Dominik Krystkowiak

BD2-Projekt Sprawozdanie, Część 5

**Temat: System rezerwacji – Fitness Klub – język Python + Flask**

Część 5: Architektura całości. Realizacja wybranych funkcji w środowisku MongoDB.

1. Funkcje realizowane przez MongoDB
   - obsługa logowania użytkowników – w bazie MongoDB przechowywane są loginy i hasła klientów (przy tworzeniu nowego rekordu w bazie MSSQL tworzony jest login – imię_nazwisko, hasło – losowy ciąg liter i liczb, pobierane jest także stworzone ID oraz typ użytkownika ustawiany jest jako klient).

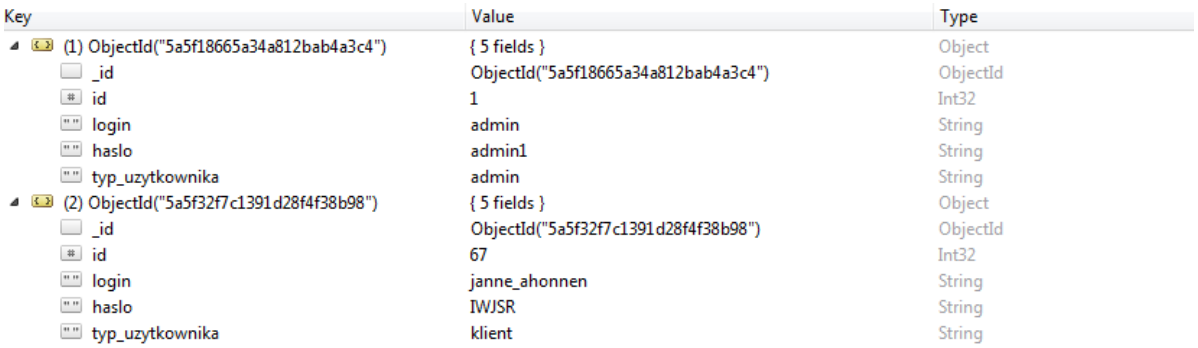
```python
from pymongo import MongoClient
import random
import string

client = MongoClient()
db = client.uzytkownicy
kolekcja = db.loginy

def dodaj_klienta_mongo(ID, imie, nazwisko):
    if len(imie)>2 and len(nazwisko)>2:
        haslo = ''.join(random.choices(string.ascii_uppercase + string.digits, k=5))
        uzytkownik = {"id": ID, "login": imie.lower() + "_" + nazwisko.lower(),
                  "haslo": haslo, "typ_uzytkownika": "klient"}
        kolekcja.insert(uzytkownik)
def usun_klienta_mongo(ID):
    try:
        kolekcja.delete_one({"id":ID})
    except:
        raise NameError("Nie ma takiego rekordu w bazie")

def zwroc_uzytkownika(login):
    try:
        uzytkownik = kolekcja.find_one({"login": login})
        return uzytkownik
    except:
        raise NameError("Nie ma takiego uzytkownika")
```

   - Wygląd bazy w MongoDB



| Key | Value | Type |
|---|---|---|
| (1) ObjectId("5a5f18665a34a812bab4a3c4") | { 5 fields } | Object |
| _id | ObjectId("5a5f18665a34a812bab4a3c4") | ObjectId |
| id | 1 | Int32 |
| login | admin | String |
| haslo | admin1 | String |
| typ_uzytkownika | admin | String |
| (2) ObjectId("5a5f32f7c1391d28f4f38b98") | { 5 fields } | Object |
| _id | ObjectId("5a5f32f7c1391d28f4f38b98") | ObjectId |
| id | 67 | Int32 |
| login | janne_ahonnen | String |
| haslo | IWJSR | String |
| typ_uzytkownika | klient | String |

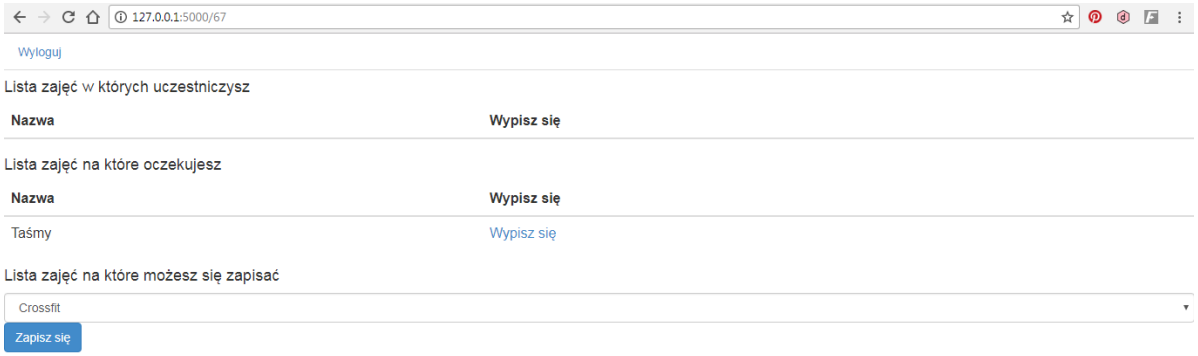
- Do bazy został dodany administrator, który jako jedyny użytkownik ma dostęp do przeglądania wszystkich tabel (poniżej kod programu umożliwiający logowanie i obsługa sesji w tym rozróżnianie klientów oraz administratora)

```
@app.route('/', methods=['GET','POST'])
def homepage():
    if request.method == 'POST':
        try:
            print(app)
            session.pop('admin', None)
            session.pop('user', None)
            uzytkownik = zwroc_uzytkownika(request.form['login'])
            if request.form['haslo'] == uzytkownik["haslo"] and "admin"==uzytkownik["typ_uzytkownika"]:
                session['admin'] = uzytkownik["id"]
                return redirect(url_for('wszyscy_klienci'))
            elif request.form['haslo'] == uzytkownik['haslo'] and "klient" == uzytkownik["typ_uzytkownika"]:
                session['user'] = uzytkownik["id"]
                return redirect(url_for('klient_widok', ID_Klienta=uzytkownik["id"]))
            else:
                flash(str("Bledne haslo"))
                return render_template('login.html')
        except:
            flash(str("Nie ma takiego uzytkownika"))
            return render_template('login.html')
    return render_template('login.html')
@app.route('/wyloguj')
def wyloguj():
    session.pop('admin', None)
    flash(str("Wylogowano. Zaloguj się ponownie"))
    return redirect(url_for('homepage'))
@app.before_request
def sesja():
    g.admin = None
    g.user = None
    if 'admin' in session:
        g.admin = session['admin']
    elif 'user' in session:
        g.user = session['user']
```

- Strony generowane dla administratora zostały zablokowane przez sprawdzenie sesji (warunek „if g.admin”) np.:

```
@app.route('/sprzet/dodaj_sprzet', methods=['POST'])
def dodaj_sprzet():
    if g.admin:
        if request.method == 'POST':
            try:
                Sprzet.dodaj(request.form['Nazwa'], request.form['Ilosc'],
                             request.form['Sala'])
                return redirect(url_for('caly_sprzet'))
            except NameError as error:
                flash(str(error))
                return redirect(url_for('dodaj_sprzet_strona'))
    return redirect(url_for('homepage'))
```

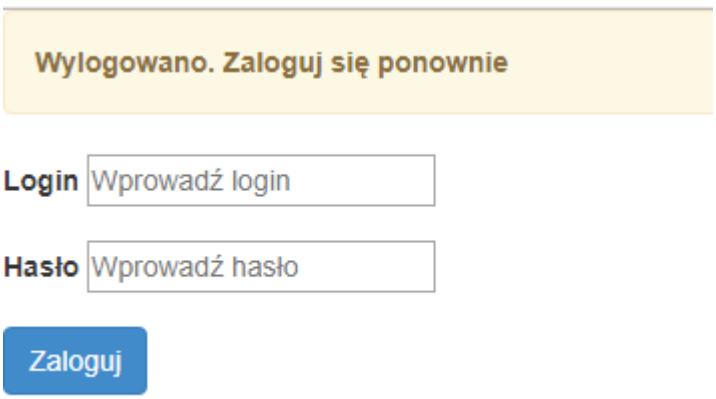
- Zalogowany klient ma dostęp wyłącznie do swoich danych




- W kodzie sprawdzany jest numer sesji (ID_Klienta) czy jest taki sam jak strona, którą chce odwiedzić klient (uniemożliwia sprawdzenie danych innego klienta)

```
@app.route('/<int:ID_Klienta>', methods=['GET','POST'])
def klient_widok(ID_Klienta):
    if g.user==ID_Klienta:
        Zajecia_ucz = klient_zajecia(ID_Klienta)
        Lista = klient_lista_oczekujaca(ID_Klienta)
        Wszystkie_zajecia = Zajecia.odczyt()
        if request.method == 'GET':
            try:
                return render_template('widok_klient.html', zajecia=Zajecia_ucz, lista=Lista, wszystkie=Wszystkie_zajecia)
            except NameError as error:
                flash(str(error))
                return render_template('widok_klient.html', zajecia=Zajecia_ucz, lista=Lista, wszystkie=Wszystkie_zajecia)
        if request.method == 'POST':
            try:
                session = create_session(bind=engine)
                q = session.query(Zajecia).filter(Zajecia.Nazwa == request.form["Zaj"])
                rekord = q.one()
                dodaj_klienta_do_zajec(ID_Klienta, rekord.ID_Zajecia)
                return render_template('widok_klient.html', zajecia=Zajecia_ucz, lista=Lista, wszystkie=Wszystkie_zajecia)
            except pyodbc.Error as ex:
                sqlstate = ex.args[0]
                if sqlstate == '42000':
                    mes = ex.args[1].split(']')
                    mes = mes[4].split('(')
                    flash(str(NameError(mes[0])))
                    return render_template('widok_klient.html', zajecia=Zajecia_ucz, lista=Lista,
                                           wszystkie=Wszystkie_zajecia)
    return redirect(url_for('homepage'))
```

- Wygląd ekranu logowania

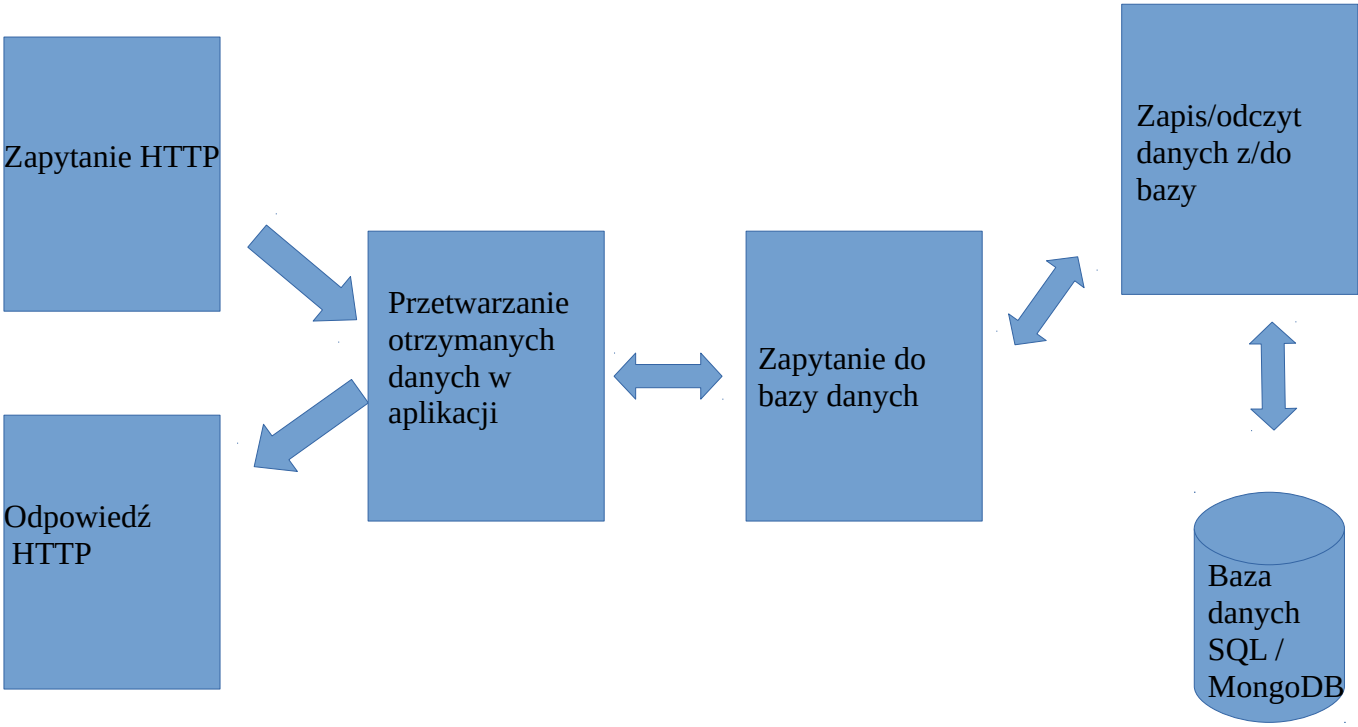


2. Architektura i schemat logiki przetwarzania całości aplikacji

   Aplikacja opiera się na obsłudze zapytań HTTP (metody POST i GET). Zapytania są przetwarzane w kodzie aplikacji, a następnie wysyłane są odpowiednie zapytania do bazy SQL/MongoDB.

   Strona internetowa generowana jest za pomocą micro frameworku Flask, baza SQL obsługiwana jest za pomocą ORM SQL Alchemy, a baza MongoDB za pomocą PyMongo.

   Schemat przetwarzania:




3. Podział funkcjonalności na część implementowaną na serwerze bazy danych i cześć zawartą w kodzie:

   Część implementowana na serwerze:

   1. Triggery – umożliwiające sprawdzenie poprawności danych lub informowanie administratora poprzez wiadomość email o nowym kliencie przy dodawaniu lub modyfikację zawartości kilku tabel podczas usuwania rekordów z tabeli.

   2. Walidacja danych za pomocą warunków spójności.

   3. Procedury – zwracające rekordy oparte na danych z różnych tabel (inner join) oraz usuwające rekordy, gdy występuje referencja klucza obcego w innych tabelach.

   4. Generowanie raportów

Część implementowana w kodzie aplikacji:

1. Konwersja danych otrzymanych z bazy oraz z zapytań HTTP.

2. Prezentacja danych w sposób czytelny dla użytkownika za pomocą generowanej strony internetowej.

3. Rozdzielenie aplikacji na widoki prezentowane administratorom i klientom.

4. Wysyłanie zapytań (dodawanie, edycja, usuwanie) do bazy SQL oraz MongoDB.

5. Obsługa błędów „wyrzucanych” przez bazę danych oraz przystępna prezentacja błędów w sposób czytelny dla końcowego użytkownika w formie np. podpowiedzi.

4. Przedyskutować korzyści wynikające z elastyczności i efektywności systemu uzyskane przez podział oprogramowania na część serwerową i część aplikacyjną (klienta).

   - Część aplikacyjna umożliwia podział użytkowników na odpowiednie kategorie i generowanie odpowiedniej treści dla nich (dane są chronione przed nieautoryzowanym dostępem). W łatwy sposób można dodać kolejne grupy użytkowników i prezentować im odpowiednie dane np. grupa trenerów.

   - Odpowiednie przetwarzanie danych oraz ich przygotowanie do wysłania do bazy danych odciąża pracę serwera (szybsza praca aplikacji).

   - Serwer bazy danych umożliwia nadanie odpowiednich uprawnień (odczyt, modyfikacja, dostęp jako administrator) do zarządzanej bazy danych.