockant les informations de l'usager.

3. Intelligence Artificielle

La tâche qui nous occupe, consiste à reconnaître des émotions dans du texte écrit, cette opération de **classification** se rapporte au traitement automatique du langage naturel. Le **NLP** (Natural Language Processing) est une sous-catégorie de l'Intelligence Artificielle au même titre que la vision par ordinateur qui analyse puis interprète des images. Le NLP, lui, traite le langage sous forme écrite pour tenter de donner à nos ordinateurs la capacité (illusoire) de nous comprendre.

Cet ensemble de techniques à la croisée entre la linguistique et l'informatique trouve une application dans des domaines tels que la traduction automatique, les Chatbot, la génération de texte, ...

> ***Choix stratégique :***
>
> ***Les sites marchands les plus prisés, étant d'envergure internationale, les avis clients sont bien souvent écrit dans plusieurs langues. Le choix a donc été fait d'entraîner le modèle à partir de textes en anglais et de traduire les commentaires "à la volée" vers cette langue afin de les analyser. Le résultat, quant à lui, sera affiché en français.***

➢ dataset

Pour entraîner notre modèle, nous devons lui fournir des exemples à partir desquels il pourra chercher à s'améliorer.

Le dataset sélectionné, vient de Kaggle et se compose de 20 000 observations constituées de phrases et réparties en six catégories représentant autant d'émotions (joie, peur, colère, surprise, amour et tristesse).

```
pd.set_option('display.max_colwidth', -1)
data.sample(n=10)
```

	text	label
9838	i can feel my blood start to boil my hands start to twitch and i suddenly get really hot	love
15774	im still using blogger to follow other blogs but i like livejournals feature of enabling private posts so i can keep just one journal without feeling inhibited about writing things i dont want to publish on the net	fear
12195	i don t know how to feel any other way about losing someone who feels like a member of my family than heartbroken	sadness
490	i am feeling more energetic more alive happier than i have in a long time	joy
14950	i usually just feel aggravated with the unprofessional attitude of the rest of the cast	anger
812	i allowed myself to eat foods that i know bother me because after all since i feel awful it may as well have come as a direct result of eating something i enjoy	sadness
4465	i hate feeling dumb i hate people who make me feel dumb or like i am being a baby	sadness
325	i feel so frightened at the thought of opening up my heart	fear
8482	i love the latter for their smooth feel and delicious flavours not to mention their awesome glossy appearance	joy
7890	i am also posting this because i am trying to work on the writing i want my students to feel passionate about	joy

Extrait du dataset.

❖ Méthodes d’import des données

Au vu des données nécessaires à l’entraînement du modèle retenu, la constitution d’une base de données analytique n’est apparue ni utile, ni pertinente.

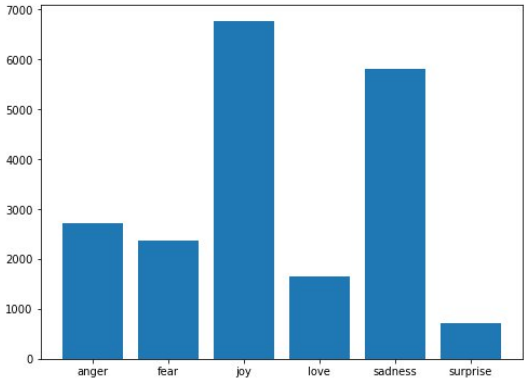
Les données sont importées directement depuis les fichiers **.csv** qui constituent l’ensemble du dataset. Bien que les données soient déjà divisées en plusieurs jeux dédiés aux différentes étapes de la préparation d’un modèle (train, test, validation), il a tout de même été décidé de les réunir en seul **DataFrame**, afin de simplifier l’analyse, les traitements et de s’assurer d’une correcte répartition des divers labels entre ces jeux.

❖ Analyse exploratoire

Nous disposons d’un dataset composé d’un corpus de 20 000 observations constituées de phrases en anglais et labellisées avec une émotion, parmi les six représentées. La distribution de ces labels n’étant pas homogène, cela sera peut-être problématique pour la détection de celles les moins représentés. On veillera à les répartir équitablement dans les différents set de données (train, test et val). L’argument **stratify** de la fonction **train_test_split()** de **Scikit-Learn** rend l’opération très simple.

```
pd.pivot_table(data, index="label", aggfunc="count")
```

label	text
anger	2709
fear	2373
joy	6761
love	1641
sadness	5797
surprise	719



Graphique représentant la distribution des catégories.

On remarque également que certaines phrases apparaissent en double, cependant, une seule se répète avec le même label. Il est décidé de garder les autres en l’état et de laisser le modèle s’arranger avec. Certaines phrases pouvant avoir plusieurs sens et être porteuses d’émotions différentes.

❖ Nettoyage et tokenisation

Étape souvent essentielle pour tout entraînement efficace d’un modèle de Machine Learning, le prétraitement des données, dans notre cas, se résume à supprimer une observation dupliquée. Les phrases quant à elles, sont déjà épurées de toute ponctuation et sans majuscules. Le nettoyage du texte sera inutile.

```
origin_text :    i am a simple sent
tokenized :      [101, 1045, 2572, 1037, 3722, 2741, 102]
```

La tokenisation, qui consiste à subdiviser les phrases en sous-ensembles pour les encoder vers un format numérique, afin de le rendre recevable par le modèle, est réalisée par une classe dédiée au modèle retenu (voir ci-après).

```
bert_tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)

input_ids=[]
attention_masks=[]
for sent in sentences:
    bert_inp = bert_tokenizer.encode_plus(
        sent,
        add_special_tokens = True,
        max_length = 150,
        pad_to_max_length = True,
        return_attention_mask = True
    )
    input_ids.append(bert_inp['input_ids'])
    attention_masks.append(bert_inp['attention_mask'])
```

Tokenisation des phrases d'entraînement.

➢ Modèle

Détecter une émotion émanant d'un texte parmi un panel prédéfini, est une tâche de classification. Le **Machine Learning** nous offre un vaste choix d'algorithmes pouvant parvenir à ce type de résultat (Naïve Bayes, Support Vector Machines, ...). Notre choix, se portera sur le **Deep Learning**. Cette approche, qui s'inspire du fonctionnement d'un cerveau, repose sur une architecture de neurones artificiels interagissant les uns avec les autres pour produire une sortie qui au fil des itérations, s'approche au plus près du résultat attendu.

❖ Transfer Learning

Le **transfert Learning** consiste à transposer les connaissances d'un modèle entraîné sur un grand nombre de données à notre propre modèle. Nous pouvons donc nous appuyer sur des connaissances généralistes pour nous spécialiser sur notre cas d'usage. Cela garantit généralement une bonne efficacité à moindre coût (en terme de temps, de puissance de calcul et de données nécessaires). C'est cette spécificité, qui fait pencher le choix en faveur du Deep Learning.

❖ BERT

Nous avons retenu **BERT** (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers**)**, le modèle de Google pour accomplir les prédictions. Publié en 2018, il a été entraîné à partir de texte de **wikipedia**.

Les **Transformers** sont des modèles qui reposent sur une architecture composée d'**encodeurs** et de **décodeurs** qui ont la particularité d'avoir une couche de "**self-attention**" ayant pour rôle de garder l'interdépendance des mots dans la représentation. Dans le cas de BERT, cette interdépendance est bidirectionnelle, elle concerne aussi bien les tokens précédents que les suivants.

BERT implémente une classe dédiée à la **tokenisation** qui à la différence de la tokenisation classique (généralement associée à de la stemmatisation ou de la lemmatisation), traite les entrées avec un algorithme de type **WordPiece** et les décomposes en "*sous-mots*". Les mots

les plus fréquent sont conservé tels quels, tandis que ceux apparaissant moins souvent, sont divisé en sous-ensemble.

```
bert_model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=num_classes)

loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')
optimizer = tf.keras.optimizers.Adam(learning_rate=2e-5,epsilon=1e-08)

bert_model.compile(loss=loss,optimizer=optimizer,metrics=[metric])
```

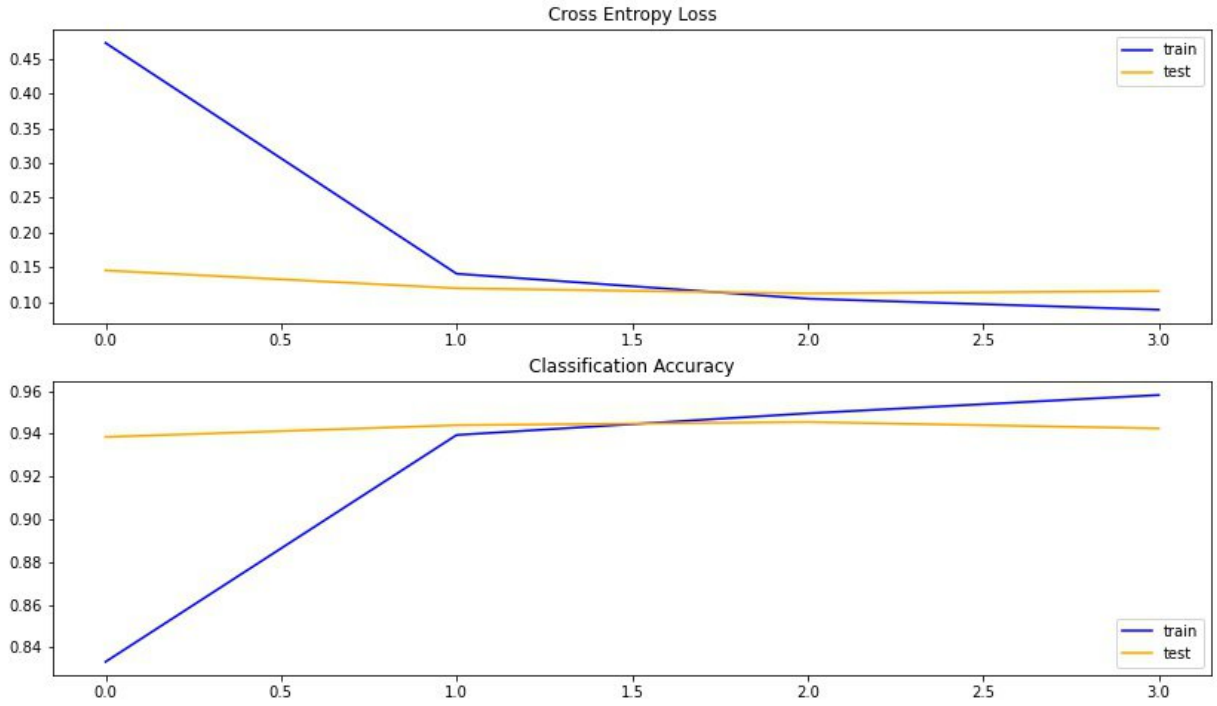
Définition du modèle.

❖ Autres outils

- **Tensorflow** et **Keras** pour la définition et l'entraînement du modèle
- **Sklearn** pour ses métriques et méthodes de split
- **Pandas** et **Numpy** pour la manipulation et la préparation des données
- **Matplotlib** pour les visualisations
- **Pickle** pour les sauvegardes intermédiaires

❖ Résultat

Après une longue phase de près de dix heures, durant laquelle quatre cycles d'entraînement (epoch) ont parcouru l'ensemble des données du jeu "**train**" pour chercher à s'améliorer tout en s'auto-évaluant grâce au jeu de "**validation**", le modèle a été évalué sur le jeu de "**test**".



Visualisation de l'évolution des scores au fil des epochs réalisée avec **Matplotlib** .

Explication des métriques :

- **Précision** : Nombre des bien prédits par rapport à tous ceux prédits comme étant de cette classe.
- **Rappel** (recall) : Nombre des bien prédits par rapport à tous ceux réellement de cette classe.
- **f1-score** : Combinaison entre la précision et le rappel, les faux négatifs ne sont pas pris en compte. Ce qui dans le cas d'un problème avec des classes déséquilibrées comme c'est le cas ici, peut s'avérer plus révélateur que l'accuracy.

Analyse des résultats :

Comme l'on pouvait s'y attendre, les classes, la moins représentées dans le dataset d'entraînement sont celles dont le f1-score est le plus faible, ces émotions sont donc moins bien reconnues.

F1 score 0.934
Classification Report

	precision	recall	f1-score	support
sadness	0.97	0.98	0.98	579
anger	0.95	0.93	0.94	271
love	0.77	0.91	0.83	164
surprise	0.79	0.81	0.80	72
fear	0.93	0.91	0.92	238
joy	0.96	0.92	0.94	676
accuracy			0.93	2000
macro avg	0.90	0.91	0.90	2000
weighted avg	0.94	0.93	0.93	2000

Score obtenu avec le jeu de données de test .

❖ Sauvegarde

Grâce aux **callbacks** définis, entre chaque epoch (cycle d'entraînement qui itère sur l'ensemble du jeu de données "**train**"), le modèle est sauvegardé si ses scores à ce moment-là sont les meilleurs. On évite ainsi les problèmes liés au sur-entraînement. Seuls les poids, qui ont étés modifiés pour s'adapter à notre tâche sont enregistrés.

4. L’application

L’application propose de détecter les émotions, soit à partir de texte libre entré directement par l’utilisateur, soit dans les avis laissés par les clients sur différentes plateformes (AirBnB ou Amazon). Dans ce dernier cas, l’URL saisie est “*scrapée”* afin de récupérer le texte des commentaires.

Ces textes sont ensuite soumis à un algorithme d’I.A. chargé d’en extraire les émotions. Puis les résultats sont présentés.

Bien que l’application soit accessible sans restrictions, l’utilisateur peut choisir de se créer un compte, dans ce cas, ses requêtes seront enregistrées dans une base de données et pourront être consultées ultérieurement.

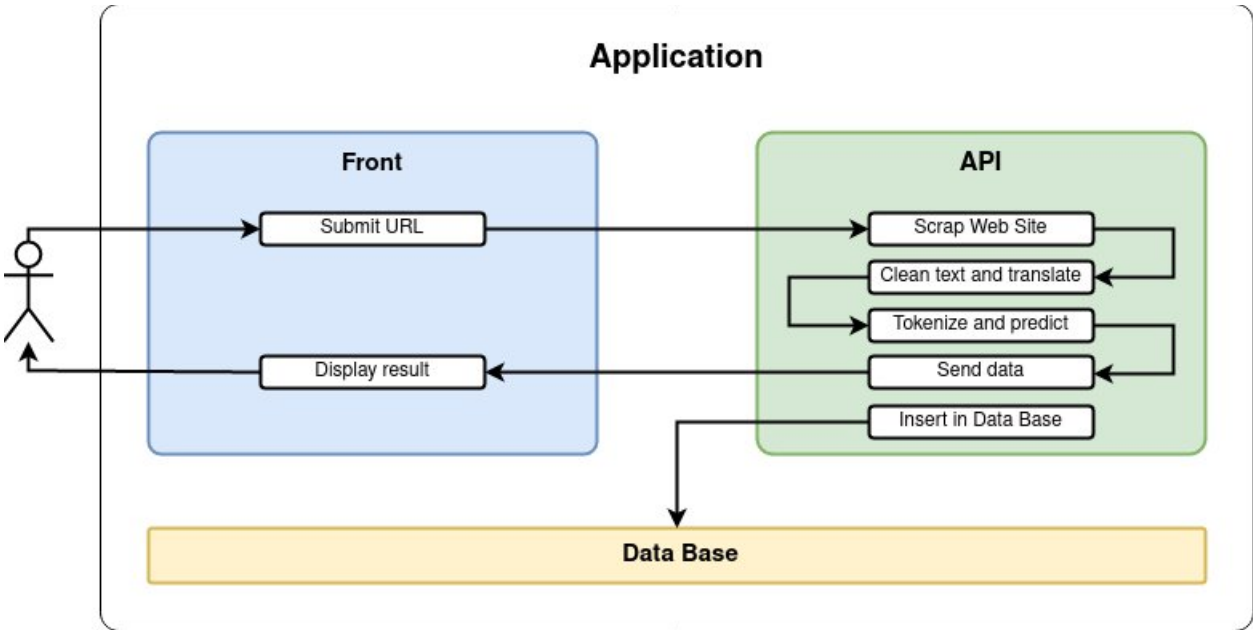
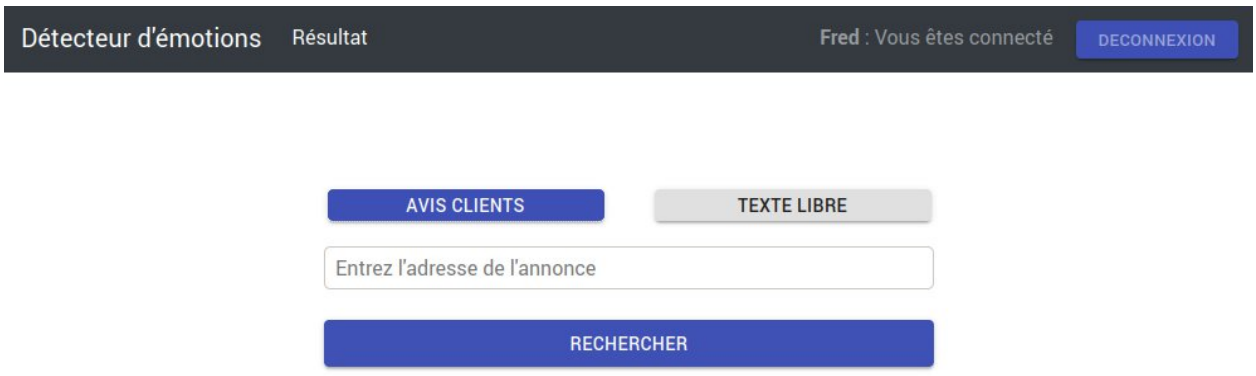


Schéma fonctionnel du processus principal de l’application.



Capture de l’écran d’accueil.

Le projet s’articule autour d’une **API** chargée d’effectuer les différents contrôles, traitements et interactions avec la base de données, d’une **interface web** permettant à l’utilisateur de gérer

son compte, de passer ses requêtes et d'en consulter les résultats, ainsi que d'une **base de données** stockant les informations des usagers.

Arborescence du projet.

➢ API

Réalisée en python grâce au micro framework web **Flask**, cette interface va permettre d'exécuter la logique au cœur de l'application. En charge du traitement des requêtes, des contrôles de sécurité ou de l'enregistrement dans la base de données, c'est la partie **Back-End**, qui sert les données au client.

Avoir un format de sortie des données standardisé en **Json** simplifie leur manipulation par la partie **Front-End** qui en retour se plie à cette norme et communique ses données dans le même format.

❖ Authentification et sécurité

L'authentification de l'utilisateur, se fait au moyen d'un **Token** et de la librairie "**flask_jwt_extended**". Elle fournit les méthodes pour générer un Token composé d'une signature, d'un **payload** contenant les données de l'utilisateur et d'une date d'expiration.

***Process** :*

- ✔ ***Lors de sa première connexion, l'usager qui souhaite créer un compte, fournis ses informations qui sont envoyées vers l'API.***
- ✔ ***On contrôle dans la base que ces données soient bien uniques avant de les enregistrer.***
- ✔ ***C'est un Hash du mot de passe qui est persisté dans la base de données.***
- ✔ ***Quand depuis l'interface, un utilisateur demande à se connecter, il fournit son email ainsi que son mot de passe.***
- ✔ ***Ce dernier est à son tour soumis au même algorithme de Hashage pour pouvoir être comparé.***
- ✔ ***Si des données de la base correspondent, un Token contenant les informations de l'utilisateur est généré, puis transmis au Front qui le stocke dans le "localStorage" du navigateur.***
- ✔ ***Tout au long de la navigation, la validité du Token est vérifiée grâce aux fonctions du module "react-jwt".***
- ✔ ***Pour chaque requête envoyée vers l'API, le Token est passé dans le Header.***
- ✔ ***L'API contrôle également la validité du Token reçu avant d'effectuer ses traitements.***

```python
@app.route("/login", methods=["POST"])
def login():
    if request.method == "POST":
        data = json.loads(request.data)

        user = User.query.filter_by(email=data["email"], password=hash_password(data["password"])).first()
        if user == None:
            return jsonify({"msg": "Email ou mot de passe incorrect"}), 401

        access_token = create_access_token(identity={"email": user.email, "username": user.username})
        return jsonify({
            "msg": f"{user.username} : vous êtes bien connecté",
            "access_token" : access_token,
        })
```

Fonction de login.

❖ Import du modèle

Pour importer le modèle précédemment entraîné, le module de classification de **BERT** est chargé depuis la librairie **transformers** de HuggingFace. Puis on lui adjoint les poids du modèle que nous avons spécialisé sur notre tâche. Cette opération étant assez longue, elle est réalisée au démarrage de l'API.

```python
def __load_model(self, path):
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
    optimizer = tf.keras.optimizers.Adam(learning_rate=2e-5,epsilon=1e-08)
    metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')

    model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased',num_labels=len(self.emotions))
    model.compile(loss=loss,optimizer=optimizer, metrics=[metric])
    model.load_weights(path)

    print("================================")
    print("\t  MODEL STARTED")
    print("================================")

    return model
```

Méthode de la classe Model appelée dés l'instanciation.

❖ Interaction avec la base de données

Le module SQLAlchemy spécialement intégré à Flask, est un ORM (object-relational mapping) qui nous facilite la manipulation de toutes les opérations du CRUD (**C**reate, **R**ead, **U**pdate, **D**elete). Les tables sont créées comme des Classes qui définissent les différents champs, leurs types, ainsi que les relations liants ces tables entre elles. Ces entités sont accessibles au travers de méthodes dont l'utilisation reste cohérente avec la syntaxe du langage. De plus, les relations définies permettent de **sérialiser** les données et d'y accéder de façon simplifiée.

```python
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password = db.Column(db.String(120), nullable=False)

    querys = db.relationship("Query", backref="user")

    def __repr__(self):
        return f'<User {self.username}>'
```

Définition de l'entité User.

Lors du premier démarrage, il est nécessaire de créer la base de données. L'application étant conteneurisée avec Docker, il faut se connecter au conteneur à l'aide de cette commande : `$ docker exec -it emotion_detector_api_1 bash` , se rendre dans le dossier /src, ouvrir un shell python, puis :

```
>>> from api import db
>>> db.create_all()
```

❖ Scraping

Lorsqu'un utilisateur saisit une URL, le Front la transmet à l'API au travers d'une requête POST. Après un contrôle de l'adresse, elle est passée à la fonction de scraping dédié au site concerné (AirBnB ou Amazon pour le moment).

C'est grâce au module **Selenium** qu'il nous est possible de récupérer le contenu des commentaires laissés par les clients. Il nous donne accès à un ensemble de méthodes nous permettant de contrôler un navigateur web de façon automatisée. Après s'être connecté à la page demandée, nous sommes capable de rechercher certaines balises HTML et d'en enregistrer le contenu, ou bien de naviguer de pages en pages en interagissant avec ces éléments, pour suivre des liens par exemple. Cela nous permet de parcourir toutes les pages de commentaires quelqu'en soit le nombre. Certains sites limitants le nombre de requêtes soumises, la connexion passe par un proxy qui change régulièrement.

Le résultat de cette collecte est retourné sous forme de liste contenant les divers avis.

```python
def amazon_scraper(url):
    all_reviews = []

    driver.get(url)
    time.sleep(1)

    title = driver.title
    rest_of_data = True
    nb_review = 0
    while rest_of_data:
        reviews = []
        try:
            next_link = driver.find_element_by_class_name("a-last")
            next_link = next_link.find_element_by_tag_name("a")
            next_link = next_link.get_attribute("href")
        except:
            rest_of_data = False

        reviews = driver.find_elements_by_class_name("review-text-content")
        for review in reviews:
            all_reviews.append(review.text)
            nb_review += 1

        if rest_of_data:
            driver.get(next_link)
        time.sleep(1)

    driver.quit()
    return all_reviews, title
```

Code permettant de récupérer les avis d'Amazon.

❖ Traitement des requêtes

Les avis retournés par la fonction de scraping sont ensuite traduits de leur langue d'origine vers l'anglais afin de pouvoir être traités convenablement par le modèle qui a été entraîné sur des textes dans cette langue. Pour cela, nous nous servons de la librairie **googletrans** qui se connecte à l'API Google translate.

> ***Choix stratégique :***
>
> ***Sans compte payant, Google limite le nombre de requêtes pouvant être soumises (un peu moins de deux cents par heure), il a donc été décidé de soumettre tous les avis concaténés afin d'économiser nos crédits.***

Cette version anglophone des commentaires est scindée au niveau des balises <END> délimitant les différents avis puis transformé en token (texte encodé sous forme numérique) par la même méthode ayant servi à préparer les données avant l'entraînement du réseau de neurones. Ainsi pré-traités, les avis peuvent êtres soumis à la fonction de prédiction implémenté dans la classe Model.

Les résultats retournés sont normalisés, c'est-à-dire que les probabilités pour chaque émotions sont ramenées entre 0 et 1 pour une meilleure interprétation.

❖ Liste des endpoints

- **POST** /join : Reçois les informations de l'utilisateur et après un contrôle, les enregistre dans la base de données et retourne un Token
- **POST** /login : Contrôle l'existence du compte et retourne un Token.
- **GET** /account : Retourne la liste des requêtes archivées pour l'utilisateur logué.
- **GET** /detail/<query_id> : Retourne la liste des résultats pour la requête demandée.
- **DELETE** /delete/<query_id> : Supprime la requête demandée et les résultats associés.
- **POST** /parse_text : Traite et analyse le texte fournis et retourne la prédiction.
- **POST** /parse_url : Scrape les avis de l'URL fournie, les traites, analyses et retourne la prédiction.

➢ Base de données

Les données des utilisateurs enregistrés sont stockées et organisées dans une base de type relationnelle. Le choix a été fait de travailler avec **MariaDB**, qui est un **Système de Gestion de Base de Données Relationnel** (SGBDR) créé suite au rachat de MYSQL et dont le mode de gouvernance par le biais d'une fondation, lui assure de rester un projet libre.

La plupart des SGBDR utilisent le langage **SQL** pour interagir avec les données présentes dans la base, cependant, la librairie SQLAlchemy que utilisée dans le projet, met à notre disposition un ensemble de méthodes couvrant toutes les opérations CRUD, et sérialise les données en fonction des relations définies, simplifiant le recours aux jonctions entre tables.

Exemple d'équivalence entre une requête SQL et une autre passée avec SQLAlchemy :

- SQLAlchemy : ***User.query.filter_by(username='fred ').first()***
- SQL : ***SELECT * FROM user WHERE username = "fred";***

```python
# Control if user's data already exist
old_username = User.query.filter_by(username=data["username"]).all()
old_email = User.query.filter_by(email=data["email"]).all()
if len(old_username) > 0 or len(old_email) > 0:
    return jsonify({"msg": "Erreur ce nom ou cet email éxiste déjà !"}), 403

# Insert user's data in database
user = User(
    username=data["username"],
    email=data["email"],
    password=hash_password(data["password"])
)
db.session.add(user)
db.session.commit()
```

Création d'un nouvel utilisateur avec SQLAlchemy.

❖ Modèle relationnel

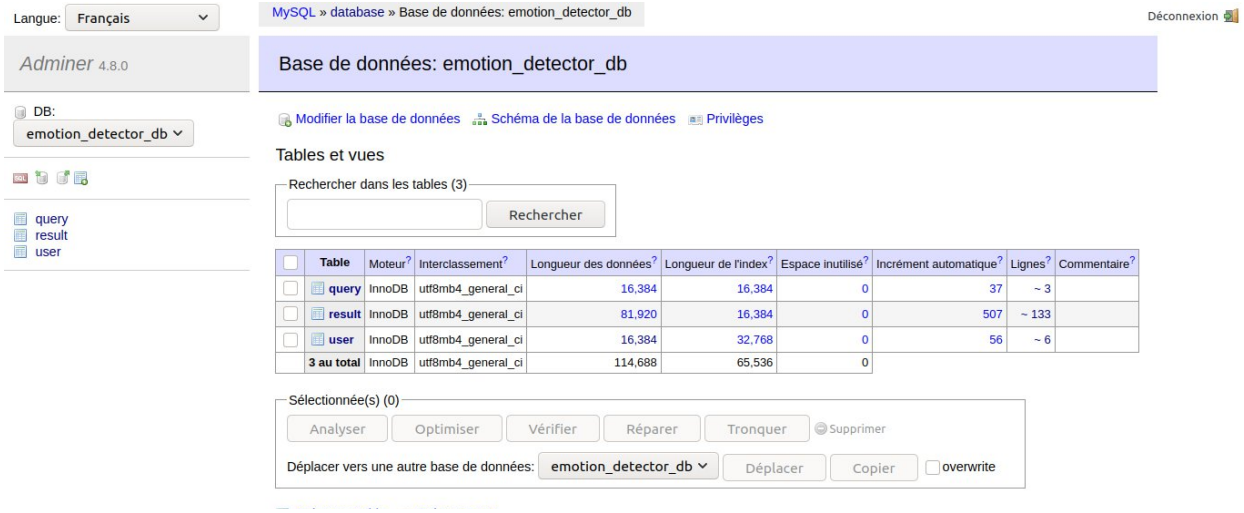
La base de données choisie, de type relationnel, respecte les principes **A.C.I.D.** qui garantissent que les transactions réalisées sont fiables.

- L'**A**tomicité signifie que la transaction est complète ou supprime toute trace de celle-ci.

- La **C**ohérence assure que toute transaction doit être valide et ne pas altérer le bon fonctionnement du système.
- L'**I**solation indique que toutes les transactions sont indépendantes les unes des autres.
- La **D**urabilité, elle, garantie la persistance des données dans la base même en cas de panne survenant immédiatement après la requête.

La base de données contient trois **tables** (user, query et result) représentées sous forme de tableau à deux dimensions, contenant elles-mêmes plusieurs **attributs** dont un identifiant unique servant de **clé primaire** et permettant de cibler avec certitude une entrée de cette table (ce champ s'auto-incrémente à chaque nouvel **enregistrement**).

Les tables "query" et "result", quant à elles, contiennent respectivement une référence vers l'identifiant des tables "user" et "query". Ces **clés étrangères** permettent de relier l'utilisateur ayant émis la demande à cette dernière et d'y rattacher les résultats.



Base de données vue depuis l'interface d'Adminer.

❖ Cardinalités

Symbolisant le nombre minimum où maximum de relations possibles entre les tables, les cardinalités sont indiquées sur le schéma de la base.

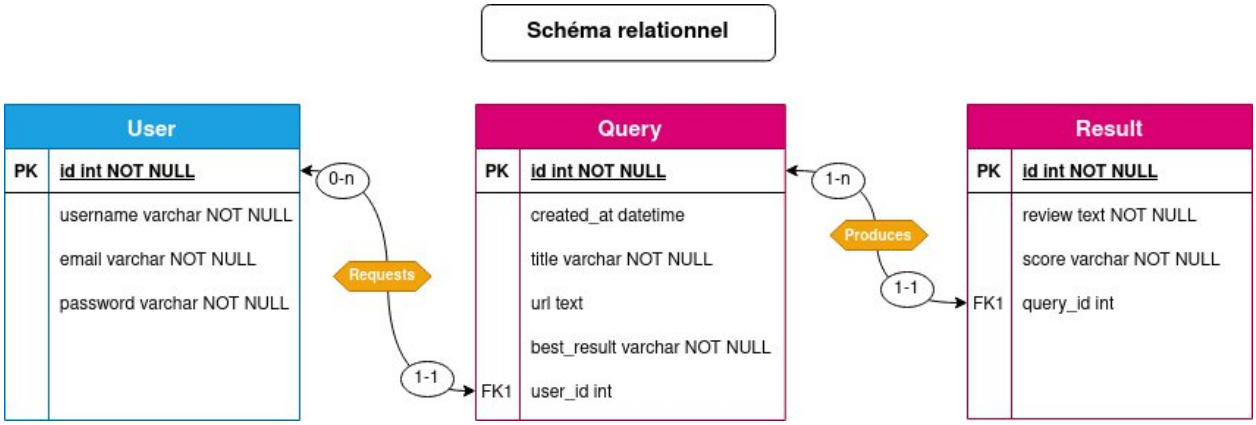


Schéma relationnel de la base de données.

❖ Sauvegardes

Les données de la base sont sauvegardées au moyen de **restic** qui réalise des sauvegardes incrémentielles et chiffrées et de **cron** pour automatiser le lancement de la tâche de façon

périodique. Le conteneur “database” contenant l’image MariaDB étant lié avec un dossier de l’hôte, les données sont automatiquement copiées dans celui-ci. C’est ce dossier qui sera sauvegardé.

- Il faut définir un dossier de sauvegarde, grâce à cette commande :

```
$ restic -r /dossier/de/sauvegarde init
```

- Après avoir rentré un mot de passe, le dossier est prêt à abriter les sauvegardes.
- Pour paramétrer cron, éditons le fichier crontab:

```
$ crontab -e
```

- Configurons la tâche et sa récurrence (elle s’exécutera tous les jours à 3 heures du matin) :

```
* 3 * * * restic -r
/chemin/vers/le/dossier/contenant/les/sauvegardes backup
/chemin/vers/le/dossier/data
```

- Le mot de passe doit être défini en tant que variable d’environnement (***RESTIC_PASSWORD***) pour être pris en compte.
- Pour contrôler les sauvegardes :

```
$ restic -r /dossier/de/sauvegarde snapshots
```

❖ Exemple de requête SQLAlchemy / SQL

- **Rechercher un utilisateur :**
  SQLAlchemy : User.query.filter_by(username=”fred”).first()
  SQL : SELECT * FROM user WHERE username = "fred";

- **Insérer un utilisateur :**
  SQLAlchemy : user = User(
  username=”fred”,
  email=”fred@email.com”,
  password=hash_password(“secret”)
)
  db.session.add(user)
  db.session.commit()
  SQL : INSERT INTO user VALUES (6, "fred ", "fred @email.com", "hash_du_password");

- **Lister les résultats d’une requête :**
  SQLAlchemy : query = Query.query.filter_by(id=1).first()
  {r.review: r.score for r in query.results}
  SQL : SELECT * FROM result LEFT JOIN query ON result.query_id = query.id WHERE query.id = 1;

- **Supprimer une requête et ses résultats :**
  SQLAlchemy : query = Query.query.filter_by(id=1).first()
  for result in query.results:
  db.session.delete(result)
  db.session.delete(query)
  db.session.commit()

SQL : DELETE FROM result WHERE query_id = 1;
DELETE FROM query WHERE id = 1;

- **Insérer les données d'une requête :**

```python
# If user is logged
if get_jwt_identity():
    # Insertion in db
    user = User.query.filter_by(username=get_jwt_identity()["username"]).first()
    query = Query(
        title=title,
        url=data["url"],
        best_result=best_result,
        user=user
    )
    db.session.add(query)

    for review, score in detailed_results.items():
        result = Result(
            review=emoji.demojize(review),
            score=json.dumps(score),
            query=query
        )
        db.session.add(result)
    db.session.commit()
```

Insertion des données avec SQLAlchemy

➢ Front

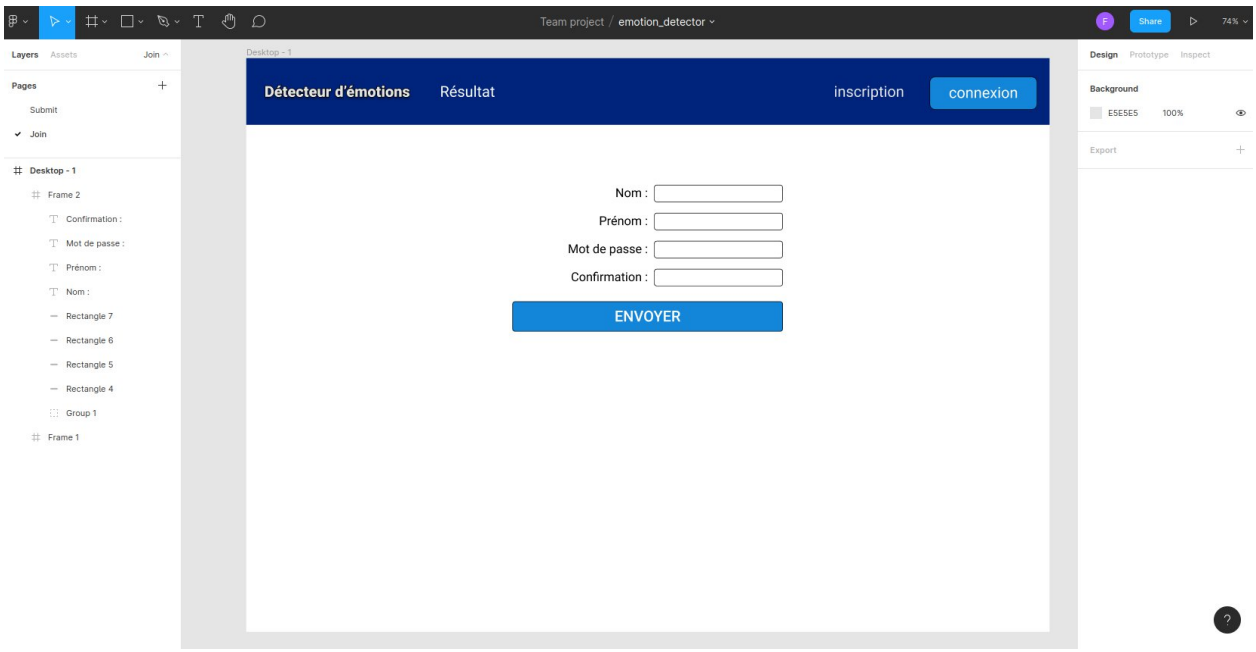
Les utilisateurs auront accès à l'application à travers un site web construit avec le Framework **React**, basé sur le langage **Javascript**, cette technologie permet de mettre en œuvre une interface dynamique et réactive.

❖ Description des pages

- **"/"** : Page d'accueil, permet à l'utilisateur de soumettre sa demande dés son arrivée sur le site.
- **"/result"** : Page dédiée à l'affichage des résultats.
- **"/signin"** : Page d'inscription.
- **"/login"** : Page de connexion.
- **"/account "** : Page permettant de consulter les archives des requêtes effectuées et leurs résultats.

❖ Maquette

Les maquettes du design ont étés réalisées avec **figma** et ont permis de penser l'**UI** (User Interface) en cohérence avec le parcours utilisateur imaginé et les spécifications fonctionnelles.



Capture d'écran de la maquette de la page d'inscription.

➢ Tests

Afin de garantir la non-régression de l'application, des **tests unitaires** couvrent l'ensemble des fonctions et des méthodes de classe de l'API. Leur exécution régulière peut nous alerter sur des dysfonctionnements survenus suite à l'ajout d'une fonctionnalité ou d'une refactorisation du code par exemple. L'identification du problème et sa localisation dans le code s'en trouveront facilités.

```
============================= test session starts ==============================
platform linux -- Python 3.7.5, pytest-6.0.2, py-1.9.0, pluggy-0.13.1
rootdir: /home/fred/Formation/Certifs/detecteur_emotion/emotion_detector/API
plugins: pylint-0.15.0, anyio-2.2.0, dash-1.4.1, Faker-4.1.2
collected 11 items

tests/test_class_model.py ....                                           [ 36%]
tests/test_class_preprocess.py .....                                     [ 81%]
tests/test_functions.py ..                                               [100%]

======================== 11 passed, 4 warnings in 18.00s =======================
```

Capture d'écran de l'exécution des tests.

Des **tests fonctionnels**, sont quant à eux effectués manuellement. Cette "recette" sert à s'assurer que le comportement attendu dans certaines circonstances se déroule conformément à nos prévisions.

➢ Déploiement

❖ Docker

Afin de simplifier les opérations de déploiement, l'application est prévue pour fonctionner avec **Docker**, une simple commande ***$ docker-compose up*** suffit à démarrer tous les conteneurs.

Outre la facilité de lancement, Docker assure également une interopérabilité accrue, notre application pouvant être exécutée sur n'importe quelle machine disposant de Docker et Docker-compose. Chaque partie de l'application tournant dans son propre conteneur, la sécurité s'en trouve elle aussi renforcée par ce cloisonnement.

Des fichiers "***Dockerfile***" paramètrent l'API et le Front en définissant les images à partir desquelles le conteneur sera construit, en copiant les dossiers requis, puis, se chargent des installations d'applications tierces (firefox utilisé par Selenium) et des librairies nécessaires, avant d'exécuter les scripts.

Le tout est orchestré par le fichier "***Docker-compose.yml***" se trouvant à la racine de l'application. Il définit également le conteneur de la base de données construit sur une image MariaDB, celui d' "***adminer***" offrant un accès à la base de données via une interface web et crée le réseau qui les relit. Pour chaque conteneur, seuls les ports strictement utiles sont ouverts et les volumes sont liés à des dossiers sur la machine hôte. Cela permet entre autres de d'assurer la persistance des données indépendamment de la vie du conteneur.

```yaml
services:
  api:
    build: API/.
    ports:
      - 5000:5000
    networks:
      - emotion_detector_network

  front:
    build: Front/.
    ports:
      - 3000:3000
    networks:
      - emotion_detector_network
```

Extrait du fichier docker-compose.yml.

➢ Monitoring

Pour suivre l'activité de l'application et s'assurer de sa disponibilité, plusieurs services tiers sont sollicités.

- **UptimeRobot**
  Envois régulièrement des ping vers l'adresse fournie et contrôle ses réponses. Dans le cas d'une indisponibilité, une alerte est envoyée par mail.

Hi,

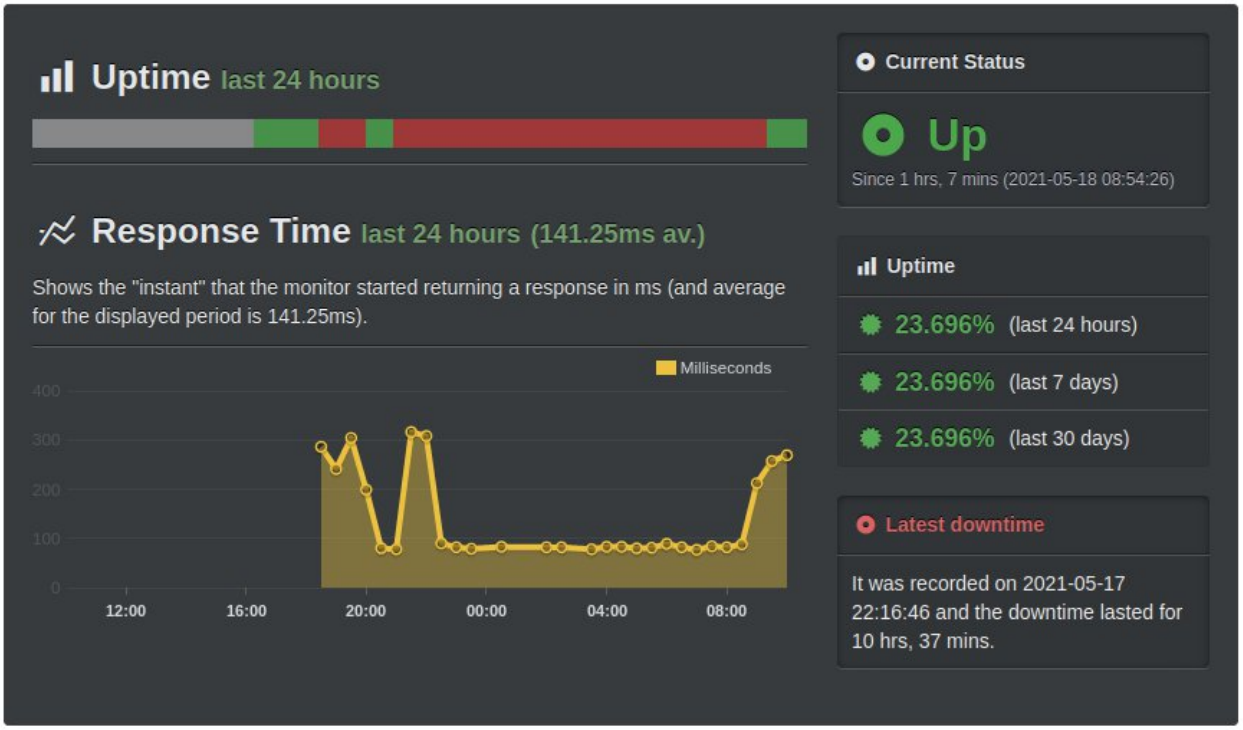
The monitor emotion-detector (http://7db907d97bc7.ngrok.io) is currently DOWN (HTTP 404 - Not Found)

Event timestamp: 2021-05-17 20:10:23 UTC+2

Uptime Robot will alert you when it is back up.

P.S. Get notified 5x faster (1-min checks + SSL monitoring) for just ~~$15~~ $7/month (50% off limited offer). Check uptimerobot.com/upgrade.

Email d'alerte envoyé par UptimeRobot.

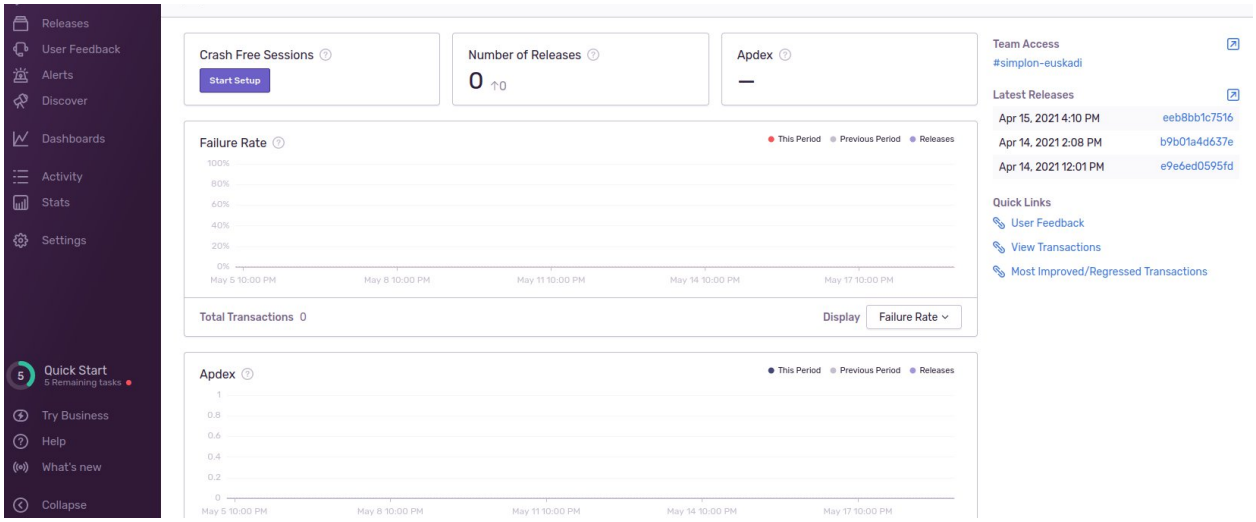


Latest Events (up, down, start, pause) Export Logs

Event	Date-Time	Reason	Duration
Up	2021-05-18 08:54:26	OK (200)	1 hrs, 7 mins
Down	2021-05-17 22:16:46	Not Found (404)	10 hrs, 37 mins

Dashboard de UptimeRobot.

- **Sentry**
  Fait remonter les erreurs, surveille les performances et bien d'autres choses. Intégré au cœur de l'application, il donne accès à ces données au travers d'une interface web et peut nous envoyer des alertes par différents moyens.



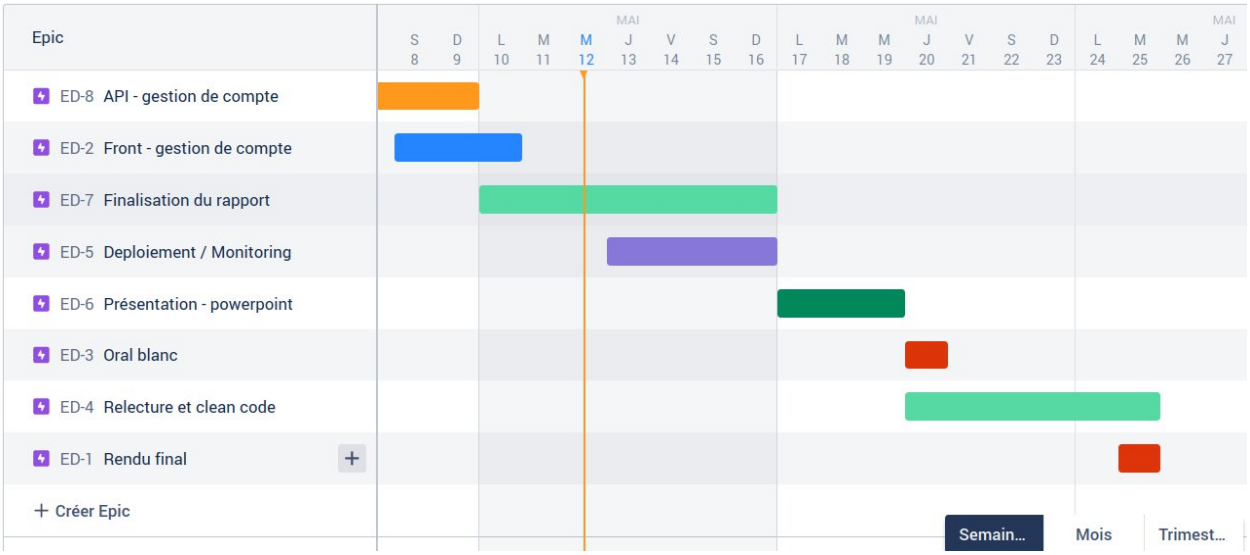
Dashboard de Sentry .

5. Gestion de projet

Le projet a été mené en suivant les bonnes pratiques de la profession et en mettant au maximum en œuvre les méthodes **Agiles**.

➤ Rétro planning

Afin de pouvoir être sûr de tenir les délais, un rétro planning partant de la date de restitution finale à servis à définir les principales étapes du projet et leurs périodes de réalisation.

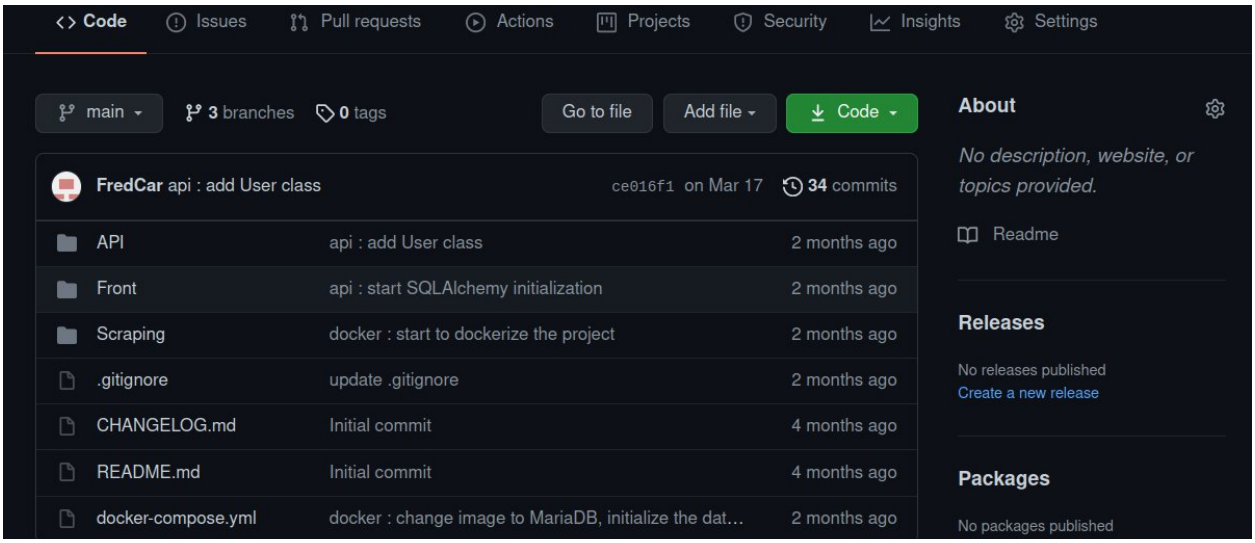


Capture d'écran du rétro planning.

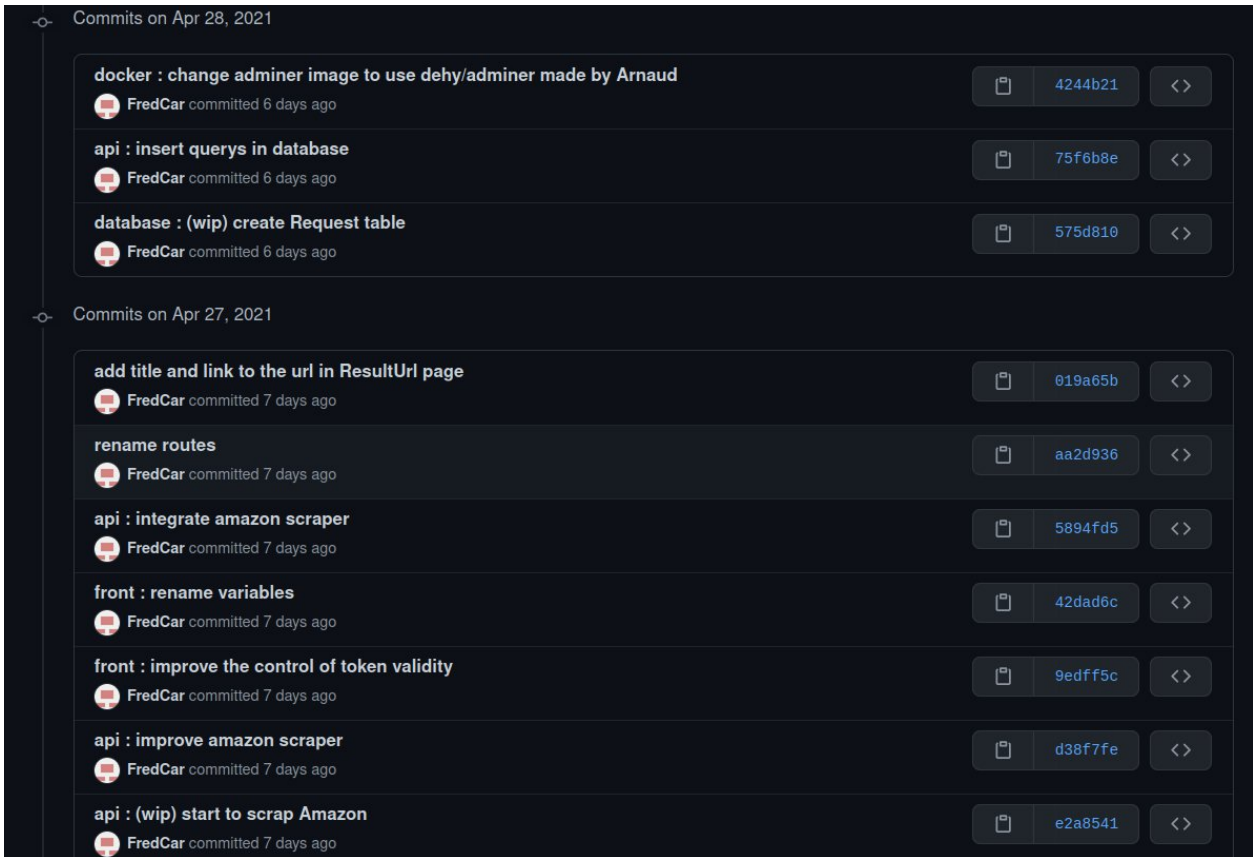
➤ Versionning avec Git et Github

Des sauvegardes fréquentes ont étés faites avec **Git** et envoyées vers un dépôt distant hébergé par **Github**. Ces commits effectués à des étapes clé de l'avancée du projet permettent également de pouvoir revenir à une version antérieure du code, cela peut s'avérer utile afin de debuguer des anomalies.

Le système des branches, quant à lui permet de travailler sur une fonctionnalité sans affecter le bon fonctionnement des autres branches de l'application.



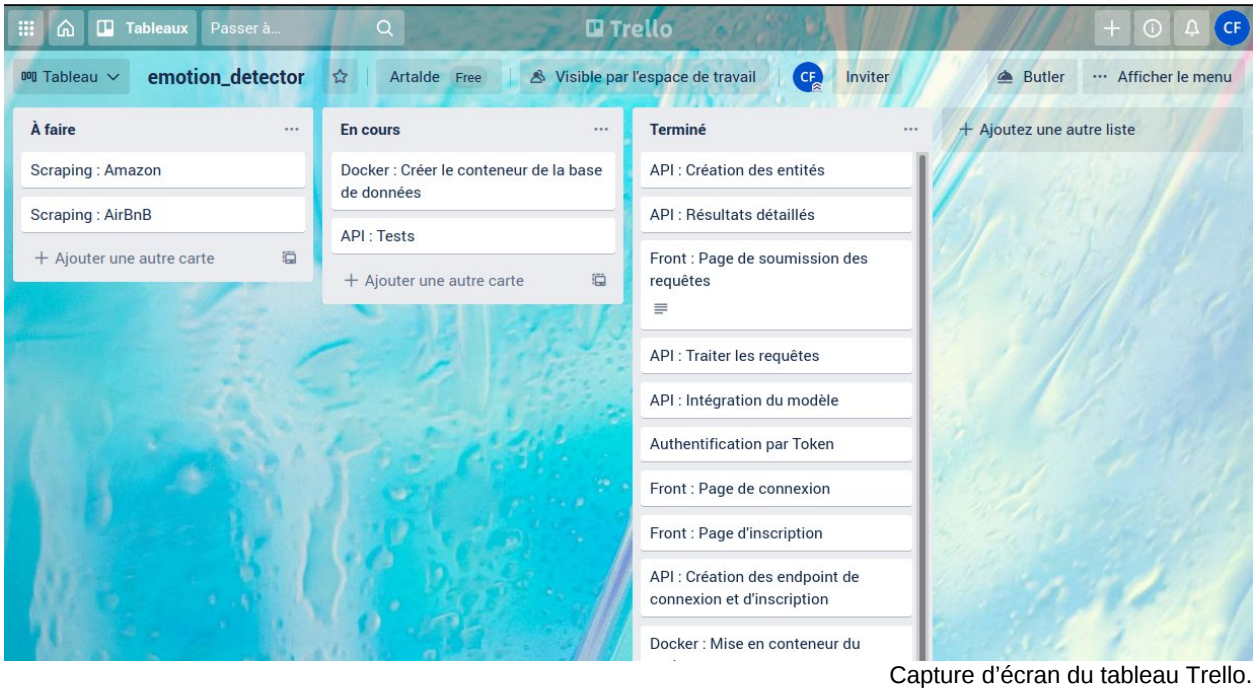
Capture d'écran du repository sur Github.



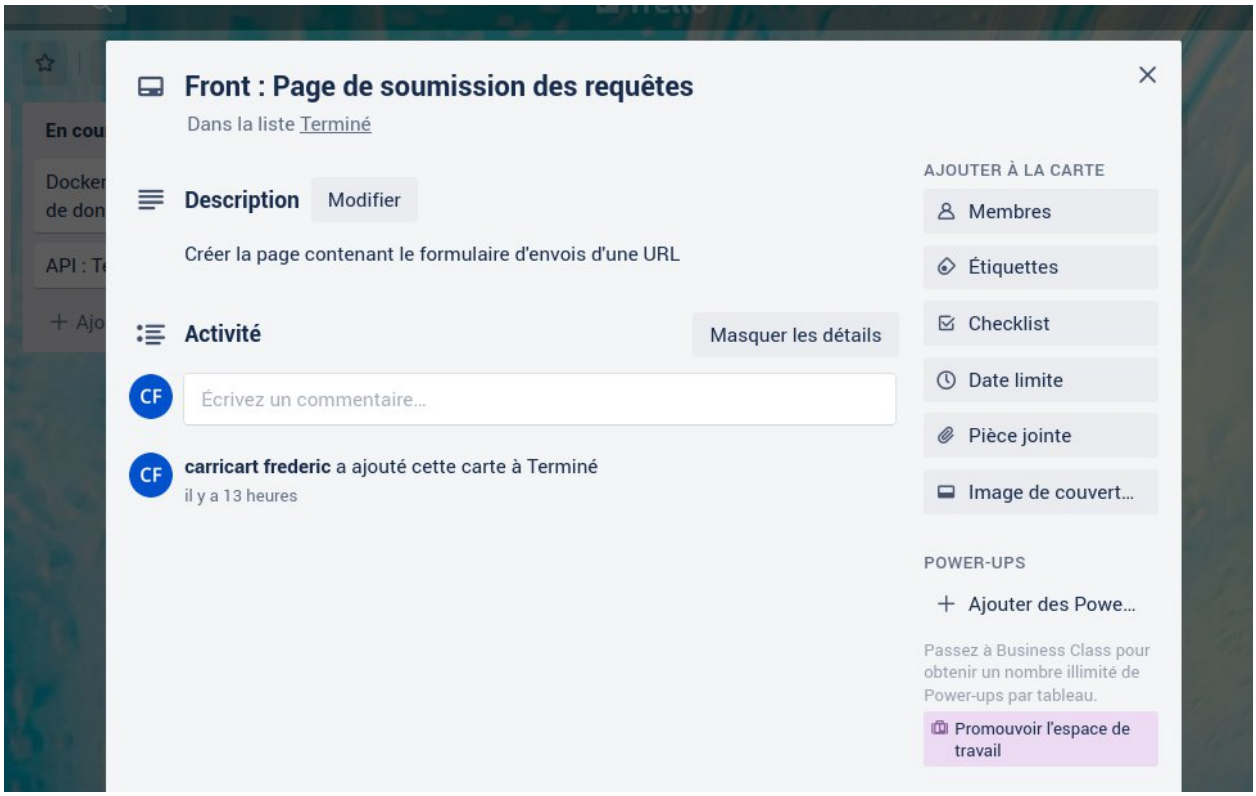
Vue sur les commits dans Github.

➢ Trello

Le suivi des tâches à effectuer s'est organisé sur Trello, un service proposant de suivre l'avancement du travail via des cartes pouvant êtres déplacées d'une colonne à une autre afin de visualiser la progression. On peut affecter une carte à tel ou tel membre de l'équipe, fixer un délai, ou bien encore discuter de cette tâche précisément dans les commentaires.



Capture d'écran du tableau Trello.



Capture d'écran d'une tâche dans Trello.

➢ Compte rendu d'avancement

Bien que le projet ait été mené seul, des comptes rendu d'avancement s'adressant à un **Product Owner** imaginaire ont étés simulés. Voici ce que cela aurait pu donner :

- **Date : Lundi 10 mai**
  Bonjour,
  La semaine passée, j'ai terminé la partie gestion de compte, aussi bien sur l'API que le Front. J'ai rencontré quelques difficultés avec un composant React dont le props ne se rechargeais pas comme attendu. Le problème a été résolu grâce aux conseils de Pierre qui m'a aidé à identifier l'origine du bug.
  Concernant le déploiement, tâche que j'attaque cette semaine, peux-tu me transmettre les identifiants du compte AWS ? Sans cela, la tâche risque de prendre du retard.
  Merci.

- **Date : Lundi 3 mai**
  Salut,
  Cette semaine a été consacrée à la résolution de bug, à du nettoyage et de l'optimisation du code. Les scrapers m'ont demandé un peu de temps et finalement, la mise en place de proxy règle le problème de connexion refusée. La rédaction du rapport a aussi débuté.
  Pour cette semaine, je dois mettre en place le système d'authentification par Token et créer la partie gestion de compte de l'utilisateur.
  Bon courage à tous !

- **Date : Lundi 26 avril**
  Bonjour,
  Le Front commence enfin à ressembler aux maquettes, l'API se structure avec de nouveaux Endpoints et la base données est correctement configurée. Bref, de grosses avancées ont eu lieu cette semaine.
  La recette réalisée en fin de semaine (merci à Mathilde, Xavier et Anne) a permis de mettre en évidence quelques bugs, certains connus, mais d'autre qui étés complètements passés sous les radars. Leurs résolutions devraient m'occuper un moment, puis j'ai aussi prévu d'attaquer le rapport.
  Je reste dispos si besoin !

6. Améliorations possibles

- Ajouter des sites cibles en ajoutant des scripts de scraping.
- Modèle entraîné en français (nécessite de constituer un dataset).
- Lever les limitations en prenant un compte payant Google.
- Donner l'occasion à l'utilisateur de labelliser des avis. Cela pourrait permettre de ré-entraîner le modèle avec des données plus pertinentes.
- Les phrases ayant servi à l'entraînement sont sans doute trop orientées autour des émotions et ont peut à voir avec le contexte d'avis sur des produits ou services. Rendant les prédictions peu précises.
- Optimiser le site React (système d'alerte, ...)