Python średnio zaawansowany

Dzień 19

Blok nr 5:

Aplikacja webowa

AGENDA

- Ciasteczka po raz drugi
- Sesje
- Wiadomości błyskowe ;-)
- Flask-Login
- Zadanie projektowe

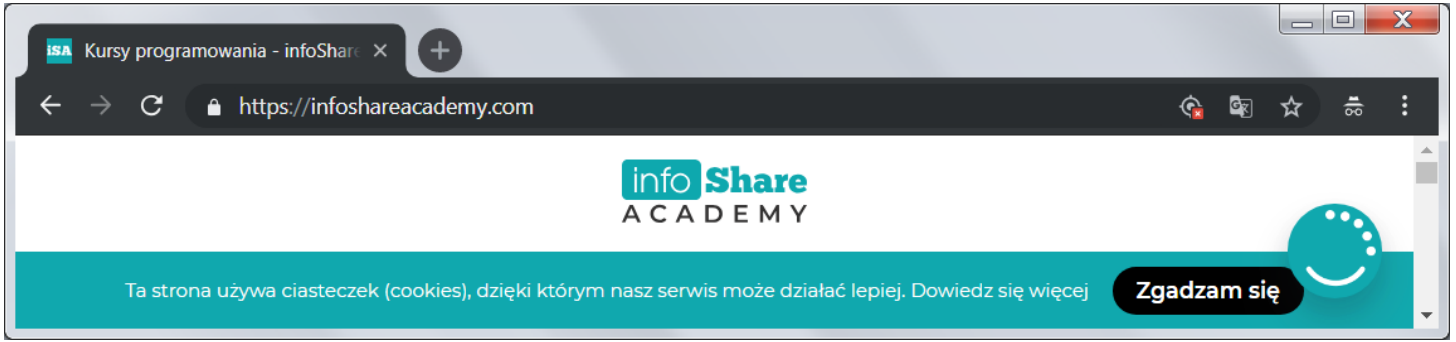
Ciasteczka po raz drugi

HTTP cookie (ostatni slajd z dnia #2)

HTTP cookie (web cookie, browser cookie) zawiera pewną niewielką część danych, którą serwer wysyła do przeglądarki użytkownika. Przeglądarka może ją zapisać i przesłać przy kolejnym zapytaniu do tego samego serwera np. zapamiętanie czy użytkownik jest już zalogowany, koszyk w sklepie internetowym.

Cookies

Protokół HTTP jest protokołem bezstanowym, więc potrzebujemy mechanizmu do śledzenia aktywności użytkowników.



Cookies nie są szczególnie bezpieczne, bo nie są odporne na modyfikacje ze strony użytkownika (F12!).

HTTP cookie

Scenariusz:

1. Cookie jest tworzone przez aplikację webową, przesyłane w odpowiedzi do przeglądarki
2. Przeglądarka umieszcza cookie w nagłówku żądań HTTP aby aplikacja mogła rozróżniać klientów

Praktyka: not_secure_cookies

Więcej informacji: http://wszystkoociasteczkach.pl/po-co-sa-ciasteczka/

Sesje

Sesje (sessions)

Informacje o stanie sesji przekazywane są w postaci zaszyfrowanego cookie.

Szyfrowanie odbywa się na podstawie sekretnego klucza aplikacji:

```
app.config['SECRET_KEY'] = ’infosharepythonsredniozaawansowany2019’
```

Flask ułatwia korzystanie z sesji i użycie sesji jest odrobinę łatwiejsze od korzystania z cookies.

Praktyka: secure sessions

Flash messages

Flash messages

Flash to rozwiązanie na przekazywanie wiadomości między kolejnymi (tylko i wyłącznie) wywoływanymi widokami w jednej sesji.

Scenariusz:

1. nadawca wywołuje metodę flash()
2. odbiorca (najczęściej w szablonie) wywołuje metodę get_flashed_messages() i przetwarza zwrócone elementy

Istnieje możliwość filtrowania wiadomości

Więcej informacji: http://flask.pocoo.org/docs/1.0/patterns/flashing/

Flask-Login

Flask-Login

Flask-Login - moduł służący do realizacji funkcjonalności związanych z zarządzaniem sesjami użytkowników aplikacji zrealizowanych we Flasku:

- logowanie, wylogowanie
- kontrola dostępu do zasobów dla uwierzytelnionych użytkowników
- realizacja funkcjonalności 'zapamiętaj mnie’

Strona modułu: https://flask-login.readthedocs.io/en/latest/

Flask-Login

Koncepcja otwartej budowy Flask-Login zakłada dowolność w zakresie mechanizmów tworzenia kont, kontroli jakości haseł, dostarczanie zasobu do przechowywania sekretów.

Innymi słowy, w zależności od tworzonego rozwiązania możemy korzystać z różnych komponentów dostarczających te dane lub usługi.

W związku z tym, wspomniane komponenty muszą realizować pewne własności (properties) i metody aby mogły być użyte z Flask-Login.

Flask-Login

Rozpoczęcie pracy:

```
from flask_login import LoginManager
```

Tak jak w przypadku obiektu reprezentującego bazę danych czy panel administracyjny, obiekt `login_manager` powinien zostać zarejestrowany w aplikacji:

```
# utworzenie login managera i rejestracja w aplikacji

login_manager = LoginManager(app)
```

Flask-Login

Flask-Login może pracować w oparciu model reprezentujący konto użytkownika rozszerzony o wspomniane własności i metody:

- is_authenticated: właściwość, zwraca True gdy użytkownik został uwierzytelniony, False w przeciwnym razie,
- is_active: właściwość, zwraca True gdy użytkownik jest aktywny, False w przeciwnym razie,
- is_anonymous: właściwość, zwraca False dla zalogowanych użytkowników, wartość True dla anonimowego użytkownika,
- get_id(): metoda zwracająca unikalny identifikator dla użytkownika. Typ: string

Flask-Login

Możesz samemu zaimplementować te 4 metody (właściwość to też metoda), ale Flask-Login dostarcza klasę UserMixin która realizuje te metody.

```
from flask_login import UserMixin

class User(UserMixin, db.Model):
```

Flask-Login

Informacja o ID zalogowanego użytkownika przechowywana jest w sesji.

ID jest tworzone gdy użytkownik zaloguje się i niszczone gdy użytkownik wyloguje się.

ID przesyłane jest przez przeglądarkę w nagłówku każdego żądania HTTP(S).

Jeśli widok jest udekorowany przez '`@login_required`', login manager zweryfikuje, czy przeglądarka posługuje się ważnym (w znaczeniu legalnym) ID.

Flask-Login

W związku z tym, musimy wyposażyć Flask-Login w metodę na sprawdzenie czy ID odpowiada użytkownikowi.

Taka metoda musi być udekorowana `@login_manager.user_loader`

oraz przekierować niezalogowanego użytkownika na odpowiedni widok służący do logowania. Realizuje się to w następujący sposób:

```
@login.user_loader
def load_user(id):
    return User.query.get(int(id))
```

Logowanie użytkownika

Logowanie realizowane jest przez metodę login_user

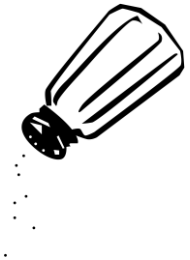
```
from flask_login import login_user

@app.route('/login', methods=['GET', 'POST'])
def login():
    if weryfikacja(haslo=„sekretna_wartość_z_bazy_danych”):

        # zalogowanie
        login_user(user)

        return redirect(url_for('index'))
```

Jak przechowywać hasło?



Nie trzymaj haseł w bazie w jawnej postaci; dobrze jak są to skróty z tych haseł (hash), a najlepiej jak są dodatkowo posolone (salted).

Wylogowanie użytkownika

Wylogowanie realizowane jest przez metodę logout_user

```
from flask_login import logout_user


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

Wymuszenie logowania

```
# wskazanie widoku służącego do logowania
login.login_view = 'login'

# udekorowanie widoków dostępnych dla zalogowanych użytkowników
from flask_login import login_required

@app.route('/sekret')
@login_required
def sekret():
```

Praktyka: web_app_login

Zadanie projektowe

Zadanie projektowe

Nasza aplikacja nie posiada funkcjonalności zakładania kont użytkowników w tabeli **Users.**

Cel:

- Podłącz Flask-Admin aby było możliwe zakładanie kont
- Dostęp do Flask-Admin powinien być również chroniony

Na początku jedyny użytkownik z dostępem do widoków chronionych to:

Login: Jan

Hasło: Nowak

Dzięki!