プログラミング演習 2(Python) レポート 2

20K1105 伊藤 葵

2021 年 7 月 31 日

1　課題

自分なりに設計した CGI プログラムを作成せよ。授業で紹介したプログラムを拡張・発展させたものでも構わない。

2　課題の目的

この課題を通し、CGI の仕組みを理解し、それを活用できるようにする。また、データのファイル形式として、CSV や XML 形式に慣れる。さらに、Web API やデータベース（このレポートでは、sqlite3）の仕組みを理解し、操作できるようにする。その他、正規表現を理解し、高度なプログラムを書けるようにする。

具体的には、以下の項目の理解・達成を目指す。

- Web アプリケーション・Web サーバーとは
- CGI による動的なコンテンツの作成
- CSV, XML 形式の理解、データの処理
- 正規表現の書き方
- Web API の仕組みの理解とその活用
- データベースの理解と操作（今回は sqlite3 ）

3　方法

このレポートでは、「今日の外食検索サイト」という名の、ホットペッパー (https://webservice.recruit.co.jp/) の API を用いた飲食店検索サイトを作成する。また、サイトには追加要素として、ログイン機能や各飲食店に「いいね！」ができる機能を搭載する。以上のサイト作成を通して、課題を達成する。

具体的に、どのようにサイトを設計したか説明する。

まず、ディレクトリの構造は以下の通りとした。

Listing 1: ディレクトリの構造

```
R02-CGI
- server.py
- shops.sqlite3
- users.sqlite3
- stylesheet.css
- cgi-bin
  - R02
    - 20K1105-r02.py
    - create_db.py
    - search.py
```

python のソースコードは、巻末付録に掲載した。

server.py は、サーバーを起動するプログラムであり、プログラミング演習 2 (Python) 第 8 回の講義で配布されたものである。20K1105-r02.py は、ログイン処理を行うプログラムであり、今回作成したサイトは server.py の実行によるサーバー起動後に、`localhost:8000/cgi-bin/R02/20K1105-r02.py` と、このファイルを指定することで接続する。`create_db.py` は、データベース作成のプログラムであり、ユーザーとそのパスワードをまとめたデータベース users.sqlite3 と、店名といいねされた回数をまとめたデータベース shops.sqlite3 の二つを作成する。stylesheet.css は、html の体裁を指定しているファイルだが、今回はうまく反映できなかった。詳しい内容は考察にて述べる。search.py は、今回の検索サイトのメインページの処理をになっているプログラムであり、現在地を geojs を用いて取得し、条件に合ったお店を検索、表示する。

次に、各ソースプログラムの詳しい内容を説明する。

server.py は、サーバーの起動と、クライアントからのリクエストに応じ必要な処理をした後、クライアントにレスポンスを返すといった挙動をしており、この挙動は `server.serve_forever()` とすることで、永遠と繰り返されるようになっている。

20K1105-r02.py は、Web サーバーがサーバーとは別に起動した Python 処理系ではじめに実行するプログラムである。サイト利用者に、ユーザー名とパスワードを入れてもらい、`create_db.py` でユーザー名の検索とそのパスワードの認証を行う。入力データ送信後の結果は 3 つ用意した。一つ目は、ユーザーが存在しない場合であり、この場合は「そのユーザーは存在しません。」と表示されるようにした。二つ目は、パスワードの認証が出来た場合である。この場合は、「認証に成功しました。」という文面に加え、検索サイトのメインページである `/cgi-bin/R02/search.py` へのリンクを表示した。三つ目は、認証に失敗した場合である。この場合は、「パスワードが違います。」と表示し、メインページへのリンクは表示されないようにした。認証のソースプログラムは、以下の通りである。

Listing 2: ユーザーとそのパスワードの認証

```
connection = sqlite3.connect("users.sqlite3")
cur = connection.cursor()
cur.execute(f"␣SELECT␣name␣FROM␣users␣WHERE␣name='{v_user}'␣")
user = cur.fetchall()
if user == []:# ユーザーが存在しない場合
    result = "<pそのユーザーは存在しません。></p>"
if user != []:
    cur.execute(f"␣SELECT␣password␣FROM␣users␣WHERE␣name='{v_user}'␣")
    pw = cur.fetchall()
    if v_password == pw[0][0]:
        result = "<p認証に成功しました。>␣<a␣href='/cgi-bin/R02/search.py進
            む'></a></p>"
    else:
        result = "<pパスワードが違います。></p>"
if v_user == v_password == '':
    result = ""
```

`v_user, v_password` という変数は、サイト利用者が入力したユーザー名とパスワードを表している。sqlite3 の `SELECT` 文を用い、 `v_user` に一致するカラム `name` のパスワードのデータをテーブル `users` から取得する。そして、取得したデータ `user` が空のリスト、すなわち何もデータが得られなかった場合は、ユーザーがいないとした。データが取得できた場合は、変数 `v_password`

と比較し、等しければ認証できたとした。

create_db.py は、データベース作成のプログラムである。ユーザーとそのパスワードをまとめたデータベース users.sqlite3 と、店名といいねされた回数をまとめたデータベース shops.sqlite3 の二つを作成した。このデータベース作成の python ソースプログラムは、サイト接続前に一度実行し、作成された二つのファイルを Listing1 に記した位置に配置する必要がある。二つのデータベース作成のソースプログラムは以下の通りである。

Listing 3: データベース作成

```
connection = sqlite3.connect("users.sqlite3")
cur = connection.cursor()

# ユーザーとそのパスワードのデータベースを作成する
cur.execute("CREATE␣TABLE␣users(name,␣password)")
def create_users(cur, name, password):
    cur.execute(f'INSERT␣INTO␣users␣VALUES("{name}",␣"{password}")')
# 名前とパスワード一覧（今回は乱数を用いて 100 件生成）
names = # 百個のユーザー名のリスト
passwords = # 百個のパスワードのリスト
for x in range(100):
    create_users(cur, names[x], passwords[x])
connection.commit()
connection.close()

# 店名といいねされた回数のデータベースを作成する
connection = sqlite3.connect("shops.sqlite3")
cur = connection.cursor()
cur.execute("CREATE␣TABLE␣shops(shop,␣nice)")
connection.commit()
connection.close()
```

users.sqlite3 では、sqlite3 の CREATE TABLE 文を用い、カラム name, password をとるテーブル users を作成した。また、乱数を用いて、百件のユーザー名とパスワードを作成し、INSERT 文によりデータを追加した。最後に、connection.commit() によるデータベースの反映をし、connection.close() でデータベースを閉じた。同様に、 shops.sqlite3 ではカラム shop, nice をとるテーブル shops を作成した。カラム shop は店名、カラム nice は、ユーザーにいいねされた数を意味する。このデータベースのデータは、メインページにてお店が検索されるたびに追加されていくという方針を取った。

最後に search.py の説明をする。これは、検索サイトのメインページの処理をしており、以下の流れで処理を行う。

1. 現在地の取得
2. 1 で得た位置情報と、サイト利用者が入力した条件にあう飲食店を検索
3. 結果を表示
4. 「いいね！」されたお店のいいね数を増やす

現在地の取得には、geojs を用いた。これは、https://get.geojs.io/v1/ip/geo.json にリクエストを飛ばすことで、だいたいの位置情報の緯度と経度を結果として返す。ここで、取得した緯度と

経度のデータと、サイト利用者が入力したお店のジャンルと検索範囲の指定を、ホットペッパーの API に渡す。ホットペッパーの API へのリクエストの飛ばし方は、以下の通りである。ホットペッパーの API 利用は無料だが、登録が必要である。

Listing 4: ホットペッパーの API の利用

```
response = requests.get(
    'http://webservice.recruit.co.jp/hotpepper/gourmet/v1/',
    params={
    'key': API キー,
    'genre': v_g,
    'lat': la,
    'lng': lo,
    'range': v_r}
    )
result = response.text
shops = collect_info(result)
return shops
```

各パラメータを説明する。key には、ホットペッパーの API 利用登録時に取得する API キーを指定する。genre には、サイト利用者が取得したジャンル v_g を指定しており、ジャンルは居酒屋、ダイニングバー・バル、創作料理、和食、洋食、イタリアン・フレンチ、中華、焼肉・ホルモン、韓国料理、アジア・エスニック料理、各国料理、カラオケ・パーティ、バー・カクテル、ラーメン、お好み焼き・もんじゃ、カフェ・スイーツ、その他グルメの計 15 種である。lat, lng はそれぞれ緯度、経度を表しており、先ほど geojs を用いて取得したデータ la, lo を指定している。最後に、range に現在地から何メートルの範囲内のお店を検索するかを指定している。また、ホットペッパーの API 使用時には、クレジットを表示する必要がある。お店の検索結果は xml 形式で返されるため、今回は正規表現を用いて必要な店舗情報を取得することとした。

Listing 5: 店舗情報の取得

```
def collect_info(xml):
    connection = sqlite3.connect("shops.sqlite3")
    cur = connection.cursor()
    name = r'</id><name>(.*?)</name>'
    name_list = re.findall(name, xml)
    station = r'<station_name>(.*?)</station_name>'
    station_list = re.findall(station, xml)
    access = r'<access>(.*?)</access>'
    access_list = re.findall(access, xml)
    open_time = r'<open>(.*?)</open>'
    open_list = re.findall(open_time, xml)
    close = r'<close>(.*?)</close>'
    close_list = re.findall(close, xml)
    wifi = r'<wifi>(.*?)</wifi>'
    wifi_list = re.findall(wifi, xml)
    price = r'</name><average>(.*?)</average>'
    price_list = re.findall(price, xml)
    shops = []
    for x in range(len(name_list)):
```

```
53          cur.execute(f"␣SELECT␣nice␣FROM␣shops␣WHERE␣shop='{name_list[x]}'␣
                ")
54          point = cur.fetchall()
55          if point == []:
56              data = [name_list[x], station_list[x], access_list[x],
57                      open_list[x], close_list[x], wifi_list[x],
58                      price_list[x], 0]
59          else:
60              data = [name_list[x], station_list[x], access_list[x],
61                     open_list[x], close_list[x], wifi_list[x],
62                     price_list[x], point[0][0]]
63          shops.append(data)
64      return shops
```

xml から取得した店舗情報は、店名、最寄り駅、行き方、開店・閉店時間、wi-fi の有無、平均予算であり、さらに「いいね！」された回数をデータベース shops.sqlite3 を用いて取得した。店舗情報の取得は、正規表現の `re.findall` によるマッチングを用いた。第一引数となるマッチングパターンは、xml ファイルのタグ< >を用いた。そして、「いいね！」された回数の取得だが、はじめてお店 A が検索された場合と既に A が検索されたことがある場合の二つの場合でデータベースの処理を変えた。初めてお店 A が検索された場合は、そのお店のデータはデータベースにはまだ存在しないため、いいね数を 0 とし、先に検索した店舗情報と合わせてリスト data を作成した。既に A が検索されたことがある場合は、上記ソースプログラム 53・54 行目でデータベースからいいね数を取得できるため、その数と店舗情報を合わせてリスト data とした。

ここで、いいね！に伴うデータベースの処理を説明する。この処理のソースプログラムは、以下の通りである。

Listing 6: いいね！によるデータベースの処理

```
65  def add_nice(v_nice):
66      connection = sqlite3.connect("shops.sqlite3")
67      cur = connection.cursor()
68      connection = sqlite3.connect("shops.sqlite3")
69      cur = connection.cursor()
70      cur.execute(f"␣SELECT␣nice␣FROM␣shops␣WHERE␣shop='{v_nice}'␣")
71      point = cur.fetchall()
72      if point == []:
73          cur.execute(f'INSERT␣INTO␣shops␣VALUES("{v_nice}",␣"1")')
74          connection.commit()
75      else:
76          p = int(point[0][0])
77          new_p = p+1
78          cur.execute(f"UPDATE␣shops␣set␣nice='{new_p}'␣where␣shop='{v_nice
                }'")
79          connection.commit()
80      connection.close()
```

shops.sqlite3 は、はじめから全国の店舗情報を含んだテーブルを作成すると、ファイルが大きくなってしまうため、お店が検索されるとその店名といいね数 0 のデータを追加していくこととした。まずはじめに、shops.sqlite3 のカーソルを取得する。もし、お店 A が初めて検索され、か

ついいね！をされた場合、INSERT 文により、店名といいね数 1 のデータを追加する。そして、お店 A がすでに検索されたことがある場合、上記のプログラム構成からデータベースにその店舗のデータは存在するため、上記ソースプログラム 70・71 行目でいいね数のデータを取得できる。いいね数を取得した場合は、sqlite3 の UPDATE 文を用いて、カラム nice (いいね数) の更新を行う。いいね数の処理は 76・77 行目で行っており、取得したデータを int 型に変換後、+1 をしている。以上のように、検索しまとめた店舗の情報を html の   構文を用い、箇条書きにて表示させた。また、検索結果の末尾に、サイト利用者がいいね出来るよう、「いいね！」ボタンを作成した。検索結果の html の作成は、以下のように行った。

Listing 7: 検索結果の html の作成

```
81 def make_ans(shops):
82   ans = ""
83   for s in range(len(shops)):
84       ans += "<p>"
85       ans += f"{s店目+1}</p>"
86       ans += "<ul>"
87       ans += f"<li店名>:␣{shops[s][0]}</li>"
88       ans += f"<li最寄り駅>:␣{shops[s][1]}</li>"
89       ans += f"<li行き方>:␣{shops[s][2]}</li>"
90       ans += f"<li開店・閉店時間>:␣{shops[s][3]}~{shops[s][4]}</li>"
91       ans += f"<li>wi-fi␣の有無:␣{shops[s][5]}</li>"
92       ans += f"<li平均予算>:␣{shops[s][6]}</li>"
93       ans += f"<li「いいね！」された回数>:␣{shops[s][7]}</li>"
94       ans += "</ul>"
95       ans += "</p>"
96   favorite = "<pお店に「いいね！」をつけませんか？><br>"
97   for s in range(len(shops)):
98       favorite += f"<p>{s店目+1}"
99       favorite += f"<button␣type='submit'␣name='nice'␣value='{shops[sいいね！][0]}'></button><br>"
100  favorite += "</p>"
101  return ans+favorite
```

変数 ans と favorite に、必要な文面を追記していった。

4 実行結果

はじめに、create_db.py を実行する。次に、作成された二つのデータベースを、server.py と同じディレクトリ配下に配置する。そして、コマンドプロンプトを用い 'python server.py' を実行、サーバーを起動した後、任意の Web ブラウザのアドレスバーに localhost:8000/cgi-bin/R02/20K1105-r02.py を入力する。

実行結果は、以下の通り。

図 1: ログイン画面

(i) 存在しないユーザー名として NoUser、(ii) 正しいユーザー名「イトウアオイ」と誤ったパスワード ppppp を入力すると、警告文が表示される。

図 2: (i) 存在しないユーザー名を入力した場合

図 3: (ii) 誤ったパスワードを入力した場合

正しいユーザー名「イトウアオイ」と正しいパスワード「Hkkx3P6SB3」を入力すると、認証される。

図 4: 認証が通った場合

進む、というハイパーリンクをクリックすると、メインページに遷移する。メインページの検索前の状態は、図 5 の通り。

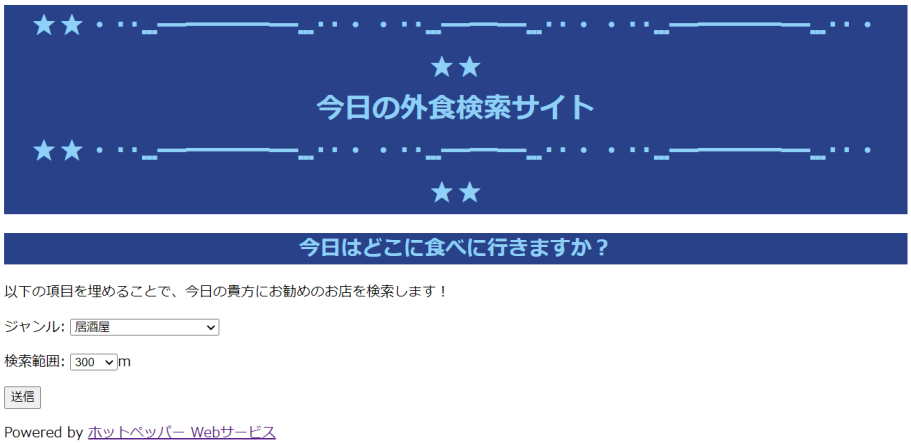

図 5: 検索前のメインページの様子

試しに、お店のジャンルを「イタリアン・フレンチ」、検索範囲を 500m に指定した際の検索結果は図 6 の通り。

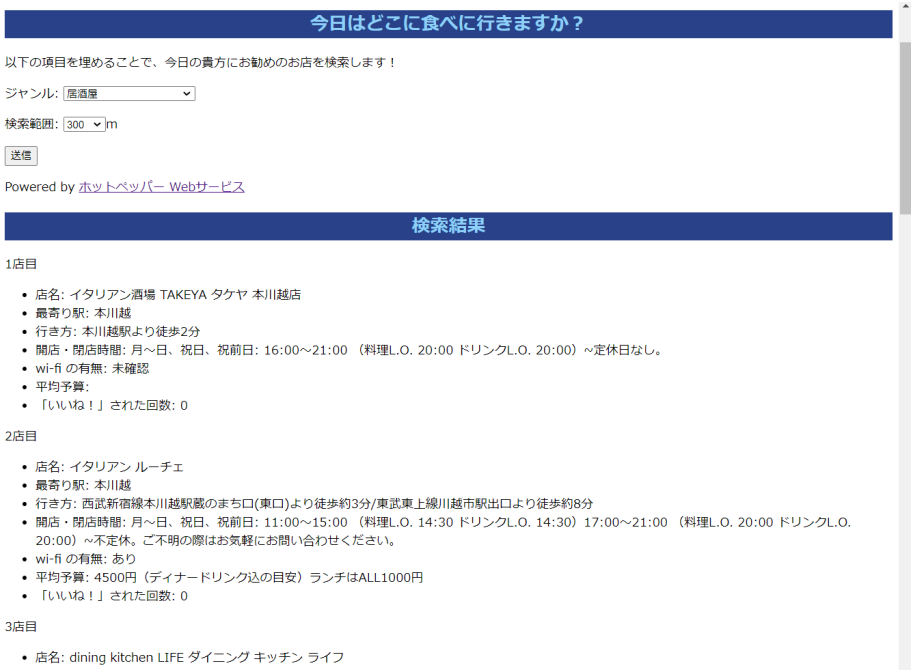

図 6: 検索結果

ページ下部に、いいね！をする欄がある。

図 7: いいね！ボタン

ここでは、一件目に表示されたお店「イタリアン酒場 TAKEYA タケヤ 本川越店」のいいね！ボタンを押し、ページをリロード、または同じ条件で店舗検索をすると、いいね数が反映されていることが分かる。(図 8)

図 8: いいね！数の反映

5 考察

はじめに、第 2 章で述べた目標の達成度について考察する。それぞれの項目の具体的な達成方法は以下の通りである。達成した項目とサイトの細かい表示結果や挙動の反省点を考える。

表 1: 目標の達成度

目標	具体的な達成方法
Web アプリケーション・Web サーバーとは	server.py,20K1105-r02.py,search.py
CGI による動的なコンテンツの作成	20K1105-r02.py,search.py
CSV, XML 形式の理解、データの処理	xml タグを用いた店舗情報のまとめ
正規表現の書き方	xml タグを用いたマッチング
Web API の仕組みの理解とその活用	ホットペッパー API
データベースの理解と操作（今回は sqlite3 ）	ユーザー認証、いいね数の処理

まず、Web API の利用である。https://webservice.recruit.co.jp/doc/hotpepper/reference.html に記載されている検索クエリ・サンプルクエリを参照することで、うまく API を用いて店舗検索ができたと私は考える。次に、ホットペッパーの API から取得したデータを適切な方法で取得できたか、という点について考察する。今回、データは xml 形式で返され、xml のタグを用いた正

規表現によるマッチングで必要な情報を取得した。この方法以外に、xml 形式の書き方で結果が返されていることを活かし、一旦 xml ファイルを作成し、そのファイルを操作するという方法も考えられる。だが、この場合、検索されるたびに xml ファイルが生成されていくため、メモリを圧迫すると考察できる。また、店舗は増えたり減ったりと動的なデータであるため、生成した xml ファイルを参照して次回、同じ場所で同じ条件で検索された際にそのファイルを参照するということもできない。ただし、xml ファイルを作成した場合、そこから CSV ファイルを作成して、データ処理を分かりやすく容易にすることができるため、メモリに余裕がある場合は、xml ファイルを作成していってもよいと思う。

サイトの細かい表示結果や挙動や表示の反省点を考える。

まず、ログイン画面についてだが、一旦送信ボタンを押すと、打ったユーザー名とパスワードが入力ボックスから消えてしまう。これは、送信ボタンを押した後に第三者が画面を覗いた際、その人がどのユーザーであるか分からないため匿名性が保持されているといえるが、反面認証が通らなかった際、ユーザー名あるいはパスワードをタイプミスしていたのか、もしくはそもそも入力を忘れていたのかが分からないため、利用しづらくなっていると考える。また、パスワードを入力した際、そのまま文字列が表示されているため、パスワードの欄は伏字となる仕様にすべきだといえる。その他、認証後ハイパーリンクをふまずに、メインページに遷移できた方が、サイトの利便性が高まると思う。次に、検索画面となるメインページの表記である。これに関して、検索条件を指定する際、ドロップダウンリストを採用することで、利用者がジャンル名や検索範囲を入力間違いし、正しく検索できなくなることを防げたと考える。今回、現在地の取得時に利用した geojs はかなり大まかな緯度・経度情報しか取得できないため、Geolocation API を使用するなど別手段から現在地取得の精度をあげるべきだといえる。検索条件を選択、送信ボタンを押して、検索結果が表示された後だが、ドロップダウンリストの表示が初期値の「居酒屋」「300m」に戻ってしまう。これでは、サイト利用者は自分がどんな条件で検索したかが分かりにくくなってしまうため、表示に改良する余地があるといえる。また、今回いいね！機能を搭載したため、それと python のソート関数を活用し、いいね数！が多い順に検索結果が表示すれば、サイト利用者にとって分かりやすいサイトとなると考える。最後に、いいね数の表示についてである。ページ下部のいいね！ボタンを押すと、自分が選んだ店舗のいいね数を増やすことができる。しかし、このままでは何度も押すことで、一人一票ではなく複数票投票できることになってしまう。また、当初は html にて、`input type="checkbox"` とすることで一度に複数店舗のいいね！が出来る仕様を試みたが、うまくデータベースに反映することができなかったため、今後の反省点の一つに挙げられる。そして、いいね数は、ページをリロードまたは同条件にて再検索をしない限り更新されないため、利用者が無事いいね！が出来たかが不鮮明となってしまっている。これに関しては、データベースのみではなく、ページ内容も同時に更新されるようプログラムを構成すべきだと考える。その他、いいね！ボタンがページ下部にあるため、このままではページを下までスクロールしない限りいいね！ボタンの存在が分かりづらい、いいね！をするにはわざわざスクロールをしなければならないといった不便さが残ってしまっている。さらに、いいね！のさらなる機能として、一度誤っていいね！をしてしまった際、取り消しをする機能が搭載されていないため、より高度なサイトを構築するためには、サイトの目的に沿ったいいね！機能を追求する必要があるといえる。その他、GUI の工夫としていいね！のマークを親指を立てた good マークのアイコンにするといった点が挙げられる。

最後に、リファクタリングの余地があるかについて考察する。今回、ページの装飾として stylesheet.css という CSS ファイルを作成し、html の head 部にて

`<link rel="stylesheet" href="http://localhost:8000/cgi-bin/R02/stylesheet.css">` と指定したが、うまく反映されなかったため、`<h1>` に表示の指定をするなど、、具体的にプログラム

の中に書き込んでしまった。この部分に関して、CSS ファイルを反映できれば、ページの体裁を変える場合にいちいちソースプログラムを読んでいく必要がなくなるため、改善できると思う。また、search.py の正規表現のマッチングを行っているメソッド collect_info() だが、殆どの場合はじめに ’変数 = r’<タグ>(.*?)</タグ>’ と変数を置き、re.findall(変数, API から取得した xml)’ という流れでマッチングをしている。ここの部分は、何らかのメソッドを立てることで、ソースプログラムをリファクタリングできると考える。また、 create_db.py だが、データベースの作成、カーソルの取得、テーブル作成までは作成した二つのデータベースに共通した処理であるため、こちらも同様にメソッドを立てることでリファクタリングができると思う。

6 まとめ

今回の CGI プログラム作成の課題を通して、Web アプリケーションとはどのようなものかということを理解し、加えて Web API の仕様やデータベースの操作方法をマスターした。また、正規表現によるさまざまなマッチング方法を習得した。今後の課題として、ユーザーのログイン機能を活かし、そのユーザーが調べたお店の検索履歴を残したり、いいね！機能のさらなる向上、さらに見やすいサイトとなるよう CSS を工夫することが挙げられる。

7 付録: ソースプログラム

今回使用・作成したソースプログラムと CSS ファイルの中身を掲載する。

Listing 8: server.py

```
import http.server

server_address = ("", 8000)
handler_class = http.server.CGIHTTPRequestHandler

server = http.server.HTTPServer(server_address, handler_class)
server.serve_forever()
```

Listing 9: 20K1105-r02.py

```
# 20K1105 伊藤葵
# レポート課題 R02
# 実行方法: http://localhost:8000/cgi-bin/R02/20K1105-r02.py
# ログイン画面
# ユーザー名: イトウアオイ
# パスワード: Hkkx3P6SB3
# ------------------------------------
import sys, io
import cgi, cgitb
import sqlite3

connection = sqlite3.connect("users.sqlite3")
cur = connection.cursor()

```

```
cgitb.enable()
sys.stdout = io.TextIOWrapper(sys.stdout.buffer,
                              encoding='utf-8')
form=cgi.FieldStorage()
# ------------------------------------
v_user = form.getvalue('user', '')
v_password = form.getvalue('password', '')
# ------------------------------------
template = """
<html><head>
  <meta charset="utf-8">
  <title> 最終課題 (20K1105) </title>
  <link rel="stylesheet" href="http://localhost:8000/cgi-bin/R02/
      stylesheet.css">
</head>
<body>
  <form method="POST" action="/cgi-bin/R02/20K1105-r02.py">
  <h1 style="color: #8ed0f7; background-color: #294188; font-family:
      fantasy; ; text-align: center★★・・・・・—————————・・・・・・—
      —・・・・・・—————————・・・・・・★★"><br>今日の外食検索サイト
  <br>★★・・・・・—————————・・・・・・—————————・・・・・・—————————・・・・・・★★
  </h1>
  <h2 style="color: #8ed0f7; background-color: #294188; font-family:
      fantasy; ; text-align: centerログイン
      "></h2>
  <pユーザー名とパスワードを入力してください。></p>
  <pユーザー名>: <input type="text" name="user">
  <pパスワード>: <input type="text" name="password">
  <p> <input type="submit"></p>
  {result}
</body>
<html>
"""
cur.execute(f"␣SELECT␣name␣FROM␣users␣WHERE␣name='{v_user}'␣")
user = cur.fetchall()
if user == []:# ユーザーが存在しない場合
    result = "<pそのユーザーは存在しません。></p>"
if user != []:
    cur.execute(f"␣SELECT␣password␣FROM␣users␣WHERE␣name='{v_user}'␣")
    pw = cur.fetchall()
    if v_password == pw[0][0]:
        result = "<p認証に成功しました。>␣<a␣href='/cgi-bin/R02/search.py進
            む'></a></p>"
    else:
        result = "<pパスワードが違います。></p>"
if v_user == v_password == '':
    result = ""


text = template.format(result = result)
print("Content-type:␣text/html\n")
```

```
print(text)
```

Listing 10: createdb.py

```
# 20K1105 伊藤葵
# レポート課題 R02
# データベース作成
# 追加・検索処理は db_add.py にて行います。
# -----------------------------------
import sqlite3
connection = sqlite3.connect("users.sqlite3")
cur = connection.cursor()

# ユーザーとそのパスワードのデータベースを作成する
cur.execute("CREATE␣TABLE␣users(name,␣password)")
def create_users(cur, name, password):
    cur.execute(f'INSERT␣INTO␣users␣VALUES("{name}",␣"{password}")')
# 名前とパスワード一覧（今回は乱数を用いて 100 件生成）
names = ['モリモトリク', 'クロサマナ', 'イトウアオイ', 'サカイユウダイ', 'コバヤシリョウ', 'マスカタイサーク', 'オガワハラシンヤ', 'シノハラサキ', 'マツカワミナト', 'トミバリワカナ', 'ヤマモトツネコ', 'ハナミズザカマモリ', 'コニシユキコ', 'ミヤガワミエコ', 'サカキラン', 'イノウエカオル', 'ヤマガタダニアオイ', 'テンノカワハルカ', 'キクフロレンティナ', 'タジマスミコ', 'ディアネット', 'コラリーヌ', 'レオノール', 'ランシーヌ', 'ジャニーヌ', 'ジャネット', 'サンドロン', 'ジャニエス', 'エルミニク', 'マガリーヌ', 'エロイネス', 'エリザボー', 'コラリーヌ', 'リリアーヌ', 'デライネス', 'カロル', 'エグラベル', 'アリアニック', 'アリアーヌ', 'カトリエル', 'エルベーレ', 'コッランコ', 'ルクウーゴ', 'ルチジェロ', 'ランベリオ', 'ルチジルド', 'ウゼビーノ', 'ディナテッロ', 'アベラード', 'ポリナルド', 'アルリエト', 'バルトビア', 'カールジョ', 'フィロルド', 'チェチェロ', 'エルマーロ', 'ルキオッレ', 'ドリアーノ', 'グアルフォ', 'ルフレンテ', 'カミラ', 'エルファニ', 'ストリーセ', 'ダクマリー', 'ルネリーデ', 'ユッテ', 'ギーゼルマ', 'ドーリーザ', 'ルトラーラ', 'ヘルミルダ', 'ルハイルゼ', 'イジドゥラ', 'ジークシア', 'ブリエラ', 'リーザンナ', 'ラッヘルタ', 'ルフリヤム', 'クダレーナ', 'フロレーア', 'ウルズザネ', 'ジャスター', 'ティクシス', 'デイモンド', 'ハーヴァン', 'ランダレル', 'ハーヴァー', 'ストフリー', 'ライアラン', 'アイザック', 'ジェラリー', 'クラレッド', 'シルヴァン', 'アルヴィン', 'トバイラス', 'パーシュア', 'メレデニス', 'クラレッグ', 'デルバジル', 'アンドニー', 'ヴァレッド']
passwords = ['5Vz140DtTl', '9ZxzUXDgQ7', 'Hkkx3P6SB3', 'u36a8Ecg4w', '0
    RM39G4Rq1', 'eTC1o1fOcE', 'bRXNCN0qfm', 'G89S0TH1He', 'lV59U040Sg', '6
    T4294b30s', 'Wr05QNlT6Y', 'QsAlVT50hc', 'kx2J4X9Kp6', '0y5bD4rr6u', '
    fCIwO0A4S9', 'lc8lIlh5w0', 'TgIkw8zi6F', '0FsmT2C2KY', '7W0Q68u0Wl', '
    24a85YygHc', '6csf3A5V7O', 'B06TQa49S0', 'rzZZ3FBkEK', 'R4UHP4JUAH', '
    70IMMg2d6R', 'vc0Ay0pVv2', '0m2SKzoPsa', 'aZBHEghiRI', 'h640F7wsAY', '
    ZI0280VW5g', 'pSO06VrqL6', 'hhNyp6fpO2', '2r03994F0p', '00d775B6oK', '
    6myLdO2oEq', '71LLhZaAy0', 'kpDF8ZUmWO', 'Sx2f807OFm', 'xcPOJWStu5', '
    omQ3BP8tR8', 'BcEryV04e8', 'itM8QjhTbR', 'wxlo6K4s4T', 'v0Ntz0lQqR', '
    KYswjD3uJ5', 'm7kLqHj9Kr', '2tBw0ppF17', 'zIhovgDsJI', 'r4d83f000J', '
    LcwmIFub6f', '0Slmo4R89F', 'PSpH1AnCDJ', 'JisxLCu90m', 'gJihFmKGgo', '
    79Wo4dNVmu', '5SAwIGaqw6', 'KSK3XpLDCF', 'xFXm1AnYW2', 'NMFkGqQQ3s', '
    EyLd0JiIMC', 'mzi00n2fb7', 'qayD00T96K', '3DrcRks39E', 'F8f10atHFq', '
    4RGKneD7n0', 'mL8iX428r0', 'Q1mLcl7DW2', 'KO00CuLr0G', 'hj927ETWBA', '
    5wn3SUc9bm', 'R2qa4FYwtN', 'X0Zol9BFXq', '0SJ3T19Ru6', 'YQTaGrO0uh', '
    Db80JeZp39', 'dVBvP51VEB', '6y3nbXX9S6', 'TiDJ6V0D30', 'B6F271puS8', '
```

```
    C6p1a1w42b', '00c046yAL0', '5k5j98CRG4', 'O2xOSG9M13', 'bReQdtoj4w', '
    0OTRSWfxPj', 'rn4aXLnBtD', 'mwk07019UE', '9F975DqtuR', '7UbTG1A5HA', '
    2x632pd3qK', '4lbPSaLy0d', 'F3Dx8xT4Wh', '3Oq0BBqjR8', '5xQOm80Ooj', '
    bh0itAg719', '6r80xE74vu', '04uS9XItk0', '0q8nj7Go7l', 'xouQn0O5BB', '
    CUtEUdtjF4']
for x in range(100):
    create_users(cur, names[x], passwords[x])
connection.commit()
connection.close()

# 店名といいねされた回数のデータベースを作成する
connection = sqlite3.connect("shops.sqlite3")
cur = connection.cursor()
cur.execute("CREATE␣TABLE␣shops(shop,␣nice)")
connection.commit()
connection.close()
```

Listing 11: search.py

```
# 20K1105 伊藤葵
# レポート課題 R02
# メインページ
# ホットペッパーの API を用い、飲食店を検索します
# いいね数の表示は、ページのリロードボタンを押してから反映されます。
# -------------------------------------
import sys, io
import requests
import cgi, cgitb
import re
import sqlite3

cgitb.enable()
sys.stdout = io.TextIOWrapper(sys.stdout.buffer,
                              encoding='utf-8')
form=cgi.FieldStorage()
# -------------------------------------
v_genre = form.getvalue('genre', '')
v_range = form.getvalue('range', '')
v_nice = form.getvalue('nice', '')
# -------------------------------------
def search_current_location():
    geo_request_url = 'https://get.geojs.io/v1/ip/geo.json'
    data = requests.get(geo_request_url).json()
    latitude = data['latitude']
    longitude = data['longitude']
    return latitude, longitude

def collect_info(xml):
    connection = sqlite3.connect("shops.sqlite3")
    cur = connection.cursor()
```

```
32      name = r'</id><name>(.*?)</name>'
33      name_list = re.findall(name, xml)
34      station = r'<station_name>(.*?)</station_name>'
35      station_list = re.findall(station, xml)
36      access = r'<access>(.*?)</access>'
37      access_list = re.findall(access, xml)
38      open_time = r'<open>(.*?)</open>'
39      open_list = re.findall(open_time, xml)
40      close = r'<close>(.*?)</close>'
41      close_list = re.findall(close, xml)
42      wifi = r'<wifi>(.*?)</wifi>'
43      wifi_list = re.findall(wifi, xml)
44      price = r'</name><average>(.*?)</average>'
45      price_list = re.findall(price, xml)
46      shops = []
47      for x in range(len(name_list)):
48          cur.execute(f"␣SELECT␣nice␣FROM␣shops␣WHERE␣shop='{name_list[x]}'␣
                ")
49          point = cur.fetchall()
50          if point == []:
51              data = [name_list[x], station_list[x], access_list[x],
52                      open_list[x], close_list[x], wifi_list[x],
53                      price_list[x], 0]
54          else:
55              data = [name_list[x], station_list[x], access_list[x],
56                     open_list[x], close_list[x], wifi_list[x],
57                     price_list[x], point[0][0]]
58          shops.append(data)
59      return shops
60
61  def shop_search(v_g, la, lo, v_r):
62      response = requests.get(
63          'http://webservice.recruit.co.jp/hotpepper/gourmet/v1/',
64          params={
65          'key': 'ホットペッパー登録時に取得したキー',
66          'genre': v_g,
67          'lat': la,
68          'lng': lo,
69          'range': v_r}
70          )
71      result = response.text
72      shops = collect_info(result)
73      return shops
74
75  def make_ans(shops):
76      ans = ""
77      for s in range(len(shops)):
78          ans += "<p>"
79          ans += f"{s店目+1}</p>"
80          ans += "<ul>"
```

```
81          ans += f"<li店名>:␣{shops[s][0]}</li>"
82          ans += f"<li最寄り駅>:␣{shops[s][1]}</li>"
83          ans += f"<li行き方>:␣{shops[s][2]}</li>"
84          ans += f"<li開店・閉店時間>:␣{shops[s][3]}~{shops[s][4]}</li>"
85          ans += f"<li>wi-fi␣の有無:␣{shops[s][5]}</li>"
86          ans += f"<li平均予算>:␣{shops[s][6]}</li>"
87          ans += f"<li「いいね！」された回数>:␣{shops[s][7]}</li>"
88          ans += "</ul>"
89          ans += "</p>"
90      favorite = "<pお店に「いいね！」をつけませんか？><br>"
91      for s in range(len(shops)):
92          favorite += f"<p>{s店目+1}"
93          favorite += f"<button␣type='submit'␣name='nice'␣value='{shops[sいいね！][0]}'></button><br>"
94      favorite += "</p>"
95      return ans+favorite
96
97  def add_nice(v_nice):
98      connection = sqlite3.connect("shops.sqlite3")
99      cur = connection.cursor()
100     cur.execute(f"␣SELECT␣nice␣FROM␣shops␣WHERE␣shop='{v_nice}'␣")
101     point = cur.fetchall()
102     if point == []:
103         cur.execute(f'INSERT␣INTO␣shops␣VALUES("{v_nice}",␣"1")')
104         connection.commit()
105     else:
106         p = int(point[0][0])
107         new_p = p+1
108         cur.execute(f"UPDATE␣shops␣set␣nice='{new_p}'␣where␣shop='{v_nice}'")
109         connection.commit()
110     connection.close()
111
112
113 # -------------------------------------
114 template = """
115 <html><head>
116   <meta charset="utf-8">
117   <title> 最終課題 (20K1105) </title>
118 </head>
119 <body>
120   <form method="POST" action="/cgi-bin/R02/search.py">
121   <h1 style="color: #8ed0f7; background-color: #294188; font-family: fantasy; ; text-align: center★★・・・・・—▬▬▬—・・・・・・・・・—▬—・・・・・・・・—▬▬▬—・・・・・・・★★"><br>今日の外食検索サイト
122   <br>★★・・・・・—▬▬▬—・・・・・・・・・—▬—・・・・・・・・—▬▬▬—・・・・・・★★
123   </h1>
124   <h2 style="color: #8ed0f7; background-color: #294188; font-family: fantasy; ; text-align: center今日はどこに食べに行きますか？
      "></h2>
125   <p以下の項目を埋めることで、今日の貴方にお勧めのお店を検索します！></p>
```

```
126   <pジャンル>:
127   <select name="genre">
128     <option value="G001居酒屋"></option>
129     <option value="G002ダイニングバー・バル"></option>
130     <option value="G003創作料理"></option>
131     <option value="G004和食"></option>
132     <option value="G005洋食"></option>
133     <option value="G006イタリアン・フレンチ"></option>
134     <option value="G007中華"></option>
135     <option value="G008焼肉・ホルモン"></option>
136     <option value="G017韓国料理"></option>
137     <option value="G009アジア・エスニック料理"></option>
138     <option value="G010各国料理"></option>
139     <option value="G011カラオケ・パーティ"></option>
140     <option value="G012バー・カクテル"></option>
141     <option value="G013ラーメン"></option>
142     <option value="G0016お好み焼き・もんじゃ"></option>
143     <option value="G014カフェ・スイーツ"></option>
144     <option value="G015その他グルメ"></option>
145   </select></p>
146   <p検索範囲>:
147   <select name="range">
148     <option value="1">300</option>
149     <option value="2">500</option>
150     <option value="3">1000</option>
151     <option value="4">2000</option>
152     <option value="5">3000</option>
153   </select>m</p>
154   <p> <input type="submit"></p>
155
156   Powered by <a href="http://webservice.recruit.co.jpホットペッパー/"> サービス Web</a>
157   <h2 style="color: #8ed0f7; background-color: #294188; font-family: fantasy; ; text-align: center検索結果
          "></h2>
158   {ans}
159 </body>
160 <html>
161 """
162 la, lo = search_current_location() # 現在地の緯度・経度を取得
163 shops = shop_search(v_genre, la, lo, v_range)
164 add_nice(v_nice)
165 ans = make_ans(shops)
166 text = template.format(ans = ans)
167 print("Content-type:␣text/html\n")
168 print(text)
```

Listing 12: stylesheet.css

```
1 H1, H2{ color: #8ed0f7; background-color: #294188 }
2 H1, H2{ font-family: fantasy }
```

```
3 H1{ text-align: center }
4 H2{ text-align: left }
```