

サインアップ、サインイン機能の実装方法

目次

- [サインアップ、サインイン機能の実装方法](#)
- [目次](#)
- [注意事項](#)
- [実装内容](#)
- [環境構築](#)
 - [Flaskアプリ起動](#)
 - [Windows\(CMD\)環境](#)
 - [Mac\(Bash\)環境](#)
- [プログラム解説](#)
 - [1. 前提](#)
 - [概要](#)
 - [main.py](#)
 - [2. プログラム実行時に処理されるプログラム](#)
 - [__init__.py](#)
 - [3. プログラム実行時にindex.htmlを表示する](#)
 - [main.py](#)
 - [4. index.htmlからadd.html\(新規登録画面\)に遷移する](#)
 - [main.py](#)
 - [index.html](#)
 - [5. add.htmlのinput要素\(テキストボックスなど\)に入力された値を取得しcheck.html\(確認画面\)に表示する](#)
 - [概要](#)
 - [main.py](#)
 - [add.html](#)
 - [check.html](#)
 - [補足\(POSTとGETの使い分けについて\)](#)
 - [6. セキュリティについて](#)
 - [GETで通信された場合](#)
 - [やり方](#)
 - [main.py](#)
 - [補足\(render_templateとredirectの違い\)](#)
 - [未入力チェック](#)
 - [やり方](#)
 - [main.py](#)
 - [add.html](#)
 - [パスワード、メールアドレスの形式チェック](#)
 - [やり方](#)
 - [main.py](#)
 - [パスワードの表示について](#)
 - [やり方](#)

- check.html
 - 補足
 - 7. データベース作成
 - 概要
 - db.py
 - __init__.py
 - 補足(CREATE文の文法等々)
 - 主キーについて
 - 基本的な文法
 - NOT NULLについて
 - IF NOT EXISTSについて
 - 8. 入力されたデータをデータベースに保存する
 - 概要
 - sessionの使い方
 - __init__.py
 - main.py
 - check.html
 - データを保存する
 - main.py
 - comp.html
 - 補足(INSERT文の文法等々)
 - 基本的な文法
 - 9. ログイン処理
 - 概要
 - POSTの場合
 - main.py
 - 補足(SELECT文の文法等々)
 - 基本的な文法
 - データを全て取得する場合
 - WHERE句
 - ORDER BY句
 - 補足(fetchall関数について)
 - GETの場合
 - 爆裂重要事項
 - main.py
 - top.html
 - 10. 既存のユーザーIDやメールアドレスを登録させない処理
 - 概要
 - main.py
- おまけ
 - 知っておいた方が良いこと
 - データベースの操作方法
 - 概要
 - 環境構築
 - コマンドプロンプトで操作する
 - Webアプリケーション開発でよく使うSQL文

- 概要
- UPDATE文
 - 爆裂重要事項
 - 基本的な文法
 - WHERE句で条件式をつける
- DROP TABLE文
 - 基本的な文法
- 補足(DELETE文について)
- sqlalchemyについて
 - 概要
 - メリット
 - デメリット
 - 文法
- flask-loginについて
 - 概要

注意事項

基本的にPCでGitHubから閲覧することをおすすめするが、移動中など、どうしてもスマホで閲覧したい場合は[このリンク\(PDF\)](#)から閲覧することをおすすめする。なお、おまけの画像のみGitHubでしか閲覧できない。

実装内容

- サインアップ(新規登録)機能
- サインイン(ログイン)機能

環境構築

事前にライブラリをインストールする必要がある。インストール方法は以下の通り。

```
pip install flask, jinja2, sqlite3
```

Flaskアプリ起動

Flaskアプリの起動方法は以下の通り。

Windows(CMD)環境

```
set FLASK_APP=flaskr  
set FLASK_ENV=development  
set FLASK_DEBUG=1  
python -m flask run
```

Mac(Bash)環境

```
export FLASK_APP=flaskr  
export FLASK_ENV=development  
export FLASK_DEBUG=1  
python -m flask run
```

プログラム解説

1. 前提

概要

main.pyにて、以下のライブラリ、モジュールをインポートすること。htmlファイルなど必要なものは各自用意し、templatesフォルダに保存すること。

main.py

```
import re
import hashlib
import sqlite3
from flaskr import app
from flask import render_template, request, redirect, url_for, session
```

2. プログラム実行時に処理されるプログラム

__init__.py

```
from flask import Flask
app = Flask(__name__)
import flaskr.main
```

3. プログラム実行時にindex.htmlを表示する

main.py

main.pyに以下を追記する。

```
@app.route('/')
def index():
    return render_template(
        'index.html'
    )
```

4. index.htmlからadd.html(新規登録画面)に遷移する

main.py

上記のindex関数とほとんど同じ。render_template関数は画面遷移する時に使用する。
main.pyに以下を追記する。

```
@app.route('/add')
def add():
    return render_template(
        'add.html'
    )
```

index.html

aタグのhref属性やformタグのaction属性に以下の様に記述する。{{}}を使えばhtmlにPythonのコードを埋め込めるようになる。

```
<!-- aタグの場合 -->
<a href="{{ url_for('add') }}">新規登録はこちら</a>
<!-- formタグの場合 -->
<form method="get" action="{{ url_for('add') }}">
    <input type="submit" value="新規登録">
</form>
```

5. add.htmlのinput要素(テキストボックスなど)に入力された値を取得しcheck.html(確認画面)に表示する

概要

- プログラムの通信方法はPOSTとする。
- @app.routeの()内にmethods=['POST']と記述することにより、POSTでの通信が可能となる。
- input要素に入力された値はrequest.form['name属性']で取得できる。
- 遷移先の画面へ値を持っていくにはrender_template関数に変数名=値で持っていける。なお、分かりやすいように値は変数を参照し、変数名と値(変数を参照する場合)は同じ名前にすることをおすすめする。
- 遷移先の画面(html)に値を表示するには{{}}内に変数名を記述することで表示できる。

main.py

main.pyに以下を追記する。

```
@app.route('/check', methods=['POST'])
def check():
    userID = request.form['userID']
    password1 = request.form['password1']
    password2 = request.form['password2']
    mail = request.form['mail']

    return render_template(
        'check.html',
        userID=userID,
        password1=password1,
        mail=mail
    )
```

add.html

```
<!-- 以下の様に必ずname属性を記述すること。以下のコードの記述場所はどこでも良い -->
<form method="post" action="{{ url_for('check') }}">
    <input type="text" name="userID">
    <input type="password" name="password1">
    <input type="password" name="password2">
    <input type="text" name="mail">
    <input type="submit" value="確認">
</form>
```

check.html

```
<!-- 以下はpタグで表示しているが、文字が表示できるタグであれば何でも良い -->
<p>{{ userID }}</p>
<p>{{ password1 }}</p>
<p>{{ mail }}</p>
```

補足(POSTとGETの使い分けについて)

input要素で入力された値を取得、保存する時の通信方法はPOST、そうでない時(画面遷移など)はGETと覚えておく程度で良い。

もう少し詳しくPOSTとGETの違いについて知りたい場合はググるかパスやFEなどの参考書を参照すると良い。

6. セキュリティについて

GETで通信された場合

現状、ブラウザのurl欄にcheck.htmlのurl(localhost:5000/check)を入力するとエラーが表示される。実際にクラッカーなどはこのエラーメッセージを読み攻撃手法を考えるらしいので、表示されないようにする。

やり方

- url欄に直接urlを入力してアクセスすると、通信方法はGETになる為、通信方法がGETの場合index.htmlなどにリダイレクトしてやれば良い。
- @app.routeのmethods=['POST']に'GET'を追加する。
- request.methodで通信方法を取得できるので、if文でPOSTでの通信の処理と、GETでの通信の処理を分けてやれば良い。
- リダイレクトはredirect関数とurl_for関数を使う。html内でも出てきたが、url_for関数は拡張子(.html)は不要である。

main.py

main.pyのcheck関数に以下を追記する。

```
@app.route('/check', methods=['POST', 'GET']) # 敢えて、GETでの通信も許可してやる。
def check():
    if request.method == 'POST':
        userID = request.form['userID']
        # 以下略
    else:
        return redirect(
            url_for('index')
        )
```

補足(render_templateとredirectの違い)

実際に試してみると分かるが、render_templateは表示されているhtmlが同じものであろうと、urlには処理された関数名が表示される。

redirectはurl_for関数に記述してある関数を実行する為、リダイレクト先の関数名が表示される。

未入力チェック

現状、何も入力していなくてもcheck.htmlに遷移してしまう。

空のデータがデータベースに保存されてしまうのはまずいので、何とかする。

入力されていないデータがある場合はadd.htmlに遷移し、エラーを表示するようにする。

やり方

- if文で変数に何も値が入っていないものを区別すれば良い。
- なぜ下記の条件式でできるのか分からない人は、Python falsyで検索すること。

main.py

main.pyに以下を追記する。

```
# 上部省略
mail = request.form['mail']
if userID and password1 and password2 and mail:
    return render_template(
        'check.html',
        userID=userID,
        password1=password1,
        mail=mail
    )
else:
    error = "全ての項目を入力してください"
    return render_template(
        'add.html',
        error=error
    )
# 以下略
```

add.html

```
{% if error %}
    <p>{{ error }}</p>
{% endif %}
<form method="post" action="{{ url_for('check') }}">
    <input type="text" name="userID">
<!-- 以下略 -->
```

パスワード、メールアドレスの形式チェック

現状、パスワードとメールアドレスは何を入力してもcheck.htmlに遷移してしまう。

パスワードに関しては、強度の高いパスワードを設定させるようにするのが望ましい。

強度が低いパスワードを入力されたらエラーを表示するようにする。

確認用で同じパスワードを入力させ比較し、違うものが入力されていたらエラーを表示するようにする。

メールアドレスに関しては正しい形式でなければエラーを表示するようにする。

やり方

- if文で正規表現と比較する。
- 以下パスワードは6文字以上半角英数字記号を使用できるようにしたものである。
※if文の条件式が長い為、横までスクロールして確認すること。

main.py

main.pyのcheck関数に以下を追記する。

```
mail = request.form['mail']

password_pattern = r'^(?=.*[0-9a-zA-Z\W]).{6,}$'
mail_pattern = r'^[\w\.-]+@[ \w\.-]+\.\w+$'
if userID and password1 and password2 and mail and password1 == password2 and
re.match(password_pattern, password1) and re.match(mail_pattern, mail):
    # 省略
elif password1 != password2:
    error = "入力されたパスワードと確認用のパスワードが異なります"
elif not(userID and password1 and password2 and mail):
    error = "全ての項目を入力してください"
elif not(re.match(password_pattern, password1)):
    error = "パスワードは次の条件を満たしている必要があります。半角英数字記号を使用、6文字以上"
elif not(re.match(mail_pattern, mail)):
    error = "正しいメールアドレスの形式ではありません。"
# if文の外に記述すること
return render_template(
    'add.html',
    error=error
)
# 以下略
```

パスワードの表示について

現状、画面遷移先で入力されたパスワードが表示される。
セキュリティ的に良くないので●などで伏せて表示されるようにする。

やり方

check.htmlにて乗算やlengthを使用し、パスワードの文字数分●を表示するようにする。

check.html

```
<p>{{ userID }}</p>  
<p>{{ '●' * user.password|length }}</p>  
<p>{{ mail }}</p>
```

補足

このやり方もパスワードの文字数が分かってしまう為、万全なセキュリティ対策とは言えない。
もしこれよりもセキュリティを万全にしたい場合は各自で調べること。

7. データベース作成

概要

データベースはSQLiteを使用する。
ユーザーテーブルを作成する。テーブル定義は以下の通り。

No.	カラム名	データ型	Not Null	デフォルト	備考
1	userID	VARCHAR	NOT NULL		ユーザーのID
2	hashed_password	VARCHAR	NOT NULL		ユーザーのハッシュ化されたパスワード
3	mail	VARCHAR	NOT NULL		ユーザーのメールアドレス

db.py

- db.pyではテーブルの作成を行う。つまり、CREATE文しか記述しない。
- `con = sqlite3.connect(DATABASE)`でデータベースと接続できる。
- `con.execute("")`の""の中にSQL文を記述する。なお、"でも動作するが、SQLの文法上"を使用することが多々ある為、""を使用することをおすすめする。
- `con.close()`でデータベースとの接続を切断できる。用事が済んだら必ず切断すること。

db.pyに以下を追記する。

```
import sqlite3

DATABASE = 'データベース名.db'

def create_expenses_table():
    con = sqlite3.connect(DATABASE)
    con.execute("CREATE TABLE IF NOT EXISTS user(userID VARCHAR PRIMARY KEY, hashed_password VARCHAR NOT NULL, mail VARCHAR NOT NULL)")
    con.close()
```

__init__.py

__init__.pyに以下を追記して、実行時にデータベースが作成されるようにする。

```
from flaskr import db
db.create_expenses_table()
```

補足(CREATE文の文法等々)

主キーについて

- PRIMARY KEYと記述する。
- 重複を許さない。
- NULL(空文字)を許さない。
- 基本的に一番最初に記述する。

基本的な文法

```
CREATE TABLE テーブル名(カラム名1 型名 PRIMARY KEY, カラム名2 型名)
```

NOT NULLについて

主キー以外でNULLを許さない場合、NOT NULLと記述する。

```
CREATE TABLE テーブル名(カラム名 型名 主キー, カラム名 型名 NOT NULL)
```

IF NOT EXISTSについて

PythonやPHPなどのプログラミング言語でデータベースを作成するプログラムを記述すると、毎回CREATE文が実行されてしまう。

2度目の実行は不要な上に、データが全て消えてしまう危険性がある為、テーブルが存在しない時だけCREATE文を実行するようにしたい。

IF NOT EXISTSを使用すれば上記のことが解決できる。

```
CREATE TABLE IF NOT EXISTS テーブル名(カラム名 型名 主キー, カラム名 型名 NOT NULL)
```

8. 入力されたデータをデータベースに保存する

概要

check.htmlからcomp.htmlに遷移するタイミングにデータを保存する。
現状、check.htmlからユーザーID、パスワードなどの情報を取得できない。
sessionを使用し、取得できるようにする。

sessionの使い方

- セッションキーを設定する。
- データを保持する時間を設定する。
- `session[配列名] = 変数`でsessionにデータを保存できる(配列名は何でもok)。
- 今回はsessionに保存したいデータが複数ある為、保存したいデータを辞書型にする。

`__init__.py`

`__init__.py`に以下を追記する。

```
from flask import Flask
app = Flask(__name__)
import flaskr.main

import os
app.secret_key = os.urandom(24) # セッションキーの設定

from datetime import timedelta
app.config['PERMANENT_SESSION_LIFETIME'] = timedelta(hours=1) # データを保持する時間の設定

from flaskr import db
db.create_expenses_table()
```

main.py

main.pyのcheck関数に以下を追記する。

```
if userID and password1 and password2 and mail and password1 == password2 and
re.match(password_pattern, password1) and re.match(mail_pattern, mail):

    user = {
        'userID' : userID,
        'password' : password,
        'mail' : mail
    }

    session['user'] = user

    return render_template(
        'check.html',
        user=user
    )
# 以下略
```

check.html

```
<p>{{ user.userID }}</p>
<p> {{ '●' * user.password|length }} </p>
<p>{{ user.mail }}</p>
```


データを保存する

- セキュリティ上、パスワードはハッシュ化する。
- hashlib.sha256(ハッシュ化したい物.encode()).hexdigest()でハッシュ化できる。
- その他、通信方法がGETだった場合の処理も実装する。

main.py

main.pyに以下を追記する。

```
@app.route('/comp', methods=['POST', 'GET'])
def comp():
    if request.method == 'POST':
        user = session.get('user')

        hashed_password = hashlib.sha256(user['password'].encode()).hexdigest()

        con = sqlite3.connect(DATABASE)
        con.execute("INSERT INTO user VALUES(?, ?, ?)", (user['userID'],
hashed_password, user['mail']))
        con.commit()
        con.close()

        return render_template(
            'comp.html'
        )
    else:
        return redirect(
            url_for('index')
        )
```

comp.html

```
<!-- 登録が完了したことの文言を表示する -->
```

補足(INSERT文の文法等々)

データを追加する際に使用する。

基本的な文法

```
INSERT INTO テーブル名(カラム名1, カラム名2) VALUE(値1, 値2);
```

9. ログイン処理

概要

- ユーザーIDとパスワードの比較はSQLのSELECT文で行う。
- ヒットすればユーザーIDとパスワードに誤りは無く、ヒットしなければどちらかが誤っている、もしくはユーザーIDが存在しないことになる。
- パスワードはハッシュ化して保存されている為、入力されたパスワードはハッシュ化してから比較しなければいけない。
- ユーザーIDとパスワードに誤りが無ければtop.htmlに遷移するようにする。
- ユーザーIDとパスワードに誤りがあればindex.htmlに遷移してエラーメッセージを表示する。

POSTの場合

main.py

main.pyに以下を追記する。

```
@app.route('/top', methods=['GET', 'POST'])
def top():
    if request.method == 'POST':
        userID = request.form['userID']
        hashed_password =
        hashlib.sha256(request.form['password'].encode()).hexdigest()

        con = sqlite3.connect(DATABASE)
        existing = con.execute("SELECT * FROM user WHERE userID = ? AND
hashed_password = ?", (userID, hashed_password)).fetchall()
        con.close()

        if existing:
            session['userID'] = userID
            return render_template(
                'top.html',
                userID=userID
            )
        else:
            error = 'ユーザーIDかパスワードが間違っています。'
            return render_template(
                'index.html',
                error=error
            )
```

補足(SELECT文の文法等々)

基本的な文法

```
SELECT カラム名1, カラム名2 FROM テーブル名;
```

データを全て取得する場合

```
SELECT * FROM テーブル名;
```

WHERE句

```
SELECT * FROM テーブル名 WHERE カラム名 = 'example'; /* カラム名の値がexampleに一致するデータを取得する */
SELECT * FROM テーブル名 WHERE カラム名 <> 'example'; /* カラム名の値がexampleに一致しないデータを取得する */
SELECT * FROM テーブル名 WHERE カラム名 < 10; /* カラム名の値が10未満のデータを取得する */
SELECT * FROM テーブル名 WHERE カラム名 <= 10; /* カラム名の値が10以下のデータを取得する */
SELECT * FROM テーブル名 WHERE カラム名 > 10; /* カラム名の値が10より大きいデータを取得する */
SELECT * FROM テーブル名 WHERE カラム名 >= 10; /* カラム名の値が10以上のデータを取得する */
```

ORDER BY句

```
SELECT * FROM テーブル名 ORDER BY カラム名 ASC; /* カラム名の値を昇順に並べて表示する(昇順の場合ASCは省略可) */
SELECT * FROM テーブル名 ORDER BY カラム名 DESC; /* カラム名の値を降順に並べて表示する(降順の場合DESCは省略可) */
```

補足(fetchall関数について)

fetchall関数ではSELECT文で取得したデータを全て変数に格納する関数である。リスト形式で取得する。なお、最初の1件だけ取得したい場合はfetchone関数を使用すれば良い。

```
data = con.execute("SELECT * FROM table").fetchall() # 取得したデータを全てdataに格納する
data = con.execute("SELECT * FROM table").fetchone() # 最初の1件だけdataに格納する
```

GETの場合

爆裂重要事項

check関数やcomp関数では通信方法がGETだった場合、単純にindex.htmlにリダイレクトしてやれば良かった。

しかし、ログイン後に別のページからtop.htmlに遷移する可能性が高い為、GETでアクセスされた場合の処理も考えなければいけない。

下記のプログラムでは、セッションがあればログインしていると見做し、top.htmlに遷移するようにしている。

対してセッションがない場合は、セッションの有効期限が切れている、もしくは、直接url欄にtop.htmlのurlを打ち込んでアクセスされたと見做し、index.htmlに遷移し、セッションが切れて自動的にログアウトしたとエラーメッセージを表示するようにしている。

なお、後者の場合も、セッションが切れて自動的にログアウトしたとエラーメッセージが表示されるが、後者は大半が悪意のあるユーザーしか行わない行為であるため、同じエラーメッセージでも良しとしている。

main.py

main.pyのtop関数に以下を追記する。

```
@app.route('/top', methods=['GET', 'POST'])
def top():
    if request.method == 'POST':
        # 省略
    else:
        if 'userID' in session:
            userID = session['userID']

            return render_template(
                'top.html',
                userID=userID
            )
        else:
            error = '長時間操作が行われなかったため、ログアウトしました。'
            return render_template(
                'index.html',
                error=error
            )
```

top.html

ただtop.htmlに遷移しただけでは本当にログインできているか分からない為、ユーザーIDを表示するようにする。

```
<p>{{ userID }}</p>
```

10. 既存のユーザーIDやメールアドレスを登録させない処理

概要

このアプリのデータベース(ユーザーテーブル)では、主キーがユーザーIDになっており、既存のユーザーIDを追加しようとするエラーになる。

また、大規模SNSは例外として、メールアドレスも重複を許さないアプリが大半である。

main.py

main.pyのcheck関数に以下を追記する。

```
@app.route('/check', methods=['POST', 'GET'])
def check():
    if request.method == 'POST':
        # 省略

        con = sqlite3.connect(DATABASE)
        existing_user = con.execute("SELECT userID FROM user WHERE userID = ?",
        (userID,)).fetchone()
        existing_mail = con.execute("SELECT mail FROM user WHERE mail = ?",
        (mail,)).fetchone()
        con.close()

        if userID and password1 and password2 and mail and password1 == password2 and
        re.match(password_pattern, password1) and re.match(mail_pattern, mail) and
        not(existing_user) and not(existing_mail):

            # 省略

        elif password1 != password2:
            # 省略
        elif not(userID and password1 and password2 and mail):
            # 省略
        elif not(re.match(password_pattern, password1)):
            # 省略
        elif not(re.match(mail_pattern, mail)):
            # 省略
        elif existing_user:
            error = "ユーザーIDが既に存在します。別のユーザーIDを設定してください。"
        elif existing_mail:
            error = "このメールアドレスは既に登録されています。"

        return render_template(
            'add.html',
            error=error
        )
    # 以下略
```

おまけ

知っておいた方がよいこと

データベースの操作方法

概要

わざわざPythonで操作しなくても、コマンドプロンプトでデータベースを操作できる。
データの閲覧や、実装前にSQL文が正しく動作するかの確認などで使用すると便利。

環境構築

手順が多い為、Googleで「SQLite パス」などで検索し、個人ブログなどから環境構築の方法を閲覧してください。

学校の場合は、環境構築不要です。

コマンドプロンプトで操作する

コマンドプロンプトで、データベースファイルがあるフォルダまで移動し以下のコマンドを実行する。

```
sqlite3 データベースファイル名.db
```

sqlite3との対話モードになり、この状態でSQL文を実行できる。

なお、デフォルトの状態だとカラム名が表示されず見辛いですが、以下のコマンドでカラム名を表示できるようになる。

```
.mode column
```

終了する時は以下のコマンドを入力する。

```
.exit
```

Webアプリケーション開発でよく使うSQL文

概要

サインアップ、サインイン機能では使わなかったSQL文の紹介

UPDATE文

データを更新、編集する際に使用する。

爆裂重要事項

Webアプリケーションの開発ではUPDATE文の文法は全て押さえていないと、ほぼ確でデータを壊すことになる。

UPDATE文に関する項目は基本的な文法だけでなく、必ず最後まで読むこと！

基本的な文法

以下の例は該当するカラムの値を全て書き換えるものである。

```
UPDATE テーブル名 SET カラム名 = 値;
```

WHERE句で条件式をつける

以下の例はWHERE句で指定した条件式に該当するデータの値を書き換えるものである。

```
UPDATE テーブル名 SET カラム名 = 値 WHERE カラム名 = "値";
```

DROP TABLE文

テーブルを削除する際に使用する。

頻繁に使うものではないが、開発中にテーブル定義が変更になったり、誤って挿入してはいけないデータを挿入してしまったりなど、テーブルを削除せざるを得ないことも多々ある。

基本的な文法

```
DROP TABLE テーブル名;
```

補足(DELETE文について)

データを削除する際に使用するDELETE文だが、Webアプリケーションの開発では使わないことが多い。

実際にデータを削除する際は削除フラグを使用し、アプリ上で表示できなくし、データベース上ではデータを保持しているケースが多い。

sqlalchemyについて

概要

sqlalchemyというSQL文を記述せずにデータベースを操作できるライブラリである。

メリット

- SQLインジェクションの対策になる
- SQL未学習の人、苦手な人は感覚的に操作できて扱いやすい(かもしれない)

デメリット


- SQLが得意な人、慣れている人はsqlalchemyの独特な文法に戸惑う人が多い(気がする)

文法

気が向いたら執筆する。

flask-loginについて

概要

hakase/画像に適当な解説を載せてるけど、PDFでは画像は見れないよ！GitHubで見てね！

もう少し真面目に回答すると、flask-loginでできることは画像の通りだが、ログイン画面の作成、ログイン可能かどうかなどの処理は自分で作成しなければいけない。

つまり、今回の場合flask-loginを使用してもしなくてもさほど変わらない。

また、認証情報の保持がSessionである為、JWTなどはflask-loginは対象外である。

でも本当の理由は勉強するのが面倒だk...