



Réalisation d'un bot informatique - Rapport final



PT3 : Réalisation d'un bot informatique

Rova R - Benoît T - Sébastien B

2019-2020 / DUT INFO 2

Sommaire	2
Remerciements	3
Introduction	4
Démarche du travail effectué et organisation	5
Aspects techniques et présentation des résultats	6
Le Framework Flask	6
Base de données Firebase	9
Fonctionnement du bot	10
L'API gmail	11
Les API Selenium, Twython et pytesseract	13
Architecture de l'application (simulée sur un téléphone)	17
Flux de navigation dans l'application : états-transitions	18
Conclusion	19
Analyse critique du projet	20
Perspectives du projet	20
Overview of the project (résumé du projet en anglais)	21
Glossaire	22
Annexe	24
Planning	24
Analyse des besoins	25
Cahier des charges fonctionnel	26
Backlog de produit	27
Mode d'emploi	28
Diagrammes : Cas d'utilisation et UML	29
Budget et coûts	30

Remerciements

Nous tenons à remercier tout particulièrement M.Finck, notre tuteur de projet, pour l'aide qu'il nous a apportée durant ces 5 mois. Sa disponibilité, sa pédagogie et ses conseils nous ont été précieux dans nos choix de conception et développement de notre application et ses fonctionnalités.

Nous remercions notre entourage. Les informations précieuses qu'ils nous ont données, nous ont permis de poursuivre ce projet dans les meilleures conditions possibles.

Introduction

Depuis peu, les réseaux sociaux sont devenus une partie intégrante de la vie en société. Les principaux utilisateurs, les jeunes, utilisent en effet de plus en plus de réseaux sociaux différents que ce soit pour interagir avec leurs amis, se tenir au courant de l'actualité ou même trouver du travail. Selon une étude réalisée en 2017, un jeune de 18 ans en Europe passe en moyenne 70 heures sur son téléphone par mois, ce qui représente presque 3 jours. C'est de là qu'est née l'idée de créer une application compagnon pour faciliter l'utilisation des réseaux sociaux et gagner du temps.

Plusieurs entreprises ont déjà mit sur le marché des applications de ce type. On peut citer notamment Hootsuite, une application créée en 2008 par Ryan Holmes qui va permettre de programmer des publications sur divers réseaux sociaux, de gérer plusieurs comptes, et d'obtenir des analyses détaillées (activité d'un utilisateur).

Cependant, ces applications s'adressent surtout aux professionnels, par exemple des entreprises qui voudraient voir l'effet de leur campagne publicitaire sur ses utilisateurs. À contrario, notre projet vise plutôt les particuliers qui voudraient juste gagner du temps et automatiser certaines de leurs actions sur le web.

Notre projet informatique consiste donc à développer une application informatique. Celle-ci prendra la forme d'un bot informatique* qui se baladera sur internet, plus précisément les réseaux sociaux, en simulant les actions des internautes humains. Une autre fonctionnalité prévue, c'est la possibilité de l'utiliser en tant que compagnon, pour qu'il puisse gérer en toute autonomie nos comptes réseaux sociaux, selon les paramètres d'entrée que nous lui passons. Il pourra ainsi, par exemple :

- publier automatiquement à la date désirée,
- suivre des comptes,
- chercher selon des des mot-clés,
- faire une synthèse des actualités,
- interagir, liker, etc.

Bref la liste est longue, d'où il nous sera nécessaire de travailler en mode Agile.

Démarche du travail effectué et organisation

Le développement d'une application informatique n'est pas une tâche aisée. De ce fait, une bonne organisation et une répartition des tâches optimisée sont nécessaires. Plusieurs méthodologies de développement existent et ont déjà fait leurs preuves. On peut notamment parler des méthodologie en cascade, en spirale ou encore en Agile. Ces méthodes dépendent souvent de la taille du projet et des exigences du client (besoins exprimés).

Dans notre cas, nous travaillons sur un petit projet, donc une méthode classique en cascade suffit. Cependant, le modèle d'application que nous souhaitons développer comporte de nombreuses fonctionnalités modulaires. De plus, nous devons délivrer continuellement et rapidement une application fonctionnelle.

Finalement, nous n'étions que 3 dans le groupe, à la fois concepteurs, développeurs, product owner et scrum master. Cela reste très compliqué à gérer donc nous avons décidé de ne garder que certains outils de la méthode Agile qui nous paraissent utiles et pertinents. Ainsi, nous avons donc entretenu un product backlog* et un sprint backlog*, cependant il n'y a pas de management visuel tel que le Scrum Board*. Pour le remplacer, nous avons utilisé une To Do List classique. Enfin, nous avons fait des mêlées périodiques durant chaque sprint* afin que chaque membre du groupe puisse communiquer leur avancement.

Pour suivre la progression du projet, nous avons mis en place un diagramme de Gantt afin de vérifier si les fonctionnalités voulues étaient bien implémentées.



Aspects techniques et présentation des résultats

Le Framework Flask

Flask est un Framework* web, plus précisément **un micro-framework**. “Micro” signifie que Flask ne fait pas tout. Il faudra installer des extensions pour pouvoir utiliser toutes les fonctionnalités possibles. Même si Flask est tourné web application, cela n’empêche pas que l’on puisse concevoir des applications qui ressemblent fortement à des applications mobiles. En alliant le langage Python et un **système de templates** HTML (avec du CSS, Js, etc.) très riche, on peut rapidement s’en sortir. Un système de **routes** permet de bien structurer et personnaliser l’application en leur associant une fonctionnalité précise. Flask n’a rien à envier aux frameworks comme Angular ou React. En effet, on a droit aux mêmes fonctionnalités telles que l’interpolation binding, property binding, ou encore les directives structurales (if, for, etc.)

Finalement, l’avantage de Flask est que **l’application tourne en tout temps**.

Un exemple concret tiré du projet : une route ‘/interface’ où l’utilisateur lancer le bot

```
157 @app.route('/interface', methods=['GET', 'POST'])
158 def hello_user():
159     """
160     Page dans laquelle l'utilisateur configure et lance le bot
161     Encore en développement
162     :return:
163     """
164     if request.method == 'POST':
165         if request.form['submit_button'] == 'go':
166             user = []
167             passw = []
168             order = request.form.getlist('accounts')
169
170             for account in order:
171                 cred = (db.collection(account).document('credentials')).get()
172                 user.append(cred.get('email'))
173                 passw.append(cred.get('password'))
174             tasks = db.collection('Tasks').stream()
175             # print(order)
176             # print(user)
177             # print(passw)
178             bot_test = Bot(user, passw, order, int(request.form['retrieve']), tasks)
179             bot_test.vivre()
180             notifications = bot_test.getNotificationTwitter()
181
182             elif request.form['submit_button'] == 'stop':
183                 raise Exception('Arreté prématurement')
184             return render_template('hello_user.html', notifications=notifications)
185     return render_template('hello_user.html')
```

Lignes	Commentaire
157 - 158	Définition de la route /interface dotée des méthodes HTTP POST et GET . La fonction à appeler lors de l'accès à la route est hello_user().
164	Condition : si la requête HTTP* appelante est de type POST.
165	Condition : si l'utilisateur appuie sur le bouton 'go' dans le formulaire de la page
168	Récupération des champs (paramètres) du formulaire de la page correspondant aux comptes à gérer, les identifiants etc.
174	Récupération de la liste des tâches dans la base de données Firestore. La fonction stream() ne récupère pas tout d'un coup mais établit un lien (flux) avec la base de données.
178 - 179	Instanciation* du bot avec la liste des identifiants, les comptes à gérer, sa durée de vie et la flux vers la base de données.
180	Récupération des notifications Twitter dans la variable 'notifications'.
184	Rendu de la page hello_user.html avec la variable 'notifications'.

La page HTML générée (en pré-rendu) :

Interface de parametrag

Comptes à gerer

☐ Twitter
☐ Facebook
☐ Gmail

Expiration du bot en minutes

placeh

Lancer

Terminer

Durant cet intervalle de temps le bot va aller executer les taches dans la base de données et récupérer les infos

Log des actions :

```

{% with messages = get_flashed_messages() %} {% if
messages %}

    {% for message in messages %}
    • {{ message }}
    {% endfor %}

{% endif %} {% endwith %} {% block body %} {%
endblock %}

```

Dernières notifications

```

{{ notifications }}

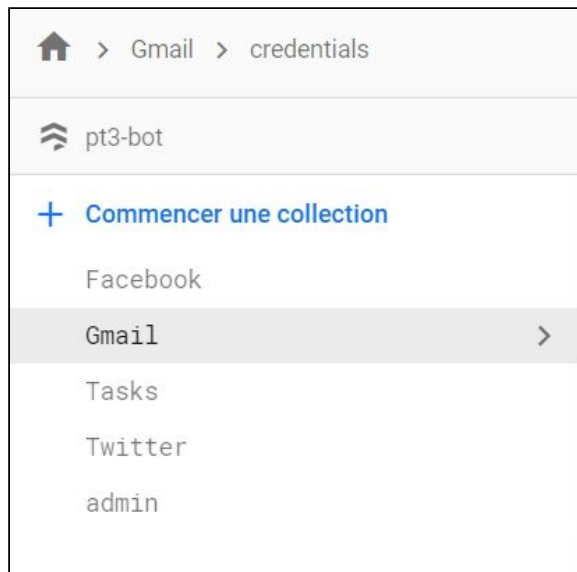
```

Une partie du code html :

```
47     <div class="form-group">
48         <label class="col-md-4 control-label" for="statea"></label>
49         <div class="col-md-8">
50             <button class="blue" id="statea" type="submit"
51                 name="submit_button" class="btn btn-success" value="go">Lancer
52             </button>
53             <button class="red" id="stateb" type="submit"
54                 name="submit_button" class="btn btn-danger" value="stop">Terminer
55             </button>
56         </div>
57     </div>
58
59 </fieldset>
60 </form>
61 </body>
62
63 <fieldset>
64     <h2>Log des actions :</h2>
65     {% with messages = get_flashed_messages() %}
66     {% if messages %}
67         <ul class="flashes">
68             {% for message in messages %}
69                 <li>{{ message }}</li>
70             {% endfor %}
71         </ul>
72     {% endif %}
73     {% endwith %}
74     {% block body %}{% endblock %}
75 </fieldset>
76 <fieldset>
77     <h2> Dernières notifications </h2>
78     <p>{{ notifications }}</p>
79 </fieldset>
```

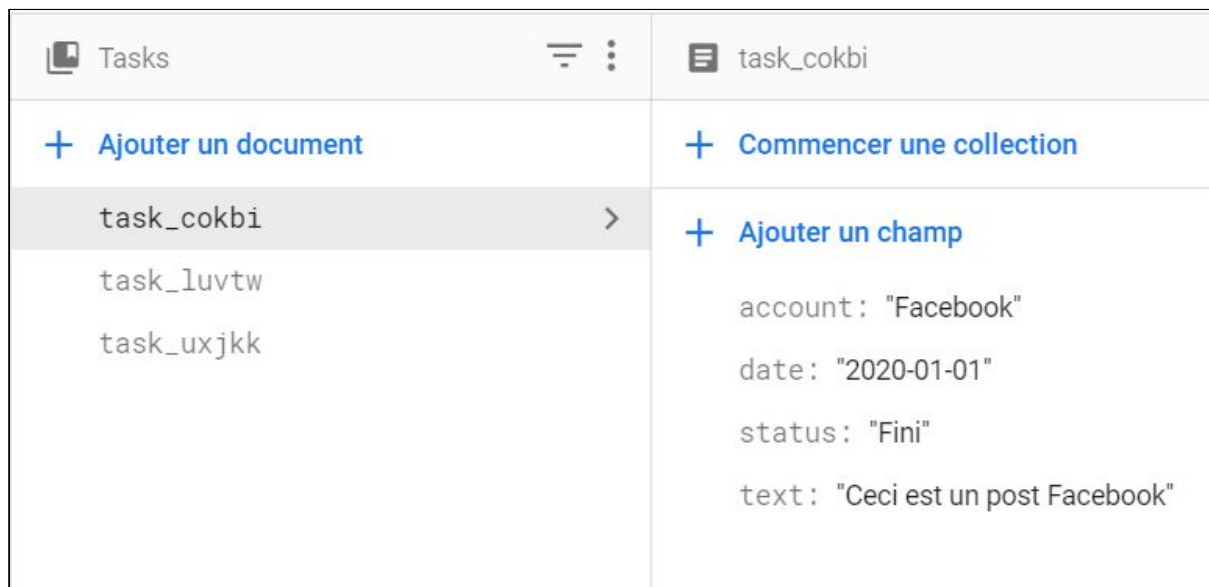
Lignes	Commentaire
65	Récupération des messages Flash*, qui permettent d'afficher des messages dans le navigateur sans rafraîchir la page. C'est le bot qui envoie ces messages : dans notre cas, les dernières actions réalisées par ce dernier.
66	Directive if qui vérifie si des messages existent bien.
68	Directive for sur la balise qui va afficher la liste de tous les messages.
69	Interpolation binding* avec la variable message pour afficher sa valeur.
78	Interpolation binding pour afficher le contenu de la variable notifications, passé en paramètre avant le rendu.

Base de données Firebase



Cloud Firestore est une base de données NoSQL construite pour des applications en général. Elle est basée sur le type Documents. Son objectif est de fournir une expérience utilisateur exceptionnelle et de simplifier le développement d'applications avec la synchronisation en direct, la prise en charge hors ligne et les transactions ACID sur des centaines de documents et de collections. Nous avons **crypté toutes les informations de connexion en SHA-256** pour un maximum de sécurité.

Le document Tasks contient la liste des tâches sous forme de collections. Une collection correspond à un enregistrement dans la table.



Fonctionnement du bot

Le bot est semi-autonome, on définit la liste des tâches qu'il doit réaliser. Une durée de vie ou il sera actif est calculée. Une boucle va aussi le maintenir en vie jusqu'à ce qu'il ait effectué cette action. Si on ajoute une deuxième action pour un temps plus lointain, cette durée de vie va être augmentée en conséquence. Ce bot est doté de toutes les fonctionnalités et API citées dans cette partie. La fonction la plus importante est **la fonction vivre()** car c'est elle qui lance le bot et le fait travailler.

L'exemple suivant nous montre ce qu'il fait dans le cas précis d'une tâche Twitter :

```
89 def vivre(self): # prend en parametres des minutes
90     """
91     Fonction a appeler pour démarrer le bot.
92     Le bot meurt au bout d'un temps défini en parametres de son constructeur.
93     Il va alors chercher dans la base de donnees toutes les taches qu'il doit executer et agir en fonction
94     du type de compte.
95     Pour le moment, il n'implémente pas encore un module d'IA.
96     Les fonctions des autres réseaux twittet, gmail, facebook, etc sont développées en parallele mais ne sont pas
97     encore integrees au bot.
98
99     :return: void
100     """
101     print('Le bot va vivre ' + str(self.suicideDay) + ' minutes')
102     heure_depart = time.time()
103     expiration = heure_depart + self.suicideDay * 60
104     print('Il est ' + str(time.strftime("%I : %M %p", time.localtime(heure_depart))) + ' actuellement')
105     print('Le bot prendra fin à ' + str(time.strftime("%I : %M %p", time.localtime(expiration))))
106
107     while time.time() < expiration:
108
109         for task in self.tasks:
110             print('Compte : ' + task.get('account') + ' Publication : ' + task.get('text'))
111             if task.get('account') == 'Twitter':
112                 self.twitter_instance.logIn2(username, password) # à cacher
113                 self.twitter_instance.post(task.get('text'))
114                 self.twitter_instance.logOut()
115                 self.save_to_log(task.get('text'), 'Twitter')
116                 flash(self.return_last_action()) # affichage de l'action
117                 self.send_sms_notification(self.return_last_action()) # envoi de notification message
118                 task.get().update({"status": "Fini"})
```

Petite explication :

Ligne 103, le bot se "suicide" au bout d'un certain temps.

Ligne 111, si la tâche est à effectuer est sur Twitter, on appelle toutes les fonctions API nécessaires. Après un succès, il l'enregistre dans un log qui sera affiché dans la page.

L'API gmail

L'API gmail est une API* proposée par la société google. Cette API permet de créer des applications qui permettent d'utiliser plusieurs fonctionnalités de gmail telles que les labels*, les mails triés en fonction de catégories...

Pour utiliser cette API, il faut obtenir les credentials* d'un compte google et les inclure au projet ainsi que l'autorisation par l'utilisateur de l'application afin que cette dernière puisse exploiter son compte.

Grâce à cette API, notre application est capable de trier les mails en fonction de l'objet ou du destinataire. Par exemple, on peut récupérer les mails qui ont dans leur objet le mot clé "google" ou "frite" ou alors on peut savoir quels sont les messages envoyés par une adresse email en particulier.

Un exemple concret tiré du projet : obtenir les différents Labels de notre boîte mail et obtenir l'ensemble des messages grâce aux labels obtenus précédemment.

```
98     def getLabels(self):
99         """ Retourne tous les labels """
100         results = self._service.users().labels().list(userId='me').execute()
101         labels = results.get('labels', [])
102
103         if not labels:
104             print('No labels found.')
105             return None
106         else:
107             self.labels = labels
108             print('Labels:')
109             for label in labels:
110                 print(label['name'])
111             return labels
112
113     def getAllMsgByLabels(self, search=""):
114         """ Retourne tous les messages """
115         labels = [ "INBOX", "CATEGORY_PERSONAL" ]
116         messages = self._ListMessagesWithLabels(labels, search)
117         for m in messages:
118             message = self._GetMessage(msg_id=m["id"])
119         return self._messages
```

Lignes	Commentaire
100	Création de la variable results qui possède la liste des différents labels qui lui sont propres.
101	Création de la variable labels qui va retourner 'results' sous la forme d'un tableau de labels
103 - 105	Si la variable labels n'existe pas alors 'None' est retourné
106 - 111	Si la variable labels existe alors les labels sont retournés
115	On sélectionne les labels "INBOX" et "CATEGORY PERSONAL"
116	Création de la variable messages qui va contenir l'ensemble des messages (mails) qui possèdent les caractéristiques précisées
117	Récupération itérative* des mails avec get()

Les API Selenium, Twython et pytesseract

Nous allons maintenant vous parler de la façon dont nous avons codé les fonctionnalités du bot, par exemple effectuer un tweet.

Nous avons commencé par créer un module* par réseau social, ainsi on a un module Twitter qui va contenir **toutes les fonctions de bases** concernant ce réseau social. Ensuite, nous avons fait des recherches sur les API existantes, afin de pouvoir **programmer facilement** nos fonctions.

Notre premier choix s'est porté sur l'**API Selenium**, qui est un framework de test : il va effectuer des actions automatiquement sur un navigateur web à notre place. Son principe est plutôt simple : il va se baser sur le code html et css d'une page web pour **rechercher des éléments** puis va effectuer **des actions** dessus (envoi de texte, clic souris).

La 1ère étape est de créer une instance* d'un navigateur internet, Firefox dans notre cas.

Ensuite, on peut lui demander de rechercher des éléments selon **des critères différents** (par chemin xpath, par nom de classe, par id, etc.). Enfin, on peut effectuer diverses actions en fonction de l'élément ciblé (envoi de texte pour un champ input, clic souris sur un lien href ou un bouton).

Explication des fonctionnalités :

1) Se connecter sur Twitter avec Selenium

Code :

```
37 def login(self, token1, token2):
38     # ...
47     opts = Options()
48     opts.headless = True
49     try:
50         self.driver = Firefox(options=opts)
51         self.driver.get("https://twitter.com/login")
52     except WebDriverException as exception:
53         print("Driver could not connect")
54         return False
55     try:
56         username_field = self.driver.find_element_by_class_name("js-username-field")
57         password_field = self.driver.find_element_by_class_name("js-password-field")
58     except NoSuchElementException as exception:
59         print("Element not found and test failed")
60         return False
61     time.sleep(2)
62     username_field.send_keys(token1)
63     self.driver.implicitly_wait(1)
64     time.sleep(2)
65     password_field.send_keys(token2)
66     self.driver.implicitly_wait(1)
67     self.driver.find_element_by_class_name("EdgeButtom--medium").click()
```

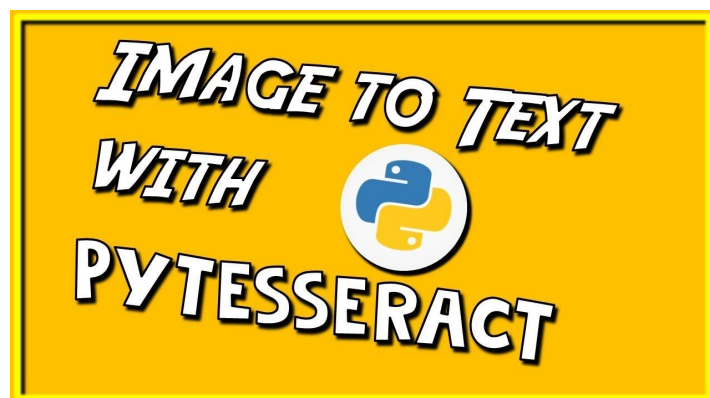
Lignes	Commentaire
47 - 48	Création d'une option pour navigateur, puis affectation de son champ headless à true (pour activer l'affichage en arrière plan).
49 - 54	Ouverture du driver* Firefox avec l'option opts, pour qu'il s'ouvre en arrière plan. Accès à la page de login de Twitter. En cas d'impossibilité d'ouverture, on retourne False.
55 - 60	Recherche des éléments d'authentification (champ username et champ password) par le nom de leurs modules. Exception : si les champs ne sont pas trouvés (mauvaise connexion, changement du site).
61 - 65	Envoi de texte dans les éléments trouvés précédemment.
67	On cherche et on clique sur le bouton de login.

Les temps d'attente `time.sleep` servent à simuler les actions d'un humain (sinon, Selenium effectue les actions trop rapidement).

Les temps d'attente `implicit_wait` sont propres à Selenium, et servent à lui donner un temps pour rechercher les éléments (il va attendre x secondes pour trouver l'élément avant de passer à la suite).

2) Obtenir les dernières notifications avec Selenium, pytesseract (OCR-Tesseract)

Dans cette fonction, on va utiliser le module `pytesseract` en plus de Selenium. Celui permet l'utilisation de l'OCR* en Python. Cette fonction et les autres ne sont utilisables qu'après avoir utilisé la fonction de connexion `login`.



```

171 def getNotifications(self):
172     """
177     self.driver.get("https://twitter.com/notifications")
178     time.sleep(2)
179
180     self.driver.implicitly_wait(10)
181     element = self.driver.find_element_by_xpath(
182         '/html/body/div/div/div/div/main/div/div/div/div[1]/div/div[2]/div/section/div')
183     location = element.location
184     size = element.size
185     png = self.driver.get_screenshot_as_png()
186     im = Image.open(BytesIO(png))
187     left = location['x']
188     top = location['y']
189     right = location['x'] + size['width']
190     bottom = location['y'] + size['height']
191
192     im = im.crop((left, top, right, bottom))
193     im.save('screenshot.png')
194     pytesseract.pytesseract.tesseract_cmd = 'C:/Program Files (x86)/Tesseract-OCR/tesseract'
195     text = pytesseract.image_to_string(Image.open('screenshot.png'))
196     return text

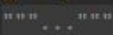
```

Lignes	Commentaire
177	Accès à la page des notifications de Twitter.
181 - 184	Recherche de la zone html contenant les notifications et création des attributs location et taille de cet élément.
185 - 186	Capture d'écran de la page par le driver puis ouverture de cette image.
187 - 193	Configuration de l'image : rognure grâce aux attributs location et size, afin de n'avoir que la zone visée. Sauvegarde de l'image en format png.
194 - 196	Indication du chemin vers Tesseract-OCR. Utilisation de l'OCR tesseract pour obtenir le texte.

3) Obtenir des tweets avec l'API Twython

Ici, on a utilisé une API de Twitter pour rechercher et renvoyer un nombre n de tweets populaires sur un thème.


```

154     def getPopularTweets(self, theme, number):
155         
164         twitter = Twython(self.APP_KEY, self.APP_SECRET)
165         results_popular = twitter.search(q=theme, result_type='popular', count=number)
166
167         return results_popular

```

Lignes	Commentaire
164	Création d'une instance Twython avec les clés de l'application créées sur le site de Twitter.
165	Utilisation d'une fonction de l'API pour chercher des tweets.
167	Les tweets sont retournés sous forme de chaîne de caractères.

Pour afficher les tweets :

```

all_tweets_popular = results_popular['statuses']

for tweet2 in all_tweets_popular:
    print(tweet2['text'])

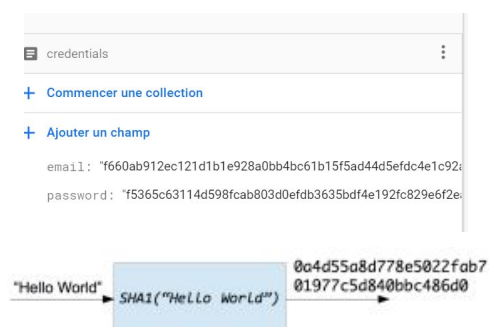
```

4) Authentification via OAuth

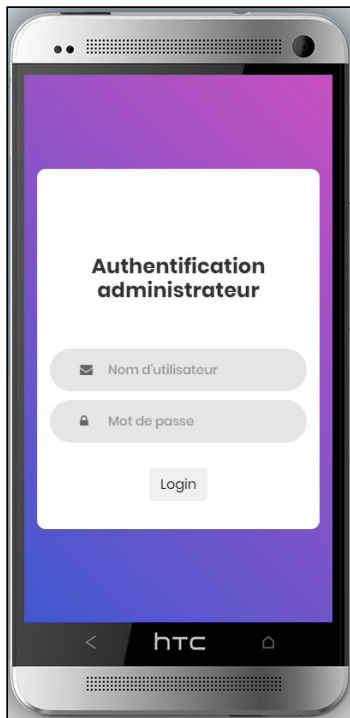
OAuth est un protocole permettant de donner des autorisations à une application ou un site, d'utiliser des API. Pour utiliser une API Twitter, il est nécessaire de passer par ce protocole afin d'obtenir le droit d'effectuer des actions sur son compte.

5) Cryptage

On a utilisé le module hashlib qui contient différents moyens de faire du hachage sécurisé d'une chaîne de caractères. Le hash des identifiants tapés dans l'application sont comparés au hash dans la base de données.



Architecture de l'application (simulée sur un téléphone)



Page de connexion administrateur



Page servant à entrer nos différents comptes



Interface de démarrage le bot

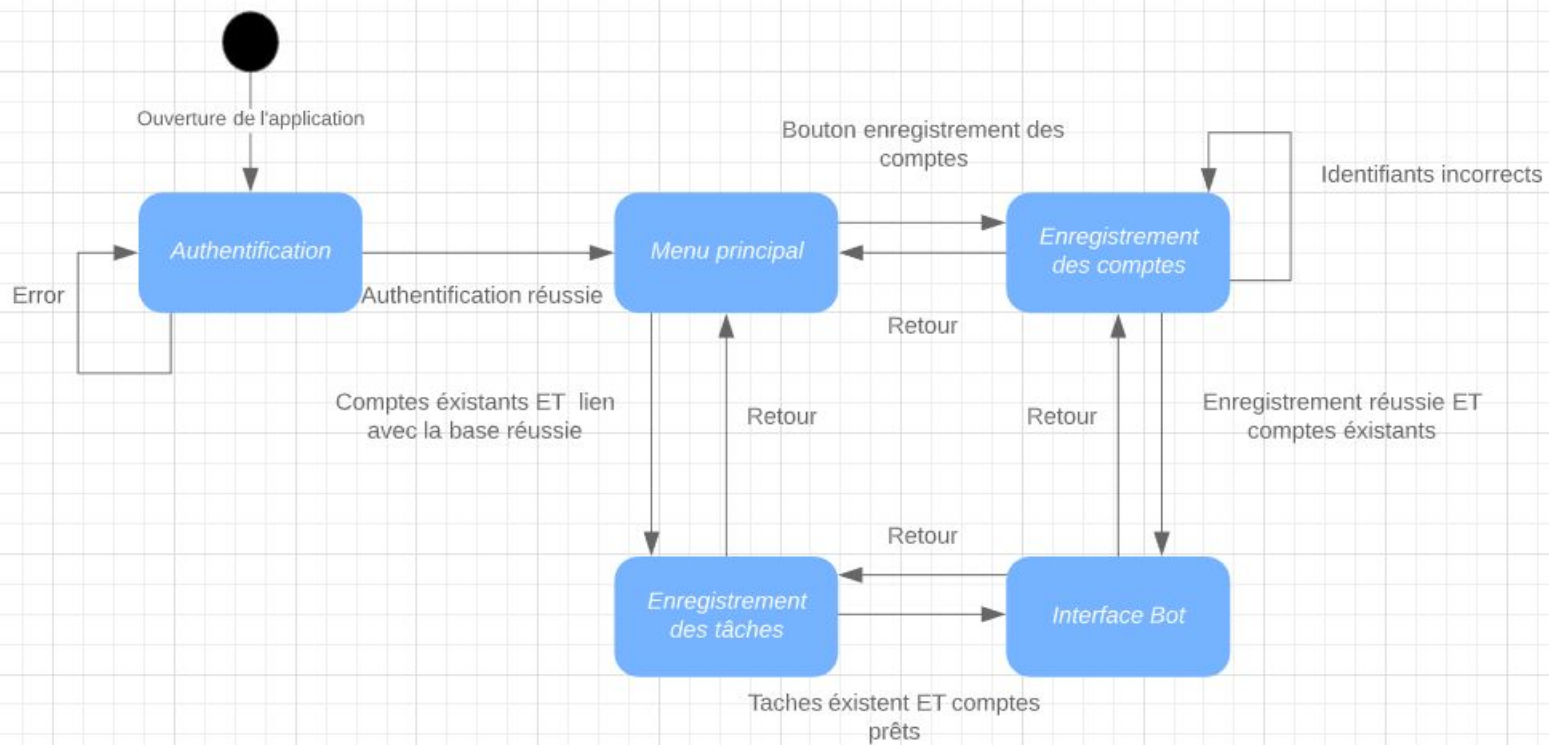


Interface d'enregistrement des tâches



Menu principal

Flux de navigation dans l'application : états-transitions



Conclusion

À l'issue de ce projet, nous avons réussi à programmer un bot capable de faire quelques actions sur les réseaux sociaux. Nous considérons ce projet comme réussi mais il reste encore améliorable avec l'ajout de nouvelles fonctionnalités et la maintenance de celles déjà en place.

Cependant, au vu du temps imposé, il était impossible d'intégrer l'ensemble des fonctionnalités que l'on avait prévu au début du projet. Ainsi, si c'était à refaire, nous reverrions nos objectifs afin qu'ils soient réalisables dans le temps donné.

Avis de Benoît : Pour moi, ce projet fût une expérience enrichissante car j'ai pu apprendre à utiliser diverses API et à comprendre des documentations qui ne sont pas toujours très bien expliquées. De plus, ce projet m'a permis d'organiser mon code afin que l'ensemble du groupe puisse utiliser mes implémentations. Enfin, ce projet m'a fait progresser dans l'apprentissage du langage python.

Avis de Sébastien : Ce projet m'a permis d'apprendre à utiliser de nouveaux outils (API) et les faire interagir ensemble. De plus, j'ai pu développer mes connaissances en Python. D'un point de vue humain, j'ai appris à travailler en équipe et à m'organiser en fonction de l'évolution du projet.

Avis de Rova : Grâce à ce projet ambitieux, j'ai pu me familiariser avec les techniques Agiles même si ce n'est que dans la forme. Par ailleurs, j'ai pu me forger un bagage technique assez conséquent qui me sera sans doute utile dans la poursuite de mes études. Finalement, j'ai appris aimer le travail d'équipe. Cela s'est vu à travers une bonne entente et une bonne communication au sein du groupe.

Analyse critique du projet

Points faibles du projet :

- Autorisations d'accès aux API parfois compliquées à obtenir (longue procédure).
- On se perd facilement dans toutes les fonctionnalités à créer et implémenter : il faut bien s'organiser.
- L'application n'est pas assez ergonomique pour le moment.
- Pas assez de tests unitaires réalisés.
- Encore beaucoup de bugs et d'exceptions non gérées.
- Nous sommes un peu livrés à nous même par rapport à la conception.

Points forts du projet :

- Sujet très libre et ouvert à de nombreuses améliorations : nouvelles API, réseaux sociaux en constante évolution, etc.
- Le projet nous permet d'entrevoir plusieurs domaines de l'Informatique: front-end, back-end, API, etc.
- Projet consistant et cohérent : nous avons réussi à faire une majeure partie de ce que nous voulions.
- Bonne communication et entente dans le groupe.
- Les cours dispensés à l'IUT (EGO, WIM, ACD, etc.) nous ont largement aidé tant au niveau gestion de projet que réalisation et conception technique.

Perspectives du projet

Un certain nombre de fonctionnalités doit encore être implémenté au sein du bot.

De plus, il restera toujours de nouvelles fonctionnalités à implémenter, ainsi que d'autres réseaux sociaux à ajouter. La programmation orientée objet facilite la maintenance et l'ajout de fonctionnalités. Dans le meilleur des cas possibles, nous pourrions vendre notre application au grand public et en faire une start-up.

Rova : "J'aimerais bien continuer à maintenir ce projet à travers le PT4. Je suis particulièrement intéressé par l'intelligence artificielle. Je pourrais développer une IA qui répondrait à des messages ou à des mails en toute autonomie"

Sebastien : "Ce projet m'a beaucoup intéressé, mais je ne peux pas le continuer en tant que PT4 vu que je pars à l'étranger pour le semestre 4 "

Benoît : "Je ne sais pas encore si je vais continuer ce projet au PT4. Si je continue ce projet en solo, je pense me focaliser sur l'ajout de fonctionnalités sur les différents réseaux sociaux et améliorer l'interface utilisateur."

Overview of the project (résumé du projet en anglais)

The purpose of this project is to build an application, more specifically a bot. This one will surf independently on the internet, within social networks such as Instagram, Facebook, Twitter and so on. Our primary mission is to **make the user's life easier**. On the second hand, it will take the part/role of **a companion**, so the user will save time.

Concretely, the bot will:

- Automatically publish on a specified date, in a specified manner
- Look for informations and resume them in a dashboard
- Interact (automatic responses, likes, following ...)

Our thought process:

First of all, we studied the current status of social networks among a large panel of users of all ages. Most of these users do not use only one social network. It can become very constraining and time consuming to manage our digital life. According to a recent study conducted in 2017, an 18-year-old in Europe spends an average of 70 hours on his or her smartphone per month, which represents almost 3 days. We therefore see this as an opportunity to build an app that keeps us informed while saving our precious time.

Working method:

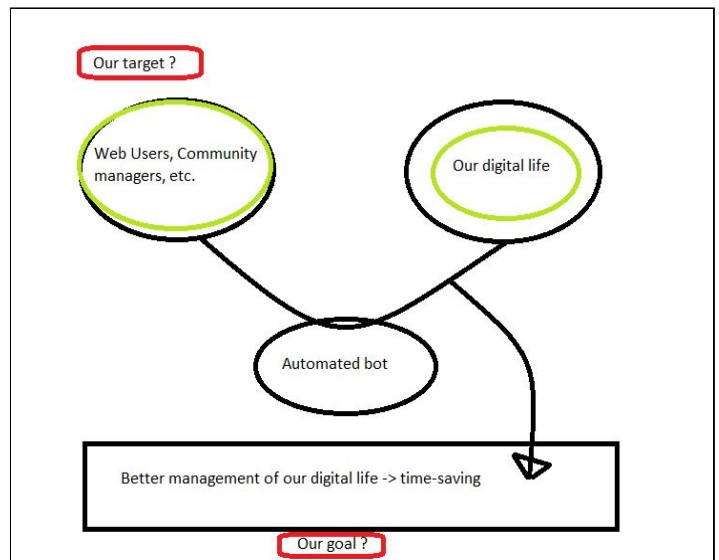
We have chosen to use the **agile methodology** instead of the classic methodology for two main reasons. On the one hand, our application will have a large number of features. On the other, we have to develop a launchable application as early as possible. We are at the same time developers, designers, product owners and scrum master*, so it is not easy to define who will do what.

However, we have decided to use a classic tool : a To Do List. This one allows us to organize tasks chronologically.

This project taught us to trust everyone's work and working in a team has been a rewarding experience. It prepares us for the world of work.

If a person in the group has difficulties on a development point, the other members of the group would help him.

Mutual help and cooperation are the two keywords of this project.



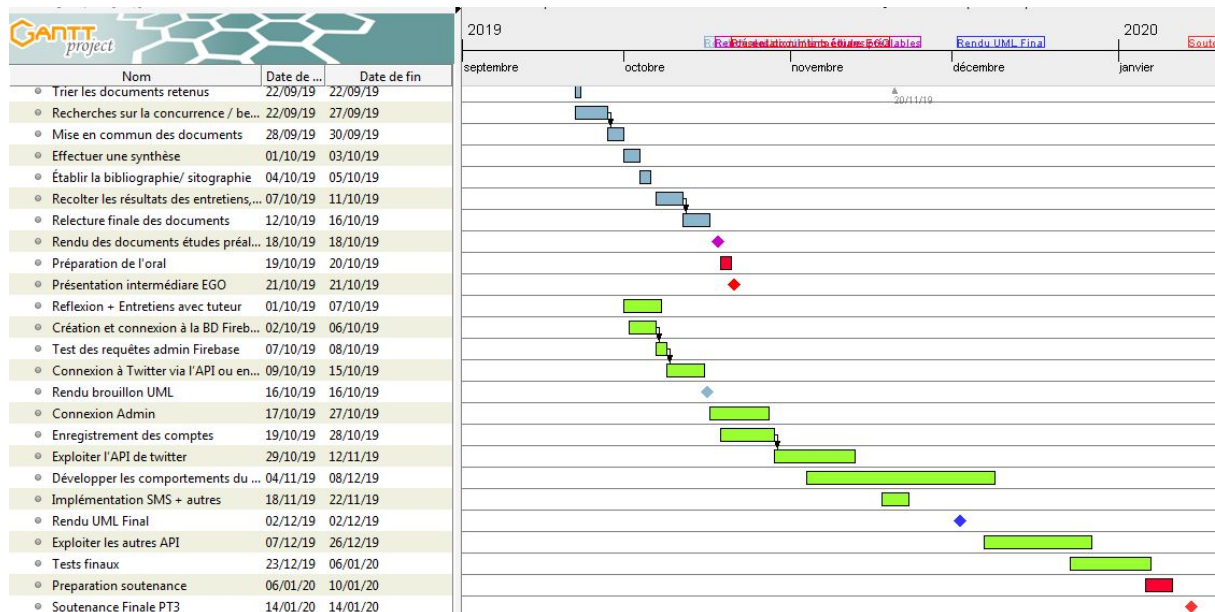
Glossaire

API	En informatique, une interface de programmation d'application est un ensemble de classes, de méthodes, de fonctions et de constantes qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels.
Bot informatique	Aussi appelé robot, le bot est un programme informatique réalisant des tâches automatisées, tel un assistant virtuel. Il peut par exemple parcourir le web pour indexer les pages dans les navigateurs, optimiser votre agenda ou encore chatter en direct réseaux sociaux.
Credentials	Identifiants de connexion.
Driver	WebDriver est un framework de tests fonctionnels issu du projet Selenium.
Framework	Ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel.
Instance/Instanciation	En programmation orientée objet, on appelle instance d'une classe un objet avec un comportement et un état, tous deux définis par la classe. On appelle instanciation le fait d'initialiser un objet.
Interpolation binding	Permet d'afficher la valeur d'une propriété / variable d'un composant / classe dans un template avec la syntaxe des accolades doubles {{}}.
Itératif	Qui est répété plusieurs fois.
Label	Une étiquette de mails qui permet de les catégoriser.
Module	Un module python est un fichier .py qui contient des définitions de fonctions, classes et des instructions.
OCR	Reconnaissance optique de caractères (ROC), en anglais optical character recognition (OCR) désigne les procédés informatiques pour la traduction d'images de textes imprimés ou dactylographiés en fichiers de texte.
Product backlog	Le backlog scrum est destiné à recueillir tous les besoins du client que l'équipe projet doit réaliser.

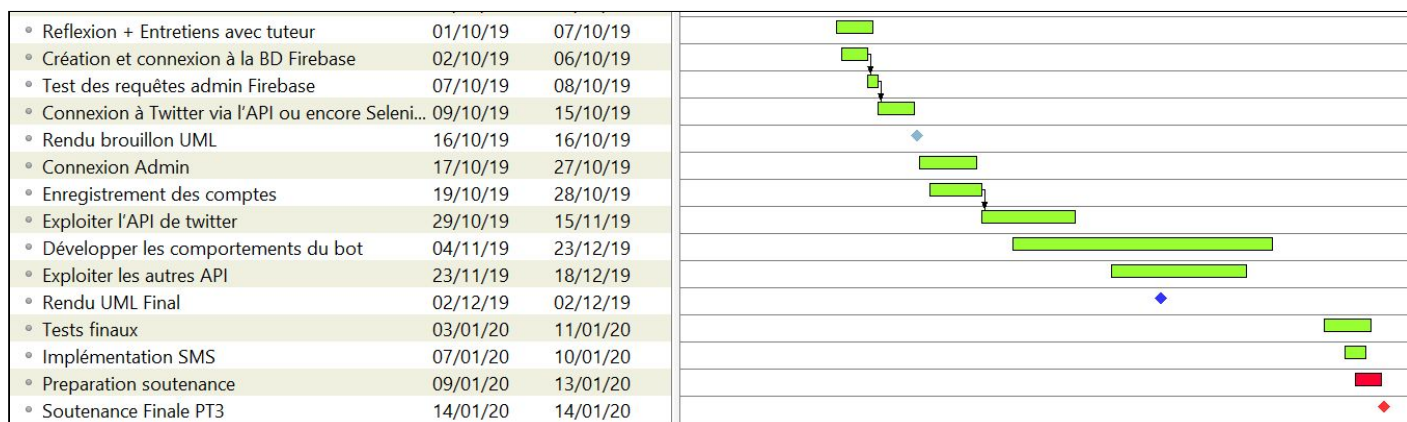
Sprint backlog	Ensemble des items pris en charge dans un sprint.
Requête HTTP	Une requête HTTP est un ensemble de lignes envoyé au serveur par le client. Il contient une ligne de requête, avec les méthodes (GET ou POST), et des champs d'en-tête.
Scrum Board	Élément permettant de suivre la progression de la réalisation des différentes tâches et user stories tout au long du sprint courant.
Scrum master	Le Scrum Master est avant tout un coach, tandis que le Product Owner est comparable à un chef de projet.
Sprint	Un sprint est une itération de développement de la méthode Scrum. Il dure généralement entre deux et quatre semaines. Les fonctionnalités réalisées durant le sprint sont sélectionnées dans le « product backlog » renseigné par le « product owner ».

Annexe

Planning



Planning prévisionnel établi au début du projet



Avancement final concernant l'aspect informatique

Analyse des besoins

Nous remarquons une tendance particulière chez les 18-25 ans, la majorité d'entre eux passent plus de 3h par jour sur les réseaux sociaux. Nous avons potentiellement 60% d'entre eux qui se disent que le temps qu'il passent en ligne est excessif. Aussi, nous remarquons que ce qu'il vont aller chercher dessus, la plupart du temps, ce sont de informations non essentielles qu'il oublient souvent après quelques temps (publications drôles, vidéos / podcast, etc.).

Certains d'entre eux proposent des solutions qui leur permettraient d'économiser leur temps : ils pensent à l'affichage du temps d'utilisation, ou encore à un blocage automatique après un certain temps.

Ce que nous pouvons retenir, c'est que dans cette tranche d'âge la notion de bot n'est pas encore très répandue. En revanche, chez ceux qui en ont déjà rencontré, ils connaissent tous les chatbots, les bots twitter qui font des retweets, ou encore les bots qui envoient des spams.

Finalement, nous pouvons donc en déduire qu'il y a encore des améliorations possibles afin d'aider l'utilisateur à gérer ses comptes : résumer les éléments essentiels, l'avertir de son utilisation, l'accompagner et l'éduquer par rapport à la place des bots sur internet.

De ces chiffres ressortent deux objectifs principaux qu'il faudra que cette application atteigne :

1. Cette application permettra à l'utilisateur d'économiser au moins 60% de son temps en faisant des rapports périodiques sur les sujets (publications drôles, famille, professionnel, etc.) que l'utilisateur définit en amont.
2. Cette application pourra gérer ses comptes réseaux sociaux (des différentes plateformes) en autonomie. Elle pourra par exemple gérer les demandes d'amis, les likes et différentes interactions, publier périodiquement.

Cahier des charges fonctionnel

Contraintes internes: L'interface doit être claire et compréhensible pour les nouveaux utilisateurs.

Contraintes temporelles: La recette informatique devra se faire en décembre, c'est-à-dire que les fonctionnalités voulues soient développées et testées d'ici janvier (se rapporter à la partie planification). Les sprints devront être respectés et des rendez-vous périodiques avec le tuteurs organisés. Travailler en mode Agile nous assurera que les besoins formulés seront comblés.

Contraintes sur les ressources humaines: Nous sommes 3 développeurs - concepteurs dans le groupe. Nous devons donc nous organiser pour distribuer au mieux les tâches. En plus du développement, nous devons faire nous même la conception ce qui n'est pas une tâche aisée. Nous devons donc diviser les différentes tâches selon les capacités de chacun afin d'avancer au plus vite. Comme nous ne travaillerons pas forcément sur les mêmes points, des problèmes pourront émerger. Nous prendrons donc le temps entre chaque sprint de voir ce que chacun a fait et essayerons de trouver des solutions afin d'aider celui qui bloquera.

Contraintes de développement: Le langage de développement, la plateforme ainsi que des objectifs intermédiaires seront imposés par le tuteur de projet. Nous aurons donc l'obligation de les respecter même s'il nous incombe de trouver par nous même comment procéder. Évidemment, ces obligations techniques seront plus détaillées dans le cahier des charges techniques mais il est important de les citer ici car le tuteur est implicitement celui qui formule les besoins; Concrètement, les consignes sont les suivantes :

- Le bot devra posséder et gérer ses propres comptes gmail, twitter, facebook, instagram.
- Il doit être capable d'envoyer des notifications et ses sms.
- Il devra posséder une base de données locale et en ligne.
- Son activité sur le net sera guidée de manière semi-automatique par un module d'IA
- Il devra utiliser les bibliothèques existantes et des API.

Contrainte externe: Nous devons respecter les lois en vigueur face à la manipulation des données à caractère personnel. Ce point est critique car notre application aura un accès total aux comptes des utilisateurs et leur vie privée. Nous devons ainsi garantir une sécurité infaillible pour protéger ces données critiques.

Limites du projet:

Pour le moment, nous ne développons pas une IA totalement autonome car ceci requiert des compétences que nous ne possédons pas encore. Nous ne réutiliserons en aucun cas et sous aucune forme les données personnelles des utilisateurs de l'application. Nous répondrons aux besoins formulés en évitant toute fonctionnalité superflue ou inutile.

Backlog de produit

En tant que	je souhaite	afin que /de	Critères d'acceptation	Estimation arbitraire de la difficulté	Status	Fin le
Administrateur	visualiser le contenu d'une base de données, et le modifier	le bot puisse stocker des données	Les données et paramètres sont bien enregistrés dans la base de données	5	Fait	06/10/2019 avance de 10 jours
Utilisateur	voir la liste de mes comptes enregistrés	je puisse en rajouter ou les supprimer	Les éléments modifiés doivent être pris en compte dans la base de données et affichés correctement après actualisation	8	Fait	30/10/2019 on time
Utilisateur	pouvoir enregistrer ses propres comptes	je puisse décider quel réseau le bot doit gérer	Les informations sont bien entrés dans la/les base de données	5	Fait	28/10/2019 on time
Utilisateur	définir des actions automatiques	je puisse gagner du temps / ne pas oublier	Je reçois des notifications	21 (à découper en sous tâches)	Fait	04/11/2019 retard de 20 jours
Utilisateur	être en sécurité	préservé mes données personnelles	Les données des comptes sont cryptés et difficile à décoder	5	Fait	05/01/2020 retard de 35 jours
Administrateur	devoir m'identifier	limiter l'accès aux pages de paramètres et de modification du comportement du bot	Un utilisateur non identifié ne doit pas pouvoir accéder aux pages autorisant une modification de données	5	Fait	30/11/2019 avance de 10 jours
Utilisateur	recevoir des sms / notifications	je soit au courant des actions du bot	le délais de réception est court	8	Fait	09/01/2020
Administrateur	pouvoir personnaliser la charte graphique du site	optimiser l'ergonomie du site	templates html /css bien séparés par dossiers et sous-dossiers	8	Fait	10/12/2019 on time
Administrateur / Utilisateur	avoir un fichier log (journal) à disposition	je soit au courant des actions du bot	tout y est écrit, avec la date et l'heure	13 (à découper en sous tâches)	Fait	10/11/2019 avance de 15 jours

Mode d'emploi

PT3-Bot-Informatique :

Nous sommes sur la version 3.7 de Python.

Pour tester, c'est simple:

-Aller dans PT3-Bot-Informatique/v3.0/src/main_pt3/

-puis lancer main.py avec l'environnement de votre choix après avoir réglé les imports et les logiciels à installer.

-Aller sur localhost sur le port 5000 (127.0.0.1:5000)

-Les identifiants de connexion sont = username:password

-Pour visualiser les changements dans la BD se connecter à Firestore avec les identifiants google suivants :

E-mail : robert.rich136@gmail.com | Mot de passe: JHvjsd65\$+

Console de la bd:

<https://console.firebase.google.com/project/pt3-bot/database/firestore>

TRÈS IMPORTANT

Il faut charger toutes les librairies et les modules avant de pouvoir tester.
(PyTesseract, twilio, etc.)

Il faut **installer manuellement certains package avec pip install**. Ex twilio, flask, Tools, pytesseract, etc.

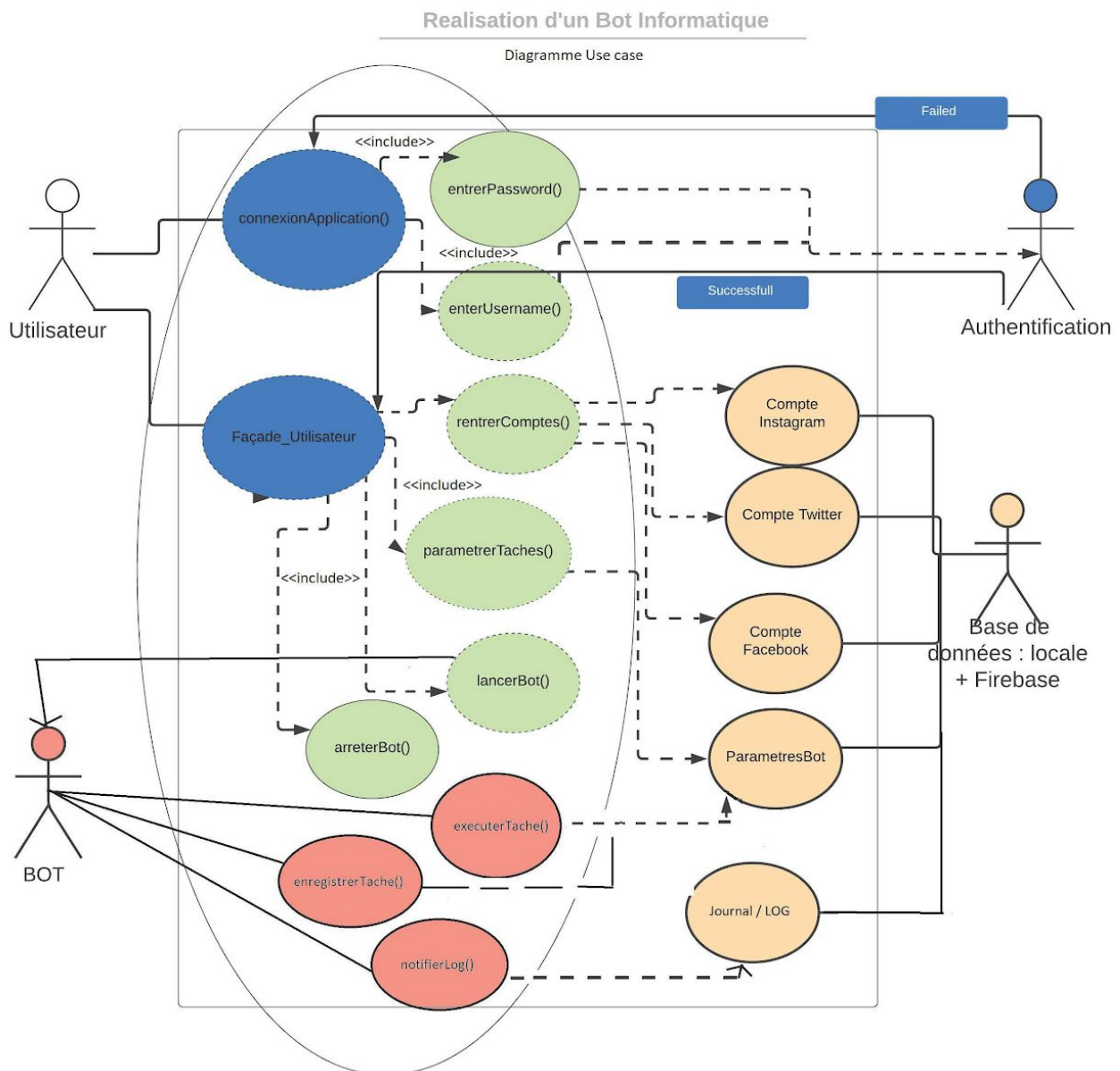
Sur PyCharm, il faut que **les packages src_bot et utility_folder soient dans le path du project interpreter** pour la version 3.0.

IL FAUT INSTALLER PYTESSERACT et éventuellement mettre à jour le chemin dans le code (module TwitterSelenium_v2.py pour la version 2.0 ou Twitter_API pour la version 3.0 fonctions getNotifications() et getMentions()). Par défaut, le chemin est : C:/Program Files (x86)/Tesseract-OCR/tesseract. **Lien pour telecharger l'OCR PyTesseract :**
<https://github.com/tesseract-ocr/tesseract/wiki/4.0-with-LSTM#400-alpha-for-windows>

Diagrammes : Cas d'utilisation et UML

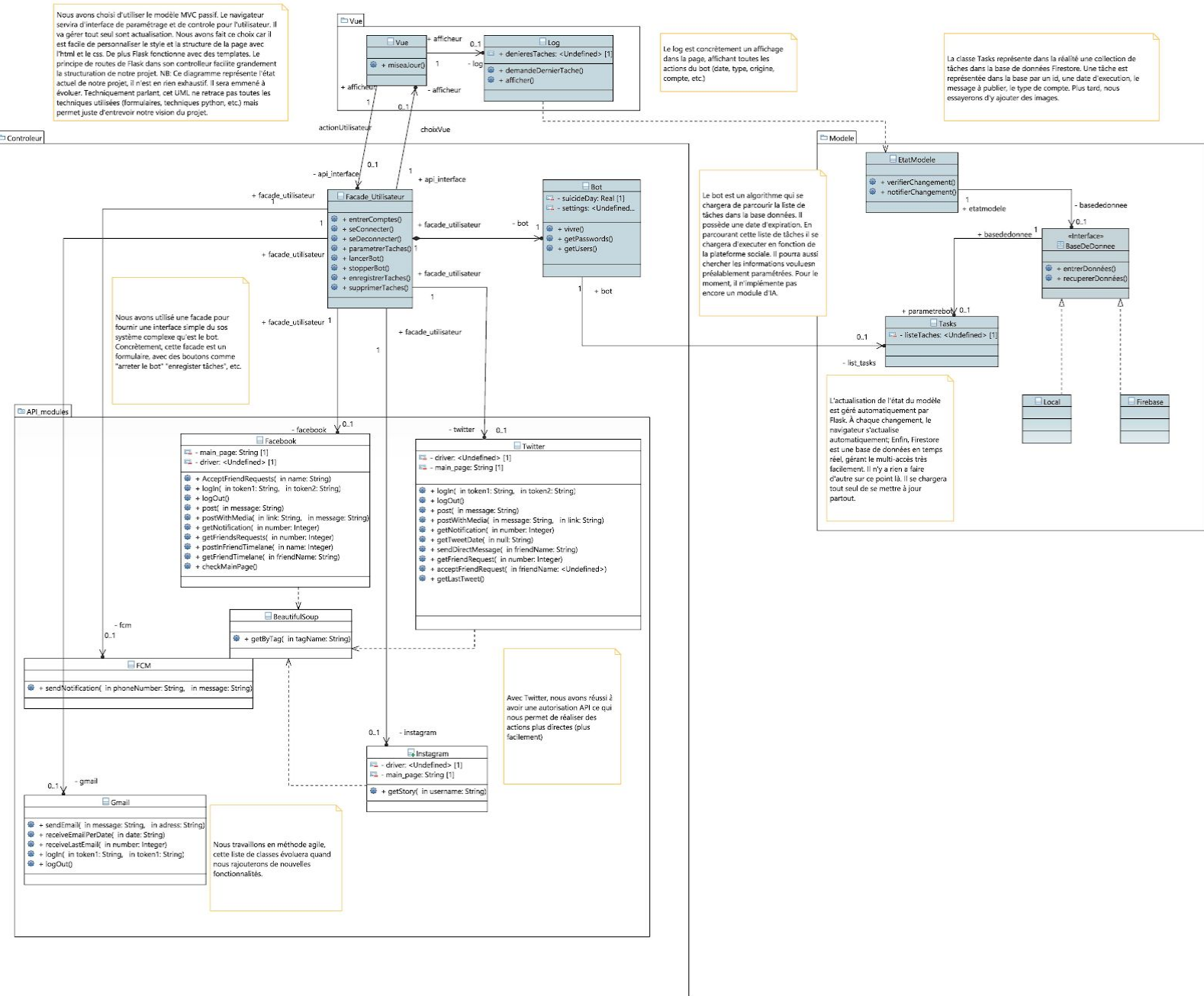
Lien pour le diagramme de cas d'utilisation en meilleure qualité :

https://drive.google.com/file/d/1KC08J884O6S4myFQCB4TfNO2ab_Aj0aA/view?usp=sharing



Lien pour voir le diagramme UML en meilleure qualité:

https://drive.google.com/file/d/1WdwmbWldVCiY8M2bkFDo_wQY9UwU6Zn/view?usp=sharing



Budget et coûts

Ce projet n'est pas soumis à des contraintes budgétaires.

Les API et les librairies que nous avons exploitées sont quasiment gratuites. Cependant, elles possèdent des fonctionnalités qui sont payantes après un certain quota d'utilisation.