

Warszawa, 23.06.2021

Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych

Programowanie aplikacji internetowych (PAINT)

Projekt: Serwis sieciowy

Dokumentacja końcowa

Zespół:
Adamiuk Zuzanna 300444
Krakowiak Aleksandra 290292
Rancew Joanna 300465

Spis treści

Cel projektu.....	3
Projekt rozwiązania.....	3
Realizacja projektu.....	3
Wygląd aplikacji	4
Dokumentacja wstępna a dokumentacja końcowa	18
Instrukcja kompilacji, uruchamiania i korzystania	19
Współpraca podczas realizacji projektu.....	20
Wnioski	20
Bibliografia	21

Cel projektu

Celem projektu było stworzenie serwisu sieciowego, który umożliwia użytkownikom pisanie i przeglądanie blogów (wpisów). Aplikacja obejmuje podstawowe funkcjonalności potrzebne do zarządzania wpisami.

Projekt rozwiązania

Dostęp dla użytkownika

Każda osoba mająca dostęp do przeglądarki może skorzystać ze stworzonego serwisu sieciowego. Użytkownik nieposiadający konta (brak rejestracji) ma możliwość przeglądu trzech ostatnich dodanych wpisów. Natomiast osoba posiadająca konto w serwisie ma możliwość zalogowania się (poprzez podanie loginu i hasła) na swój panel, gdzie może tworzyć nowe, edytować i usuwać wpisy. Posiada również uprawnienia do przeglądania wszystkich blogów innych użytkowników. Istnieje również opcja założenia nowego konta.

Bezpieczeństwo

W celu zapewnienia bezpieczeństwa dostępu do konta użytkowników, podczas transmisji oraz w bazie danych przechowującej informacje o użytkownikach hasła są zaszyfrowane przy pomocy hash. Porównywanie hasła podczas logowania także nie zdradza jego pierwotnej treści.

Realizacja projektu

Wykorzystane technologie

Backend aplikacji został napisany w języku *Python* z wykorzystaniem:

- *Flask* - możliwość tworzenia aplikacji internetowych,
- *sqlite3* – moduł, który zapewnia interfejs SQL,
- *SQLAlchemy* – narzędzie wykorzystywane do tworzenia krótszych zapytań SQL,
- *flask_sqlalchemy* – rozszerzenie *Flask* przez dodanie *SQLAlchemy*,
- *bcrypt* - umożliwia stosowanie funkcji hashującej i porównującej hashe

Do przechowywania danych wykorzystano relacyjną bazę danych SQLite. Stworzono dwie tabele: *wpis* oraz *user*. *User* to tabela odpowiedzialna za przechowywanie loginu oraz zaszyfrowanego hasła użytkownika. *Wpis* to tabela posiadająca informację o danym wpisie (tytuł, data utworzenia, treść, login oraz id użytkownika, który stworzył wpis).

Frontend aplikacji został stworzony w języku HTML5 z arkuszem stylów CSS. Z zewnętrznych rozszerzeń wykorzystano zestaw czcionek ze strony Google Fonts.

Środowisko pracy

Kod źródłowy aplikacji został napisany w PyCharm, zaś do obsługi bazy danych wykorzystano *Driver SQLite*. Wszyscy wykonawcy korzystali z jednakowego środowiska. Spójność tworzonego kodu zapewniono przy systemie kontroli wersji Git.

Adres repozytorium projektu:

<https://gitlab-stud.elka.pw.edu.pl/jrancew/paint-300444-290292-300465>

Wygląd aplikacji

Podczas projektowania interfejsu graficznego aplikacji starano się zachować jak najprostsze rozwiązania, jednocześnie nie ograniczając funkcjonalności aplikacji. Aby zapewnić estetykę projektowanego serwisu, zdecydowano zastosować skromną paletę barw, która nie powinna budzić skrajnych emocji u użytkowników.



Rysunek 1. Paleta kolorów użyta do realizacji projektu

Na każdej podstronie zamieszczono pasek nawigacji, który dzięki parametrowi *position* ustawionemu na *fixed*, utrzymuje się stale na ekranie użytkownika - jest to powszechnie stosowane rozwiązanie ułatwiające nawigację po stronie. W arkuszu stylów określono również czcionkę (Roboto), która dzięki selektorowi ***, została zastosowana we wszystkich elementach na stronie.

Strona startowa

Po uruchomieniu aplikacji (app.py), użytkownik zostaje przekierowany na stronę startową, na której wyświetlane są 3 najnowsze posty zamieszczone na stronie (posty wszystkich użytkowników). Strona główna ma jedynie na celu zaprezentowanie możliwości platformy – sposobu wyświetlania notatek. W prawym górnym rogu strony (również z parametrem *position fixed*) znajduje się link przekierowujący do strony umożliwiającej logowanie. Aby wyróżnić link, został on zapisany pogrubioną czcionką z dodatkowym parametrem *hover*, który sprawia, że po najejaniu na niego myszą zmienia on kolor na pomarańczowy, zgodny z ustalonym standardem. Podobny zabieg wykorzystano przy projektowaniu paska nawigacji – dla parametru *hover* pojawia się pomarańczowe tło z zaokrąglonymi krawędziami oraz kolor czcionki zmienia się na kolor biały. Ponadto dodano parametr *transition*, dzięki któremu zapewniono płynne zmiany w wyglądzie wspomnianych linków.



Rysunek 2. Zrzut ekranu porównujący wygląd menu

```

<div class="menu">
    <div class="login-wrapper">
        <a class="logout" href="/login">Zaloguj</a>
    </div>
    <h1 id="menuLabel"><a href="/">BlogoSfera</a></h1>
    <div class="navbar">
        <a href="/all">Przeglądaj</a>
        <a href="{url_for('add')}">Dodaj</a>
        <a href="/mypage">Moje wpisy</a>
    </div>
</div>

```

Fragment kodu 1. Menu - kod HTML

```

#menuLabel {
    color: orange;
    font-size: 60px;
    top: 0;
    text-align: center;
    margin-bottom: 15px;
    margin-top: 10px;
}
.menu {
    overflow: hidden;
    display: block;
    width: 100%;
    top: 0;
    margin: 0 auto;
    position: fixed;
    background: white;
    padding-bottom: 10px;
}
.navbar {
    font-weight: bold;
    width: 800px;
    margin: 0 auto;
}

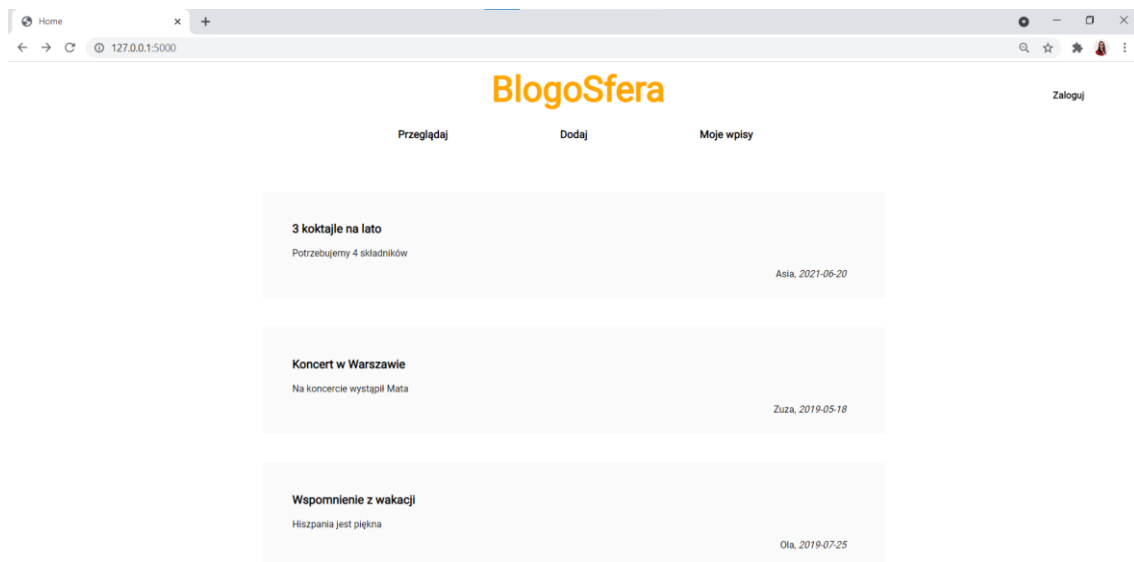
```

```

.navbar a {
    font-size: 18px;
    float: left;
    display: block;
    background: white;
    text-align: center;
    padding: 15px 55px 15px 55px;
    text-decoration: none;
    border-radius: 25px;
    transition: background-color 0.8;
    margin-left: 5%;
    margin-right: 5%;
}
.navbar a:hover {
    background: orange;
    color: white;
    border-radius: 25px;
}

```

Fragment kodu 2. Menu - arkusze stylów CSS



Rysunek 3. Zrzut ekranu strony startowej

W celu uzyskania dostępu do wszystkich opcji w menu, niezbędne jest założenie konta oraz zalogowanie. Na tej stronie nie widnieją przyciski do edycji oraz usuwania notatek, ponieważ, oczywiście, niezalogowany użytkownik nie może wykonywać tych operacji.

```
{% if data %}

    {% for wpis in data %}

        <div class="post">

            <div>

                <h3 id="title">{{wpis.tytul}}</h3>

            </div>

            <p> {{ wpis.tresc }} </p>

            <div class="signature-wrapper">

                <h4 id="signature">{{ wpis.autor }}, <i>{{ wpis.data }}</i></h4>

            </div>

        </div>

    {% endfor %}

{% else %}

    <h2> No posts on BlogoSfera </h2>

{% endif %}
```

Fragment kodu 4. Strona startowa - frontend.

```
@app.route('/')

def welcome():

    data = BlogSfera.query.order_by(desc(BlogSfera.id)).limit(3).all()

    return render_template('mainpage.html', data=data)
```

Fragment kodu 3. Strona startowa - backend

Logowanie

Panel logowania jest bardzo prosty - jest to formularz z polami do wpisania loginu oraz hasła. Poniżej przycisku „Zaloguj”, znajduje się subtelne przekierowanie do strony rejestracji, gdyby użytkownik nie posiadał konta. Zależności między wielkością elementów zostały określone jako wartości procentowe, dzięki czemu interfejs aplikacji dobrze się prezentuje niezależnie od wielkości okna przeglądarki.



Rysunek 4. Zrzut ekranu strony do logowania

```
@app.route('/login', methods=["GET", "POST"])
def login():
    if request.method == 'POST':
        login = request.form['login']
        password = request.form['password'] [...]
        user = User.query.filter_by(login=login).first() # szukanie loginu w bazie
        if user is not None: # warunek jak znajdziemy login
            hashed_password = user.password
            if bcrypt.checkpw(password.encode('utf8'), hashed_password):
                session['logged_in'] = True
                session['user_id'] = user.id
                session.permanent = True
                return redirect(url_for('my')) # login jest to przekierowanie na stronę
            else: [...]
        else: [...]
    return render_template('login.html')
```

Fragment kodu 5. Logowanie - backend

```

<div class="post">
  <h3 id="new-post-header">Logowanie</h3>
  <form action="" method = "POST">
    <div class="login-form">
      <label>Login</label>
      <input class="login-input" type="login" name="login">
      <label>Hasło</label>
      <input class="login-input" type="password" name="password">
    </div>
    <div class="button-holder">
      <button type="submit" class="btn-login" value = "Submit">Zaloguj</button>
    </div>
  </form>
  <div class="no-account">
    <p>Nie masz jeszcze konta?</p>
    <a href = "{url_for('register') }"><b>Zarejestruj się </b></a></p>
  </div>
</div>

```

Fragment kodu 6. Logowanie - frontend

Ponadto, formularz został wyposażony w szereg komunikatów, w przypadku błędnego wypełnienia – czy wszystkie pola zostały wypełnione oraz czy dane są poprawne.

The image displays two screenshots of the 'BlogoSfera' web application's login interface. Both screenshots show the site's header with the logo 'BlogoSfera' and navigation links: 'Przeglądaj', 'Dodaj', and 'Moje wpisy'. Below the header is a navigation bar with an orange button labeled 'Wypełnij wszystkie pola' (Fill all fields) in the top screenshot and 'Błąd, spróbuj ponownie' (Error, try again) in the bottom screenshot. The main content area features a light gray box titled 'Logowanie' (Login) containing a 'Login' label and a text input field. In the bottom screenshot, the error message is visible above the input field.

Rysunek 5. Zrzuty ekranu przedstawiającego komunikaty błędów


```
@app.route('/login', methods=["GET", "POST"])
def login():
    [...]
    if not login or not password:
        flash('Wypełnij wszystkie pola')
        return redirect(url_for('login'))
    [...]
    else:
        flash('Błąd, spróbuj ponownie')
        return redirect(url_for('login'))
```

Fragment kodu 7. Generowanie komunikatów - backend

```
<!-- flash messages for html -->
{% for message in get_flashed_messages() %}
    <div class="alert-error"> {{ message }}</div>
{% endfor %}
<!-- end of flash messages for html -->
```

Fragment kodu 8. Wyświetlanie komunikatów - frontend

Analogicznie zaimplementowano generację komunikatów na pozostałych stronach.

Rejestracja

Panel rejestracji jest analogiczny do interfejsu strony logowania. Oczywiście, zostały zamienione opcje „Zaloguj” na „Zarejestruj”. Dodatkowo, po zarejestrowaniu użytkownik zostaje automatycznie przekierowany na stronę logowania. Strona również została opatrzona kontrolą błędów.

Rysunek 6. Zrzut ekranu strony do rejestracji

```

@app.route('/register', methods=["GET", "POST"])
def register():
    if request.method == 'POST':
        login = request.form['login']
        password = request.form['password']
        [...]
        user = User.query.filter_by(login=login).first
        [...]
        password = bcrypt.hashpw(password.encode('utf8'), bcrypt.gensalt(16))
        newUser = User(login=login, password=password)
        DataBase.session.add(newUser)
        DataBase.session.commit()

        flash('Konto zostało stworzone, zaloguj się')
        return redirect(url_for('login'))
    return render_template('register.html')

```

Fragment kodu 10. Rejestracja - backend

```

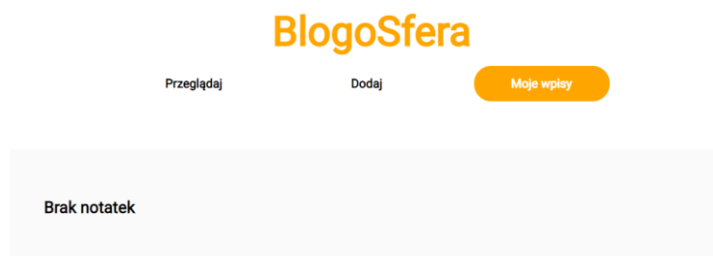
<div class="post">
    <h3 id="new-post-header">Rejestracja</h3>
    <form action="" method = "POST">
        <div class="login-form">
            <label>Login</label>
            <input class="login-input" type="text" name="login">
            <label>Hasło</label>
            <input class="login-input" type="password" name="password">
        </div>
        <div class="button-holder">
            <button type="submit" class="btn-login" value = "Submit">Zarejestruj się</button>
        </div>
    </form>
    <div class="no-account">
        <p>Masz już konto?
            <a href="{{url_for('login')}}"><b>Zaloguj się </b></a></p>
    </div>

```

Fragment kodu 9. Rejestracja - frontend

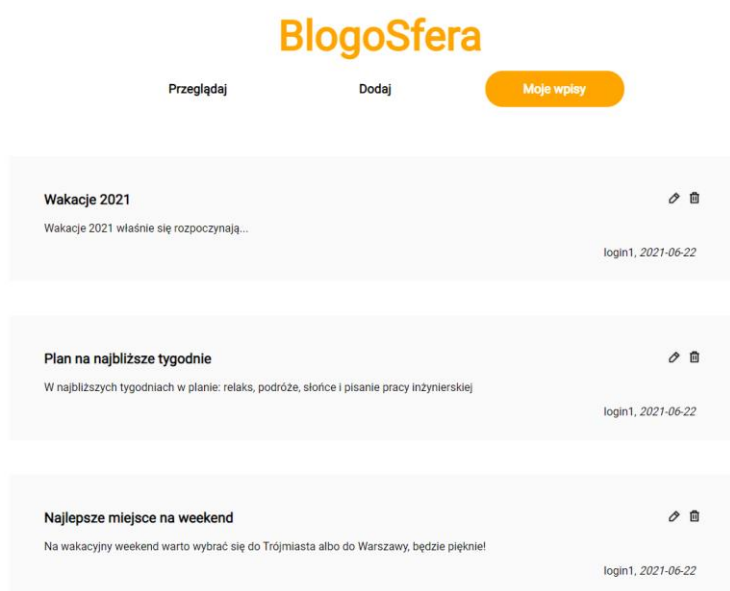
Strona użytkownika

Według projektu, na stronie użytkownika znajdują się wszystkie utworzone przez niego wpisy. W przypadku braku wpisów wyświetla się jedynie napis „Brak notatek”.



Rysunek 7. Zrzut ekranu strony użytkownika, gdy nie udostępnił jeszcze żadnych wpisów

Jeżeli zalogowany użytkownik utworzył wcześniej wpisy, są one wyświetlane jeden pod drugim, zgodnie z kolejnością dodania. Dodatkowo, przy każdym wpisie wyświetlano są ikony umożliwiające usunięcie oraz modyfikację notatek.



Rysunek 8. Zrzut ekranu strony użytkownika z istniejącymi wpisami

Ponadto, podobnie jak na pozostałych stronach w prawym dolny rogu bloku z wpisem, wyświetlany jest login autora oraz data utworzenia wpisu pobrane z bazy danych.

```

@app.route('/mypage')
def my():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    user = session['user_id']

    data = BlogSfera.query.filter_by(user_id=user).order_by(desc(BlogSfera.data)).all()

    return render_template('mypage.html', data=data)

```

Fragment kodu 11. Strona użytkownika - backend

```

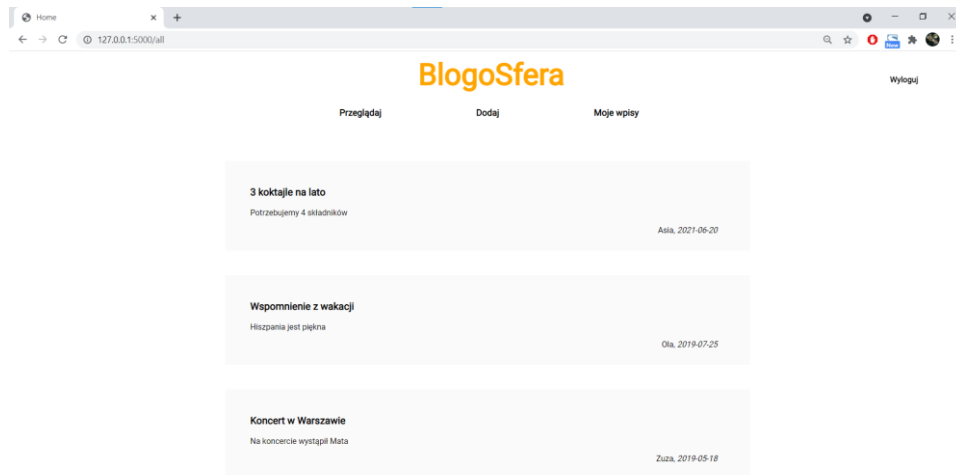
{% if data %}
    {% for wpis in data %}
        <div class="post">
            <div>
                <div class="icon-wrapper">
                    <a href="/modify/{{wpis.id}}"> </a>
                    <a href="/delete/{{wpis.id}}" [...]> </a>
                </div>
                <h3 id="title">{{wpis.tytul}}</h3>
            </div>
            <p>{{ wpis.tresc }} </p>
            <div class="signature-wrapper">
                <h4 id="signature">{{ wpis.autor }}, <i>{{ wpis.data }}</i></h4>
            </div>
        </div>
    {% endfor %}
{% else %}
    <div class="post">
        <h2> Brak notatek </h2>
    </div>
{% endif %}

```

Fragment kodu 12. Strona użytkownika - frontend

Strona ze wszystkimi wpisami

Dla zalogowanych użytkowników przewidziano opcję przeglądania wszystkich wpisów zarejestrowanych autorów. Oczywiście, bez możliwości edytowania czytanych postów.



Rysunek 9. Zrzut ekranu strony ze wszystkimi wpisami

```
@app.route('/all')
def all():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    data = BlogSfera.query.order_by(desc(BlogSfera.data)).all()

    return render_template('all.html', data=data)
```

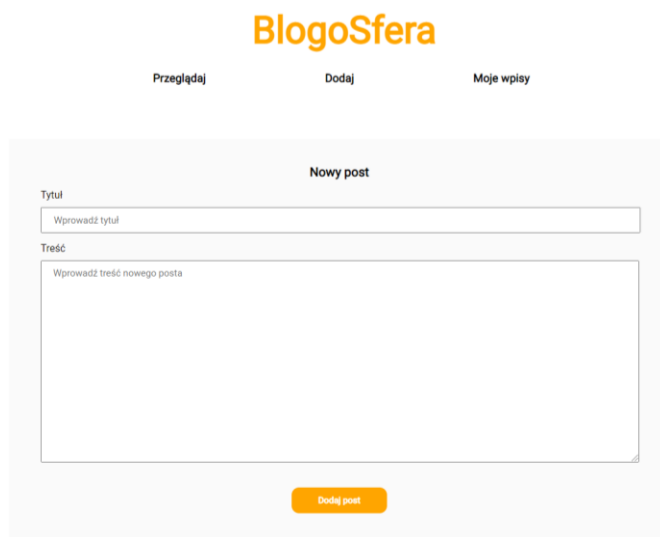
Fragment kodu 13. Wszystkie wpisy - backend

```
{% if data %}
    {% for wpis in data %}
        <div class="post">
            <div><h3 id="title">{{ wpis.tytul }}</h3></div>
            <p>{{ wpis.tresc }}</p>
            <div class="signature-wrapper">
                <h4 id="signature">{{ wpis.autor }}, <i>{{ wpis.data }}</i></h4>
            </div>
        </div>
    {% endfor %}
{% else %}
    <h2> Brak wpisów na BlogoSferze </h2>
{% endif %}
```

Fragment kodu 14. Wszystkie wpisy - frontend

Dodawanie notatki

Zalogowani użytkownicy mają możliwość dodawania wpisów na swoją stronę. Po wybraniu opcji „Dodaj” w menu, użytkownik zostaje przekierowany na poniższą stronę.



The screenshot shows the 'Nowy post' (New post) form in the BlogoSfera application. At the top, the 'BlogoSfera' logo is visible on the left, and a 'Wyloguj' (Logout) link is on the right. Below the logo, there are three navigation links: 'Przeglądaj' (Browse), 'Dodaj' (Add), and 'Moje wpisy' (My posts). The 'Dodaj' link is highlighted. The main form is titled 'Nowy post' and contains two input fields: 'Tytuł' (Title) with the placeholder 'Wprowadź tytuł' and 'Treść' (Content) with the placeholder 'Wprowadź treść nowego posta'. At the bottom of the form is an orange button labeled 'Dodaj post' (Add post).

Rysunek 10. Zrzut ekranu strony z dodawaniem notatki

Jeżeli użytkownik nie jest zalogowany, a wybierze opcję „Dodaj” zostaje przekierowany na stronę logowania. Zalogowany użytkownik po zapisaniu wpisu, zostaje przekierowany do strony ze swoimi wpisami.

```
@app.route('/new', methods=["GET", "POST"])
def add():
    [...]

    if request.method == 'POST':
        tytul = request.form['title']
        tresc = request.form['content']
        data = datetime.date.today()
        user_id = session["user_id"]
        user = User.query.filter_by(id=user_id).first()
        login = user.login

    [...]

    newPost = BlogSfera(tytul=tytul, data=data, tresc=tresc, user_id=user_id, autor=login)
    DataBase.session.add(newPost)
    DataBase.session.commit()

    flash('Post został opublikowany')
    return redirect(url_for('my'))

return render_template('new.html')
```

Fragment kodu 15. Dodawanie wpisu - backend

```

<div class="post">

    <h3 id="new-post-header">Nowy post</h3>

    <form class="new-post" method="post" id="new-post-form">

        <label>Tytuł</label>

        <input class="title-input" type="text" name="title"
placeholder="Wprowadź tytuł">

        <label>Treść </label>

        <textarea class="post-input" name="content" form="new-post-form"
placeholder="Wprowadź treść nowego posta"></textarea>

        <div class="button-holder">

            <button type="submit" class="btn-submit">Dodaj post</button>

        </div>

    </form>

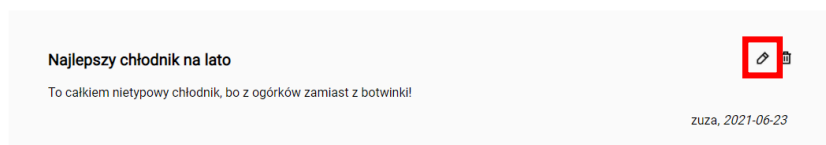
</div>

```

Fragment kodu 16. Dodawnia wpisu - frontend

Modyfikacja notatki

Strona modyfikacji postu jest wyświetlana, po wybraniu ikony ołówka w prawym górnym rogu wpisu.



Rysunek 11. Ikona modyfikacji wpisu

Użytkownik może zmienić tytuł oraz treść wpisu, jednocześnie automatycznie przy zapisie jest też zmieniana data dodania wpisu na aktualną. Po przekierowaniu na stronę edycji, w oknie tytuł oraz treść, wyświetlane są dotychczasowe wartości tych sekcji. Dzięki temu, użytkownik może zmienić pojedyncze znaki w już istniejącej treści, zamiast pisać cały tekst od nowa. Jeżeli niezalogowany użytkownik wpisze w pasku adres strony modyfikacji, zostanie przekierowany na stronę logowania. Jeżeli jednak zalogowany użytkownik wpisze adres wskazujący na chęć modyfikowania nieswojego wpisu, zostanie wyświetlony stosowny komunikat.

Modyfikuj post

Tytuł

Treść

To całkiem nietypowy chłódnik, bo z ogórków zamiast z botwinki!

Zapisz zmiany

Rysunek 12. Zrzut ekranu strony modyfikacji

```
@app.route('/modify/<int:id>', methods=["GET", "POST"])
def modify(id):
    [...]
    updatePost = BlogSfera.query.filter_by(id=id).first()
    if request.method == 'POST':
        [...]
        updatePost.tytul = request.form['title'] # pobranie nowych informacji
        updatePost.treść = request.form['content']
        updatePost.data = datetime.date.today()
    [...]
        DataBase.session.commit() # zatwierdzenie nowych treści
        flash('Post został zaktualizowany')
        return redirect(url_for('my'))
    [...]
    else:
        return render_template('modify.html', updatePost=updatePost)
```

Fragment kodu 17. Modyfikacja wpisu - backend


```

<div class="post">

    <h3 id="new-post-header">Modyfikuj post</h3>

    <form class="new-post" method="post" id="new-post-form">

        <label>Tytuł</label>

        <input class="title-input" type="text" name="title" value="{{
updatePost.tytul }}">

        <label>Tresc</label>

        <textarea class="post-input" name="content" form="new-post-form" >{{
updatePost.tresc }}</textarea>

        <div class="button-holder">

            <button type="submit" class="btn-submit">Zapisz zmiany</button>

        </div>

    </form>

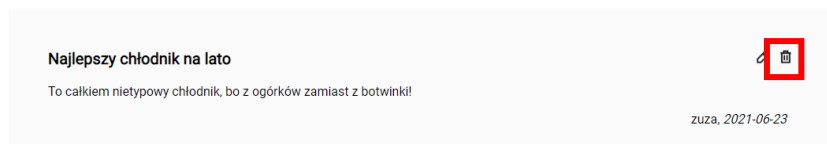
</div>

```

Fragment kodu 18. Modyfikacja wpisu - frontend

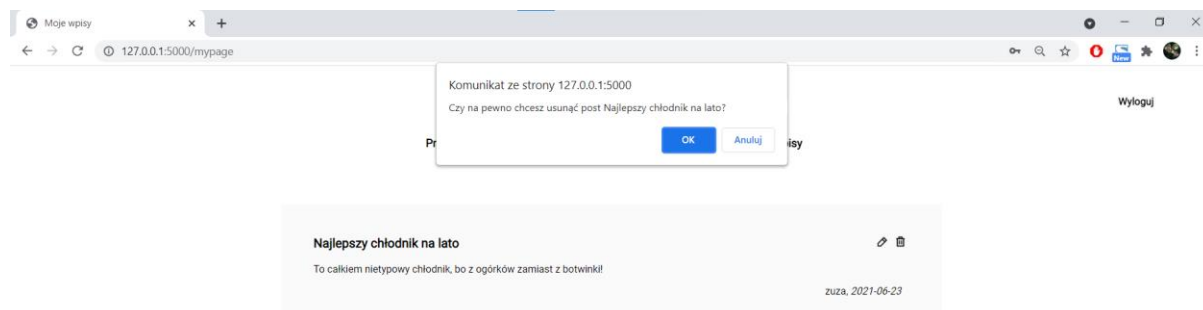
Usuwanie notatki

Obok ikony modyfikacji, przy postach użytkownika wyświetlana jest ikona usuwania wpisu.



Rysunek 13. Ikona usuwania wpisu

W celu uniknięcia nieumyślnego usuwania postów, postanowiono, aby zaimplementować okno z potwierdzeniem żądania. Po wybraniu ikony usuwania, na górze ekranu pojawia się okno informujące o żądaniu oraz wyświetlany jest tytuł usuwanego wpisu. Po wybraniu przez użytkownika opcji „Ok” post jest usuwany z bazy.



Rysunek 14. Zrzut ekranu okna z potwierdzeniem żądania

```

@app.route('/delete/<int:id>', methods=["GET"])

def delete(id):

    [...]

    wpis = BlogSfera.query.filter_by(id=id).first()

    [...]

    BlogSfera.query.filter_by(id=id).delete()

    DataBase.session.commit()

    return redirect(url_for('my'))

```

Fragment kodu 19. Usuwanie wpisu - backend

```

<div class="icon-wrapper">

    <a href="/modify/{{wpis.id}}"> </a>

    <a href="/delete/{{wpis.id}}" onclick="return confirm('Czy na pewno chcesz usunąć post
    {{ wpis.tytul }}?')">
    </a>

</div>

```

Fragment kodu 20. Usuwanie wpisu - frontend

Podobnie jak w przypadku strony edycji wpisu – przy wpisaniu ręcznie adresu usuwania wpisu, sprawdzane jest czy użytkownik jest zalogowany i jest autorem notatki.

Baza danych

Struktura bazy danych zakłada dwie rozdzielne bazy: bazę użytkowników oraz bazę wpisów. Rekord w bazie użytkowników zawiera id, login oraz zaszyfrowane hasło. W bazie wpisów każdy wpis opisany jest poprzez atrybuty: tytuł, data utworzenia, treść oraz login użytkownika, który stworzył wpis.

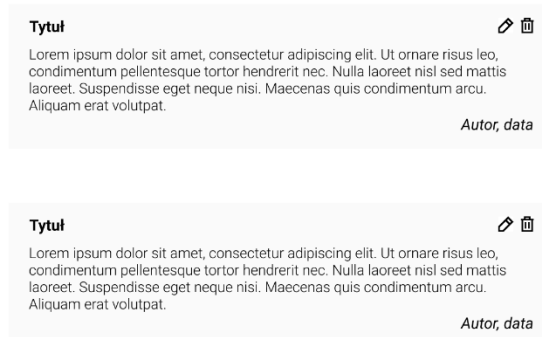
Komunikacja z bazą danych pozwala na poprawne logowanie oraz rejestrację użytkowników, a także operację na wpisach: tworzenie, modyfikację, usuwanie wpisu.

Dokumentacja wstępna a dokumentacja końcowa

Podczas realizacji projektu rozszerzono założenia dotyczące interfejsu graficznego projektowanej aplikacji. Stworzona przez nas aplikacja jest intuicyjna, a także wygodna dla użytkowników. Tworzyliśmy ją z myślą, o jak najlepszym *user experience*. Ostateczny wygląd interfejsu jest bardzo zbliżony to naszych wstępnych założeń:

BlogoSfera

przeglądaj dodaj post



Rysunek 15. Wstępny projekt interfejsu graficznego aplikacji

Względem początkowych założeń wprowadziłyśmy dodatkowe udogodnienia:

- tworzenie, modyfikacja lub usunięcie wpisu nie wymaga ponownego wprowadzania hasła – wystarczy zalogowanie się, co powoduje uwierzytelnienie sesji i w ramach tej sesji wykonuje się powyższe operacje,
- ze względu na ochronę przed atakami typu „brute force” błędy przy logowaniu lub rejestracji są komunikowane użytkownikowi niebezpośrednio (informacja o tym, które pole zostało niepoprawnie uzupełnione nie jest mu znana),
- w menu pojawił się dodatkowy przycisk do logowania się, co ułatwia użytkownikom obsługę aplikacji,
- przyciski przekierowujące do logowania się lub rejestracji są umieszczone w widocznych dla użytkownika miejscach.

Instrukcja kompilacji, uruchamiania i korzystania

Przed uruchomieniem aplikacji, należy zainstalować poniższe pakiety wraz danymi instrukcjami:

```
npm install Flask
npm install sqlite3
pip install SQLAlchemy
pip install flask-sqlalchemy
pip install py-bcrypt
```

Zaleca się uruchomienie aplikacji w PyCharm, umożliwia to bezpośrednie zainicjalizowanie *Driver SQLite*.

Pierwszym etapem uruchomienia aplikacji jest stworzenie bazy danych z przykładowymi danymi. Uruchomienie pliku *sql.py*. Następnie należy uruchomić *app.py*. Podczas uruchamiania tego pliku powinien zostać wyświetlony poniższy komunikat z linkiem.

```
* Serving Flask app 'app' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 141-980-122
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Rysunek 16. Komunikat podczas uruchamiania aplikacji

Link <http://127.0.0.1:5000/> należy kliknąć. Następnie w przeglądarce uruchomi się stworzona aplikacja internetowa. Korzystanie z aplikacji zostało omówione w rozdziale *Realizacja projektu*.

Zalecenie:

Przy pierwszym uruchomieniu aplikacji należy stworzyć swojego własnego użytkownika, aby zobaczyć działanie całej aplikacji. Bazowi użytkownicy nie mają zahaszowanych haseł, co powoduje błędne działanie aplikacji.

Współpraca podczas realizacji projektu

Realizacja projektu umożliwiła nam rozwój naszych umiejętności. Podczas tworzenia dokumentacji początkowej dokonaliśmy podziału obowiązków, co umożliwiło nam określić pewne cele dla każdej z nas.

Zuzanna Adamiuk, jako Senior Frontend Developer, kierowała pracami nad frontendem aplikacji i przydzielała pozostałym członkom grupy ewentualne dodatkowe zadania w tym obszarze. Dodatkowo Zuzanna dbała o integrację zespołu poza aspektem pracy nad projektem, zachowując work- life balance.

Joanna Rancew jako Senior Backend Developer zespołu, kierowała pracami nad backendem, delegowaniem zadań w tym zakresie oraz przygotowaniem i dostosowaniem narzędzi do komunikacji w zespole.

Aleksandra Krakowiak jako Senior Database Developer, kierowała pracami nad dostępem do bazy danych, jej przygotowaniem oraz dbaniem o dobre relacje w zespole. Dodatkowo Aleksandra pełniła obowiązki Junior Backend Developer we współpracy z Joanną. Aleksandra była także odpowiedzialna za rekrutację członków zespołu.

Wnioski

Realizacja projektu pozwoliła nam rozwinąć umiejętności pracy w grupie, zarządzania czasem oraz umiejętność delegacji zadań. Każda z nas poznała nowe technologie, funkcje i nauczyła dobrych praktyk przy tworzeniu aplikacji webowych.

Końcowy efekt: wygląd oraz funkcjonalność aplikacji są naszym zdaniem satysfakcjonujące. Nie tylko osiągnęliśmy założone na początku projektu cele, ale także udoskonaliłyśmy serwer o kilka bardzo istotnych funkcjonalności. Aplikacja jest odporna na najczęstsze błędy, wygodna dla użytkowników i spełnia odpowiednie funkcje.

Bibliografia

1. <https://flask.palletsprojects.com/en/2.0.x/#>
2. <https://flask-sqlalchemy.palletsprojects.com/en/2.x/#>