# ~~ Rapport 3 - Projet IA ~~

| | |
|---|---|
| ➢ BERGONZI Vinicius<br>➢ LENING Steve<br>➢ MASI Alessio<br>➢ THEUBO Ghislain | **ISI 3A - G5** |

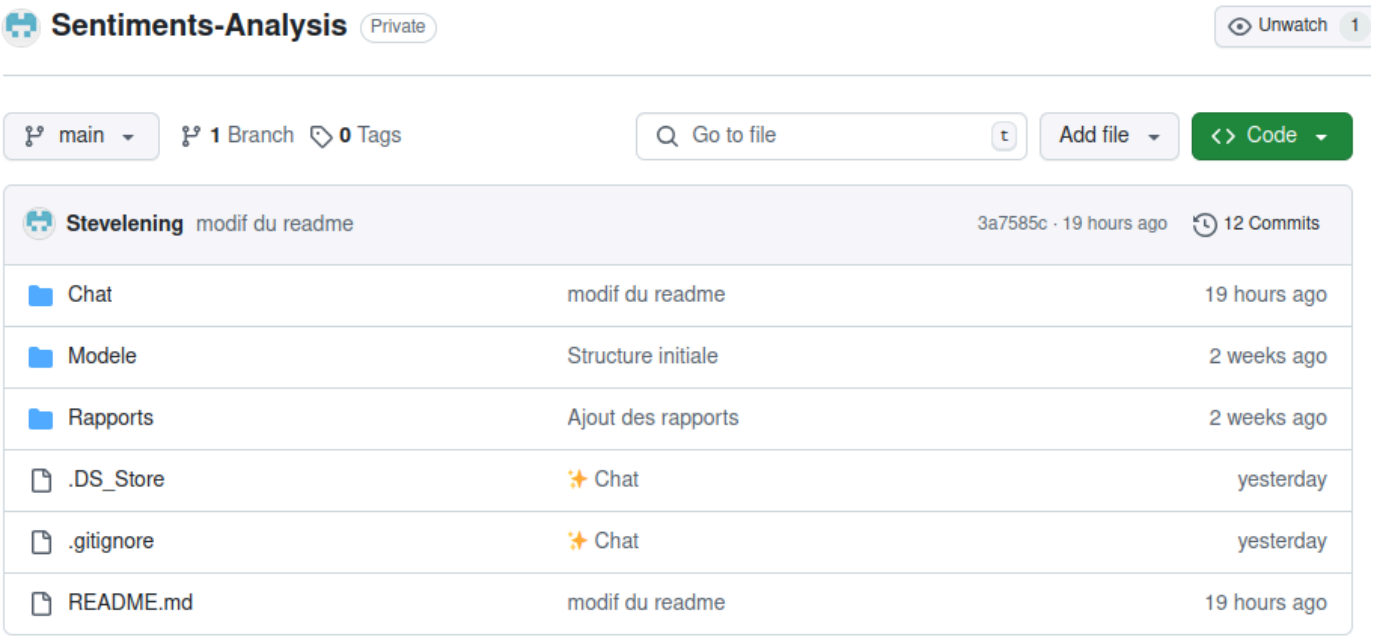
## Architecture de la solution proposée

Nous avons mis en place un repository GitHub pour la création et la collaboration des membres de l'équipe sur ce projet.

### Structure du repository :

- ❖ Un répertoire **Chat/** qui contient l'essentiel de l'application web de chat
- ❖ Un répertoire **Modele/** qui contient le modèle d'IA que nous utilisons pour l'analyse et la classification des messages échangés dans le chat.
- ❖ Un répertoire **Rapports/** qui contient tous les rapports aux formats word et pdf
- ❖ Un fichier **README.md** permettant une prise en main rapide de l'application par les utilisateurs.

Voici un aperçu de la structure de notre repository GitHub :



## Application web :

### 1. Architecture du chat :

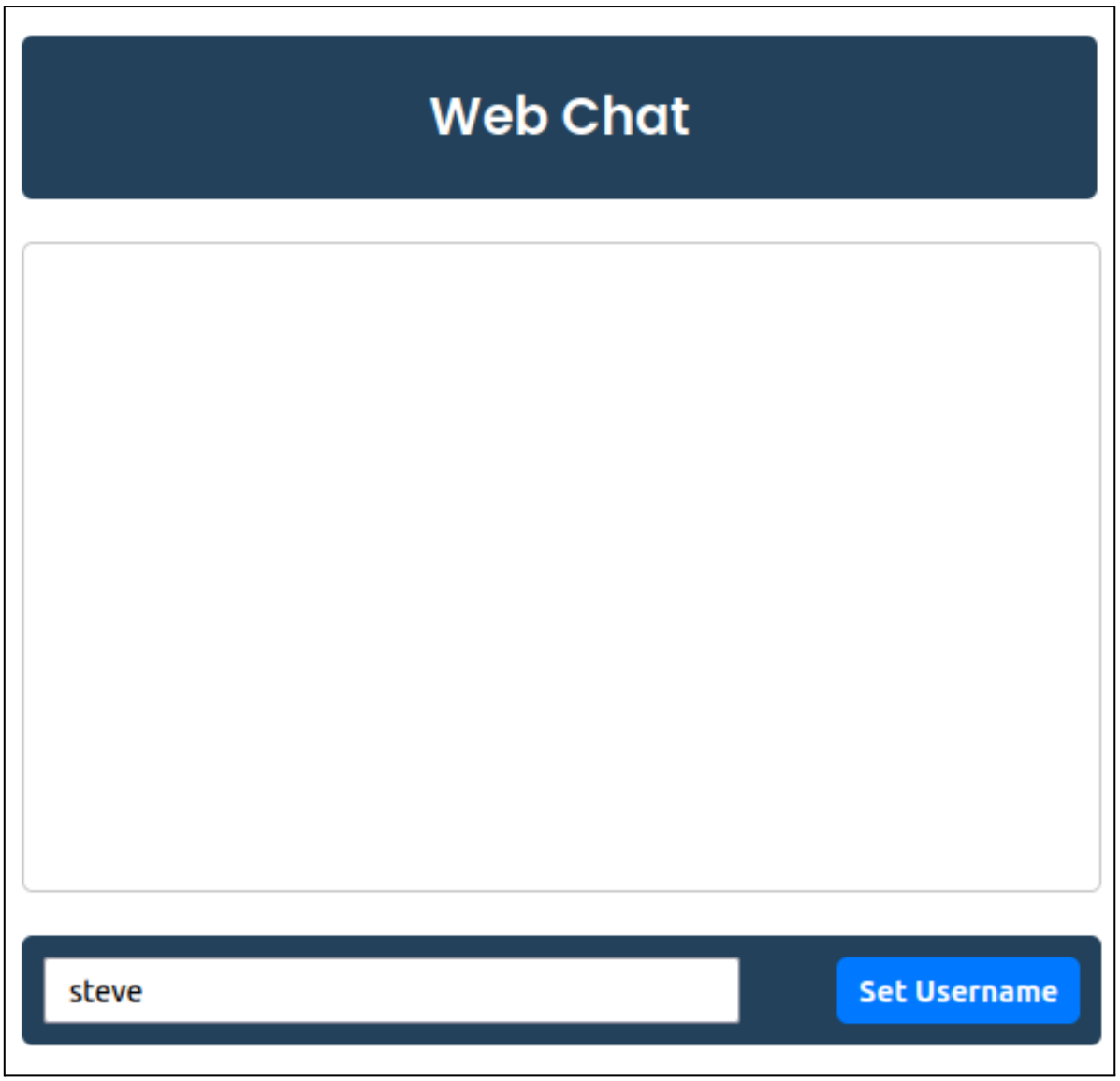
- **Frontend** : html5, css3 et Javascript
- **Backend** : NodeJs et express
- **BD** : SQLite3

### 2. Comment lancer l'application :

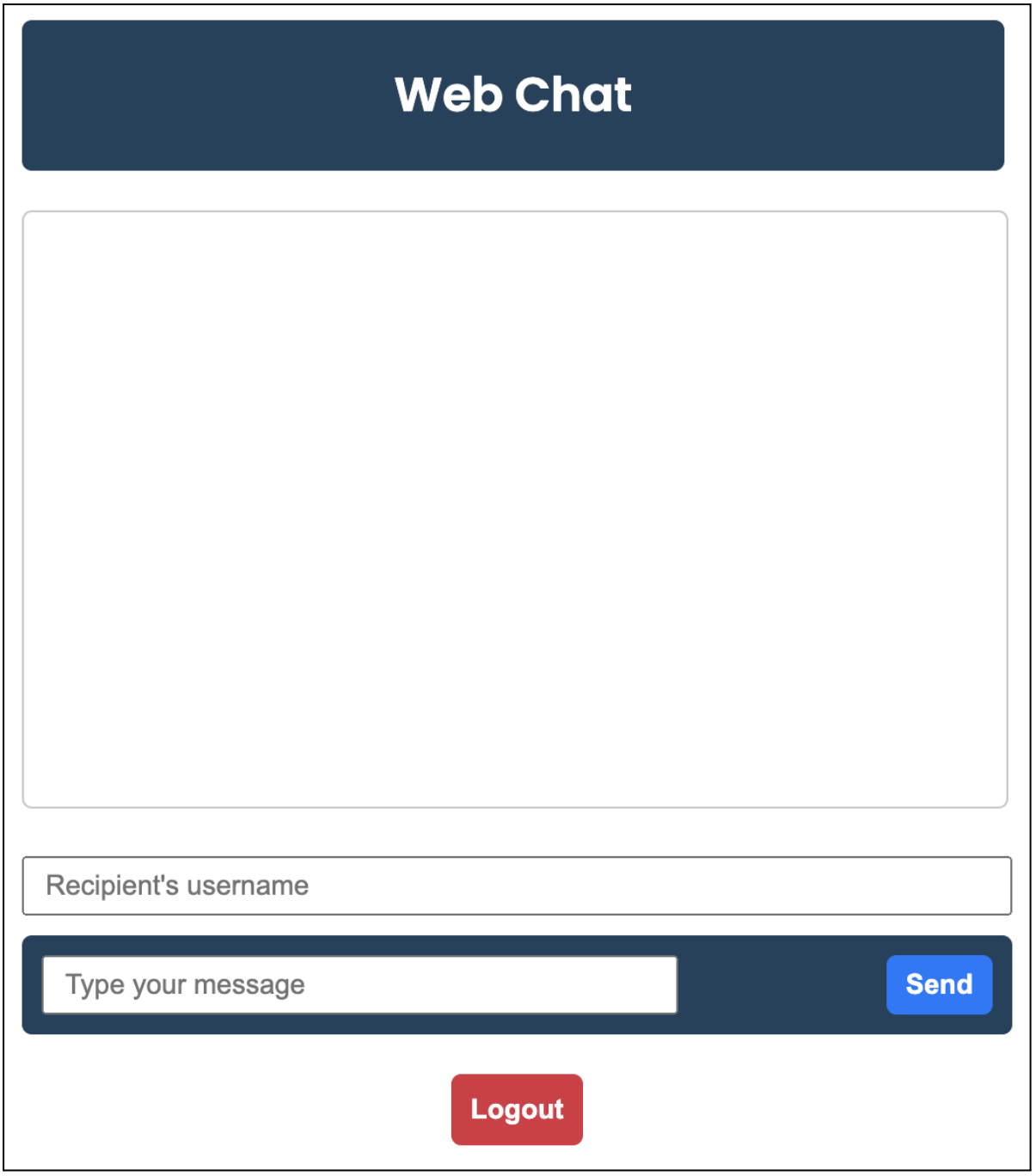
- Se déplacer dans le répertoire **Chat/** en exécutant la commande `cd Chat`
- Exécuter la commande `npm install` pour installer les dépendances nécessaires à l'exécution du serveur.
- Exécuter la commande `node index.js` pour lancer le serveur express. Si tout se passe bien, vous verrez apparaître le message **`Server running at http://localhost:3000`** dans votre terminal.
- Dans le navigateur, aller à l'adresse http://localhost:3000.
- Vous pouvez maintenant utiliser le Chat.

### 3. Description du fonctionnement Chat

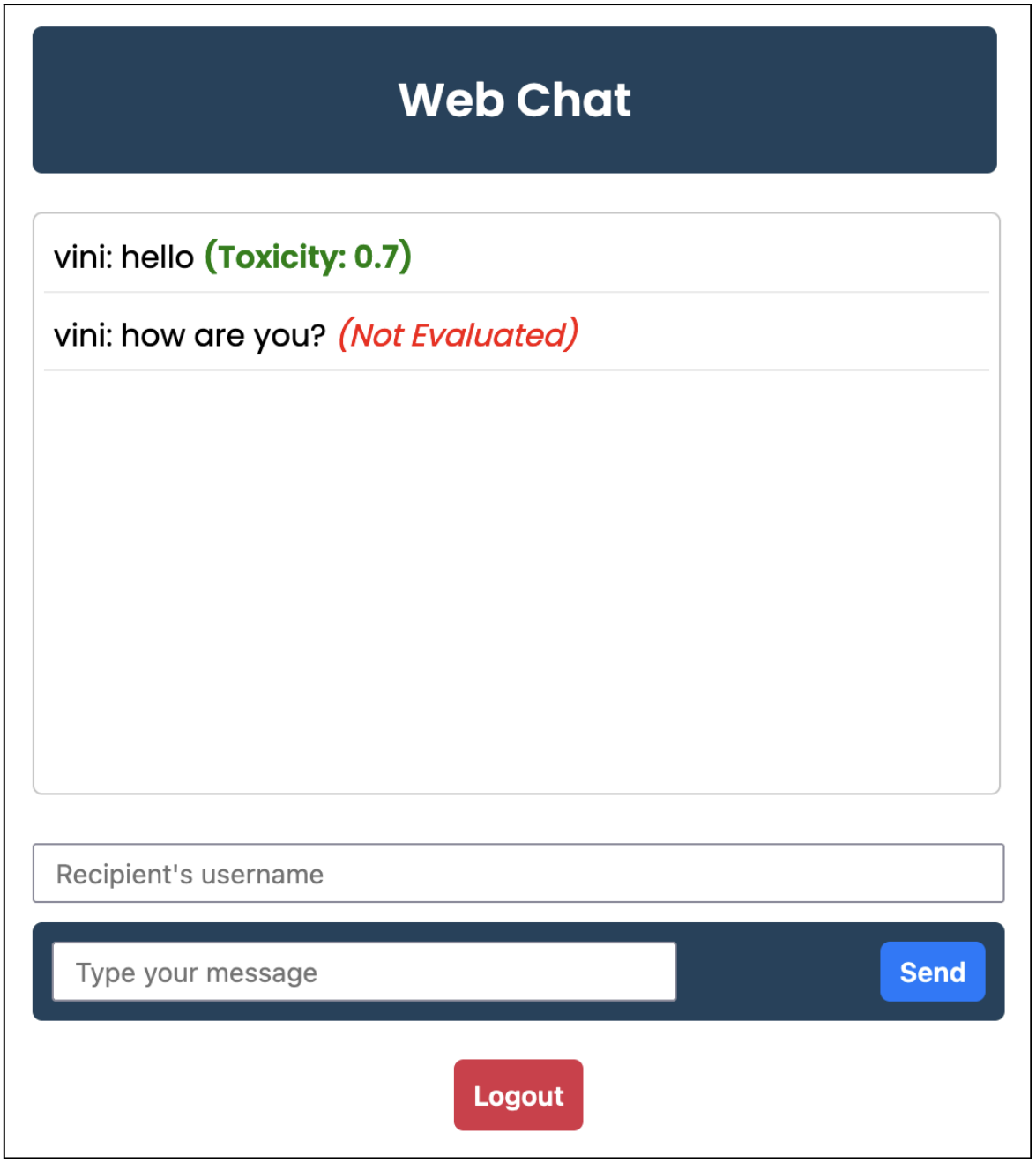
Le webchat est l'infrastructure que le modèle va utiliser pour collecter les données d'évaluation auprès des utilisateurs. L'idée est de fournir une interface graphique aux utilisateurs interagissant les uns avec les autres par échanges de messages dans l'application, tout en exécutant le modèle en arrière-plan, sans aucune interférence avec les actions des utilisateurs (l'exécution du modèle est complètement transparente pour les utilisateurs du Chat). Nous utilisons une architecture client/serveur et à l'heure actuelle, le chat en ligne utilise les technologies HTML5, CSS3 et JavaScript pour l'interface qui est **le client**, ainsi que Express et NodeJS pour le backend qui est **le serveur**. Nous utilisons SQLite3 comme **base de données** afin de stocker les messages des utilisateurs.
Ci-dessous se trouve un aperçu du prototype initial de l'application, où l'utilisateur peut choisir son propre nom d'utilisateur qui sera stocké dans la session du navigateur et affiché une fois que le message arrivera à destination. Afin de simuler un environnement réel pour cette application, il est nécessaire d'utiliser deux navigateurs différents pour créer des clients différents, puisque le nom de l'utilisateur est stocké dans la session du navigateur et que pour un même navigateur et une même adresse, on ne peut pas avoir deux sessions différentes.




Lorsque l'utilisateur renseigne son nom d'utilisateur et clique sur le bouton `Set Username`, un utilisateur portant ce nom est créé dans la base de données et ajouté à la session du navigateur. Cet utilisateur est ensuite redirigé vers la page ci-dessous où il peut échanger avec les autres utilisateurs du Chat.




Si le serveur est configuré et opérationnel, une fois que l'utilisateur envoie un message à un autre client, l'utilisateur final doit recevoir le message, le nom de l'utilisateur qui l'a envoyé et un indicateur indiquant si le message a déjà été évalué par le modèle ou si cette évaluation est en attente. Si le message a déjà été évalué, l'interface va indiquer le niveau de toxicité trouvé par le modèle.




Le serveur est conçu suivant une architecture RestAPI qui propose deux endpoints principaux pour la collecte et l'évaluation des données (messages utilisateur) : `/messages` et `/evaluate`. Le premier se charge de fournir tous les messages qui n'ont pas encore été évalués, cela peut se faire en ajoutant simplement un flag « evaluated » sur chaque message. Le second reçoit un `messageId` et un `toxicity` qui est en fait le score de toxicité et qui contient la valeur calculée par le modèle dans la base de données. Cette valeur est ensuite affichée à l'utilisateur. Si le message n'est pas encore évalué, l'utilisateur verra un indicateur `(Not evaluated)` devant le message.

## Modèle

**Architecture du modèle :**

- Modèle de base **:** [Deberta v3 small]
- Données utilisé **:** [Text moderation 410k]

L'architecture utilisée pour le moment est relativement simple et est constituée de 3 couches avec des fonctions d'activation ReLU.

```
self.classifier = torch.nn.Sequential(
                    torch.nn.Linear(768, 384),
                    torch.nn.ReLU(),
                    torch.nn.Dropout(0.2),
                    torch.nn.Linear(384, 192),
                    torch.nn.ReLU(),
                    torch.nn.Dropout(0.2),
                    torch.nn.Linear(192, 18))
```

Le modèle possède en sortie 18 labels qui correspondent aux 18 catégories disponibles dans le jeu de données utilisé.

Pour obtenir des prédictions avec le modèle, il suffit d'écrire le message qu'on veut analyser et le modèle nous permet d'obtenir le score pour chaque catégorie.

```
print(pipe("I love this movie!"))
```

```
tensor([[0.0317, 0.0133, 0.0119, 0.0042, 0.0264, 0.0063, 0.0092, 0.0124, 0.0090,
         0.0236, 0.0233, 0.0069, 0.0055, 0.0177, 0.0126, 0.0057, 0.0039, 0.0081]])
```

L'intégration entre le modèle et l'API chat sera effectuée en appelant la méthode proposée avec le message à analyser. Après on analysera le score proposée par le modèle et si ce dernier est supérieur à une certaine valeur, le message sera signalé au modérateur qui pourra ensuite l'analyser et prendre des dispositions le cas échéant.