Spécifications fonctionnelles et techniques - Extension Chrome avec API d'Analyse de Sentiments (VADER + AWS EC2)

1. Introduction

1.1 Contexte

Le projet consiste à développer une extension Chrome permettant de détecter en temps réel la négativité des messages saisis sur les réseaux sociaux. L'analyse de sentiment sera réalisée via une API basée sur le modèle **VADER**, hébergée sur **AWS EC2 (Ubuntu)**.

1.2 Objectifs du projet

1. **API d'analyse de sentiment** avec VADER, déployée sur **AWS EC2** pour analyser le texte envoyé par l'extension.
2. **Extension Chrome** qui capture les messages en temps réel sur les réseaux sociaux, les envoie à l'API pour analyse, et affiche une notification si un texte négatif est détecté.

1.3 Périmètre du projet

- Création de l'API avec FastAPI et VADER.
- Extension Chrome pour capturer et analyser les textes en temps réel.
- Interaction fluide entre l'extension et l'API.

2. Description fonctionnelle

2.1 Fonctionnalités principales

1. **Extension Chrome** :
    - **Capture en temps réel** des textes saisis dans les zones de texte (Facebook, Twitter).
    - **Envoi des textes à l'API** pour analyse via une requête HTTP POST.
    - **Affichage d'une notification** lorsque l'API détecte du contenu négatif.
2. **API d'analyse de sentiment (VADER)** :
    - **Analyse des textes** à l'aide du modèle VADER et retour d'un score de sentiment via un endpoint REST.
    - **Retour des résultats** avec un format JSON indiquant le score de polarité.

2.2 Scénarios d'utilisation

- **Rédaction de commentaire Facebook** : Le texte est capturé et analysé en temps réel. Si du contenu négatif est détecté, une notification est envoyée à l'utilisateur avant la publication du message.
- **Rédaction de tweet** : La même logique s'applique, avec une notification suggérant la reformulation.

2.3 Cas d'utilisation non pris en charge

- Les messages cryptés et privés ne sont pas analysés.
- L'analyse multilingue n'est pas supportée.

3. Description technique

3.1 Architecture générale

3.1.1 Composants :

1. **API d'analyse de sentiment** :
    - Hébergée sur **AWS EC2 (Ubuntu)**.
    - Utilise **FastAPI** pour gérer les requêtes REST.
    - Utilise **VADER** pour l'analyse de sentiment.
2. **Extension Chrome** :
    - **Content script** qui capture le texte dans les zones de saisie des réseaux sociaux.
    - **Appel à l'API** via une requête HTTP POST pour analyser le texte.
    - **Affichage des notifications** si le texte est jugé négatif.

3.1.2 Schéma global de l'architecture :

- **Utilisateur** : Tape un message sur une plateforme sociale.
- **Content script** : Capture le texte saisi et l'envoie à l'API pour analyse.
- **API FastAPI** : Reçoit le texte, le traite avec VADER, et renvoie un score de sentiment.
- **Extension Chrome** : Reçoit la réponse et affiche une notification si nécessaire.

3.2 Description détaillée des fichiers

3.2.1 Fichiers de l'API

1. **Fichier** `app.py` :
    - **Rôle** : Fichier principal qui contient la logique de l'API et les endpoints. Il doit gérer la réception des requêtes, l'analyse de sentiment avec VADER, et le retour des résultats.
    - **Méthodes à implémenter** :
        - **POST** `/analyze` : Reçoit du texte, analyse avec VADER, et retourne un score de sentiment.
            - **Entrée** : JSON contenant une clé `text` avec le texte à analyser.
            - **Sortie** : Un objet JSON contenant le score de sentiment retourné par VADER (positif, neutre, négatif).
        - **Modèle de données (input)** :
            - Champ `text` : Représente le texte à analyser. Type : chaîne de caractères (string).
        - **Modèle de données (output)** :
            - `neg` : Score de négativité (float entre 0 et 1).

- `neu` : Score de neutralité (float entre 0 et 1).
- `pos` : Score de positivité (float entre 0 et 1).
- `compound` : Score global de polarité, entre -1 (négatif) et 1 (positif).

2. **Fichier `requirements.txt`** :
   - **Rôle** : Spécifie les dépendances Python nécessaires.
   - **Contenu** :
     - **fastapi** : Framework pour l'API REST.
     - **uvicorn** : Serveur ASGI pour servir l'API.
     - **vaderSentiment** : Librairie pour l'analyse de sentiment.
3. **Fichier `start.sh`** :
   - **Rôle** : Script pour démarrer l'API avec Uvicorn.
   - **Contenu** :
     - Lancer Uvicorn pour démarrer l'API sur le port 8000.

3.2.2 Configuration AWS EC2

1. **Instance EC2 (Ubuntu)** :
   - **Rôle** : Héberger l'API sur AWS.
   - **Étapes** :
     - Créer une instance EC2 (Ubuntu).
     - Installer **Python** et **pip**.
     - Cloner le dépôt de code de l'API.
     - Installer les dépendances via **pip**.
     - Configurer le **pare-feu** pour ouvrir les ports nécessaires (port 8000 pour Uvicorn).
2. **Sécurité** :
   - Configurer un groupe de sécurité pour autoriser le trafic HTTP entrant sur le port 8000.

3.2.3 Fichiers de l'extension Chrome

1. **Fichier `manifest.json`** :
   - **Rôle** : Fichier de configuration de l'extension.
   - **Contenu** :
     - Déclaration des **permissions** : Accès aux pages des réseaux sociaux (Facebook, Twitter).
     - Déclaration du **content script** : Indiquer quel script doit être injecté dans les pages.
     - Permissions pour faire des **requêtes réseau** vers l'API.
2. **Fichier `content.js` (Content Script)** :
   - **Rôle** : Capturer le texte en temps réel et interagir avec l'API d'analyse de sentiment.

- **Méthodes à implémenter** :
  - **Capture du texte** : Surveiller les événements de saisie dans les champs de texte (`<textarea>`, `<input>`).
  - **Envoyer à l'API** : Envoyer le texte à l'API via une requête HTTP POST.
  - **Gérer la réponse de l'API** :
    - Si la réponse contient un score de négativité élevé (ex : `compound < -0.05`), déclencher une notification.

3. **Fichier `popup.html`** :
   - **Rôle** : Interface graphique pour contrôler l'extension (activer/désactiver).
   - **Contenu** :
     - Un bouton bascule (switch ON/OFF) permettant à l'utilisateur de désactiver l'extension.
4. **Fichier `popup.js`** :
   - **Rôle** : Gérer le comportement du popup (activation/désactivation de l'extension).
   - **Méthodes à implémenter** :
     - **Gestion de l'état** : Permettre à l'utilisateur de basculer entre l'activation et la désactivation de l'extension.
     - **Stockage de l'état** : Utiliser **chrome.storage** pour stocker l'état d'activation de l'extension (ON/OFF).
5. **Fichier `background.js` (optionnel)** :
   - **Rôle** : Gérer les événements en arrière-plan, comme la gestion de l'état actif de l'extension entre les onglets.
   - **Méthodes à implémenter** :
     - Maintenir l'état de l'extension activée ou désactivée, même lorsque l'utilisateur change d'onglet.

3.2.4 Notifications

1. **Utilisation de l'API de notifications Chrome** :
   - **Rôle** : Afficher une notification lorsque l'API renvoie une réponse indiquant que le texte est négatif.
   - **Contenu de la notification** :
     - Titre de la notification (ex : "Message négatif détecté").
     - Texte d'explication invitant à reformuler (ex : "Essayez de reformuler votre message pour le rendre plus positif.").

3.3 Modèles de données

1. **Modèle envoyé à l'API** :
   - **Requête HTTP POST** au format JSON.

- Contient une clé `text` avec le texte à analyser.

2. **Modèle retourné par l'API** :
   - **Réponse HTTP** au format JSON.
   - Champs :
     - `neg` : Score de négativité (entre 0 et 1).
     - `neu` : Score de neutralité (entre 0 et 1).
     - `pos` : Score de positivité (entre 0 et 1).
     - `compound` : Score global (entre -1 et 1, où -1 est très négatif, 1 est très positif).

4. Interfaces utilisateur

4.1 Description des interfaces

- **Popup** : Interface utilisateur permettant d'activer ou désactiver l'extension via un interrupteur (ON/OFF).
- **Notifications** : Affichage d'une alerte lorsqu'un texte négatif est détecté.

4.2 Diagrammes des interfaces

- **Popup** : Bascule ON/OFF.
- **Notifications** : Alerte contextuelle avec suggestion de reformulation.

5. Contraintes techniques

5.1 Sécurité et confidentialité

- Aucune donnée personnelle ne doit être stockée par l'API ou l'extension.

5.2 Compatibilité

- **Navigateurs supportés** : Chrome version 90+.
- **Systèmes d'exploitation** : Windows, macOS, Linux.

6. Gestion de projet

6.1 Planning

1. **Jour 1** : Développement de l'API avec FastAPI et VADER.
2. **Jour 2** : Développement de l’extension Chrome.
3. **Jour 3** : Tests d'intégration et déploiement sur AWS EC2.

6.2 Ressources et compétences

- **Développeur back-end** : Expérience avec Python, FastAPI et AWS EC2.
- **Développeur front-end** : Compétences avec JavaScript et Chrome Extensions API.

7. Livrables

7.1 Liste des livrables

- API FastAPI avec VADER déployée sur AWS EC2.
- Extension Chrome avec manifest, content script, popup, et gestion des notifications.
- Documentation d'installation et d'utilisation.

7.2 Documentation

- **Manuel utilisateur** : Instructions pour utiliser l'extension Chrome.
- **Documentation technique** : Instructions pour déployer l'API sur AWS EC2.

8. Validation

8.1 Tests fonctionnels

- Tests de capture de texte sur Facebook et Twitter.
- Vérification des notifications en cas de message négatif.

8.2 Tests de performance

- Mesure du temps de réponse de l'API.

8.3 Critères d'acceptation

- Le texte doit être analysé en moins de 1 seconde.
- Les notifications doivent s’afficher correctement et être pertinentes.
- L'extension doit fonctionner sans erreur sur plusieurs navigateurs Chrome.