Guide de Mentorat : Projet Python Flask

Outil de Gestion des Objets Perdus et Retrouvés

Technologies : Python 3.12, Flask, SQLite, HTML/CSS, Git/GitHub

Préparé par **R Yanis Axel DABO**
Developer Logiciel, Etudiant en L2 de Genie Logiciel

June 5, 2025

Contents

1 Introduction

Ce guide est conçu pour vous accompagner dans la création d'un **Outil de Gestion des Objets Perdus et Retrouvés** avec Python et Flask. L'objectif est d'apprendre à développer une application web, gérer une base de données, et collaborer via Git et GitHub, tout en adoptant des bonnes pratiques. Chaque section est détaillée avec des explications simples, des exemples de code complets avec mise en surbrillance syntaxique, et des conseils pour éviter les erreurs courantes.

1.1 Objectifs pédagogiques

- Apprendre à créer une application web avec Flask et l'architecture MVC.
- Comprendre comment stocker et gérer des données avec SQLite.
- Maîtriser Git et GitHub pour le travail en équipe.
- Développer des compétences en programmation Python (nommage, documentation).
- Apprendre à déboguer et présenter un projet.

1.2 Présentation du projet

L'application permet aux utilisateurs de :

- S'inscrire et se connecter de manière sécurisée.
- Déclarer des objets perdus ou retrouvés via des formulaires simples.
- Consulter tous les objets sur une page d'accueil.
- (Bonus) Identifier des correspondances entre objets perdus et retrouvés.

2 Technologies Utilisées

- **Python 3.12** : Langage clair et adapté aux débutants.
- **Flask** : Framework web léger pour créer des applications rapidement.
- **SQLite** : Base de données simple, stockée dans un fichier.
- **HTML/CSS** : Interface utilisateur basique et stylée.
- **Git/GitHub** : Outils pour versionner et collaborer.
- **Google Meet** : Réunions d'équipe pour la coordination.

3 Structure du Projet

3.1 Arborescence

Voici comment organiser les fichiers du projet :

```
lost_and_found/
        app.py                % Point d'entrée de l'application Flask
        models.py             % Modèles de base de données
        templates/            % Fichiers HTML pour l'interface
            base.html          % Template de base pour la mise en page
            login.html         % Page de connexion
            register.html      % Page d'inscription
            home.html          % Page d'accueil
            lost.html          % Formulaire pour objets perdus
            found.html         % Formulaire pour objets retrouvés
            match.html         % Page de correspondance (bonus)
        static/               % Fichiers statiques (CSS)
            style.css          % Styles de l'application
        instance/             % Base de données SQLite
            database.db        % Fichier de la base de données
        .gitignore            % Fichiers à ignorer par Git
        requirements.txt      % Dépendances Python
        README.md             % Documentation du projet
```

3.2 Explication des fichiers

- `app.py` : Configure Flask, définit les routes (URLs), et gère la logique.
- `models.py` : Définit les structures de données pour SQLite (utilisateurs, objets).
- `templates/` : Contient les fichiers HTML rendus dynamiquement.
- `static/style.css` : Définit l'apparence visuelle (couleurs, mise en page).
- `instance/database.db` : Fichier de la base de données SQLite.
- `.gitignore` : Exclut les fichiers temporaires ou sensibles.
- `requirements.txt` : Liste les bibliothèques Python nécessaires.
- `README.md` : Explique comment installer et exécuter le projet.

3.3 Conseils pour l'organisation

- Créez les dossiers `templates` et `static` dès le début.
- Nommez les fichiers clairement (ex. : `lost.html` au lieu de `page1.html`).
- Tenez le `README.md` à jour pour aider vos coéquipiers.

4 Concepts Fondamentaux

4.1 Architecture MVC

L'architecture MVC (Modèle-Vue-Contrôleur) organise le code :

- **Modèle** : Gère les données (ex. : utilisateurs, objets dans `models.py`).
- **Vue** : Affiche les données à l'utilisateur (ex. : pages HTML dans `templates/`).
- **Contrôleur** : Traite les actions de l'utilisateur (ex. : routes dans `app.py`).

Exemple : Quand vous déclarez un objet perdu, le contrôleur enregistre les données dans le modèle et met à jour la vue.

4.2 Gestion des sessions

Les sessions permettent de savoir qui est connecté. Flask stocke des informations (comme l'ID utilisateur) dans un cookie sécurisé. Exemple :

```
session['user_id'] = user.id  # Enregistre l'utilisateur connecté
```

4.3 Base de données SQLite

SQLite est une base de données simple qui stocke tout dans un fichier (`database.db`). SQLAlchemy traduit les objets Python en tables SQL, rendant la gestion des données plus intuitive.

4.4 Templates Jinja2

Jinja2 insère des données dynamiques dans le HTML. Exemple :

```
<p>Bienvenue, {{ session['username'] }} !</p>
```

Ici, `{{ session['username'] }}` affiche le nom de l'utilisateur connecté.

5 Étapes de Développement

Voici les étapes pour construire l'application, expliquées pour les débutants :

5.1 Étape 1 : Configurer l'environnement

1. Téléchargez et installez Python 3.12 depuis python.org.
2. Créez un environnement virtuel pour isoler les dépendances :

```
python3 -m venv venv
source venv/bin/activate  % Linux/Mac
venv\Scripts\activate     % Windows
```

3. Installez les bibliothèques nécessaires :

```
pip install flask flask-sqlalchemy werkzeug
pip freeze > requirements.txt
```

4. Vérifiez que `requirements.txt` contient `flask`, `flask-sqlalchemy`, et `werkzeug`.

5.2 Étape 2 : Initialiser Git

1. Configurez votre identité Git :

```
git config --global user.name "Votre Nom"
git config --global user.email "votre.email@example.com"
```

2. Créez un dépôt local :

```
git init
```

3. Créez un dépôt sur GitHub nommé `lost-and-found`.

4. Liez le dépôt local à GitHub :

```
git remote add origin
    https://github.com/votre_utilisateur/lost-and-found.git
```

5. Faites un premier commit et poussez :

```
git add .
git commit -m "Initialisation du projet Flask"
git push -u origin main
```

6 Code Complet et Explications

Chaque fichier est présenté avec son code complet, des commentaires détaillés, et des explications pour aider les étudiants à comprendre ce qu'ils écrivent.

6.1 Fichier : app.py

Ce fichier est le cœur de l'application, gérant la configuration Flask, les routes, et la logique.

```
# Importation des modules nécessaires
from flask import Flask, render_template, request, redirect, url_for, session
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash, check_password_hash
from difflib import SequenceMatcher
from datetime import datetime
import os

# Création de l'application Flask
app = Flask(__name__)
app.config['SECRET_KEY'] = 'votre_cle_secrete_ici'  # Clé secrète pour
    sécuriser les sessions
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///instance/database.db'  #
    Chemin de la base de données
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False  # Désactiver les
    avertissements inutiles
```

```
db = SQLAlchemy(app)  # Initialisation de SQLAlchemy pour gérer la base de données

# Importer les modèles après l'initialisation de db
from models import User, LostItem, FoundItem

# Créer les tables de la base de données
with app.app_context():
    db.create_all()

@app.route('/')
def home():
    """Affiche la page d'accueil avec tous les objets perdus et retrouvés."""
    lost_items = LostItem.query.all()  # Récupère tous les objets perdus
    found_items = FoundItem.query.all()  # Récupère tous les objets retrouvés
    return render_template('home.html', lost_items=lost_items, found_items=found_items)

@app.route('/register', methods=['GET', 'POST'])
def register():
    """Permet à un utilisateur de s'inscrire."""
    if request.method == 'POST':
        username = request.form['username']  # Récupère le nom d'utilisateur du formulaire
        password = request.form['password']  # Récupère le mot de passe
        if not username or not password:
            return render_template('register.html', error="Tous les champs sont requis")
        if User.query.filter_by(username=username).first():
            return render_template('register.html', error="Nom d'utilisateur déjà pris")
        hashed_password = generate_password_hash(password, method='pbkdf2:sha256')  # Hache le mot de passe
        user = User(username=username, password=hashed_password)  # Crée un nouvel utilisateur
        db.session.add(user)  # Ajoute à la base de données
        db.session.commit()  # Valide les changements
        return redirect(url_for('login'))  # Redirige vers la page de connexion
    return render_template('register.html')  # Affiche le formulaire d'inscription

@app.route('/login', methods=['GET', 'POST'])
def login():
    """Permet à un utilisateur de se connecter."""
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
```

```
53          user = User.query.filter_by(username=username).first()  # Cherche
    l'utilisateur
54          if user and check_password_hash(user.password, password):  # Vérifie le
    mot de passe
55              session['user_id'] = user.id  # Stocke l'ID utilisateur dans la
    session
56              session['username'] = user.username  # Stocke le nom d'utilisateur
57              return redirect(url_for('home'))  # Redirige vers l'accueil
58          return render_template('login.html', error="Identifiants incorrects")
59      return render_template('login.html')
60
61  @app.route('/logout')
62  def logout():
63      """Déconnecte l'utilisateur."""
64      session.pop('user_id', None)  # Supprime l'ID utilisateur
65      session.pop('username', None)  # Supprime le nom d'utilisateur
66      return redirect(url_for('home'))  # Redirige vers l'accueil
67
68  @app.route('/lost', methods=['GET', 'POST'])
69  def lost():
70      """Permet de déclarer un objet perdu."""
71      if 'user_id' not in session:
72          return redirect(url_for('login'))  # Vérifie si l'utilisateur est
    connecté
73      if request.method == 'POST':
74          description = request.form['description']
75          if not description:
76              return render_template('lost.html', error="La description est
    requise")
77          lost_item = LostItem(description=description,
    user_id=session['user_id'])
78          db.session.add(lost_item)
79          db.session.commit()
80          return redirect(url_for('home'))
81      return render_template('lost.html')
82
83  @app.route('/found', methods=['GET', 'POST'])
84  def found():
85      """Permet de déclarer un objet retrouvé."""
86      if 'user_id' not in session:
87          return redirect(url_for('login'))
88      if request.method == 'POST':
89          description = request.form['description']
90          if not description:
91              return render_template('found.html', error="La description est
    requise")
92          found_item = FoundItem(description=description,
```

```
    user_id=session['user_id'])
93          db.session.add(found_item)
94          db.session.commit()
95          return redirect(url_for('home'))
96      return render_template('found.html')
97
98  @app.route('/match')
99  def match():
100     """Identifie les correspondances entre objets perdus et retrouvés."""
101     if 'user_id' not in session:
102         return redirect(url_for('login'))
103     lost_items = LostItem.query.all()
104     found_items = FoundItem.query.all()
105     matches = []
106     for lost in lost_items:
107         for found in found_items:
108             similarity = SequenceMatcher(None, lost.description,
    found.description).ratio()
109             if similarity > 0.6:  # Seuil de similarité
110                 matches.append((lost, found, similarity))
111     return render_template('match.html', matches=matches)
112
113 if __name__ == '__main__':
114     app.run(debug=True)  # Lance le serveur en mode débogage
```

Explication pour les étudiants :

- Ce fichier configure Flask et définit toutes les routes (URLs) de l'application.
- Chaque fonction (ex. : `home`, `register`) correspond à une page ou une action.
- Les commentaires expliquent chaque étape (ex. : hachage des mots de passe, gestion des sessions).
- `debug=True` aide à voir les erreurs pendant le développement.

6.2 Fichier : models.py

Définit les structures de données pour la base de données.

```
1  from flask_sqlalchemy import SQLAlchemy
2  from datetime import datetime
3
4  # Initialisation de SQLAlchemy
5  db = SQLAlchemy()
6
7  class User(db.Model):
8      """Représente un utilisateur dans la base de données."""
```

```
    id = db.Column(db.Integer, primary_key=True)  # ID unique pour chaque utilisateur
    username = db.Column(db.String(80), unique=True, nullable=False)  # Nom d'utilisateur unique
    password = db.Column(db.String(120), nullable=False)  # Mot de passe haché

class LostItem(db.Model):
    """Représente un objet perdu."""
    id = db.Column(db.Integer, primary_key=True)  # ID unique
    description = db.Column(db.String(200), nullable=False)  # Description de l'objet
    date = db.Column(db.DateTime, default=datetime.utcnow)  # Date de déclaration
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    # Lien vers l'utilisateur

class FoundItem(db.Model):
    """Représente un objet retrouvé."""
    id = db.Column(db.Integer, primary_key=True)
    description = db.Column(db.String(200), nullable=False)
    date = db.Column(db.DateTime, default=datetime.utcnow)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
```

Explication :

- `User` stocke les informations d'authentification.
- `LostItem` et `FoundItem` stockent les descriptions et dates des objets.
- `user_id` lie chaque objet à l'utilisateur qui l'a déclaré.

6.3 Fichier : templates/base.html

Template de base pour une mise en page cohérente.

```
<!DOCTYPE html>
<html>
<head>
  <title>Outil de Gestion des Objets Perdus</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
  <header>
    <h1>Outil de Gestion des Objets Perdus et Retrouvés</h1>
    <nav>
      <a href="{{ url_for('home') }}">Accueil</a>
      {% if 'user_id' in session %}
        <a href="{{ url_for('lost') }}">Déclarer Perdu</a>
        <a href="{{ url_for('found') }}">Déclarer Retrouvé</a>
```

```
        <a href="{{ url_for('match') }}">Correspondances</a>
        <a href="{{ url_for('logout') }}">Déconnexion ({{
    session.get('username', '') }})</a>
      {% else %}
        <a href="{{ url_for('login') }}">Connexion</a>
        <a href="{{ url_for('register') }}">Inscription</a>
      {% endif %}
    </nav>
  </header>
  <main>
    {% if error %}
      <p style="color: red;">{{ error }}</p>
    {% endif %}
    {% block content %}{% endblock %}
  </main>
</body>
</html>
```

Explication :

- Fournit une structure HTML commune avec une barre de navigation.
- `{% if 'user_id' in session %}` affiche des liens différents selon si l'utilisateur est connecté.
- `{% block content %}` permet aux autres pages d'insérer leur contenu.

6.4 Fichier : templates/home.html

Page d'accueil affichant les objets.

```
{% extends 'base.html' %}
{% block content %}
  <h2>Accueil</h2>
  {% if 'username' in session %}
    <p>Bienvenue, {{ session['username'] }} !</p>
  {% else %}
    <p>Veuillez vous connecter pour déclarer des objets.</p>
  {% endif %}
  <h3>Objets Perdus</h3>
  <ul>
    {% for item in lost_items %}
      <li>{{ item.description }} (Signalé le {{ item.date.strftime('%Y-%m-%d')
    }})</li>
    {% endfor %}
  </ul>
  <h3>Objets Retrouvés</h3>
  <ul>
    {% for item in found_items %}
```

```
      <li>{{ item.description }} (Signalé le {{ item.date.strftime('%Y-%m-%d')
    }})</li>
    {% endfor %}
  </ul>
{% endblock %}
```

Explication :

- Hérite de `base.html` pour la mise en page.
- Affiche un message personnalisé et la liste des objets perdus/retrouvés.

6.5 Fichier : templates/register.html

Formulaire d'inscription.

```
{% extends 'base.html' %}
{% block content %}
  <h2>Inscription</h2>
  <form method="POST">
    <label>Nom d'utilisateur : <input type="text" name="username"
    required></label><br>
    <label>Mot de passe : <input type="password" name="password"
    required></label><br>
    <input type="submit" value="S'inscrire">
  </form>
{% endblock %}
```

Explication : Formulaire simple avec validation HTML (`required`) pour l'inscription.

6.6 Fichier : templates/login.html

Formulaire de connexion.

```
{% extends 'base.html' %}
{% block content %}
  <h2>Connexion</h2>
  <form method="POST">
    <label>Nom d'utilisateur : <input type="text" name="username"
    required></label><br>
    <label>Mot de passe : <input type="password" name="password"
    required></label><br>
    <input type="submit" value="Se connecter">
  </form>
{% endblock %}
```

Explication : Formulaire de connexion avec validation des champs.

6.7 Fichier : templates/lost.html

Formulaire pour déclarer un objet perdu.

```
{% extends 'base.html' %}
{% block content %}
  <h2>Déclarer un Objet Perdu</h2>
  <form method="POST">
    <label>Description : <textarea name="description"
    required></textarea></label><br>
    <input type="submit" value="Déclarer">
  </form>
{% endblock %}
```

Explication : Formulaire pour soumettre une description d'objet perdu.

6.8 Fichier : templates/found.html

Formulaire pour déclarer un objet retrouvé.

```
{% extends 'base.html' %}
{% block content %}
  <h2>Déclarer un Objet Retrouvé</h2>
  <form method="POST">
    <label>Description : <textarea name="description"
    required></textarea></label><br>
    <input type="submit" value="Déclarer">
  </form>
{% endblock %}
```

Explication : Formulaire similaire pour les objets retrouvés.

6.9 Fichier : templates/match.html

Page pour les correspondances (bonus).

```
{% extends 'base.html' %}
{% block content %}
  <h2>Correspondances Potentielles</h2>
  <ul>
    {% for lost, found, similarity in matches %}
      <li>
        Objet perdu : {{ lost.description }}<br>
        Objet retrouvé : {{ found.description }}<br>
        Similarité : {{ (similarity * 100)|round(2) }}%
      </li>
    {% endfor %}
  </ul>
{% endblock %}
```

Explication : Affiche les correspondances entre objets perdus et retrouvés, avec un score de similarité.

6.10 Fichier : static/style.css

Feuille de style pour l'interface.

```
body {
  font-family: Arial, sans-serif;  /* Police claire et lisible */
  margin: 0;
  padding: 20px;
  background-color: #f9f9f9;  /* Fond clair */
}
header {
  background-color: #4CAF50;  /* Vert pour len-tête */
  color: white;
  padding: 10px;
  text-align: center;
}
nav a {
  color: white;
  margin: 0 15px;
  text-decoration: none;
  font-weight: bold;
}
nav a:hover {
  text-decoration: underline;  /* Effet au survol */
}
main {
  max-width: 800px;  /* Largeur maximale pour la lisibilité */
  margin: 20px auto;
  padding: 10px;
  background-color: white;
  border-radius: 5px;
  box-shadow: 0 0 10px rgba(0,0,0,0.1);  /* Ombre légère */
}
form {
  display: flex;
  flex-direction: column;
  gap: 10px;  /* Espacement entre les éléments */
}
input, textarea {
  padding: 10px;
  border: 1px solid #1e1e;
  border-radius: 4px;
}
input[type="submit"] {
```

```
  background-color: #4CAF50;
  color: white;
  border: none;
  padding: 10px;
  cursor: pointer;
}
input[type="submit"]:hover {
  background-color: #45a049;  /* Vert plus foncé au survol */
}
ul li {
  padding: 10;
  border-bottom: 1px solid #eee;
}
```

Explication :

- Définit un style moderne et responsive.
- Utilise des commentaires pour expliquer chaque règle CSS.
- Assure une interface claire pour les débutants.

6.11 Fichier : .gitignore

Exclut les fichiers inutiles du versionnement.

```
__pycache__/
# Fichiers temporaires Python
*.pyc
venv/
# Base de données
instance/*
*.db
*.sqlite3
```

Explication : Évite de versionner les fichiers temporaires ou sensibles.

6.12 Fichier : requirements.txt

Liste les dépendances.

```
flask
flask-sqlalchemy
werkzeug
```

Explication : Permet d'installer les bibliothèques avec `pip install -r requirements.txt`.

6.13 Fichier : README.md

Documente le projet.

```
# Outil de Gestion des Objets Perdus et Retrouvés

## Description
Une application web Flask pour déclarer et gérer des objets perdus et retrouvés.

## Installation
1. Clonez le dépôt :
   ```
   git clone https://github.com/votre_utilisateur/lost-and-found
   cd lost-and-found
   ```
2. Créez un environnement virtuel :
   ```
   python3 -m venv venv
   source venv/bin/activate # Linux/Mac
   venv\Scripts\Scripts\activate # Windows
   ```
3. Installez les dépendances :
   ```
   pip install -r requirements.txt
   ```
4. Lancez l'application :
   ```
   python app.py
   ```

## Fonctionnalités
- Inscription et connexion sécurisées.
- Déclaration d'objets perdus ou retrouvés.
- Affichage des objets sur la page d'accueil.
- Correspondance des objets (bonus).

## Technologies
- Python 3.12, Flask, SQLite, HTML/CSS, Git/GitHub
```

7 Git et GitHub

7.1 Initialisation

- **Configurer :**

```
git config --global user.name "Votre Nom"
git config --global user.email "votre.email@example.com"
```

- **Créer un dépôt local :**

```
git init
```

- Créez un dépôt sur GitHub nommé `lost_and_found`.
- **Liez le dépôt :**

```
\item git remote add origin https://github.com/votre_utilisateur/lost-and-fou
```

- **Faites un premier commit et push :**

```
git add .
git commit -m "Initialisation du projet Flask"
git push -u origin main
```

7.2 Collaboration

- **Cloner** : `git clone https://github.com/votre`$_u$`tilisateur/lost-and-found.git`Mettre à jour
  `git pull origin main`
- **Pousser** :

```
\item git add .
git commit -m "Ajout de la page d'accueil"
git push origin main
```

7.3 Branches

- **Créer une branche :**

```
1    \item git checkout -b feature/authentification
```

- **Pousser la branche :**

```
    \item git add .
    git commit -m "Ajout de l'inscription"
    \texttt{git push origin feature/authentification}
\end{itemize}
    \item Créez un \textit{pull request} sur GitHub et assignez un coéquipier pou
```

7.4 Résolution des conflits

- **Si un conflit survient :**
- 1. Ouvrez le fichier en conflit (marqué par =====).

```
\item git add .
git commit -m "Résolution du conflit"
git push
```

7.5 Bons messages de commit

- Soyez précis : `"Ajout du formulaire de déclaration d'objet perdu"`.
- Commettez après chaque petite fonctionnalité.
- Utilisez `git status` pour vérifier ce que vous versionnez.

8 Bonnes Pratiques Python et Flask

- **Nommage** : Utilisez des noms clairs (`lost_item` au lieu de `x`).
- **Documentation** : Ajoutez des docstrings à chaque fonction/classe.
- **Sécurité** : Hachez les mots de passe, validez les entrées.
- **Tests** : Testez chaque route après l'avoir codée (ex. : `http://localhost:5000/`).
- **Débogage** : Utilisez `print()` ou `app.run(debug=True)` pour repérer les erreurs.

9 Bonus : Améliorations et Présentation

9.1 Améliorations possibles

- Ajouter un champ pour des images (stockées comme BLOB dans SQLite).
- Implémenter une recherche par mot-clé avec :

```
LostItem.query.filter(LostItem.description.contains('mot'))
```

- Ajouter des notifications par email avec `smtplib` (vérifiez les limites du serveur).
- Créer un tableau de bord utilisateur pour gérer ses déclarations.

9.2 Présentation

- Faire une démonstration en direct via Google Meet, montrant chaque page.
- Expliquer qui a codé quoi (ex. : "Marie a fait l'authentification").
- Montrer le dépôt sur GitHub : commits, branches, pull requests.
- Parler des défis (ex. : conflits Git) et comment ils ont été résolus.
- Préparer un diaporama simple pour structurer la présentation.

10 Dépannage pour Débutants

Cette section est étendue avec des exemples concrets pour aider les étudiants à résoudre les erreurs courantes et augmenter le nombre de pages.

10.1 Erreurs Flask et Python

- **Erreur : "No such table"** : Assurez-vous que `db.create_all()` est exécuté dans `app.py`.
  - **Vérification** : Ajoutez ce code dans `app.py` avant les routes :

```
with app.app_context():
    db.create_all()
```

  - Si l'erreur persiste, supprimez `database.db` et relancez l'application.
- **Erreur : 404 Not Found** : Vérifiez les URLs dans `app.py` et `url_for`.
  - Exemple : Si `http://localhost:5000/login` renvoie 404, assurez-vous que la route existe :

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    return render_template('login.html')
```

  - Vérifiez aussi dans `base.html` que $\texttt{url}_f or('login')estutiliscorrectement$.
- **Erreur : "KeyError: 'user$_id$'"** : $Celasignifiequelasessionn'apasd'IDutilisateur.$
  - **Solution** : Assurez-vous que l'utilisateur est connecté avant d'accéder à `session['user`$_id$`']` :

10.2 Erreurs Git

- Problèmes de poussée : "non-fast-forward" : Cela indique un conflit avec le dépôt distant.
  - Solution : Mettez à jour votre dépôt local :

```
git pull origin main --rebase
git push origin main
```

  - Si un conflit apparaît, ouvrez les fichiers affectés, résolvez les différences, puis :

```
git add .
git rebase --continue
git push
```

- Erreur : "Permission denied (publickey)" : Problème avec la configuration SSH.

```
\item \textbf{Vérification} : Assurez-vous que votre clé SSH est ajoutée à G
\texttt{ssh -T git@github.com}
\item \textbf{Si nécessaire, générez une nouvelle clé :}
\texttt{ssh-keygen -t ed25519 -C "votre.email@example.com"}
\texttt{pcat ~/.ssh/id_ed25519.pub}
```

`Copiez la clé publique et ajoutez-la dans GitHub > Settings > SSH and GPG keys.`

10.3 Erreurs de base de données

- **Erreur : "OperationalError : database is locked"** : SQLite ne gère pas bien les accès concurrents.
  - **Solution** : Redémarrez l'application et évitez d'ouvrir `database.db` dans un autre programme.
- **Erreur : "IntegrityError"** : Violation de contrainte (ex. : nom d'utilisateur déjà pris).
  - **Vérification** : Dans `register`, vérifiez l'existence de l'utilisateur :

```
    if User.query.filter_by(username=username).first():
        return render_template('register.html', error="Nom dutilisateur déjà pris")
```

10.4 Autres problèmes courants

- **L'application ne démarre pas** : Vérifiez les dépendances dans `requirements.txt` et l'environnement virtuel :

```
pip install -r requirements.txt
```

- **Le style CSS ne s'applique pas** : Assurez-vous que `static/style.css` est lié :

```
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
```

- **Les templates ne s'affichent pas** : Vérifiez que les fichiers HTML sont dans le dossier `templates`.

11 Exemples de Code Avancés

Pour enrichir le projet et augmenter le nombre de pages, voici des exemples de fonctionnalités avancées que les étudiants peuvent implémenter.

11.1 Recherche par mot-clé

Ajoutez une route pour rechercher des objets par mot-clé dans la description.

```
@app.route('/search', methods=['GET', 'POST'])
def search():
    """Permet de rechercher des objets perdus ou retrouvés par mot-clé."""
    if request.method == 'POST':
        keyword = request.form['keyword']
        if not keyword:
```

```
            return render_template('search.html', error="Veuillez entrer un mot-clé")
        lost_items = LostItem.query.filter(LostItem.description.contains(keyword)).all()
        found_items = FoundItem.query.filter(FoundItem.description.contains(keyword)).all()
        return render_template('search.html', lost_items=lost_items, found_items=found_items, keyword=keyword)
    return render_template('search.html')
```

Fichier : templates/search.html

```
{% extends 'base.html' %}
{% block content %}
  <h2>Recherche d'Objets</h2>
  <form method="POST">
    <label>Mot-clé : <input type="text" name="keyword" required></label>
    <input type="submit" value="Rechercher">
  </form>
  {% if keyword %}
    <h3>Résultats pour "{{ keyword }}"</h3>
    <h4>Objets Perdus</h4>
    <ul>
      {% for item in lost_items %}
        <li>{{ item.description }} ({{ item.date.strftime('%Y-%m-%d') }})</li>
      {% endfor %}
    </ul>
    <h4>Objets Retrouvés</h4>
    <ul>
      {% for item in found_items %}
        <li>{{ item.description }} ({{ item.date.strftime('%Y-%m-%d') }})</li>
      {% endfor %}
    </ul>
  {% endif %}
{% endblock %}
```

Explication :

- Cette fonctionnalité permet de filtrer les objets par mot-clé.
- La méthode `contains` de SQLAlchemy effectue une recherche insensible à la casse.
- Ajoutez un lien dans `base.html` pour accéder à cette page :

```
    <a href="{{ url_for('search') }}">Rechercher</a>
```

11.2 Tableau de bord utilisateur

Créez une page où les utilisateurs peuvent voir leurs propres déclarations.

```
@app.route('/dashboard')
def dashboard():
    """Affiche les déclarations de lutilisateur connecté."""
    if 'user_id' not in session:
        return redirect(url_for('login'))
    user_id = session['user_id']
    lost_items = LostItem.query.filter_by(user_id=user_id).all()
    found_items = FoundItem.query.filter_by(user_id=user_id).all()
    return render_template('dashboard.html', lost_items=lost_items,
   found_items=found_items)
```

Fichier : templates/dashboard.html

```
\begin{verbatim}
{% extends 'base.html' %}
{% block content %}
  <h2>Tableau de Bord</h2>
  <h3>Vos Objets Perdus</h3>
  <ul>
    {% for item in lost_items %}
    <li>{{ item.description }} ({{ item.date.strftime('%Y-%m-%d') }})</li>
    {% endfor %}
  </ul>
  </h3>Vos Objets Retrouvés</h3>
  <ul>
    {% for item in found_items %}
    <li>{{ item.description }} ({{ item.date.strftime('%Y-%m-%d') }})</li>
    {% endfor %}
  </ul>
{% endblock %}
\end{verbatim}
```

Explication :

- Cette page affiche uniquement les objets déclarés par l'utilisateur connecté.
- Ajoutez un lien dans `base.html` :

```
    <a href="{% url_for('dashboard') %}">Tableau de bord</a>
```