



Programmer's guide

**Valentin Arioti, Augustin Nogue,
Babacar Sow, Innocent Ye**

UNamur
Belgique
Mai 2022

Table des matières

1	Introduction	2
2	Sources	2
2.1	Backend	3
2.2	Frontend	3
2.3	Plugins	3
3	Documentation	4
4	Développeurs	4
5	Commanditaire	4
6	Environnement	5
7	Testing	5
8	Architecture	6
9	Déploiement	7
9.1	Localhost	7
9.2	Localhost et redirection en ligne	7
9.3	Hébergeurs en ligne	7
9.4	Google API et Gmail	8
9.4.1	Google API	8
9.4.2	Gmail	10

1 Introduction

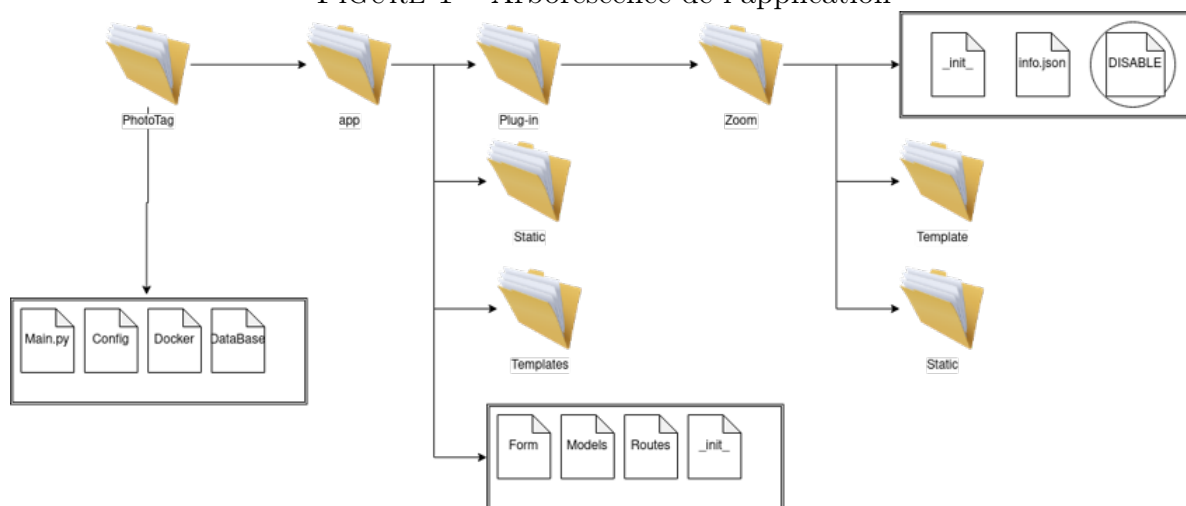
PhotoTag est une application d'annotation d'images et de vidéos, intelligente et collaborative. Elle permet de :

1. Visualiser et annoter des photos et des vidéos
2. Collaborer avec d'autres utilisateurs en temps réel
3. Ajouter, activer et désactiver facilement des plugins

Pour plus d'informations sur le fonctionnement de l'application, vous pouvez regarder une vidéo de démonstrations disponible à l'adresse suivante : <https://youtu.be/GZDZ6BKirKo>

2 Sources

FIGURE 1 – Arborescence de l'application



Dans le dossier principal, se trouvent :

1. Le fichier **main.py** qui va permettre de lancer l'application web.
2. Le fichier **config.py** qui va permettre de configurer le module. sqlalchemy. C'est dans ce fichier qu'il faut mettre l'URI de la base de données SQL. Pour l'instant c'est le fichier **users.db**, qui est créé dans le dossier principal à la première initialisation de l'application, qui sert de base de données.
3. Le fichier **dockerfile** qui va permettre de lancer l'app depuis un conteneur docker.
4. Le fichier **uwsgi.ini** qui sera utile pour héberger l'app sur un serveur.
5. Le dossier **test** qui contient quelques tests unitaires sur les méthodes de liées à la base de données.
6. Le dossier **app** qui contient tout le code de l'application web, expliqué dans les sous-sections suivantes.

2.1 Backend

1. **__init__.py** : initialise l'app, ces paramètres, et ces modules annexes (login et plugin manager, socketio, sqlalchemy).
2. **routes.py** : contient toutes les routes de l'app organisée par catégories.
3. **models.py** : modélisation des tables SQL en classes Python.
4. **forms.py** : formulaires utilisés par l'app.
5. **utils** : contient des méthodes secondaires utilisées dans les fichiers précédents
 - **keygenerator.py** : génère une clé pour le reset de password
 - **smtpConfig.py** : contient les identifiants du compte Google utilisé pour envoyer les mails
 - **utilsPhotoTag.py** : contient les méthodes qui permettent de couper une vidéo en images

2.2 Frontend

Pour le frontend, **HTML**, **Javascript** et **CSS** ont été utilisé car.

Les pages **HTML** se trouvent dans le dossier *app/templates* et sont organisés en catégories. La page *parent.html* contient la navbar principale et est héritée par les autres pages qui ne sont pas relatives à un projet. La page *parent2.html* contient la navbar principale et une navbar relative au projet et est héritée par les pages sont relatives à ce projet.

Les scripts **Javascript** se trouvent dans le dossier *app/static/js*. La plupart de ces scripts sont des outils pour la page d'annotation et se trouvent dans le sous-dossier *annotation*.

Les feuilles de style **CSS** se trouvent dans le dossier *app/static/css*

2.3 Plugins

La librairie [flask-plugin](#) permet à l'app de gérer facilement les plugins. Voici les étapes à suivre pour ajouter, activer et désactiver un plugin facilement à l'app **PhotoTag**.

1. Dans le dossier *app/plugins*, créer un nouveau dossier pour le nouveau plugin.
2. Dans ce dossier, ajouter un fichier **info.json** qui contient les informations du plugin (identifier, name, author, license, description, version, ...)
3. Créer un fichier **__init__.py** et dans celui-ci, créer une classe qui hérite de *AppPlugin*, il faut également créer un **Blueprint**.
4. Dans ce fichier, il est possible d'implémenter des nouvelles routes (pour faire de nouvelles pages ou ajouter des sections à des pages existantes via l'héritage) ou bien des événements qui vont aller modifier directement le comportement de pages existantes. Cette deuxième solution permet d'ajouter des éléments sur une même page, depuis plusieurs plugins différents.
5. Pour le code frontend de ces plugins, il est également possible d'utiliser les sous-dossiers **templates** et **static**. Il est possible également d'écrire directement du HTML dans la méthode Python grâce à **render_template_string**.

6. Pour voir les éléments du plugin dans l'app via les évènements, il est nécessaire de les connecter à une fonction via à un nom d'évènement.

```
connect_event("tmpl_before_content_annotate_annel", inject_zoom_script)
```

7. Ce nom d'évènement est utilisé ensuite dans le code grâce de l'app pour afficher le code du/des plugin(s) qui ont une connexion avec cet évènement.

```
emit_event("tmpl_before_content_annotate_annel")
```

8. Pour désactiver un plugin, il suffit de rajouter un fichier vide nommé **DISABLE** dans le dossier du plugin à désactiver, et ensuite, relancer l'app pour que cela prenne effet.

3 Documentation

La documentation nécessaire pour ce projet se trouve à ces endroits :

1. **Démo** : Une vidéo de démonstration de l'app est disponible : <https://youtu.be/GZDZ6BKirKo>.
2. **Programmer's guide** : Le guide pour programmeurs se trouve à l'emplacement `doc/programmerguide.pdf`
3. **Flask (Python)** : Documentation pour [Flask](#).
 - `flask_login` : Documentation pour la librairie [flask_login](#).
 - `flask-wtf` : Documentation pour la librairie [flask-wtf](#).
 - `flask-plugins` : Documentation pour la librairie [flask-plugins](#).
 - `flask_sqlalchemy` : Documentation pour la librairie [flask_sqlalchemy](#).
 - `flask-socketio` : Documentation pour la librairie [flask-socketio](#).
4. **yagmail (Python)** : Documentation pour la librairie [yagmail](#).
5. **google-auth (Python)** : Documentation pour la librairie [google-auth](#).
6. **opencv-python (Python)** : Documentation pour la librairie [opencv-python](#).
7. **unittest (Python)** : Documentation pour la librairie [unittest](#).
8. **Déploiement Flask** : Documentation pour [déployer une app Flask](#)

4 Développeurs

Valentin Arioti, Augustin Nogue, Babacar Sow, Innocent Ye : étudiants en Master à l'UNamur (Belgique)

5 Commanditaire

Le projet **PhotoTag** a été proposé par Guillaume Maître dans le cadre du cours INFOM114 (Ingénierie du logiciel et laboratoire) à l'UNamur.

6 Environnement

Le serveur **Flask** est facile à déployer sur tous les OS, nous l'avons testé sur Windows 10 et Ubuntu 20, mais cela devrait également fonctionner sur MacOS. Nous avons choisi de faire un serveur web car cela facilite la collaboration de plusieurs utilisateurs en temps-réel, et nous avons choisi de le réaliser avec **Flask** car ce framework possède de nombreux modules utiles pour ce projet.

Pour programmer avec Python, nous avons utilisé un environnement virtuel avec *virtualenv* pour éviter les problèmes de dépendances. Pour ce qui est de la version de **Python** et de **pip**, c'est la 3.8 qui a été utilisée. Les versions des différentes librairies nécessaires sont indiquées dans le fichier *requirements.txt*.

7 Testing

Pour les tests unitaires avec **unittest**, il faut faire une classe de test pour chaque script à tester. Cette classe doit hériter de la classe **unittest.TestCase**.

Ensuite dans cette classe, il faut définir une méthode pour chaque test unitaire. Pour lancer les tests, il faut avoir ce code dans le script :

```
if __name__ == "__main__":  
    unittest.main()
```

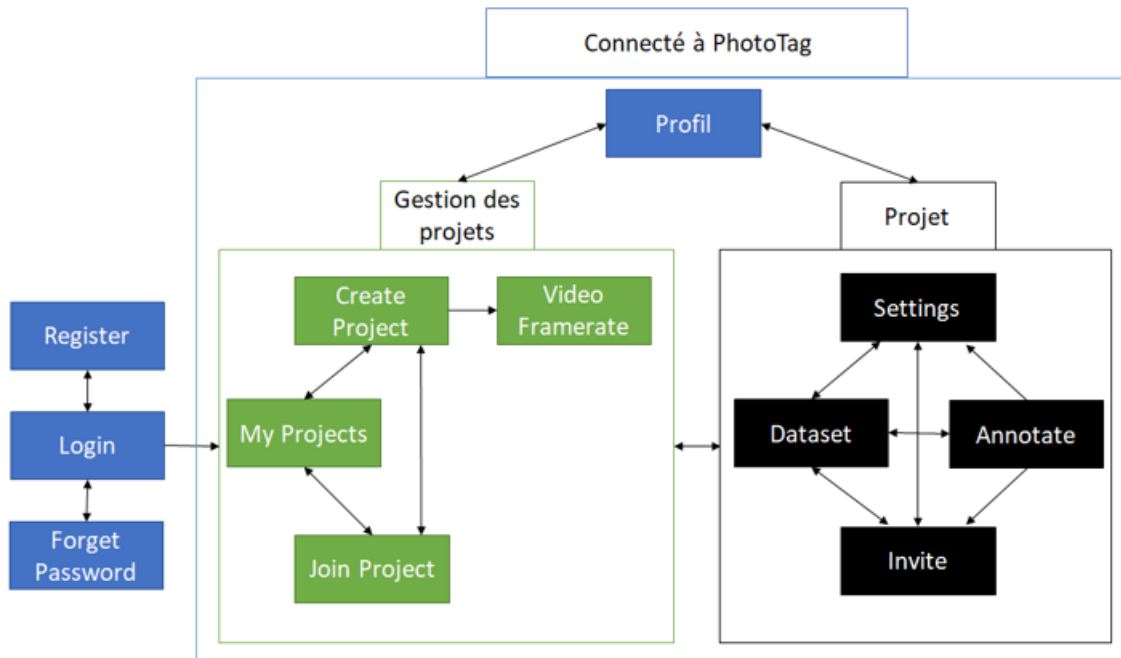
Et ensuite, lancer le script normalement avec **Python**.

Les tests unitaires réalisés portent sur des méthodes relatives aux bases de données. Il y a peu de tests unitaires car nous avons plutôt opté pour des tests fonctionnels (vérifier que l'utilisateur n'a pas accès à des projets dont il n'est pas membre, vérifier que les éléments visuels s'affichent bien, vérifier que l'application a un comportement normal, pas trop lent, intuitif, ...).

8 Architecture

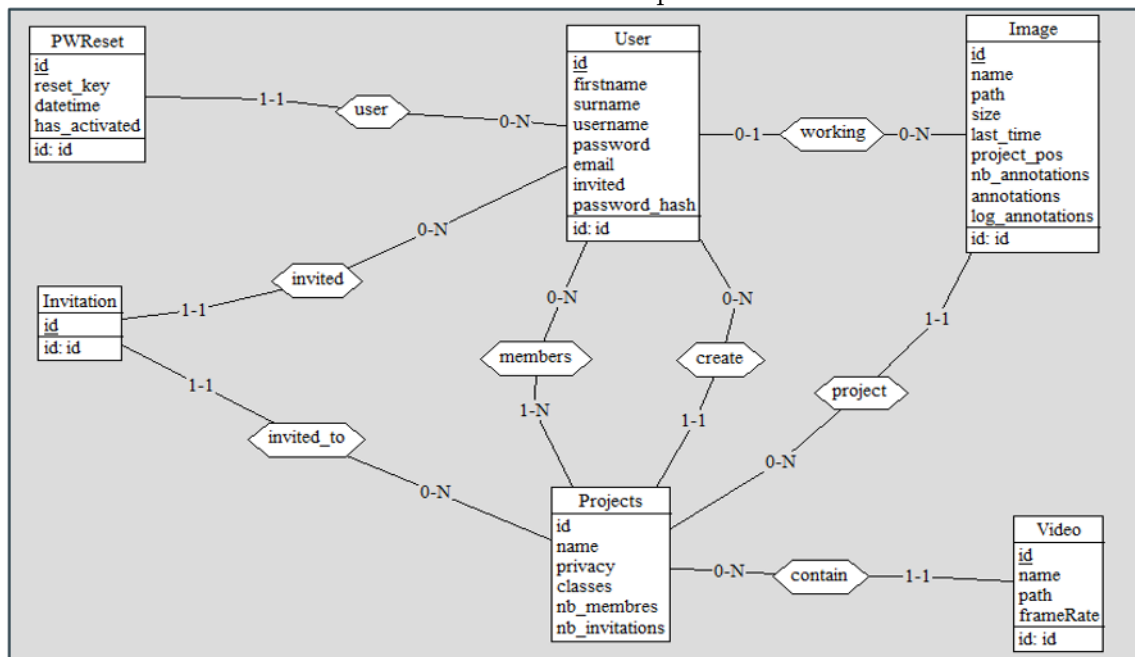
Vous trouverez ci-dessous l'architecture du site web **PhotoTag** :

FIGURE 2 – Architecture du site



Vous trouverez ci-dessous le schéma conceptuel de la base de données relationnelle de **PhotoTag** :

FIGURE 3 – Schéma conceptuel de la BD



9 Déploiement

Dans cette section, vous trouverez des informations pour déployer le serveur facilement.

9.1 Localhost

1. Installer les librairies nécessaires : `pip install -r requirements.txt`
2. Lancer le fichier main : `python main.py`
3. Le site est disponible à l'adresse indiquée dans le terminal.

```
* Serving Flask app 'app' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

9.2 Localhost et redirection en ligne

Pour tester notre application en ligne et en temps-réel, nous avons utilisé l'outil [ngrok](#) qui permet de créer un pont entre une app locale et internet. Cet outil est disponible sur tous les OS. Pour mettre l'app en ligne, voici les étapes à suivre :

1. Créer un compte sur [ngrok.com](#) et installer ngrok pour l'OS désiré.
2. Sur Windows, le téléchargement est une archive zip qui contient le fichier **ngrok.exe**. Il suffit d'extraire le fichier et l'exécuter. Cela va ouvrir une invite de commande propre à ngrok où il faudra écrire les commandes suivantes. (Sur Linux, tout se fait directement dans le terminal).
3. La première fois qu'est utilisé **ngrok** sur un appareil, il faut lui donner un token d'authentification disponible sur le compte de l'utilisateur. Cela se fait via la commande `ngrok config add-authtoken <token>`
4. Pour créer le pont entre l'app locale et une adresse en ligne, il faut avoir déployer le site en localhost et utiliser la commande `ngrok http <port>`. Dans notre cas, le port est 5000.

Bien entendu, cette solution n'est pas idéale en production 24/7. Pour cela, il vaut mieux utiliser un serveur Apache, ou un hébergeur en ligne.

9.3 Hébergeurs en ligne

Nous avons également hébergé notre application sur [pythonanywhere](#) pour vérifier que tout fonctionnait correctement sur hébergeur. L'application est disponible à l'adresse suivante : <https://phototag.pythonanywhere.com/>. Voici les étapes que nous avons suivies pour faire fonctionner l'app sur cet h"bergeur :

1. Créer un compte sur le site.
2. Ajouter une nouvelle app web qui utilise Flask et Python 3.8.
3. Créer une nouvelle base de données MySQL (dans l'onglet *Databases*).

4. Pour déposer les fichiers de notre app dans le dossier mysite, il est possible de les cloner depuis le repo Github.
5. Dans le fichier **config.py**, il faut changer l'adresse URI de la base de données en indiquant les informations ci-dessous pour connecter l'app à la base de données MySQL créée.

```
class Config(object):
    user, password = '██████████', '██████████'
    host = '██████████.mysql.pythonanywhere-services.com'
    db = '██████████$default'

    SECRET_KEY = binascii.hexlify(os.urandom(24))
    SQLALCHEMY_DATABASE_URI = 'mysql://{0}:{1}@{2}/{3}'.format(user, password, host, db)
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    os.environ['OAUTHLIB_INSECURE_TRANSPORT'] = '1'
```

6. Pour que Google Login fonctionne, il faut rajouter une redirection vers la nouvelle adresse de l'app dans l'API Google. Ceci est expliqué dans la sous-section suivante([Google API](#)).

Sur le lien, vous trouverez de plus amples informations sur comment déployer une app Flask <https://flask.palletsprojects.com/en/2.1.x/deploying/>.

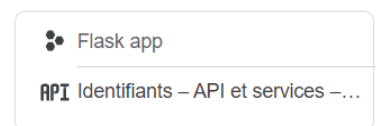
9.4 Google API et Gmail

Pour utiliser l'API de Google et pour envoyer des mails de récupération de mot de passe, nous avons créé un nouveau compte Google dont l'adresse est **pphototag@gmail.com**.

9.4.1 Google API


Pour s'enregistrer via Google Login, nous avons enregistré notre application sur "Google cloud Platform" avec compte Google que nous avons pour cette dernière. Lorsqu'on héberge l'application sur un autre site ou sur un serveur, il faut également changer le lien d'hébergement sur Google Cloud, afin que le callback puisse se faire entre Google et notre application pour pouvoir récupérer les informations dont nous avons besoin. Voici comment procéder :

1. Rendez-vous sur le site de [Google Cloud](#)
2. Connectez vous sur le compte Google crée sur l'application avec les identifiants envoyés par mail (Figure 4(a)).

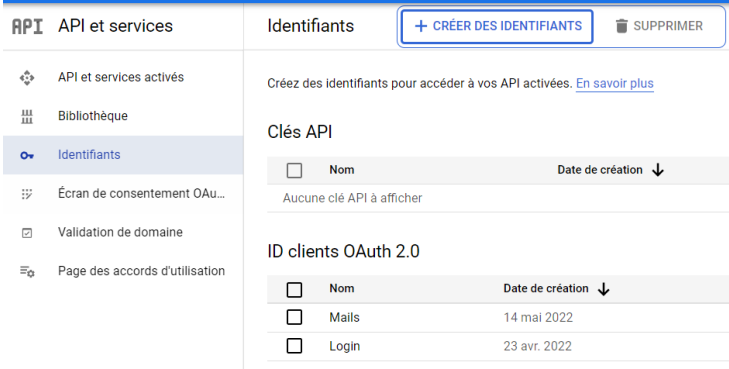


3. Allez maintenant dans la partie **API et Services : identifiants**.
4. Ensuite, allez dans la partie "**ID clients OAuth 2.0**" et sélectionnez Login.(Figure 4(b))
5. Dans la partie "**Origines JavaScript autorisées**", mettez le lien de votre site web. Pour cela, cliquez sur le bouton **+ AJOUTER UN URI** et remplissez ensuite le champ avec la nouvelle adresse du site web. Dans notre cas, le site est hébergé sur le site : (Figure 4(c))


6. Enfin dans la partie "**URI de redirection autorisés**", cliquez également sur "ajouter un uri", et mettez le lien de votre site suivi de "/login/google/login/callback". Cela permettra à Google d'avoir un callback lorsque l'on nous demande de sélectionner le compte avec lequel nous voulons nous connecter.
7. Enfin, appuyez sur le bouton **ENREGISTRER** pour confirmer vos changements. (voir Figure 4(d)).



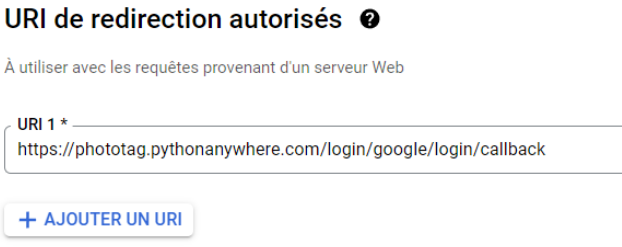
(a) connexion google cloud



(b) identifiants



(c) uri java script



(d) google callback

FIGURE 4 – Google Login

9.4.2 Gmail

Pour envoyer les mails de récupération de mot de passe, nous avons utilisés la librairie *yagmail*. Pour pouvoir envoyer le mail, il est nécessaire de paramétrer SMTP via à la méthode suivante : `yagmail.SMTP(user=<email>, password=<password>)`

Cela nécessite de laisser le mot de passe dans le code. Nous avons choisi de le mettre dans un fichier à part pour que le client puisses le cacher plus facilement par la suite. Nous conseillons d'ailleurs de changer le mot de passe lors du déploiement. Nous avons hésité à cacher le mot de passe et l'envoyer directement au client pour qu'il le rajoute lui-même dans le code par la suite, mais comme cette app a été développée dans le cadre d'un cours, nous avons préféré le laisser dans le code pour que d'autres personnes puissent lancer l'app.

Il y a également la possibilité d'utiliser le moyen d'authentification OAuth2 à la place du mot de passe, cela nécessite un "handshake" entre l'application et l'api Google. Vous trouverez plus d'informations via la documentation de [yagmail](#).