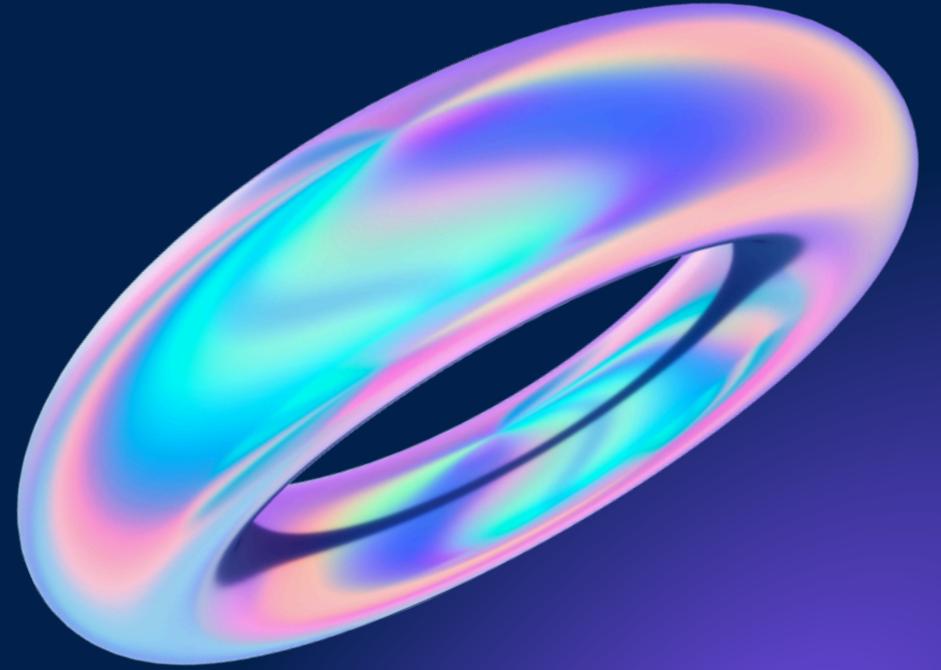


# TaskForge

## aplikacja webowa we flasku



# Czym jest Flask?



Flask jest mikroframeworkiem do tworzenia aplikacji webowych w języku Python.



Jego cechą charakterystyczną jest minimalistyczna natura, co oznacza, że zapewnia podstawowe narzędzia i struktury potrzebne do budowy aplikacji

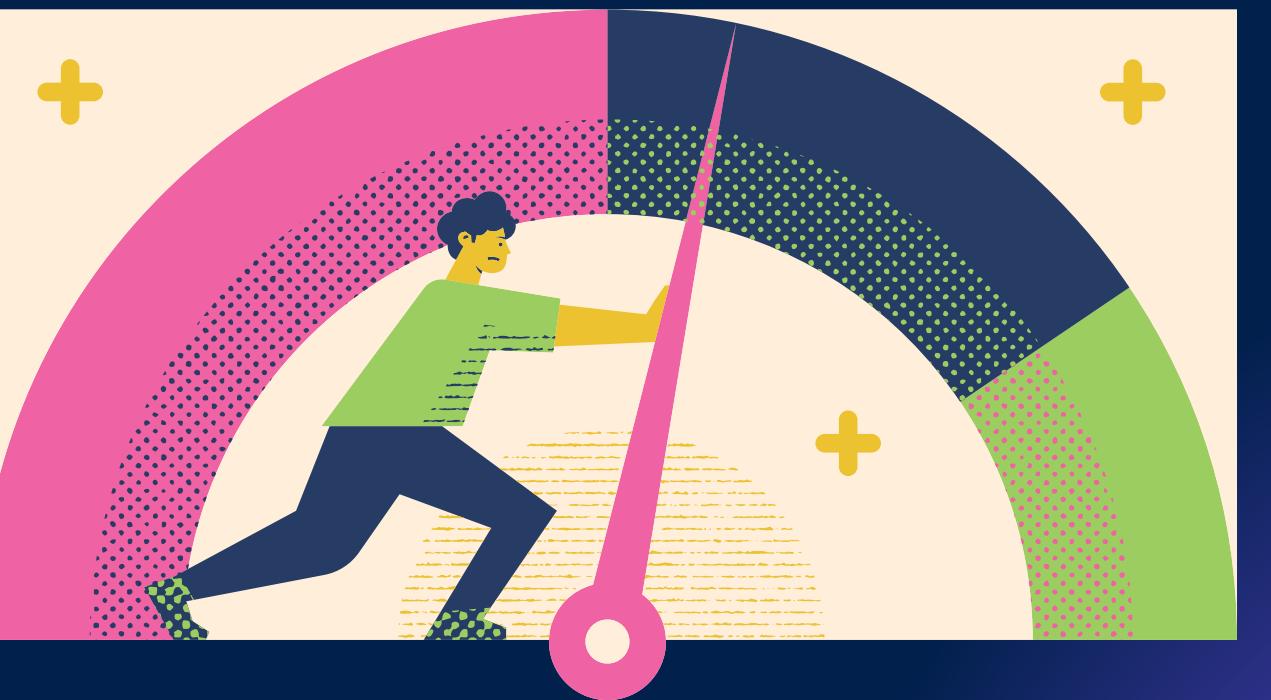


**Armin Ronacher**

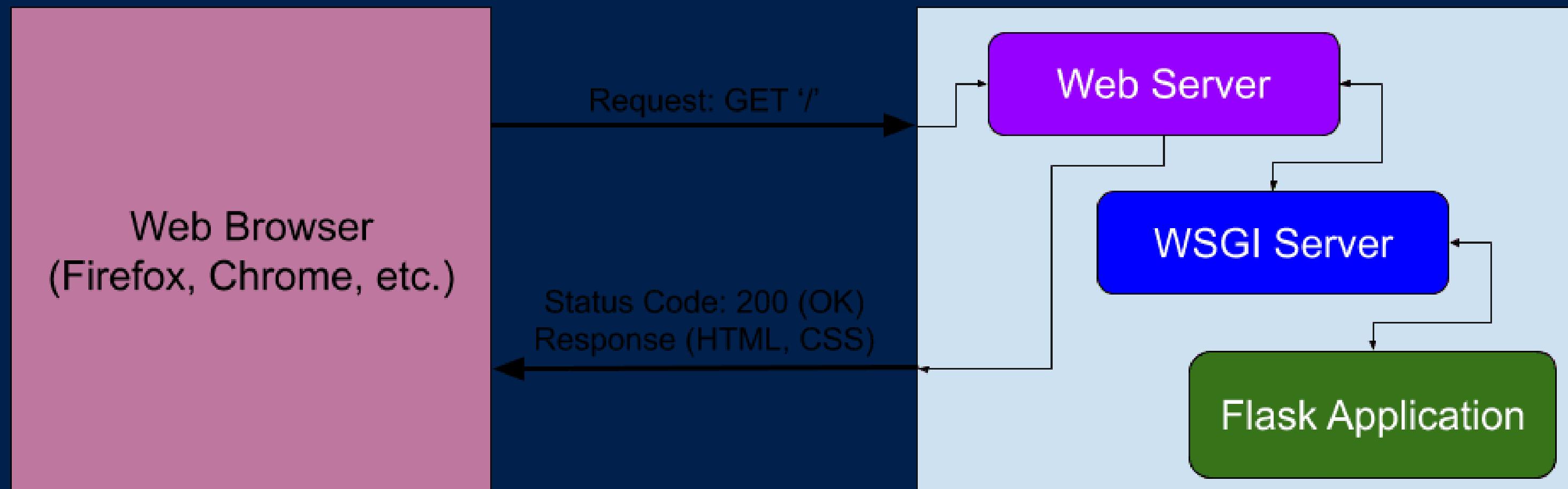
Flask został stworzony przez Armina Ronachera i po raz pierwszy wydany w 2010 roku. Od tego czasu zdobył dużą popularność wśród programistów Pythona i jest wykorzystywany przez wiele firm i projektów na całym świecie.

# Dlaczego warto używać flaska?

- Łatwy
- Lekki
- Elastyczny
- Zintegrowany
- Dobrze udokumentowany
- Wsparcie społeczności



# Request Processing



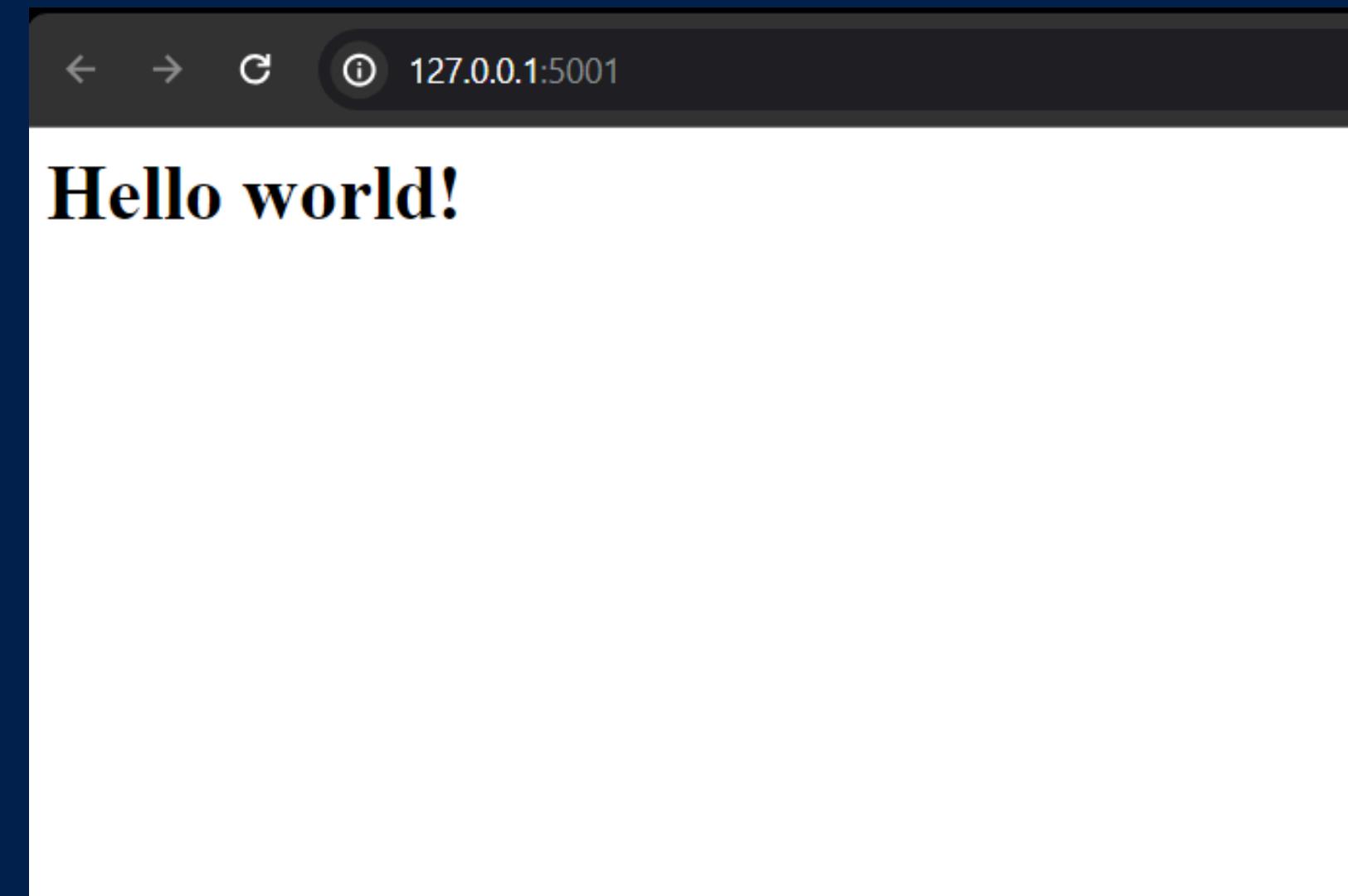
# Podstawy routingu w Flasku

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<h1>Hello world!</h1>

if __name__ == '__main__':
    app.run(debug=True)
```



```
PS C:\Users\pawel\OneDrive\Pulpit\Flask> python.exe .\app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5001
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 118-866-184
```

# Szablony i renderowanie

## 1. Jak używać szablonów HTML w Flasku?

- Flask umożliwia używanie szablonów HTML, co pozwala na separację logiki aplikacji od warstwy prezentacji.

Hello, World!

## 2. Przekazywanie danych z widoku do szablonu i ich renderowanie

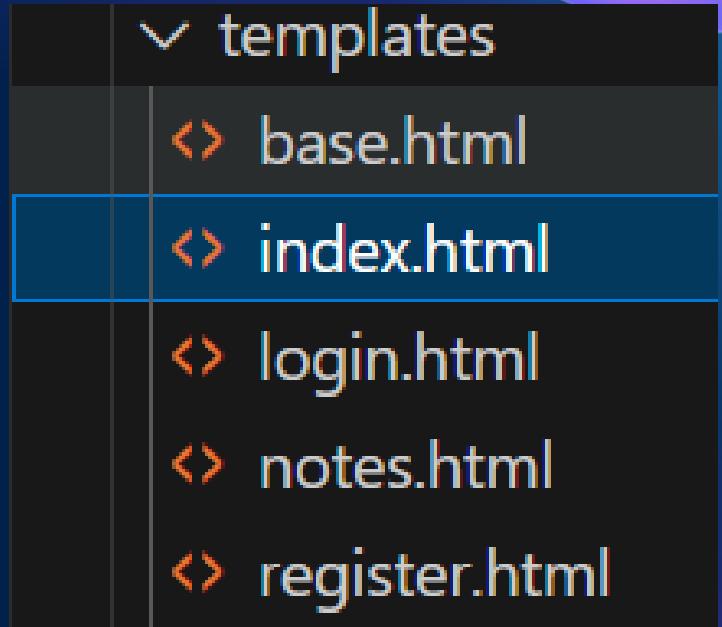
- Przedstawienie sposobu przekazywania danych z widoku do szablonu za pomocą mechanizmu kontekstu aplikacji i ich renderowania przy użyciu silnika szablonów Flaska.

```
< from flask import render_template
  from flask import Flask

  app=Flask(__name__)

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

  if __name__ == '__main__':
      app.run(debug=True)
```



A screenshot of a code editor showing two files: 'hello.html' and 'app.py'. The 'hello.html' file contains the following code:

```
<h1>Hello, World!</h1>
```

The 'app.py' file contains the following code:

```
from flask import render_template
from flask import Flask

app=Flask(__name__)

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

if __name__ == '__main__':
    app.run(debug=True)
```

# Komponenty Flask

繩  
Jinja



Werkzeug



IT'S DANGEROUS  
... so better sign this



# MarkupSafe



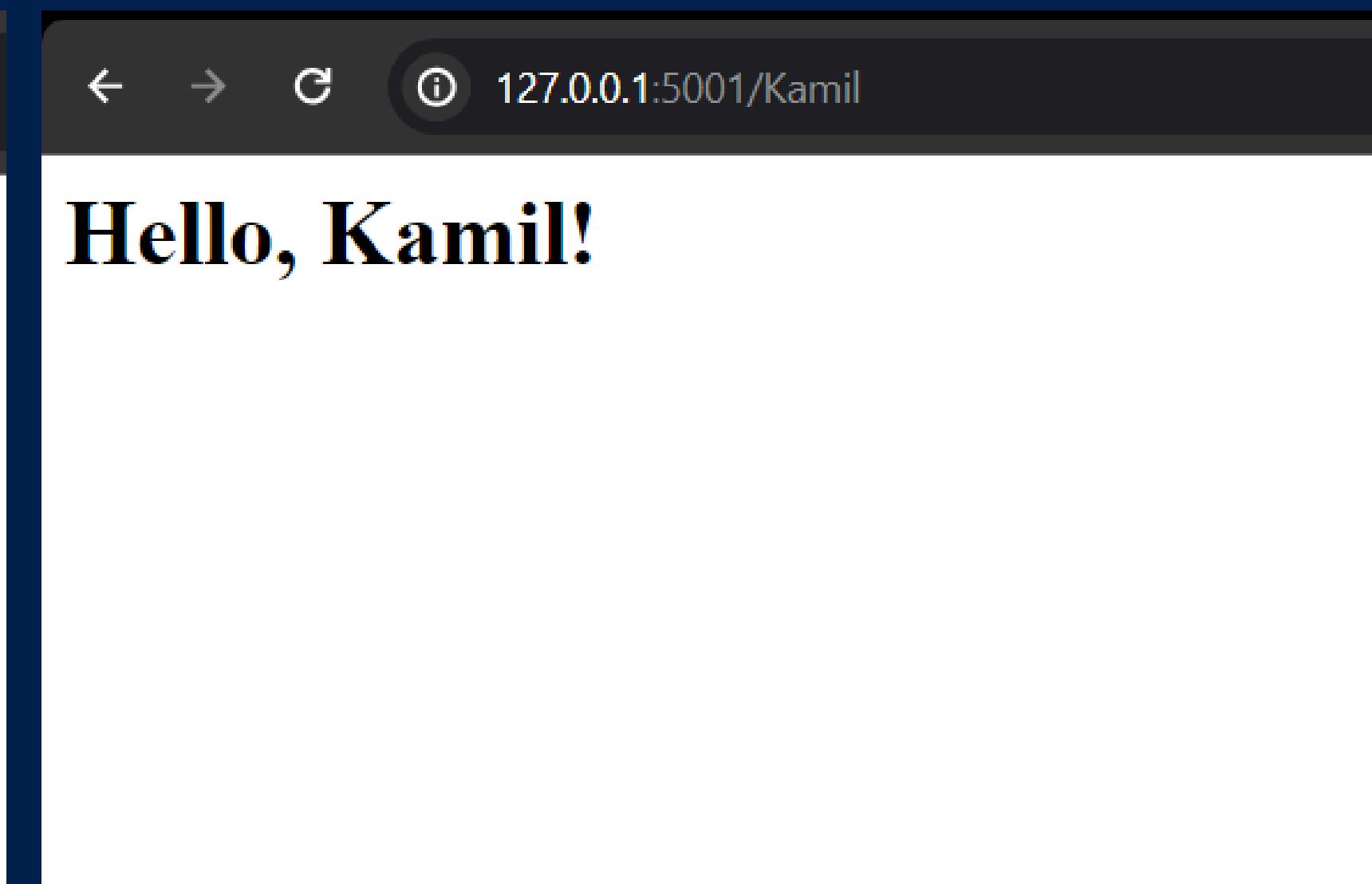
MarkupSafe - biblioteka obsługująca ciągi znaków w Pythonie, rozszerzająca typ string oznaczając jego zawartość jako "bezpieczną". Dzięki temu unika podwójnego unikania znaków specjalnych, co jest przydatne w zapobieganiu atakom XSS (Cross-Site Scripting).



# Przechwytywanie wartości

```
from markupsafe import escape

@app.route("/<name>")
def hello(name):
    return f"Hello, {escape(name)}!"
```



# Jinja 2

Jinja, jako silnik szablonów dla języka Python, oferuje szereg funkcjonalności, które ułatwiają generowanie dynamicznych treści w aplikacjach internetowych.

## Funkcjonalności:

兩  
Jinja

```
Templates > index.html
{% extends "base.html" %}

{% block title %}
Home Page {% endblock %}

{% block content %}

```

### Dziedziczenie szablonów

Pozwala na tworzenie hierarchii szablonów, gdzie jedne szablony dziedziczą cechy i bloki innych szablonów. Jest to użyteczne do unikania powtarzania kodu HTML.

```
Templates > index.html
{% extends "base.html" %}

{% block title %}
Home Page {% endblock %}

{% block content %}

```

### Bloki zawartości:

Umożliwia definiowanie bloków w szablonach, które mogą być nadpisane w dziedziczących szablonach.

# Jinja 2



```
<meta charset="UTF-8">
<meta name="viewport" co
<title>Flask App</title>
ad>
y>
<h1>Witaj, {{ user }}!</
dy>
m1>
```

## Wstawianie zmiennych

Jinja pozwala wstawiać zmienne dynamicznie w szablonach za pomocą {{ }}

```
if todo.priority == 'low' %}
    <span>Low</span>
elif todo.priority == 'medium'
    <span>Medium</span>
elif todo.priority == 'high' %
    <span>High</span>
endif %}
```

## Wyrażenia logiczne

Pozwala na wykonywanie operacji logicznych w szablonach za pomocą bloków

```
>Lista użytkowników:</h1>
>
{%
    for user in users %}
        <li>{{ user.name }}</li>
    {% endfor %}
</ul>
```

## Pętle

umożliwia iterację przez listy, słowniki i inne iterowalne obiekty

# Werkzeug

pip install Werkzeug

Werkzeug to zestaw narzędzi WSGI dla Pythona, który zapewnia obsługę żądań i odpowiedzi HTTP, routowanie URL, obsługę sesji, narzędzia do debugowania i testowania aplikacji internetowych. Jest często wykorzystywany jako podstawa do budowy aplikacji webowych w Pythonie.

```
from werkzeug.security import generate_password_hash, check_password_hash  
from flask import Flask  
  
new_user = User(username=username, password=generate_password_hash(password, method='pbkdf2:sha256'))  
db.session.add(new_user)  
  
if check_password_hash(user.password, password):  
    login_user(user, remember=True)
```

Werkzeug



# Obsługa formularzy

Importowanie modułów:

```
from flask import Flask, render_template, redirect, url_for, request, flash,
```

```
@auth.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')
```

```
<h1>Register</h1>
<form method="POST">
    <div class="form-group">
        <label for="username" class="forms-inputs">Username</label>
        <input type="text" name="username" id="username" class="form-control">
    </div>
    <div class="form-group">
        <label for="password" class="forms-inputs">Password</label>
        <input type="password" name="password" id="password" class="form-control">
    </div>
```

# HTTP requests

Metody HTTP to standardowe sposoby, za pomocą których żądania HTTP (HTTP requests) mogą być przekazywane do serwera w celu uzyskania lub modyfikowania zasobów.

Metody GET i POST są używane w protokole HTTP do komunikacji między klientem a serwerem.

- Metoda GET jest używana do żądania danych z określonego zasobu.
- Metoda POST jest używana do przesyłania danych do serwera w celu przetworzenia ich przez serwer.

Są jeszcze takie metody jak PUT, DELETE, PATCH, HEAD, OPTIONS, TRACE, CONNECT aby się dowiedzieć więcej zachęcamy do odwiedzenia dokumentacji flaska <https://flask.palletsprojects.com/en/3.0.x/>

Metoda login i register używają zarówno metody GET, jak i POST, aby obsłużyć zarówno wyświetlanie formularza, jak i przetwarzanie danych przesyłanych przez formularz. Metoda GET jest używana, gdy użytkownik otwiera stronę logowania lub rejestracji, a metoda POST jest używana, gdy użytkownik wysyła formularz z danymi logowania lub rejestracji.

```
@auth.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')

        user = User.query.filter_by(username=username).first()
        if user:
            if check_password_hash(user.password, password):
                login_user(user, remember=True)
                return redirect(url_for('views.home'))
            else:
                flash('Invalid password. Please try again.', 'error')
        else:
            flash('User not found.', 'error')

    return render_template('login.html', user=current_user)
```

```
@auth.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')
        user = User.query.filter_by(username=username).first()
        if len(username) < 2:
            flash('The username must be greater than 1 character.', category='error')
        elif len(password) < 8:
            flash('The password must be blalblala.', category='error')
        elif user:
            flash('The username is already taken. Please choose a different one.', category='error')
        else:
            new_user = User(username=username, password=generate_password_hash(password, method='pbkdf2:sha256'))
            db.session.add(new_user)
            db.session.commit()
            login_user(new_user, remember=True)
            flash('Account created.', category='success')
            return redirect(url_for('auth.login'))

    return render_template('register.html', user=current_user)
```

```
@auth.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')

        user = User.query.filter_by(username=username).first()
        if user:
            if check_password_hash(user.password, password):
                login_user(user, remember=True)
                return redirect(url_for('views.home'))
            else:
                flash('Invalid password. Please try again.', 'error')
        else:
            flash('User not found.', 'error')

    return render_template('login.html', user=current_user)
```

Po zalogowaniu zostajemy przekierowani na stronę główną

# Bazy danych we flasku

pip install flask-sqlalchemy

- Flask umożliwia integrację z różnymi bazami danych, w tym SQLite, MySQL, PostgreSQL i wiele innych.

## W pliku \_\_init\_\_

```
from flask_sqlalchemy import SQLAlchemy  
  
db = SQLAlchemy()  
DB_NAME = "database.db"
```

Inicjalizacja bazy danych

```
app.config['SQLALCHEMY_DATABASE_URI'] = f'sqlite:///{{DB_NAME}}'  
db.init_app(app)
```

Konfiguracja flask do korzystania z bazy danych

```
create_database(app)
```

Tworzenie bazy danych, jeśli nie istnieje:

## W pliku models

```
from . import db # importujemy db z pliku __init__.py  
from flask_login import UserMixin # dziedziczymy po Us  
from sqlalchemy.sql import func  
  
class User(db.Model, UserMixin):  
    id = db.Column(db.Integer, primary_key=True) # pri  
    username = db.Column(db.String(100), unique=True)  
    password = db.Column(db.String(100))  
    tasks = db.relationship('Task', backref='user')  
    notes = db.relationship('Note', backref='user')
```

Definicja klasy User która jest modelem danych z wykorzystaniem SQLAlchemy

# Bazy danych we flasku

```
tasks = db.relationship('Task', backref='user')
notes = db.relationship('Note', backref='user')
```

**Tworzenie relacji między klasą Note oraz Task**

```
new_user = User(username=username, password=generate_password_hash(password, method='pbkdf2:sha256'))
db.session.add(new_user)
db.session.commit()
```

**Tworzenie nowego użytkownika poprzez konstruktor .**

# Flask-Admin

pip install Flask-Admin

```
from flask_admin import Admin  
from flask_admin.contrib.sqla import ModelView
```

- Flask-Admin jest rozszerzeniem Flask, które umożliwia szybkie tworzenie panelu administracyjnego opartego na interfejsie użytkownika. Pozwala to na łatwe zarządzanie danymi w aplikacji Flask bez konieczności pisania dużo kodu.

The screenshot shows a user management interface. At the top, there's a navigation bar with tabs: Admin (which is active), Home, User (selected), Task, and Note. Below the tabs, there are three buttons: List (2), Create, and With selected. The main area displays a table with three columns: a checkbox column, Username, and Password. There are two rows of data:

<input type="checkbox"/>	Username	Password
<input type="checkbox"/>	PabloEscobar	pbkdf2:sha256:600000\$J
<input type="checkbox"/>	khebda	pbkdf2:sha256:600000\$S

```
login_manager = LoginManager()  
login_manager.login_view = 'auth.login'  
login_manager.init_app(app)  
  
@login_manager.user_loader  
def load_user(id):  
    return User.query.get(int(id))  
  
# Inicjalizacja panelu admina  
admin = Admin(app)  
  
# Dodanie modeli do panelu admina  
admin.add_view(ModelView(User, db.session))  
admin.add_view(ModelView(Task, db.session))  
admin.add_view(ModelView(Note, db.session))
```

# Flash messages

**Flash w Flasku to mechanizm, który umożliwia przechowywanie komunikatów, takich jak komunikaty o sukcesie lub błędach, przez jeden request HTTP i wyświetlanie ich w kolejnym requestie HTTP. Jest to przydatne do przekazywania komunikatów użytkownikowi po wykonaniu jakieś akcji, na przykład po zalogowaniu się, wysłaniu formularza lub wykonaniu innej operacji.**

# Przykłady Flash messages

```
@auth.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')
        user = User.query.filter_by(username=username).first()
        if len(username) < 2:
            flash('The username must be greater than 1 character.', category='error')
        elif len(password) < 8:
            flash('The password must be longer than 7 characters.', category='error')
        elif user:
            flash('The username is already taken. Please choose a different one.', category='error')
        else:
            new_user = User(username=username, password=generate_password_hash(password, method='pbkdf2:sha256'))
            db.session.add(new_user)
            db.session.commit()
            login_user(new_user, remember=True)
            flash('Account created.', category='success')
    return redirect(url_for('auth.login'))
```

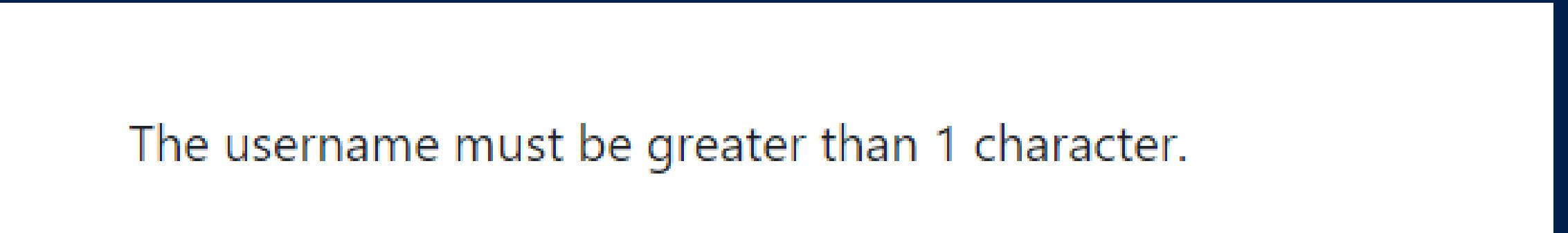
## Register

Username

Password

If you have an account click [here](#) login.

Po naciśnięciu przycisku Register otrzymujemy:



# Źródła:

<https://flask.palletsprojects.com/en/3.0.x/>

<https://www.youtube.com/watch?v=dam0GPOAvVI>

<https://www.turing.com/kb/build-flask-routes-in-python>



# Część praktyczna





Thank You