



# Flask

Jan Guz  
Maciej Klimek

## > Cel prezentacji:

- # Co to jest Flask ?

- # Historia

- # Podstawowe cechy i działanie

- # Szablony

- # Najciekawsze moduły i ich cechy

- # Bezpieczeństwo Flask

- # Wady/Zalety

- # Flask vs. Django

- # Połączenie Flaska z odrębnym frontendem

> Co to jest Flask?



**moz://a**

**Uber**

**NETFLIX**

## > Komponenty

# Werkzeug (WSGI) - obsługa części sieciowej

# Jinja - silnik szablonów

# MarkupSafe - zabezpieczenie przed atakami XSS

# ItsDangerous - cookie sessions, weryfikowanie tokenów



# Werkzeug



# IT'S DANGEROUS

... so better sign this

## > Historia

- # 2004 - powstanie Pocoo
- # 2007 - Werkzeug
- # 2008 - Jinja
- # 2010 - pierwsze wydanie Flask
- # 2016 - rozwiązanie Pocoo, powstanie Pallets
- # 2018 - wsparcie dla Bootstrap
- # 2024 - 3 lutego - najnowsze wydanie Flaska (wersja 3.0.2)



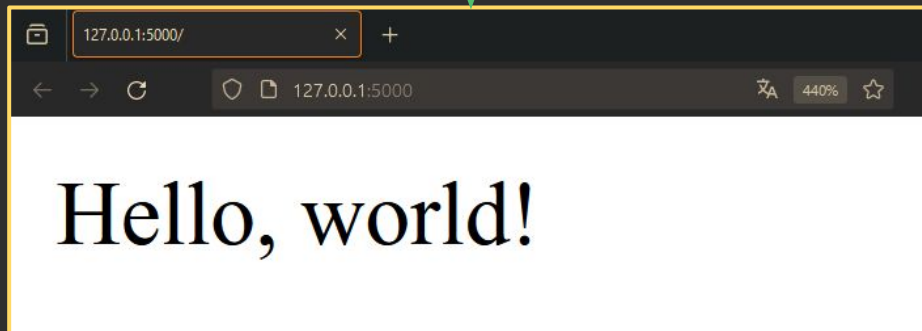
Armin Ronacher - twórca Flaska

## > Podstawowe działanie:

```
app.py > ...  
1  from flask import Flask  
2  
3  app = Flask(__name__)  
4  
5  
6  @app.route("/") ← Decorator  
7  def hello():  
8      return f"Hello, world!"  
9  
10 if __name__ == "__main__":  
11     with app.app_context():  
12         app.run(debug=True)  
13
```

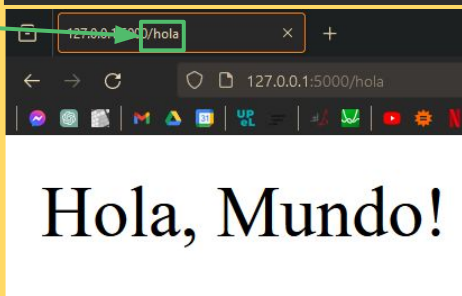
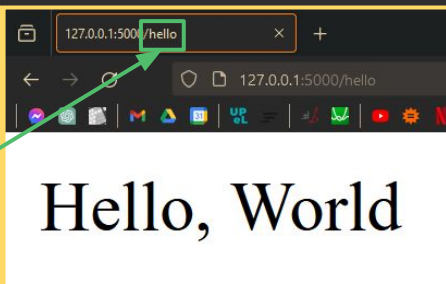
```
> python.exe .\app.py
```

```
* Running on http://127.0.0.1:5000  
Press CTRL+C to quit  
* Restarting with stat  
* Debugger is active!  
* Debugger PIN: 937-608-091
```

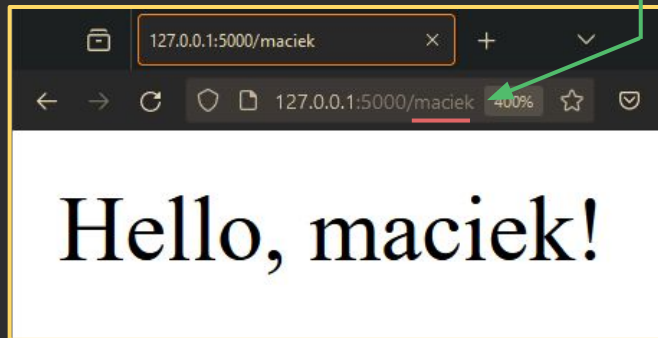


## > Routing i przechwytywanie wartości:

```
app.py > ...  
1 from flask import Flask  
2 from markupsafe import escape  
3  
4 app = Flask(__name__)  
5  
6  
7 @app.route('/hello')  
8 def hello_english():  
9     return 'Hello, World'  
10  
11  
12 @app.route('/hola')  
13 def hello_spanish():  
14     return 'Hola, Mundo!'  
15  
16  
17 if __name__ == "__main__":  
18     with app.app_context():  
19         app.run(debug=True)  
20
```



```
app.py > ...  
1 from flask import Flask  
2  
3 app = Flask(__name__)  
4  
5  
6 @app.route("/<name>")  
7 def hello(name):  
8     return f"Hello, {name}!"  
9
```



## > Szablony

# Ułatwiają pracę w plikach .html

# Oparte na silniku Jinja

# Przykładowe funkcjonalności:

- dziedziczenie
- wstrzykiwanie zmiennych
- pętle
- instrukcje warunkowe

✓ templates

<> base.html

<> contact.html

<> index.html

<> shop.html



## > Szablony - dziedziczenie

▼ static

# style.css

▼ templates

<> base.html

<> contact.html

<> index.html

<> shop.html

 app.py

templates > <> base.html > ...

```
1  <!DOCTYPE html>
2  <html lang="en">
3  > <head> ...
8  </head>
9  <body>
10     <div class="menu">
11         <ul>
12             <li><a href="/">Strona Główna</a></li>
13             <li><a href="shop">Sklep</a></li>
14             <li><a href="contact">Kontakt</a></li>
15         </ul>
16     </div>
17     <div class="centertext">
18         <h1>{{name}}</h1>
19     </div>
20     <div class="centertext">
21         {% block content %}
22         {% endblock %}
23     </div>
24 </body>
25 </html>
```


templates > <> index.html > ...

```
1  {% extends "base.html" %}
2  {% block content %}
3
4  {% for post in posts %}
5      <div class="post">
6          <h2>{{ post.title }}</h2>
7          <p>{{ post.content }}</p>
8      </div>
9  {% endfor %}
10 {% endblock %}
```

## > Szablony - wstrzykiwanie zmiennych

```
@app.route('/contact')
def contact():
    return render_template('contact.html', name="Kontakt", adres="Kalinowa 7", phone="530708858")
```

```
templates > <> base.html > ...
2   <html lang="en">
9   <body>
18  <div class="centertext">
19  |   <h1>{{name}}</h1>
20  | </div>
21  <div class="centertext">
22  |   {% block content %}
23  |   {% endblock %}
24  | </div>
25  </body>
26  </html>
```

```
templates > <> contact.html > ...
1   {% extends "base.html" %}
2
3   {% block content%}
4   <style>
5   |   body {
6   |       background-color:  rgb(0, 225, 255);
7   |   }
8   | </style>
9   | <p>Telefon:{{ phone }}</p>
10  | <p>Adres: {{adres}}</p>
11  | </div>
12  {% endblock %}
```

## > Szablony - pętle

```
26 @app.route('/')
27 def index():
28     return render_template('index.html', name="Strona główna", posts=posts_list)
```

templates > <> index.html > ...

```
1 {% extends "base.html" %}
2 {% block content%}
3
4 {% for post in posts %}
5 <div class="post">
6     <h2>{{ post.title }}</h2>
7     <p>{{ post.content }}</p>
8 </div>
9 {% endfor %}
10 {% endblock %}
```

```
posts_list = [
    {
        'title': 'Paragraf 1',
        'content': 'Rój pszczoł staje się n
    },
    {
        'title': 'Paragraf 2',
        'content': 'Jeżeli rój osiadł w cud
    },
    {
        'title': 'Paragraf 3',
        'content': 'Jeżeli rój osiadł w cud
    }
]
```

## > Szablony - instrukcje warunkowe

```
34 @app.route('/shop')
35 def shop():
36     return render_template('shop.html', name="Sklep", products=products_list)
```

templates > <> shop.html > ...

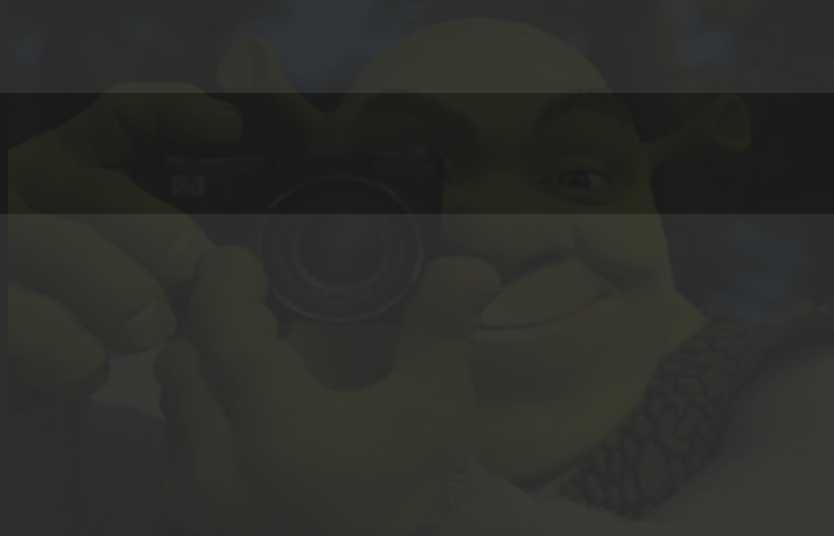
```
1  {% extends "base.html" %}
2  {% block content%}
3  <h3>Available Products</h3>
4  <ul>
5      {% for product in products %}
6          <li {% if product.stock == 0 %}class="not-available"{% endif %}>
7              {{ product.name }} - {{ product.price }}zł {% if product.stock == 0 %}(Niedostępny){% endif %}
8          </li>
9      {% endfor %}
10 </ul>
11 {% endblock %}
```

## > Message Flashing

# System flashowania w zasadzie umożliwia zapisanie wiadomości (do cookies) na końcu jednego requesta i dostęp do niej w następnym requestcie, (tylko w następnym).

```
@app.route('/like_facebook', methods=['POST'])
def like_facebook():
    flash('Dzięki za polubienie!')
    return redirect(url_for('contact'))
```

```
{% with messages = get_flashed_messages() %}
  {% if messages %}
    <ul>
      {% for message in messages %}
        <li>{{ message }}</li>
      {% endfor %}
    </ul>
  {% endif %}
{% endwith %}
```



## > HTML Escaping - MarkupSafe

```
app.py > ...
1  from flask import Flask
2
3  app = Flask(__name__)
4
5
6  @app.route("/")
7  def hello():
8      return f"<h1 style='color:red;'>ATAKUJE CIE!</h1>"
9
10
11 if __name__ == "__main__":
12     with app.app_context():
13         app.run(debug=True)
14
```

**ATAKUJE CIE!**

```
app.py > ...
1  from flask import Flask
2  from markupsafe import escape
3
4  app = Flask(__name__)
5
6
7  @app.route("/")
8  def hello():
9      return escape("<h1 style='color:red;'>ATAKUJE CIE!</h1>")
10
11
12 if __name__ == "__main__":
13     with app.app_context():
14         app.run(debug=True)
15
16
```

`<h1 style='color:red;'>ATAKUJE CIE!</h1>`



## > Obsługa metod HTTP, moduł request

```
app2.py > ...
1 from flask import Flask, render_template, request
2
3 app = Flask(__name__)
4
5
6 @app.route('/login', methods=['GET', 'POST'])
7 def login():
8     if request.method == 'POST':
9         # Logika sysetmu logowania
10
11         username = request.form['username']
12         password = request.form['password']
13         print(f"Username: {username}, Password: {password}")
14         return "Login successful!"
15
16     else:
17         return render_template('login_template.html')
18
19
20 if __name__ == '__main__':
21     app.run(debug=True)
22
```

Login

Witaj na naszej stronie!

Zaloguj się by uzyskać dostęp.

Username:

Password:

Login

Login successful!

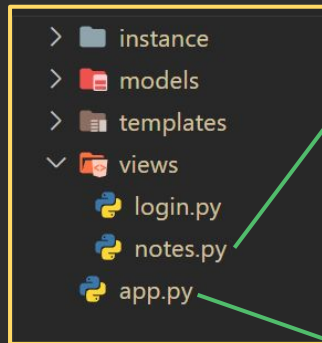
Username: martin, Password: potezne\_haslo123

[02/Apr/2024 23:31:05] "POST /login HTTP/1.1" 200 -

[02/Apr/2024 23:30:08] "GET /login HTTP/1.1" 200 -

## > Blueprints

- # **Modularność** - dzielą aplikację na moduły, które ułatwiają zarządzanie kodem
- # **Łatwa integracja** - “podpięcie” do aplikacji w jednej funkcji
- # **Wielokrotne użycie** - z jednego blueprinta może korzystać wiele aplikacji
- # **Niezależność** - mogą działać w izolacji od innych części aplikacji w osobnym, równoległym procesie



```
views > notes.py > ...
1  from flask import Blueprint, render_template,
2  from models.models import Note, db
3
4
5  bp = Blueprint('notes_bp', __name__)
6  |
7  @bp.route('/')
8  > def notes(): ...
16
17  @bp.route('/add_note', methods=['POST'])
18  > def add_note(): ...
32
33
```

```
from views.notes import bp as bp_notes
app.register_blueprint(bp_notes)
```



## > ORM, moduł SQLAlchemy

# Obsługa baz danych SQL z poziomu pythona:

- zestawianie sesji
- CRUD (Create, Read, Update, Delete)

# **Object-Relational Mapping** - mapowanie danych z bazy do zmiennych w aplikacji

# **Definiowanie atrybutów zmiennych:**

- **nullable** - czy rekord może być równy NULL
- **unique** - czy rekord musi być unikalny (np. login)
- **primary\_key** - rekord służący jako ID całego wpisu

czy jest true



## > WTForms - moduł flask-wtf

### # Walidacja i renderowanie formularzy:

- możemy przekazywać struktury formularzy bezpośrednio do szablonów gdzie później będą renderowane

### # Ochrona przed atakami CSRF (Cross-Site Request Forgery)



## > Sesje, moduł Session

# Generowanie klucza sesji - przechowywany w cookies

# Przechowywanie informacji o sesji w strukturze słownika:

- nazwa użytkownika
- ustawienia sesji
- preferencje

# Bezpieczeństwo - podpisywanie danych kluczem aby zapobiec ich modyfikacji

# Przekazywanie danych pomiędzy zapytaniami HTTP



## > Bezpieczeństwo - moduł Security

# Wprowadza mechanizmy bezpieczeństwa takie jak:

- login and registration (with tracking)
- authentication:
  - token based
  - session based
  - two-factor
- password hashing
- password recovery



## Flask-Security

Flask-Login

Flask-Mailman

Flask-Principal

Flask-WTF

itsdangerous

passlib


QRCode

webauthn

authlib

## > Bezpieczeństwo - logowanie

```
@app.route('/')
@login_required _____ panel logowania
def home():
    return render_template('index.html')
```



The screenshot shows a web browser window with the address bar containing the URL `http://127.0.0.1:5000/login?next=/`. The page has a yellow border and a dark header area. The main content area is white and contains the following elements:

- A large, bold, black heading "Login" on the left.
- A text input field labeled "Email" with a light blue border.
- A text input field labeled "Password" with a light blue border.
- A checkbox labeled "Remember Me" with an unchecked box.
- A rounded rectangular button labeled "Login" with a light blue border.

dostęp

# Welcome to the Home Page, jan@test.pl

You are logged in!

[Logout](#)

127.0.0.1:5000

## > Bezpieczeństwo - uprawnienia

```
@app.route('/admin')
@login_required
@roles_required('admin')
def admin_panel():
    return "Welcome to admin panel"
```

role == admin

**Welcome to admin panel**

127.0.0.1:5000/admin

role != admin

**Forbidden**

You don't have the permission to access the requested resource. It is either read-protected or not readable by the server.

127.0.0.1:5000/admin

## > Bezpieczeństwo - CSRF

```
csrf = CSRFProtect(app)
```

```
@app.route("/form", methods=['GET', 'POST'])  
def form():  
    if request.method == 'POST':  
        name = request.form['Name']  
        return (' Hello ' + name + '!!!')  
    return render_template('form.html')
```

```
<form action="{{ url_for('protected_form') }}" method="POST">  
    <label for="Name">Your Name Please ? </label>  
    <input type="text" name="Name">  
    <input type="hidden" name="csrf_token" value = "{{ csrf_token() }}" />  
    <button type="submit">Submit</button>  
</form>
```

```
<form action="{{ url_for('unprotected_form') }}" method="POST">  
    <label for="Name">Your Name Please ? </label>  
    <input type="text" name="Name">  
    <button type="submit">Submit</button>  
</form>
```

Your name: (secure)

Your name: (insecure)

Hello Gniewomir!!!

**Bad Request**

The CSRF token is missing.

## > Flask - czy warto?

### > Zalety:

- # very lightweight framework
- # niski próg wejścia, prostota
- # wsparcie dla wielu rozszerzeń,
- # pełna kompatybilność z WSGI

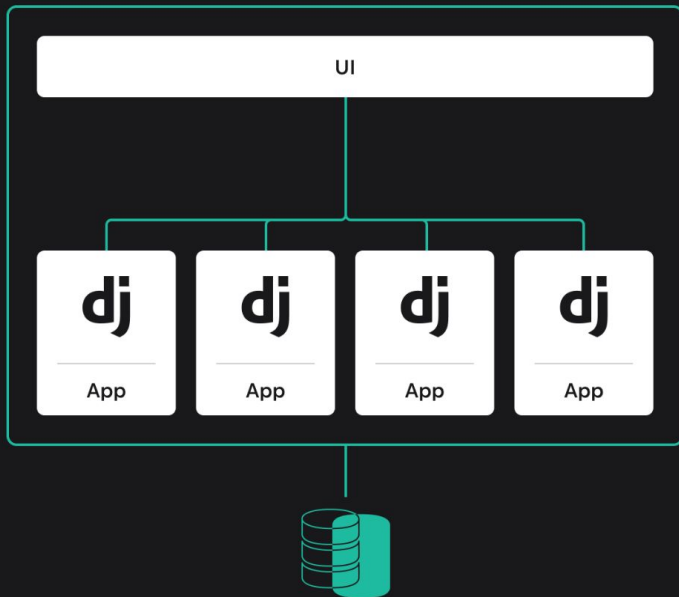
### > Wady:

- # mało wbudowanych funkcji
- # problemy ze skalowalnością
- # nieduża społeczność
- # słabo rozwinięte funkcje bezpieczeństwa



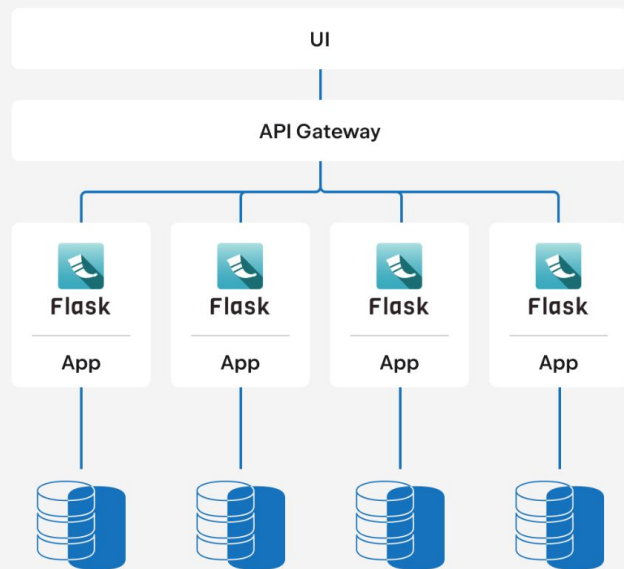
## > Flask vs. Django

Monolithic architecture



vs

Microservice architecture



## > Flask vs. Django

```
{% extends "base.html" %}

{% block content %}
    <h1>Hello, {{ username }}</h1>

    <ul>
        {% for product in product_list %}
            <li>{{ product.name }}: {{ product.price }}</li>
        {% empty %}
            <li>No products available.</li>
        {% endfor %}
    </ul>

    {{ my_date|date }}
{% endblock content %}
```

Django

```
{% extends "base.html" %}

{% block content %}
    <h1>Hello, {{ username }}</h1>

    <ul>
        {% for product in product_list %}
            <li>{{ product.name }}: {{ product.price }}</li>
        {% else %}
            <li>No products available.</li>
        {% endfor %}
    </ul>

    💡 {{ my_date.strftime() }}
{% endblock content %}
```

Jinja2

	> Flask	> Django
# Templates	Jinja2, format kompatybilny z wieloma innymi frameworkami	własny format Django
# URLs	Uproszczony system definiowania endpointów pisany w jednym pliku. Zgodny z architekturą REST	Bardziej skomplikowany system, zapewniający dużo lepszą skalowalność. Nie w pełni zgodny z architekturą REST
# Databases	Wsparcie ORM poprzez rozszerzenia, umożliwia używanie NoSQL	Wbudowane ORM, ciężko używać niewspieranych baz danych
# Authentication	Dostępne rozszerzenia, np. Flask-Admin, Flask-Login, Flask-Security	Wbudowane systemy, np. Django auth, Django admin
# Architecture	Stworzona z myślą o mikroserwisach, zgodna z REST, łatwo rozszerzalna do pewnego poziomu	Stworzona dla aplikacji monolitycznych. Lepiej dostosowana do dużych projektów.
# Learning Curve	Bardzo niski próg wejścia, prostą aplikację można stworzyć w bardzo krótkim czasie. Bardzo dobry framework do nauki podstawowych konceptów	Dużo wyższy próg wejścia. Konsekwentna architektura ułatwia późniejsze skalowanie projektów.

# Bibliografia

- <https://pythonistaplanet.com/what-can-you-do-with-flask/>
- <https://blog.jetbrains.com/pycharm/2023/11/django-vs-flask-which-is-the-best-python-web-framework/>
- [brat sztuczny](#)
- <https://flask.palletsprojects.com/en/3.0.x/>
- [https://pl.wikipedia.org/wiki/Flask\\_\(framework\)](https://pl.wikipedia.org/wiki/Flask_(framework))
- <https://kinsta.com/blog/flask-vs-django/>