

Assistant Messenger

Livrable final - Rapport technique

Table des matières

1. Présentation	3
1.1. Messenger	3
1.2. Contexte initiale du projet	3
1.3. Découpe du projet.....	3
2. Rétroplanning	4
3. Récapitulatif des fonctionnalités développées.....	5
4. Ajustement des objectifs en cours de projet	6
5. Conception.....	6
5.1. Architecture du code source	6
5.2. Extraction des messages - Server	6
5.2.1. Extraction des données	6
5.2.2. Server Express.....	8
5.3. Affichage des métadonnées - Site.....	8
5.3.1. Composition du site	8
5.3.2. Développement du site	9
5.4. Analyse de sentiments.....	11
5.4.1. Le principe	11
5.4.2. La pratique	12
5.4.3. Le résultat.....	13
6. Déploiement & Utilisation du site	14
6.1. Déploiement pour utilisation du site et serveur	14
6.2. Déploiement pour l'analyse de sentiment	14
6.3. Mode d'emploi du site / récupération des messages.....	15
7. Conclusion.....	16

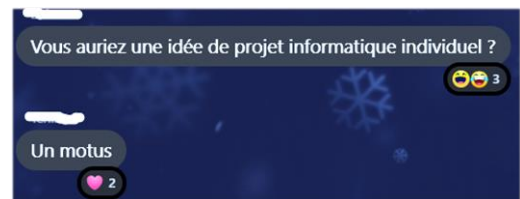
1.Présentation

1.1.Messenger

Messenger est un système de messagerie instantanée, créé par Facebook, accessible dans la plupart des environnements et systèmes d'exploitation. Cette messagerie permet de créer des conversations à plusieurs personnes, des groupes. Il n'y a pas de limite de nombre de personnes dans un groupe et lorsque quelqu'un envoie un message sur cette conversation tous les autres membres sont notifiés et le message apparaît dans le fil de discussion.

Plusieurs fonctionnalités ont été ajoutées permettant de faciliter les échanges et notamment le partage des émotions.

L'une d'entre elles est particulièrement utilisée : c'est la possibilité d'ajouter une réaction à un message spécifique, matérialisée par un émoticône accolé au dit message.



1.2.Contexte initial du projet

Ce principe de conversation de groupe peut engendrer un nombre important de messages reçus pour une personne. Ce sont généralement des conversations entre amis, groupe de travail, de familles, etc. Les personnes appartenant à plusieurs groupes actifs peuvent vite être submergées par le nombre important de messages qu'ils reçoivent de ces conversations.

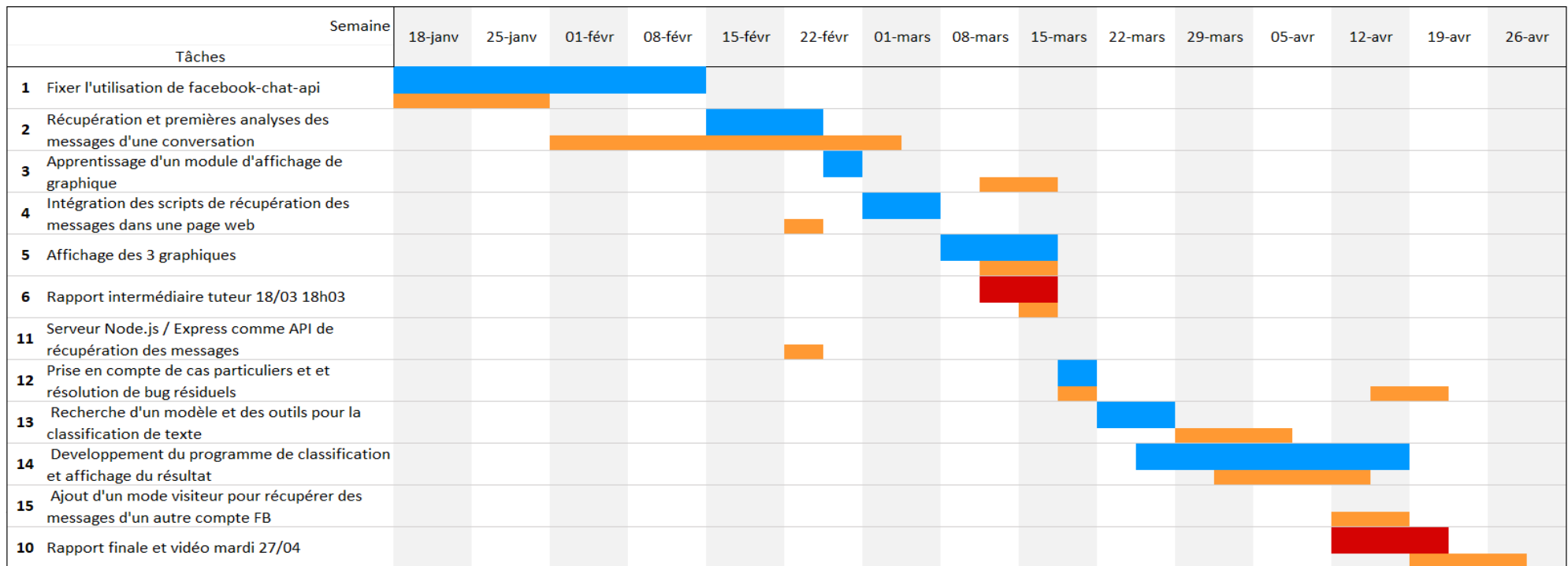
Etant donné que ces messages sont envoyés sur des groupes ils concernent souvent qu'une partie des membres, l'autre partie reçoit des notifications pour rien. Il est alors fastidieux de lire des messages qui ne nous sont pas adressés. D'où l'idée de créer un assistant capable d'analyser les messages d'une conversation de groupe Messenger afin de fournir un résumé d'informations me permettant de juger en un instant s'il est nécessaire que je me mette à jour d'une conversation.

1.3.Découpe du projet

Ce projet se découpe en trois grandes parties qui ont été développées les unes après les autres :

- 1) La récupération des messages,
- 2) L'affichage d'analyses simples et de graphiques à partir des métadonnées associées aux messages,
- 3) L'analyse du contenu des messages avec une algorithmique d'apprentissage automatique.

2. Rétroplanning



- : Période de travail effectif sur les différentes tâches
- : Prévision initiale des périodes de développement
- : Prévision initiale des périodes d'écriture des rapports

La tâche **1** a été annulée car elle menait à une impasse, la double authentification de Facebook rendait caduque l'utilisation de l'api. J'ai donc développé mon propre script de récupération des messages en tâche **2**.

J'ai développé l'API (tâche **11**) en début de projet car j'ai récemment eu l'occasion de travailler sur une API (Node/Express) et il m'a paru tout aussi rapide de développer directement cette API en même temps que le site VueJS (tâche **4**) plutôt que d'intégrer mon script de récupération de message au site web.

Aussi il m'a semblé plus naturel de découvrir Plotly directement sur mes besoins d'où les tâches **3** et **5** en parallèle.

De même pour les tâches **13** et **14** où je découvrais les technos en les appliquant à mes besoins.

3. Récapitulatif des fonctionnalités développées

1	Récupérer les derniers messages d'une conversation Messenger avec : - L'auteur, - la date de réception, - le contenu, - les réactions.	Validé	
2	Graphique montrant l'évolution du nombre de messages reçus au cours du temps.	Validé	
3	Graphique montrant l'évolution du nombre de réactions données au cours du temps.	Validé	
4	Graphique montrant l'évolution les différents interlocuteurs au cours du temps.	Modifié Validé	Remplacé par : « Graphique montrant les différents interlocuteurs sur l'ensemble des messages récupérés » Les interlocuteurs changent peu en 50, 100 voire 200 messages.
5	Possibilité de changer facilement de conversation de groupe.	Validé	Via un menu latéral sticky.
6	Identifier, grâce aux réactions, les passages de la conversation comme étant sérieux ou moins sérieux.	Validé	Matérialisé sous la forme d'un graphique montrant les différents interlocuteurs sur l'ensemble des messages récupérés
7	Regrouper les messages grâce à la fréquence de réception des messages	Non traité	Je fais le choix de me concentrer sur l'analyse du contenu des messages.
8	Visualiser l'ensemble des messages	Ajouté	Par simplicité pour visualiser rapidement les messages pertinents après avoir analysé les graphiques.
9	Afficher les messages exceptionnels : - le plus de pouces, - le plus de réactions, - le plus long message.	Ajouté	Ces messages sont souvent les points centraux d'une discussion.
12	Possibilité de rentrer ses identifiants Facebook et l'ID d'une de ces conversations de groupe sur le site pour faire l'analyse	Ajouté	Fonctionnalité ajoutée pour permettre à d'autres personnes d'utiliser cet outil.
10	Développer une API Express pour récupérer les messages via requêtes http	Objectif secondaire Validé	J'ai développé cette fonctionnalité car elle ne me demandait pas plus de

			temps que de faire la récupération des messages directement dans le site.
11	Classifier les messages et identifier les sujets abordés.	Modifié Validé	Remplacé par : « Analyse de sentiment sur l'ensemble d'une conversation »

Les objectifs principaux établis dans le cahier des charges sont complétés.

Les fonctionnalités 4 et 7 ont été revues à la baisse afin de développer les fonctionnalités 8 et 9 qui m'ont paru plus pertinentes lors du développement.

4. Ajustement des objectifs en cours de projet

Initialement la troisième partie sur l'analyse du contenu des messages était peu défini. Je souhaitais me servir du temps qu'il me resterait après les 2 premières phases pour m'initier à l'intelligence artificielle au travers des messages récupérés en première partie. Ceci étant dans l'optique de fournir à l'utilisateur une analyse plus pertinente de ses messages non lus dans une conversation de groupe.

Finalement j'ai choisi de m'initier à l'IA avec de l'analyse de sentiment sur l'ensemble d'une conversation et non sur les derniers messages reçus. L'Assistant Messenger est alors plus proche d'un Analyseur de conversation que d'un véritable assistant. Dans le cadre de ce projet informatique individuel et du peu de connaissance en IA que j'avais en abordant cette deuxième partie il m'a paru plus intéressant de faire une analyse des sentiments sur des conversations plutôt que d'aborder des outils de génération automatique de résumé par exemple.

5. Conception

5.1. Architecture du code source

Le code source disponible sur GitHub (<https://github.com/plorgue/assistant-messenger>) est séparé en 3 dossiers pour chacune des parties.

- `server/` : contient l'api qui renvoie les messages extraits
- `site/` : contient le site qui traite et affiche les messages
- `sentiments/` : contient les scripts d'analyse de sentiments

5.2. Extraction des messages – Server

5.2.1. Extraction des données

Pour récupérer les données d'un service tel que les messages d'une conversation Messenger plusieurs possibilités s'offrent à nous. Elles sont répertoriées dans le tableau suivant :

Technique	Difficulté de mise en place	Stabilité de la solution.	Efficacité	Applicable pour une conversation Messenger
1 Utiliser une API publique du service (ici Facebook)	Simple, il y a souvent de la documentation fournie.	Stable dans le temps	Optimisée	L'API de Facebook ne permet pas de récupérer les messages de Messenger
2 Simuler l'application client du service (ici le site web Messenger à travers un navigateur) et récupérer les données dans les requêtes HTTP	Difficile, il faut analyser le comportement de l'application client et l'ensemble des requêtes qu'elle effectue au serveur. Il n'y a pas de documentation.	Instable, si le service change sa gestion des requêtes. Solution non commercialisable.	Presque aussi efficace que l'API publique	Il y a un système d'authentification difficile à reproduire, du moins avec mes compétences.
3 Extraire les données de l'application client.	Modéré. Dans le cas d'une application web, extraire les données d'une page HTML est plutôt simple techniquement mais reste assez long à mettre en place.	Instable. Si le service change sa façon d'organiser les données dans l'application. Solution non commercialisable	Lent pour extraire beaucoup de données. Il faut charger toute l'application client pour y extraire qu'une petite partie des données.	Messenger dispose d'une application web pour suivre ces messages. Cette solution est donc pertinente pour ce projet.

Malheureusement Facebook ne permet pas de récupérer facilement les données de conversations Messenger grâce à leur API publique.

Dans mes recherches j'ai découvert deux projets open-source sur GitHub qui avaient pour ambition de récupérer les messages d'une conversation Messenger à partir des identifiants de l'utilisateur: [fbchat](#) en python et [facebook-chat-api](#) en JavaScript. Ces deux solutions ne sont pas maintenues et bien que j'ai réussi à récupérer quelques données avec [facebook-api-chat](#) une première fois, les fois suivantes Facebook a bloqué l'accès, voyant que l'appareil n'était pas habituel. Ces projets utilisent la méthode 2.

Finalement je me suis rabattu sur la troisième solution : la récupération des messages dans la page HTML (web scrapping), bien que plus fastidieuse elle fonctionne.

Techniquement j'utilise Puppeteer, une librairie Node permettant de contrôler facilement un navigateur Headless (sans l'affichage dans une fenêtre) généralement utilisé pour tester les application web.

Je simule alors le comportement d'une personne qui se connecterait en entrant identifiant/mot de passe et cliquerait sur tel ou tel bouton. Une fois sur la page de la conversation il faut scroller les messages pour en récupérer plus.

Une fois la conversation scrollée à souhait je récupère les nœuds HTML contenant les messages avec le sélecteur : `'div[data-testid="outgoing_group"]'` puis avec une succession de règles et d'expressions régulières je récupère le contenu et les diverses métadonnées du message.

Le script de scrapping est dans `server/utils/scrapping.js` et des exemples de messages récupérés sont dans `server/store/`.

En terme de rapidité, récupérer une cinquantaine de messages prend quelques minutes. J'ai pu extraire au maximum 8223 messages en 15h avec environ 1400 scroll.

5.2.2. Server Express

Dans l'optique de permettre l'affichage de l'Assistant Messenger sur plusieurs plateformes (site web, appli mobile) et ayant déjà des connaissances dans la librairie Express avec Node j'ai choisi de développer une API. Trois requêtes GET sont possibles (`server/api/routes.js`) pour:

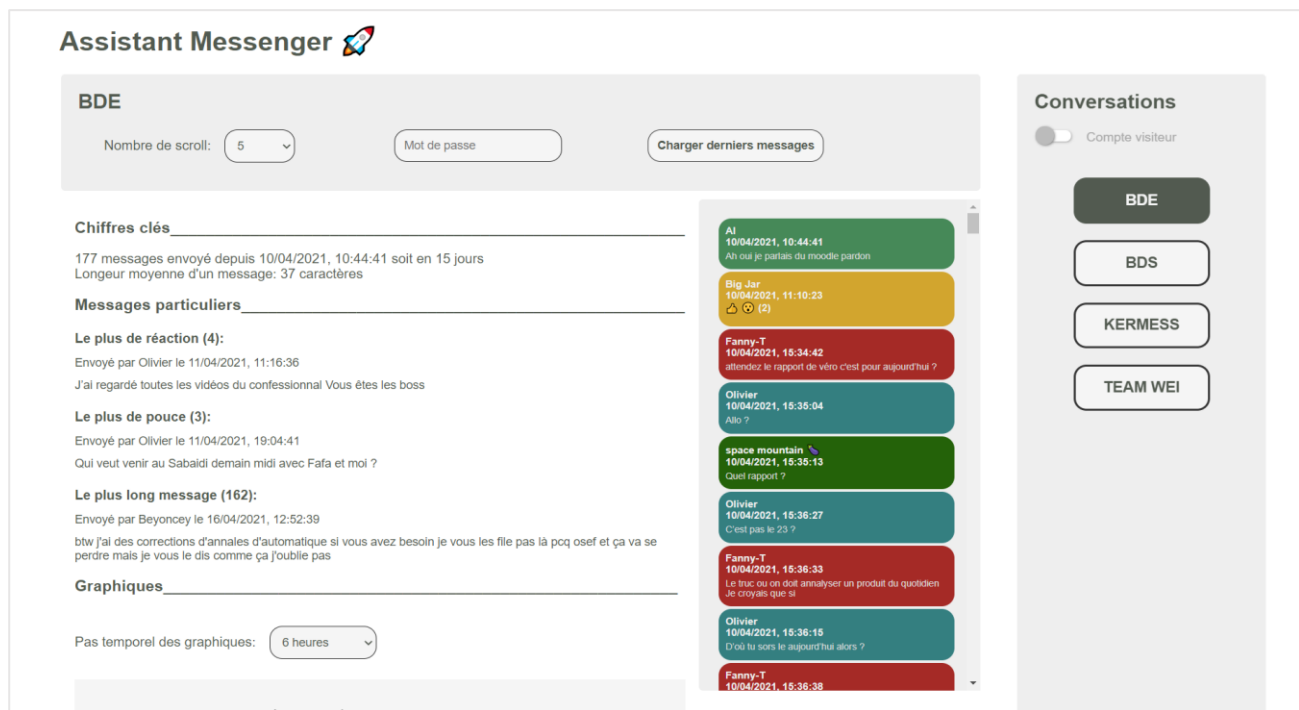
- récupérer des messages sur mon compte avec un mot de passe différent de mon mot de passe Facebook (pour ne pas stocker en clair mon mot de passe Facebook dans le code source sur GitHub),
- récupérer des messages déjà extraits des pages web et stockés dans des fichiers JSON,
- récupérer des messages d'une conversation de groupe d'un autre compte.

5.3. Affichage des métadonnées – Site

5.3.1. Composition du site

Le site n'a qu'une seule page composée :

- d'une zone verticale à droite pour sélectionner la conversation à utiliser,
- d'une barre horizontale en haut pour choisir le nombre de scroll (proportionnel au nombre de messages récupérés), entrer mon mot de passe personnel et un bouton pour récupérer les messages,
- d'une zone centrale dans laquelle sont représentées les messages.



Capture d'écran après réception de messages

Une fois les messages chargées la zone centrale s'affiche. On y retrouve :

- Le nombre de messages chargés,
- La date de réception du message le plus ancien et depuis combien de temps,
- La longueur moyenne d'un message,
- Le message ayant reçu le plus de réaction,
- Le message ayant reçu le plus de pouce 👍,
- Le message le plus long,
- Un histogramme du nombre de messages reçus par tranche horaire (durée préalablement choisie),
- Un histogramme du nombre de réactions reçus par tranche horaire avec la part de réaction typée sérieuse (j'ai sélectionné une dizaine de réactions)
- Un histogramme du nombre de messages envoyés par chaque personne
- L'ensemble des messages reçus dans l'ordre chronologique

J'ai choisi ces différentes représentations dans l'objectif de différencier les sujets sérieux des plus légers, avec comme principe que les messages « sérieux » sont les plus indispensables à lire. Par expérience et en observant mes conversations j'ai constaté plusieurs faits à propos des messages concernant sujets sérieux :

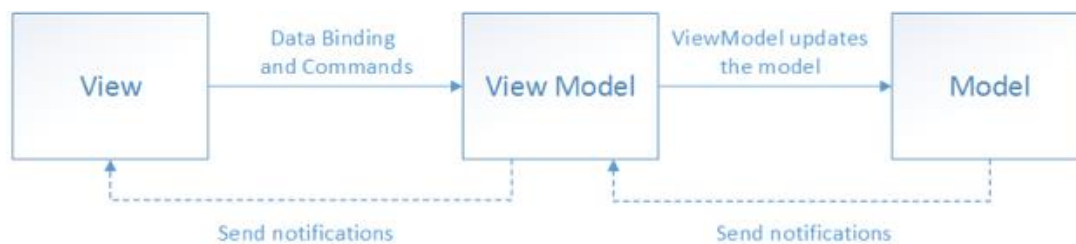
- Ils reçoivent peu de réaction et principalement des pouces,
- Plus le message est long plus il a des chances d'être sérieux,
- Ce sont souvent les mêmes personnes qui parlent de sujets sérieux.

5.3.2. Développement du site

VueJS

Le site est développé avec VueJS, c'est est un framework Javascript frontend pour créer des interfaces utilisateurs. Il est extrêmement facile à prendre en main et permet de découper le site

en plusieurs composants autonomes et réutilisables. Ce framework s'inspire du patron d'architecture MVVM.



Le data binding permet de coupler les données aux DOM ce qui rend l'application réactive aux changements.

J'utilise aussi Vuex pour stocker le model principal (les messages de la conversation sélectionnée) pour le rendre plus facilement accessible à l'ensemble des composants du site.

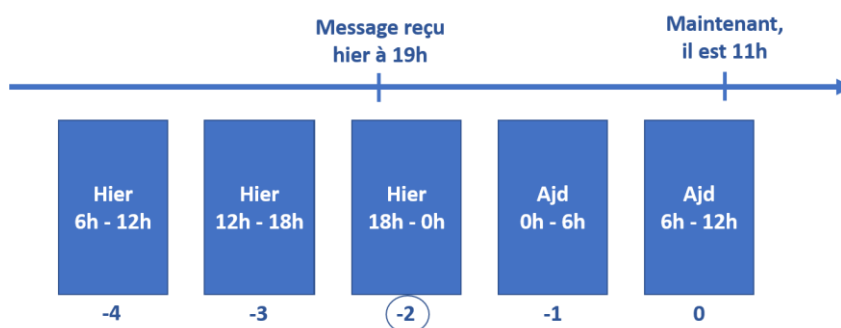
Le composant principal est Dashboard dans `site/src/views/`. Les sous-composants sont dans `site/src/components/`.

Plotly

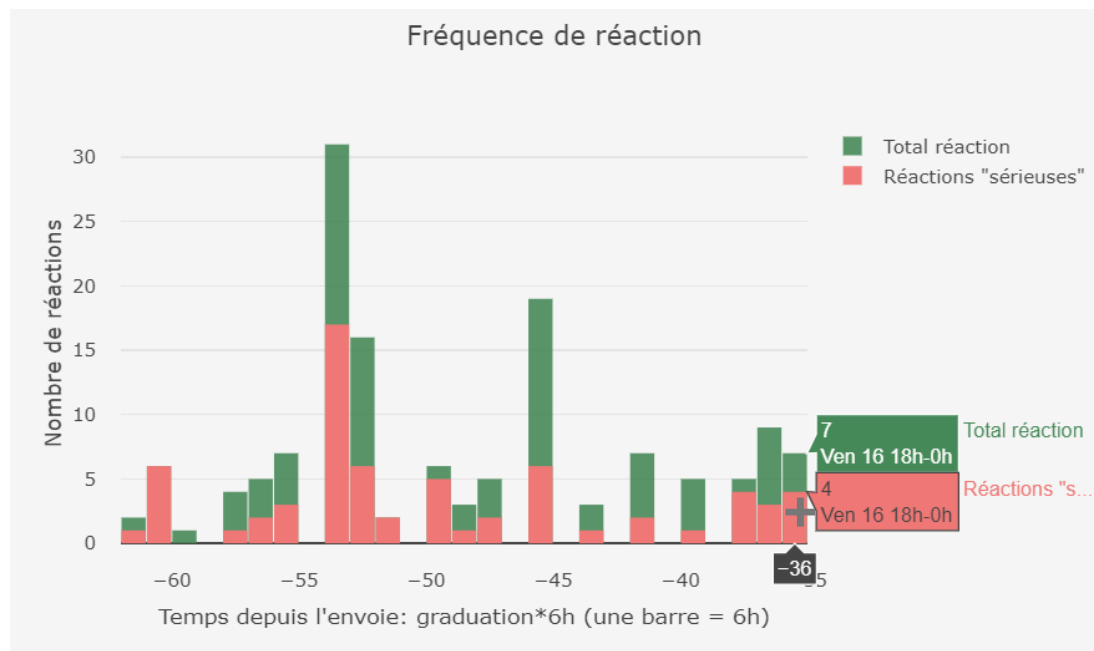
Pour l'affichage des histogrammes j'utilise Plotly. Il y a une importante phase de préparation des données. L'utilisateur choisit l'intervalle de temps (ex : 6h) qui correspondra à une barre dans l'histogramme. Pour 6h une journée est donc décomposé en 4 barres (0h-6h, 6h-12h, ..) .

Il faut convertir les dates des messages en un nombre: le nombre d'intervalle de temps depuis l'instant présent.

Ex : Un message a été reçu hier à 19h. Il est 11h et la largeur d'un intervalle choisi est 6h. Le message se trouve donc dans l'intervalle « Hier entre 18h et 0h » soit le 3eme intervalle (les autres étant « Ajd 0h-6h » et « Ajd 6h-12h »). Ce message se voit donc associer le nombre -2 car l'indexation commence à 0 (et non 1) et le chiffre est négatif pour avoir l'instant présent à droite de l'histogramme.



Les largeurs des barres vont de 15min à 1 journée. Lorsqu'on passe la souris sur une barre une bulle indique l'intervalle correspondant. Je n'ai pas réussi à modifier les valeurs d'abscisses pour avoir un affichage plus pertinent.



5.4. Analyse de sentiments
















































5.4.1. Le principe

Pour la dernière phase du projet je souhaitais analyser le contenu des messages pour m'initier à l'intelligence artificielle au travers d'un projet concret. Les objectifs de cette partie donc ont été définis tardivement, à la mi-projet. Après quelques temps de recherche en NLP (Traitement naturel du langage) j'ai choisi de faire de l'analyse de sentiments sur les messages.

D'un côté on associe à chaque message un sentiment positif 1 ou négatif 0 et de l'autre on extrait de chaque message les entités citées (personnes, lieux, objets/concepts marquants). Ainsi on obtient une correspondance sentiment / entité. En comptant le nombre de sentiments positifs / négatifs associés à chaque entité puis en faisant la différence on peut classer les entités de la plus appréciée à la plus décriée.

Par exemple imaginons que sur le schéma ci-dessous un rectangle est un mot, une ligne de rectangle une phrase et les rectangles de couleurs des entités. Chaque phrase est déterminée comme étant positive (P) ou négative (N).

On peut alors distinguer les entités le plus souvent associées à un sentiment positif de celles plus souvent associées à un sentiment négatif.

P								P	N	P-N		
P									0	2	-2	
N									2	0	2	
N										1	1	1
P									1	1	1	
N												

Ici le orange est la couleur préférée alors que le rouge est la moins appréciée.

Pour un corpus de messages provenant d'une conversation Messenger cela permet de mettre en avant ce que le groupe préfère et ce qu'il aime le moins sur la période couverte par les messages.

De plus j'ai choisi d'associer à une entité plusieurs mots lui faisant référence. Par exemple, j'ai associé au concept « covid 19 » les mots « pandémie », « coronavirus », « covid », etc.

5.4.2. La pratique

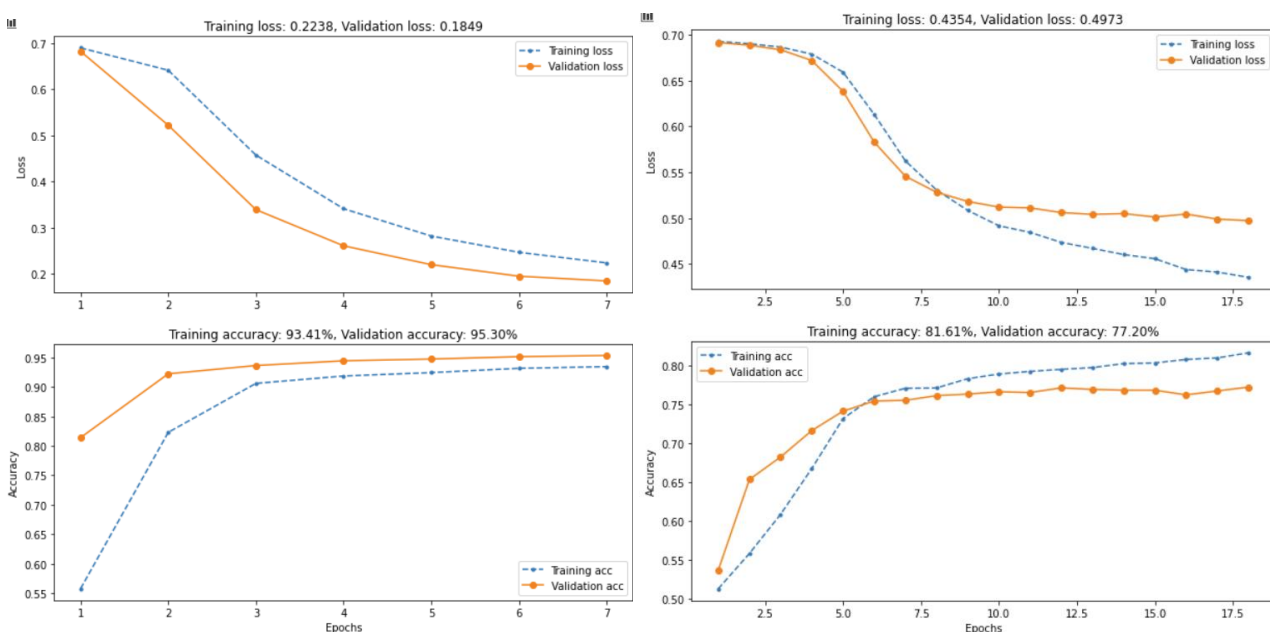
Classification messages positifs et négatifs

L'analyse de sentiment est développée en Python avec la plateforme TensorFlow. Je me suis inspiré du travail de [Theophile Blard](#) pour fine tune un modèle [camembert](#) pré-entraîné.

Pour l'entraînement j'ai utilisé différents jeux de données : d'abord des [tweets français](#) puis des [commentaires allocine](#) (utilisé par Théophile Blard).

J'ai annoté 280 messages issus de la conversation que je souhaitais analyser. Ces messages m'ont permis de déterminer quels hyperparamètres (nombre de données d'entraînement, taille des batch, coefficient d'apprentissage, nombre d'époques) étaient les plus efficaces pour l'entraînement de mon modèle.

J'ai obtenu de meilleurs résultats en fin d'entraînement (sur les sets de données test), avec les commentaires allocine (F1-score de 95% contre 80% pour les tweets). Les courbes ci-dessous représentent l'entraînement avec les deux jeux de données avec des hyperparamètres qui m'ont parus optimaux. Cependant il s'avère que sur mon corpus de messages annotés, le modèle entraîné avec les commentaires allocine perd totalement en efficacité avec un score qui tombe à 62% alors qu'avec les tweets il reste stable autour des 79%. Ce résultat était prévisible, les commentaires allocine sont beaucoup plus longs et se rapprochent donc moins des messages de conversations que des tweets.



Courbe d'entraînement (allocine à gauche et tweets à droite)

J'ai donc gardé le modèle entraîné avec les tweets.

Techniquement j'ai profité d'avoir une carte graphique (Nvidia GTX 1650 Ti avec 4Go de mémoire dédiée) pour accélérer l'entraînement du modèle avec CUDA. La mise en place m'a demandé quelques ajustements pour accorder les versions des différents outils mais l'effort en valait la peine. Cette opération m'a permis de réduire drastiquement le temps d'entraînement en passant de plusieurs heures sur le CPU à quelques dizaines de minutes avec le GPU Nvidia.

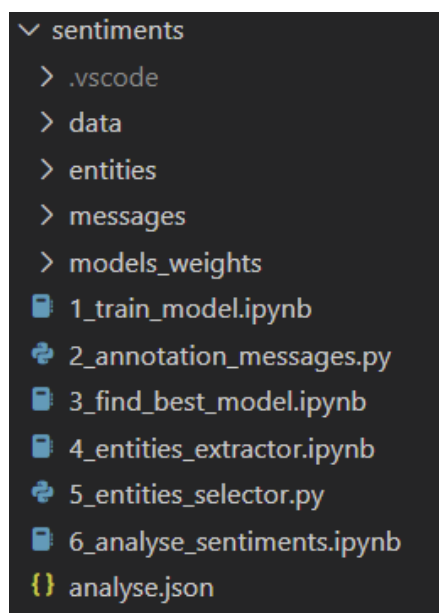
Extraction des entités

Pour récupérer les entités des messages j'ai utilisé la librairie spaCy qui contient cette fonctionnalité. Cependant cet outil est entraîné sur des textes mieux structurés que les messages que je lui ai fournis. Un travail de sélection à la main des entités extraites par spaCy était nécessaire.

Sur 8223 messages spaCy a extrait 1681 entités. Après sélection, regroupement et ajout personnel j'en ai gardé environ 450.

Description du dossier *sentiment*/ du code source

Toute la partie analyse de sentiment est dans le dossier `sentiments/`.



- `data/` contient les données d'entraînement
- `entites/` contient les fichiers contenant les entités extraites
- `messages/` contient les messages annotés et non annotés
- `models_weights/` contient les fichiers de poids du modèle après entraînement avec différents hyperparamètres
- `1_train_model.ipynb` est un notebook permettant l'entraînement du modèle
- `2_annotation_messages.py` est un script python facilitant l'annotation avec la console
- `3_find_best_model.ipynb` est un notebook qui teste sur les messages annotés les différents réglages du modèle issu des entraînements avec différents hyperparamètres
- `4_entities_extractor.ipynb` est un notebook qui extrait des messages les entités avec spaCy et effectue un premier trie
- `5_entities_selector.py` est un script python qui facilite la sélection et le regroupement des entités avec la console
- `6_analyse_sentiment.ipynb` est un notebook qui applique l'analyse de sentiment sur les messages non annotés d'une conversation et classe ainsi les entités contenues dans la conversation. Les entités utilisées sont celles pré-extraites avec 4 et 5.

5.4.3. Le résultat

Après exécution des différents scripts j'ai obtenu un classement des entités pour ma conversation BDE regroupant 8223 messages sur 4 mois. Les résultats sont visibles dans le notebook `6_analyse_sentiment.ipynb`.

Le modèle prédit beaucoup plus de messages positifs que négatifs, ce qui traduirait le fait que les membres de cette conversation seraient plutôt optimistes. Je note que certaines thématiques

chères au BDE ressorte bien dans les entités les plus appréciées. Cependant d'autres entités ne me paraissent pas à la bonne position. Est-ce que le modèle a fait de mauvaises prédictions ou mon ressenti est-il faux ?

Il est difficile d'évaluer quantitativement la justesse de ces résultats. Si je devais me risquer à faire une analyse qualitative je dirais que malgré un modèle avec des scores améliorables les résultats me paraissent pertinents et cohérents avec mon ressenti en tant que membre de la conversation. Tirer des conclusions de ces résultats se rapprocherait presque à de l'astrologie.

6. Déploiement & Utilisation du site

6.1. Déploiement pour utilisation du site et serveur

Prérequis :

- Avoir Node.js installé → nodejs.org
- Avoir un gestionnaire de paquet JS installé, j'utilise NPM → npmjs.com
- Avoir installé VueCLI (vuejs.org) → `npm install -g @vue/cli`

Procédure :

- Cloner le dépôt GitHub: [plorgue/assistant-messenger \(github.com\)](https://github.com/plorgue/assistant-messenger) ou télécharger les dossiers `server/` et `site/` sur le [Google Drive](https://drive.google.com)
- Exécuter la commande `npm install` successivement dans les deux répertoires `server/` et `site/`
- Pour lancer l'API exécuter : `node .\server.js` dans le répertoire `server/`
- Pour lancer le serveur local permettant de visionner le site Vue exécuter `npm run serve` dans le répertoire `site/`
- Dans un navigateur Edge, Chrome, ou Firefox (il est peu probable que ça fonctionne avec Internet Explorer) entrer l'url : <http://localhost:8080>

6.2. Déploiement pour l'analyse de sentiment

Pour exécuter les Notebook sur sa machine :

- Avoir un environnement [Anaconda](https://anaconda.org) actif sur lequel est installé le package [Jupyter](https://jupyter.org)
- Utiliser un éditeur de texte compatible comme VS Code ou JupyterLab
- Pour exécuter les notebooks sur GPU il faut installer CUDA et TensorFlow pour GPU. Attention il faut que les versions de Python, CUDA et TensorFlow soient compatibles. La documentation [TensorFlow](https://www.tensorflow.org/install/gpu) explique la procédure d'installation

Pour exécuter les Notebook sur Google Colab :

- Aller sur [Google Drive](https://drive.google.com)
- Ouvrir les Notebooks avec Google Colab
- Pour les exécuter il faut modifier la variable `chemin_dossier_contenant_projet` avec le chemin, dans votre Drive du dossier contenant le dossier `sentiments/`

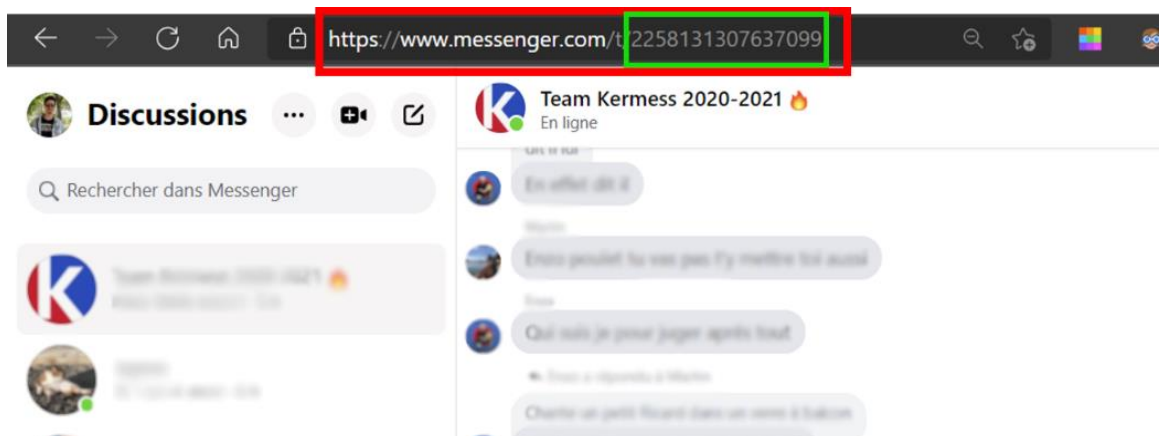
6.3.Mode d'emploi du site / récupération des messages

Ce projet est avant tout développé pour un usage personnel. Mes conversations sont donc près rentrées. Il n'était pas initialement prévu que ce site soit utilisé pour d'autres comptes. J'ai cependant ajouté en fin de projet la possibilité pour un autre utilisateur d'entrer ses identifiants et l'id d'une de ses conversations pour utiliser cet outil.

De plus je propose d'afficher des messages sans avoir à saisir une quelconque information pour avoir un aperçu du rendu rapidement.

Les trois manières de récupérer et d'afficher des messages sont donc les suivantes:

Messages récupérés	Procédure	Remarques
Les derniers messages d'une de mes conversations pré-enregistrées	1) Sélectionner une conversation à droite 2) Rentrer un mot de passe personnel dans la barre horizontale 3) Cliquer sur « Charger derniers messages »	Fonctionnalité principale
Messages stockés sur le serveur	Identique à la procédure précédente en laissant le mot de passe personnel vide	Fonctionnalité de démonstration, pour charger des messages rapidement
Messages d'une conversation d'un autre compte	1) Activer le switch « compte visiteur » 2) Compléter les identifiants. 3) Cliquer sur « Valider » puis sur « Charger derniers messages » 4) Patienter le temps de l'extraction des messages côté serveur	Fonctionnalité de démonstration. « Id conversation » correspond au nombre dans l'URL d'une conversation de groupe (cf image ci-dessous)



Identifiant de conversation essentiel pour récupérer les messages

Remarques supplémentaires :

- La récupération des messages fonctionne uniquement sur des conversations de groupe,

- Avec une connexion trop lente il arrive que Puppeteer n'arrive pas à se connecter à Messenger ou qu'aucun message ne se charge. Dans ces cas-là, il ne faut pas hésiter à essayer plusieurs fois en actualisant le site et en relançant le server avec `node /server.js`

7. Conclusion

Amélioration

Je propose plusieurs pistes d'amélioration et de nouvelles fonctionnalités à développer :

- L'ajout d'un graphique montrant l'évolution de la longueur moyenne des messages sur un intervalle de temps donné. (J'ai eu l'idée de cette représentation lors de l'écriture de ce rapport, elle me paraîtrait tout-à-fait intéressante et est simple à développer avec les composants VueJS déjà développés)
- Il serait intéressant d'améliorer le scrapping pour remonter plus loin dans le temps et récupérer plus de messages. La fonction « Recherche » dans la conversation Messenger permettrait de commencer l'extraction à partir d'un message antérieur. Plus de messages permettraient de faire une analyse de sentiment sur les entités plus pertinente.
- Pour améliorer les performances de l'analyse de sentiment on pourrait annoter plus de messages et entraîner le modèle directement sur des messages de conversations.
- Sur l'interface web il faudrait simplifier la procédure de récupération des messages d'un compte autre que le mien « mode visiteur ».

Changement d'échelle

Il me paraît difficile de proposer cet outil à un plus large public et d'autant plus de le monétiser. L'extraction des messages d'une page HTML est lourde et trop instable. Même si on récupère directement les messages dans les requêtes faites par le navigateur il est peu probable que Facebook accepte cette pratique. Enfin il faudrait que les utilisateurs acceptent de fournir leurs identifiants Facebook à un service étranger à Facebook. Pour proposer une solution publique de ce projet il faudrait travailler avec Facebook pour récupérer efficacement et légalement les messages.

Revue des objectifs du projet

Je considère avoir rempli l'objectif principal que je m'étais fixé en début de projet, à savoir récupérer les derniers messages de mes conversations de groupe Messenger puis les traiter et les afficher de manière pertinente pour faciliter leur revue.

Ayant complété cet objectif dans les temps j'ai pu aborder l'objectif secondaire qui était de m'initier à l'intelligence artificielle à travers un projet concret : l'analyse du contenu des messages. Je considère cet objectif en grande partie atteint. J'aurais aimé pouvoir aborder d'autre modèle que BERT et comparer leur efficacité dans ce projet.