Objectifs de la feuille

- Le modèle User
- Authentification
- Déconnexion
- Protection des vues
- Redirection automatique

Introduction

Nous allons maintenant ajouter l'authentification des utilisateurs de notre app. Attention : le serveur de développement sert les pages avec HTTP, c'est à dire que la connexion est non sécurisée et que les mots de passe vont donc circuler en clair ! Il ne faut jamais faire cela en déploiement ! Il faut impérativement utiliser HTTPS !

```
$ pip install flask-login
```

Le modèle User de Flask

Nous allons devoir ajouter un modèle User dans la base de données avec un identifiant et un mot de passe crypté avec SHA256 dans le fichier models.py:

```
class User(db.Model):
    Login = db.Column(db.String(50), primary_key=True)
    Password = db.Column(db.String(64))
```

Quand on lit la doc pour flask-login, on apprend que la classe User doit offrir une certaine API qu'on peut obtenir, partiellement implémentée, par la classe UserMixin. On fait donc les modifications suivantes :

```
from flask_login import UserMixin

class User(db.Model, UserMixin):
    Login = db.Column (db.String(50), primary_key=True)
    Password = db.Column (db.String(64))

    def get_id(self):
        return self.Login
```

Nous venons d'ajouter un modèle, mais la table correspondante n'existe pas encore dans la base de données. C'est la fonction db.create_all( ) qui permet de créer les tables manquantes. Nous allons donc ajouter une commande syncdb pour invoquer cette fonction dans le fichier commands.py:

```
@app.cli. command ()
def syncdb():
    '''Creates all missing tables . '''
    db.create_all()
    lg.warning('Database synchronized!')
```

Maintenant, on peut l'utiliser : $ flask syncdb
On peut alors vérifier que la table User a bien été créée dans le fichier monApp.db

Pour ajouter des utilisateurs dans notre base de données, nous allons réaliser une commande newuser :

```
@app.cli.command()
@click.argument('login')
@click.argument('pwd')
def newuser (login, pwd):
    '''Adds a new user'''
    from . models import User
    from hashlib import sha256
    m = sha256()
    m.update(pwd.encode())
    unUser = User(Login=login ,Password =m.hexdigest())
    db.session.add(unUser)
    db.session.commit()
    lg.warning('User ' + login + ' created!')
```

Ajoutons maintenant un utilisateur Cricri avec mot de passe azerty123 : $ flask newuser Cricri azerty123

C'est à vous de jouer !

Ajoutez une commande newpasswrd pour changer le mot de passe d'un utilisateur :

$ flask newpasswrd Cricri legarsdu41

Formulaire d'authentification

Pour activer le plugin, nous devons ajouter ces quelques lignes dans app.py :

```
from flask_login import LoginManager
login_manager = LoginManager(app)
```

Puis nous devons fournir un callback pour charger un utilisateur étant donné son identifiant. Ce callback sera utilisé par flask pour l'instance de User d'un utilisateur authentifié, étant donné son cookie de session. Dans le fichier models.py :

```
from .app import login_manager
@login_manager.user_loader
def load_user(username):
    return User.query.get(username)
```

Dans forms.py, vous devez créer un formulaire pour permettre à un utilisateur de s'authentifier. On lui adjoindra une méthode get_authenticated_user( ) qui vérifie que l'utilisateur existe, et que son mot de passe est correct et renvoie l'instance de User représentant cet utilisateur. Si l'utilisateur n'existe pas ou que son mot de passe est faux, il renvoie None.

```
from wtforms import PasswordField
from . models import User
from hashlib import sha256

class LoginForm(FlaskForm):
        Login = StringField ('Identifiant')
        Password = PasswordField ('Mot de passe')

        def get_authenticated_user (self):
                unUser = User.query.get(self.Login.data)
                if unUser is None:
                        return None
                m = sha256 ()
                m.update(self.Password.data.encode())
                passwd = m.hexdigest()
                return unUser if passwd == unUser.Password else None
```

Maintenant que nous avons le formulaire, il faut créer une vue pour l'utiliser dans views.py :

```
@app.route ("/login/", methods =("GET","POST" ,))
def login():
        unForm = LoginForm ()
        unUser=None
        if unForm.validate_on_submit():
                unUser = unForm.get_authenticated_user()
                if unUser:
                        login_user(unUser)
                        return redirect (url_for("index",name=unUser.Login))

        return render_template ("login.html",form=unForm)
```

Et n'oublions pas le template login.html :

```
{% extends "base.html" %}

{% block main %}
<form class="form-horizontal" role="form" method ="POST" action ="{{ url_for ('login') }}">
        {{ form.hidden_tag() }}
        <div class ="form-group">
                <label for="username" class ="col-sm-2 control-label">{{ form.Login.label }}</label >
                <div class ="col-sm-10">
                        {{ form.Login (size=50, class_="form-control") }}
                </div >
        </div >

        <div class ="form-group">
                <label for="password" class ="col-sm-2 control-label">{{ form.Password.label }}</label >
                <div class ="col-sm-10">
                {{ form.Password (size=50, class_ ="form-control ") }}
                </div >
        </div >

        <div class ="form-group">
                <div class ="col-sm-offset-2 col-sm-10">
                        <input class ="btn btn-success" type="submit" value ="Se connecter">
                </div >
        </div >
</form >
{% endblock %}
```

Pour que l'utilisateur puisse constater qu'il est bien connecté, on va ajouter son nom, à droite dans la barre de navigation du template base.html, ainsi qu'un lien pour le déconnexion.
Donc dans le , on va rajouter à la fin :

```
<ul class="nav navbar-nav navbar-right">
    {% if current_user.is_authenticated %}
        <a class="navbar-brand"><em>Vous êtes connecté <strong>{{ current_user.Login}}</strong></em></a>
        <li class="nav-item"><a class="nav-link" href="{{ url_for ('logout') }}">Déconnexion</a></li >
    {% else %}
        <li class="nav-item"><a class="nav-link" href="{{ url_for ('login') }}">Se connecter</a></li >
    {% endif %}
</ul>
```

Et pour que la déconnexion fonctionne, il faut rajouter une vue de logout :

```
from flask_login import logout_user

@app.route ("/logout/")
def logout():
        logout_user()
        return redirect ( url_for ('index'))
```

Protection des vues nécessitant une authentification

Dans le template auteur_list.html, nous ne mettrons le lien permettant d'accéder à la page d'édition et de suppression que si l'utilisateur est authentifié :

```
{% if current_user.is_authenticated %}
<a href="{{ url_for('updateAuteur', idA=unAuteur.idA) }}" class="btn btn-warning">Editer <i class="fas fa-pen"></i></a>
<a href="{{ url_for('deleteAuteur', idA=unAuteur.idA) }}" class="btn btn-danger">Supprimer <i class="fas fa-trash"></i></a>
{% endif %}
```

Attention ! Ceci n'est pas suffisant car un utilisateur pourrait taper l'url de la page d'édition d'un auteur dans la barre d'adresse de son navigateur web. Il faut donc protéger la vue d'édition. On peut le faire très facilement avec un décorateur @login_required devant la vue concernée :

```
from flask_login import login_required

@app.route('/auteurs/<idA>/update/')
@login_required
def updateAuteur(idA):
    unAuteur = Auteur.query.get(idA)
    unForm = FormAuteur (idA=unAuteur.idA , Nom=unAuteur.Nom)
    return render_template("auteur_update.html",selectedAuteur=unAuteur, updateForm=unForm)
```

C'est à vous de jouer !

Protégez toutes les vues d'insertion, de modification et de suppression d'auteurs. Faites de même pour les livres, sans oublier les liens d'accès à ces pages. En somme, un visiteur non identifié a le droit uniquement de consulter.

Mise en place de la redirection automatique

C'est bien beau tout cela mais le message renvoyé pourrait être plus accueillant…Pour rendre les choses plus pratiques, on voudrait que lorsque, lorsqu'une vue est ainsi protégée et que l'utilisateur n'est pas encore authentifié, il soit automatiquement redirigé vers la vue de login. Ceci peut être réalisé très simplement par la configuration suivante dans le fichier app.py :

```
login_manager.login_view = " login"
```

On informe ainsi le login_manager du nom de la fonction réalisant la vue de login. Dans notre cas, Cette fonction s'appelle login.

Une fois que l'utilisateur s'est authentifié, il faudrait qu'il soit redirigé vers la page à laquelle il tentait initialement d'accéder. Lors de la redirection initiale vers la page de login, l'url d'où on vient est précisé par le paramètre next de la requête HTTP.

Il faudrait se souvenir de ce paramètre dans notre formulaire de login pour pouvoir, après un login réussi, rediriger l'utilisateur vers la page à laquelle il tentait initialement d'accéder. On va donc ajouter un champ dans notre formulaire LoginForm :

```
class LoginForm(FlaskForm):
    Login = StringField ('Identifiant')
    Password = PasswordField ('Mot de passe')
    next = HiddenField()
```

Et on modifie notre vue login de la manière suivante :

```
@app.route ("/login/", methods =("GET","POST" ,))
def login():
    unForm = LoginForm ()
    unUser=None
    if not unForm.is_submitted():
        unForm.next.data = request.args.get('next')
    elif unForm.validate_on_submit():
        unUser = unForm.get_authenticated_user()
        if unUser:
            login_user(unUser)
            next = unForm.next.data or url_for("index",name=unUser.Login)
            return redirect (next)

    return render_template ("login.html",form=unForm)
```

Vérifiez que les redirections fonctionnent.