

# **FLASK FRAMEWORK PRE PYTHON**

Metodiky s příklady

Vypracovali:

Tomáš Halgaš  
Dominika Molitorisová  
Svetlana Vojtková

## Obsah

Flask úvod .....	3
Nastavenie prostredia .....	3
Nastavenie virtuálneho prostredia.....	4
Príklad – Hello world! .....	5
Šablóny vo Flask (templates).....	7
Štruktúra priečinku projektu .....	7
Statické súbory .....	7
Príklad – Vykresľovanie šablón.....	7
Príklad – Šablóny zobrazujúce obsah premenných .....	8
Príklad – Šablóny s presmerovaním medzi stránkami.....	11
Príklad – Pridanie štýlov .....	13
Requirements.txt .....	14
Deployment flask aplikácie na linuxový server .....	15
Inštalácia Gunicorn .....	16
Prístup k flask aplikácii využitím Gunicorn.....	16
Použitie Gunicorn ako systemd služby .....	17
Konfigurácia Nginx.....	19
Inštalácia nginx balíka.....	19
Konfigurácia nginx .....	19
Záver .....	20
Deployment flask aplikácie využitím Heroku .....	20
Heroku .....	20
Postup deploymentu s Git.....	21
Postup deploymentu využitím GitHubu .....	24

## Flask úvod

Flask je využívaný na vytváranie webových aplikácií. Je to mikro framework pre Python založený na Werkzeug WSGI softvérového vybavenia a nástroja šablón Jinja2.

### Čo je to web framework (webový rámec)?

- Rámec webovej aplikácie je zbierka knižníc a modulov, ktoré pomáhajú vývojárom vytvoriť „business“ vrstvu bez toho, aby sa museli starať o protokol, správu vlákien, atď.

### Čo je to WSGI?

- Web Server Gateway Interface je štandard pre vývoj webových aplikácií v Pythone
- Špecifikácia pre univerzálne rozhranie medzi webovým serverom a webovými aplikáciami

### Čo je to Werkzeug?

- Werkzeug je jeden z pokročilejších modulov WSGI
- Obsahuje rôzne nástroje, ktoré uľahčujú vývoj webových aplikácií

### Čo je Jinja2?

- Nástroj na vykresľovanie šablón
- Vykresľuje webové stránky pre server s ľubovoľným obsahom
- Flask vykresľuje šablóny založené na HTML pomocou Jinja2

### **Výhody Flasku:**

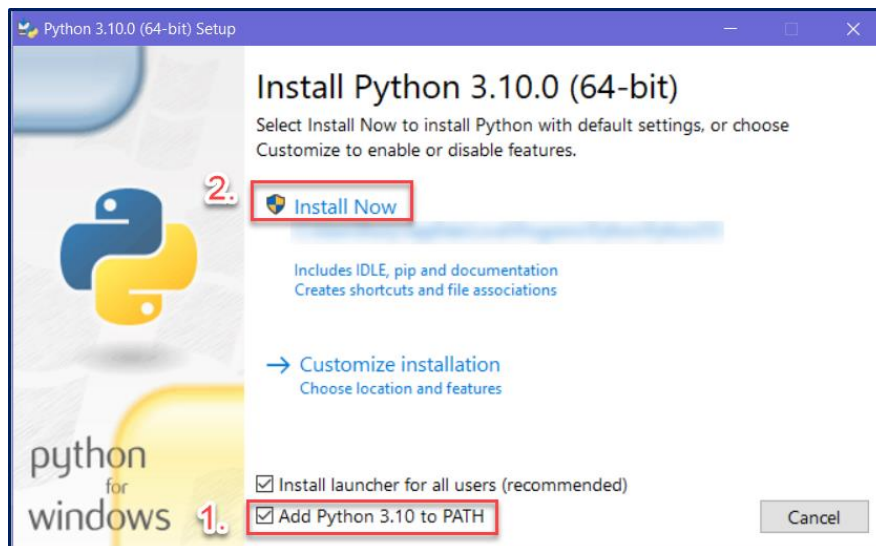
- Jednoduché použitie
- Voľnosť pri budovaní štruktúry webovej aplikácie

Aj keď Flask nemá presné pravidlá ako budovať aplikáciu, je potrebné dizajnovať ju tak, aby bolo možné orientovať sa v aplikácii aj pri zvýšenej komplexnosti.

## Nastavenie prostredia

- potrebná verzia Pythonu Python2.7 a viac

Odporúčame stiahnuť najnovšiu verziu (v čase písania 3.10) z <https://www.python.org/downloads/>. Pri inštalácii pre Windows potrebné zvoliť možnosť pre pridanie Pythonu do PATH (viď. obrázok).



Príkazom `$ pip -V` sa uistíte, že pip bolo nainštalované správne

```
C:\Users\svetl>pip -V
pip 22.0.3 from C:\Users\svetl\AppData\Roaming\Python\Python310\site-packages\pip (python 3.10)
```

## Nastavenie virtuálneho prostredia

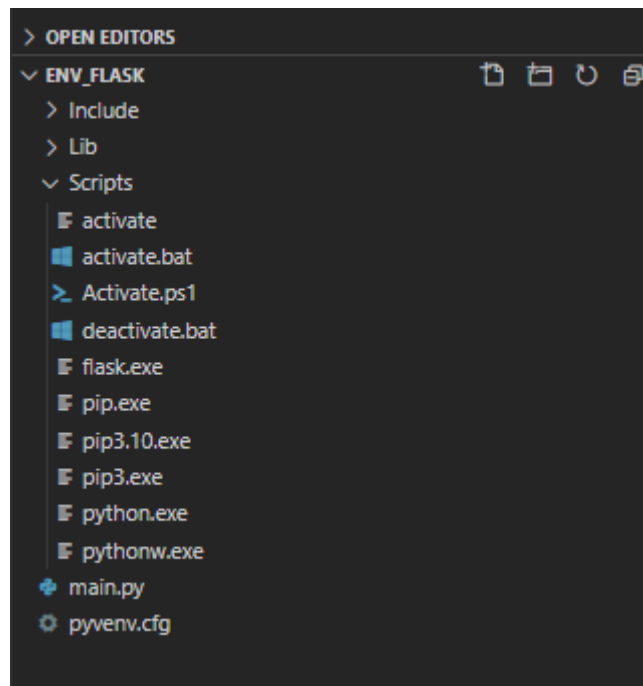
Využitie virtuálneho prostredia zaručuje v prípade operačného systému Linux, že Python na úrovni OS nebude narušený. A v prípade iných OS zaisťuje, že nastavenia a súbory sú vo virtuálnom prostredí nedotknuté, čo je užitočné, keď rôzne projekty potrebujú rôzne verzie balíčkov.

- Ak chcete pracovať s ,venv' musíte mať verziu Pythonu 3.2 a viac

Virtuálne prostredie vytvoríme príkazom `$ python -m venv .`

```
PS C:\Users\svetl\PycharmProjects\env_flask> python -m venv .
```

Keďže sme už v priečinku projektu použili sme `.,` čím sa vytvorí virtuálne prostredie v aktuálnom priečinku. Ak chcete vytvoriť svoje virtuálne prostredie na inom mieste, namiesto bodky je potrebné napísať celú cestu do cieľového adresára.



Po vykonaní príkazu sa nám vytvoria priečinky a súbory (Include, Lib\site-packages, Scripts, pyvenv.cfg...). Na aktivovanie virtuálneho prostredia je potrebné spustiť 'activate' súbor, ktorý spustí virtuálne prostredie. V tomto prípade sa 'activate' nachádza v priečinku Scripts a vieme ho spustiť príkazom `$ ./Scripts/activate`. Ako môžete vidieť na obrázku (env\_flask) pred príkazovým riadkom symbolizuje, že máme aktivované virtuálne prostredie. V prípade ak chceme vypnúť prostredie, môžeme to vykonať príkazom `$ deactivate`.

```
PS C:\Users\svetl\PycharmProjects\env_flask> ./Scripts/activate
(env_flask) PS C:\Users\svetl\PycharmProjects\env_flask> |
```

Flask nainštalujeme príkazom `$ pip install Flask`, vykonanie príkazu nainštaluje knižnicu Flask do prostredia v ktorom sa nachádzame.

```
PS C:\Users\svetl\PycharmProjects\env_flask> ./Scripts/activate
(env_flask) PS C:\Users\svetl\PycharmProjects\env_flask> pip install Flask
Collecting Flask
  Using cached Flask-2.1.1-py3-none-any.whl (95 kB)
Collecting Werkzeug>=2.0
  Using cached Werkzeug-2.1.1-py3-none-any.whl (224 kB)
Collecting Jinja2>=3.0
  Using cached Jinja2-3.1.1-py3-none-any.whl (132 kB)
Collecting itsdangerous>=2.0
  Using cached itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting click>=8.0
  Using cached click-8.1.2-py3-none-any.whl (96 kB)
Collecting colorama
  Using cached colorama-0.4.4-py2.py3-none-any.whl (16 kB)
Collecting MarkupSafe>=2.0
  Downloading MarkupSafe-2.1.1-cp310-cp310-win_amd64.whl (17 kB)
Installing collected packages: MarkupSafe, colorama, Werkzeug, Jinja2, itsdangerous, click, Flask
Successfully installed Flask-2.1.1 Jinja2-3.1.1 MarkupSafe-2.1.1 Werkzeug-2.1.1 click-8.1.2 colorama-0.4.4 itsdangerous-2.1.2
WARNING: You are using pip version 21.0.1; however, version 22.0.4 is available.
You should consider upgrading via the 'c:\users\svetl\pycharmprojects\env_flask\scripts\python.exe -m pip install --upgrade pip' command.
(env_flask) PS C:\Users\svetl\PycharmProjects\env_flask> |
```

## Príklad – Hello world!

Pre vytvorenie kódu v príklade je využívané Visual Studio Code. Vytvorili sme si najprv súbor 'main.py' a v ňom je nasledujúci kód (obrázok):

```

main.py x
main.py
1  from flask import Flask
2  app = Flask(__name__)
3
4
5  @app.route("/")
6  def hello():
7      return "Hello World!"
8
9
10 if __name__ == "__main__":
11     app.run()
12

```

Po uložení kódu, spustíme súbor nasledujúcim príkazom `$ python main.py`.

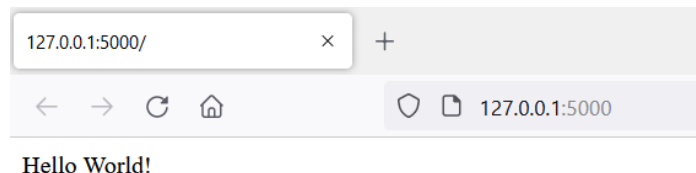
**Poznámka:** Nenazývajte súbor Flask, mohlo by to viesť ku konfliktu so samotným Flask.

```

(env_flask) PS C:\Users\svetl\PycharmProjects\env_flask> python main.py
* Serving Flask app 'main' (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: off
* Running on http://127.0.0.1:5000 (Press CTRL+C to quit)

```

Ako môžeme vidieť v konzole, súbor je spustený na adrese 127.0.0.1:5000. Na obrázku nižšie môžete vidieť ako vyzerá spomenutá localhost URL.



## Syntax kódu

1. „from flask import Flask“ - import z balíka Flask do súboru
2. „app = Flask(\_\_name\_\_)“ – pomocou tohto riadku vytvoríme objekt triedy Flask, argument „\_\_name\_\_“ pošleme predvolenému konštruktoru, čo pomôže Flasku hľadať šablóny a statické súbory
3. „@app.route(„/“)“  
     def hello():  
         return „Hello world!“ – dekodér trasy, umožní Flasku nasmerovať prevádzku stránky na funkciu (hello). Vykoná sa všetko, čo sa nachádza vo funkcii hello. V tomto prípade sa na stránku posiela výstup „Hello world!“.
4. V main funkcii „app.run()“ spustí server na lokálnom zariadení.

Na ladenie (debug) je možné pozmeniť `app.run(debug=True)`. Debuggovanie sleduje error a zobrazí ich. Stále keď je vykonaná zmena, aplikácia sa automaticky reštartuje a zmeny sú hneď vykonané.

```

10  if __name__ == "__main__":
11      app.run(debug=True)
12

```

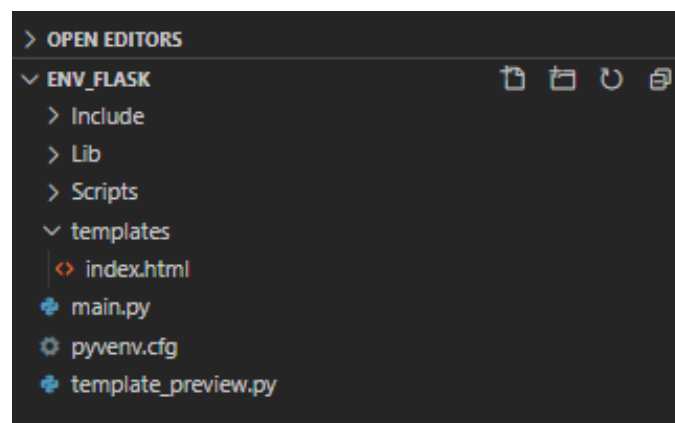
## Šablóny vo Flask (templates)

Systém webových šablón označuje zobrazenie stránky HTML koncovému používateľovi. To sa skladá z 3 častí:

- základný nástroj šablón v tomto prípade Jinja2
- zdroj údajov (HTML súbor)
- procesor šablón – Python umožní spracovanie šablón

## Štruktúra priečinku projektu

V priečinku projektu budete musieť vytvoriť priečinok 'templates' alebo 'Templates'. Táto štruktúra cesty je nevyhnutná, aby Jinja2 mohla identifikovať a spracovať HTML súbory.



## Statické súbory

Všetky statické súbory ako sú obrázky, videá, súbory CSS, JS musia byť rovnako ako HTML súbory umiestnené pod presným názvom priečinka – v tomto prípade 'static'.

## Príklad – Vykresľovanie šablón

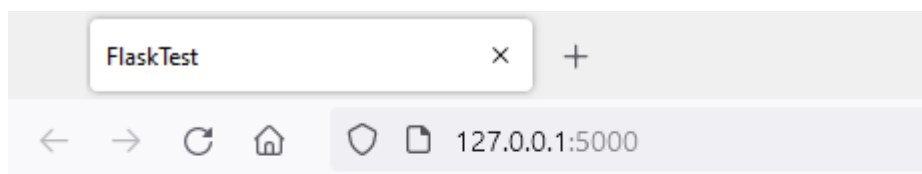
Flask funkcia `render_template()` pomáha vykresliť HTML súbor, ktorý je vo funkcií zadany. Na prvom obrázku môžete vidieť kód súboru `template_preview.py`, kde je využitá funkcia `render_template` s odkazom na `index.html`. Na ďalšom obrázku je zobrazený obsah súboru `index.html`.

```
template_preview.py X
template_preview.py
1 from flask import Flask, render_template
2
3 app = Flask(__name__)
4
5
6 @app.route('/')
7 def index():
8     return render_template('index.html')
9
10
11 if __name__ == "__main__":
12     app.run(debug=True)
13
```

```
index.html X
templates > index.html > html
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <title>FlaskTest</title>
6 </head>
7
8 <body>
9     <h2>Vitajte</h2>
10    <p>Flask: S využitím html šablóny(template)</p>
11
12 </body>
13
14
15 </html>
```

```
(env_flask) PS C:\Users\svetl\PycharmProjects\env_flask> python template_preview.py
```

Po vykonaní príkazu `$ python template_preview.py` sa zobrazí obsah HTML súboru, v našom prípade na url 127.0.0.1:5000. (viď obrázok Vitajte)



## Vitajte

Flask: S využitím html šablóny(template)

### Príklad – Šablóny zobrazujúce obsah premenných

Pre ďalšiu ukážku si vytvoríme nový HTML súbor v priečinku `templates` s názvom `welcome.html`. Pre jej zobrazenie potrebujeme definovať v Python súbore cestu k danej stránke. Vytvoríme na to novú funkciu `welcome()`, ktorá má anotáciu `@app.route('/welcome')`. Rovnako v predchádzajúcom príklade vracia funkciu `render_template()`, teraz s parametrom novo vytvoreného súboru `welcome.html`.



```

template_preview.py X
template_preview.py
1  from flask import Flask, render_template
2
3  app = Flask(__name__)
4
5
6  @app.route('/')
7  def index():
8      return render_template('index.html')
9
10 @app.route('/welcome')
11 def welcome():
12     return render_template('welcome.html')
13
14
15 if __name__ == "__main__":
16     app.run(debug=True)

```

Flask šablóny nám umožňujú zobrazovať na stránke aj obsahy premenných. Pre ich využitie je potrebné pridať do HTML kódu odkaz na premennú umiestnenú v {{ ... }}. Premennú, ktorú chceme posunúť HTML súboru uvedieme ako parameter metódy *render\_template()*. Príklad:

- Python súbor:

```

@app.route('/welcome')
def welcome():
    username = 'Peto Velky'
    return render_template('welcome.html', username = username)

```

- HTML súbor:

```

template_preview.py welcome.html X
templates > welcome.html > div.userObject
1  <!doctype html>
2  <title>Welcome Page</title>
3
4  <div class="username"><b>Vitaj</b> {{username}}</div>
5

```

Okrem takýchto typov premenných môžeme v HTML zobraziť aj obsah definovaného poľa (napr. čísla). HTML súboru ho pošleme rovnakým spôsobom ako atribút v metóde *render\_template()*. Avšak v HTML súbore chceme prvky poľa vypisovať po jednom – každé číslo vo vlastnom elemente. Na to je potrebné použiť slučku, kt. ich vypíše všetky od prvého po posledné. Vo Flask šablónach sa na to používa syntax:

```

{% for prvok in pole %}

<p> {{ prvok }} </p>

{% endfor %}

```

V prípade, že HTML súboru pošleme pole *array*, tak náš HTML kód môže vyzeráť nasledovne:

```

<div class="array">
    {% for number in array %}
    <p>Cislo {{ number }}</p>
    {% endfor %}
</div>

```

Ďalším príkladom je využitie podmieneného zobrazovania, t. j. daný obsah sa objaví len ak je splnené určitá podmienka (resp. je možné pridať, čo sa zobrazí v opačnom prípade). Syntax je nasledovná:

```

{% if a > 2 %}

<p> {{ a }} je vacsie ako 2 </p>

```

```
{% else %}
```

```
<p> {{ a }} je mensie ako 2 </p>
```

```
{% endif %}
```

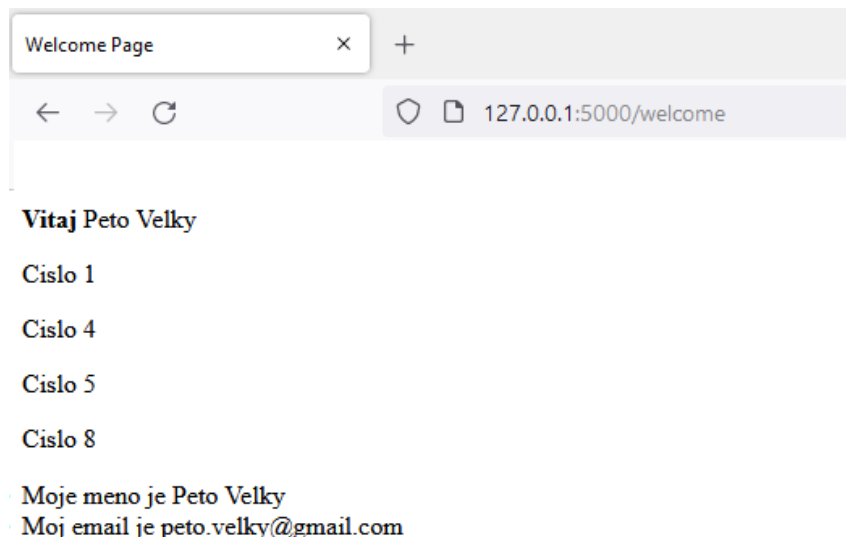
V našom príklade môžeme podmienky zobraziť pre vypísanie objektu užívateľa (tvorí ho jeho meno a email) – premenná *userObject*. Následne by celý Python súbor vyzeral nasledovne:

```
template_preview.py X welcome.html
template_preview.py
1 from flask import Flask, render_template
2
3 class Person:
4     def __init__(self, name, email):
5         self.name = name
6         self.email = email
7
8 app = Flask(__name__)
9
10 @app.route('/')
11 def index():
12     return render_template('index.html')
13
14 @app.route('/welcome')
15 def welcome():
16     username = 'Peto Velky'
17     array = [1,4,5,8]
18     userObject = Person(username, "peto.velky@gmail.com")
19     return render_template('welcome.html', username = username, array = array, userObject = userObject)
20
21 if __name__ == "__main__":
22     app.run(debug=True)
```

Obsah HTML súboru je potrebné upraviť pridaním podmienky vypísania údajov o užívateľovi, iba ak má premenná priradenú hodnotu (*userObject* sa nerovná hodnote *None*). Výsledok vyzerá takto:

```
template_preview.py X welcome.html X
templates > welcome.html > ...
1 <!doctype html>
2 <title>Welcome Page</title>
3
4 <div class="username"><b>Vitaj</b> {{username}}</div>
5
6 <div class="array">
7     {% for number in array %}
8     <p>Cislo {{ number }}</p>
9     {% endfor %}
10 </div>
11
12 <div class="userObject">
13     {% if userObject %}
14     <li><span> Moje meno je {{ userObject.name }}</span>
15     <li><span>Moj email je {{ userObject.email }}</span>
16
17     {% else %}
18     <p>Nemam user object</p>
19     {% endif %}
20 </div>
21
```

Po spustení skriptu príkazom `$ python template_preview.py` sa nami vytvorený obsah zobrazí na URL <http://127.0.0.1:5000/welcome>. Ukážka je na nasledujúcom obrázku.



## Príklad – Šablóny s presmerovaním medzi stránkami

V predchádzajúcom príklade sme vytvorili 2 HTML súbory, avšak zo základnej stránky sme sa nevedeli dostať na stránku welcome, a naopak. V tomto prípade sa pozrieme na to, ako je možné zabezpečiť presmerovanie medzi stránkami využitím Flasku.

V úlohe chceme vytvoriť webstránku s 3 podstránkami, pričom všetky stránky budú mať rovnakú štruktúru a meniť sa bude len obsah hlavnej časti. Flask umožňuje vytvorenie **base šablóny**, v ktorej sa definujú štruktúra stránky. Obsah súboru `website_base.html` (tiež vytvorený v priečinku `templates`) je na obrázku:

```
base.html X
templates > base.html > ...
1
2 <!DOCTYPE html>
3 <html lang="en">
4 <head>
5 |   <meta charset="UTF-8">
6 |   <title>{% block title %} {% endblock %} - Flask</title>
7 </head>
8 <body>
9 |   <nav>
10 |     <a href="#">Home</a>
11 |     <a href="#">About</a>
12 |     <a href="#">Articles</a>
13 |   </nav>
14
15   <hr>
16
17   <div class="content">
18 |     {% block content %} {% endblock %}
19   </div>
20 </body>
21 </html>
22
```

Dôležité je pochopiť tieto 2 časti Jinja šablónovacieho systému, kt. budú v ďalších šablónach menené:

- `{% block title %} {% endblock %}` - slúži ako placeholder pre názov stránky. V ďalších šablónach nebudeme potrebovať prepisovať celý obsah `<head>` elementu.
- `{% block content %} {% endblock %}` - ďalší placeholder, kt. predstavuje hlavný obsah stránky a bude menený v šablónach potomkov = tie šablóny, ktoré extendujú (rozširujú) `website_base.html`

Teraz môžeme vytvoriť prvú šablónu, ktorá bude rozširovať `website_base.html`. Používa sa na to syntax `{% extends 'website_base.html' %}` a špecifický obsah umiestnime medzi `{% block content %} {% endblock %}`. Prvá šablóna reprezentujúca domovskú stránku `website_home.html` vyzerá nasledovne:

```
website_home.html x
templates > website_home.html > ...
1  {% extends 'website_base.html' %}
2
3  {% block content %}
4      <h1>{% block title %} Domov {% endblock %}</h1>
5      <h2>Vitaj na uvodnej stranke!</h2>
6      <h3>Sucasny cas je: {{ utc_dt }}</h3>
7  {% endblock %}
8
```

V riadku 6 si chceme zobrazíť aktuálny čas pomocou premennej `utc_dt`. Obsah premennej si pošleme z Python súboru (potrebne vytvoriť nový skript `template_website.py`). V ňom potrebujeme získať knižnicu **datetime** pomocou `import datetime`. Následne premennú `utc_dt` definujeme ako `datetime.datetime.utcnow()`, čo predstavuje aktuálny čas. Nový Python súbor vyzerá nasledovne:

```
template_website.py x
template_website.py
1  from flask import Flask, render_template
2  import datetime
3
4  app = Flask(__name__)
5
6  @app.route('/')
7  def home():
8      return render_template('website_home.html', utc_dt=datetime.datetime.utcnow())
9
10
11 if __name__ == "__main__":
12     app.run(debug=True)
```

Spustení príkazu `$ python template_website.py` sa hlavná stránka nachádza na URL <http://127.0.0.1:5000/>. Ukážka je na nasledujúcom obrázku.



Podobným spôsobom si vytvoríme aj šablóny pre stránky **About** a **Articles** (nová šablóna a metóda). Obe HTML šablóny extendujú *base.html* a menia obsah hlavnej časti. Ak si po pridaní týchto dvoch stránok vyskúšame prekliknutie z našej úvodnej stránky na sekciu About alebo Articles (viď. obrázok vyššie), tak zistíme, že presmerovanie nefunguje. Dôvodom je, že v *base.html* odkaz (*href* atribút) zatiaľ nedokazuje na dané stránky. Flask na to využíva syntax `{{ url_for('nazov_metody') }}`. Názov metódy pre domovskú stránku je v našom prípade *home*. Správne presmerovanie vyzerá:

```
<a href="{{ url_for('home') }}">Home</a>
<a href="{{ url_for('about') }}">About</a>
<a href="{{ url_for('articles') }}">Articles</a>
```

**Poznámka:** Ak je použité presmerovanie ako znázornené vyššie, je nutné aby sa metóda v Python skripte pre sekciu About nazývala *about* a pre Articles mala názov *articles*.

## Príklad – Pridanie štýlov

Vytvorená stránka nemá žiadne štýly, a preto sa na ich pridanie sústreďí tento príklad. Najprv chceme pridať štýly pre navigačnú lištu, ktorá je rovnaká na všetkých stránkach. Vytvoríme si súbor *style.css* a umiestnime ho do priečinku *static*. V navigácii chceme zmeniť len rozloženie elementov, farbu a veľkosť textu. Na to postačí takýto kód:

```
static > style.css > ...
1  nav {
2      display: flex;
3  }
4
5  nav a {
6      color: lightseagreen;
7      font-size: 3em;
8      text-decoration: none;
9      margin-left: auto;
10     margin-right: auto;
11 }
12
```

Tento súbor so štýlmi je potrebné prepojiť s *website\_base.html*. Použijeme na to nasledujúci kód, kt. umiestnime na 3. riadok súboru:

```
<link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
```

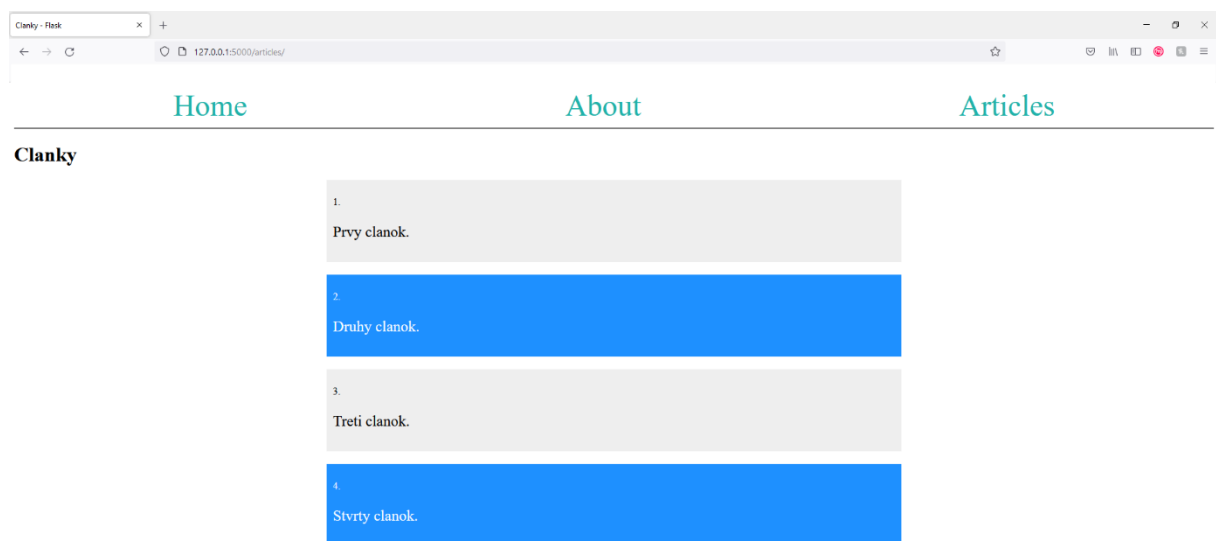
Ďalej chceme pridať štýly pre sekciu s článkami, avšak štýlu budú podmienené určitou podmienkou. Z toho dôvodu budú štýly pridané priamo do *website\_articles.html*. Predtým však potrebujeme definovať pole s článkami v Python súbore:

```
@app.route('/articles/')
def articles():
    articles = ['Prvy clanok.',
                'Druhy clanok.',
                'Treti clanok.',
                'Stvrty clanok.'
               ]
    return render_template('website_articles.html', articles = articles)
```

Ak už máme dáta, ktoré chceme v sekcii zobraziť, môžeme prejsť k pridaniu štýlov. Povedzme, že našim cieľom je, aby každý článok obsahoval poradové číslo a mal rôznu farbu pozadia a textu na základe poradia čísla (sivé pozadie a čierny text pre články s párnym poradovým číslom, modré pozadie a biely text pre články s párnym poradovým číslom). Môžeme to docieľiť takýmto zápisom:

```
website_articles.html X
templates > website_articles.html > ...
1 {% extends 'website_base.html' %}
2
3 {% block content %}
4 <h1>{% block title %} Clanky {% endblock %}</h1>
5 <div style="width: 50%; margin: auto">
6     {% for article in articles %}
7         {% if loop.index % 2 == 0 %}
8             {% set bg_color = '#1E90FF' %}
9             {% set txt_color = 'white' %}
10        {% else %}
11            {% set bg_color = '#eee' %}
12        {% endif %}
13
14        <div style="padding: 10px; margin: 20px; background-color: {{ bg_color }}; color: {{ txt_color }}">
15            <p>{{ loop.index }}.</p>
16            <p style="font-size: 24px">{{ article }}</p>
17        </div>
18    {% endfor %}
19 </div>
20 {% endblock %}
21
```

Ako v predchádzajúcich príkladoch využijeme for cyklus a na základe poradia, kt. určuje *loop.index* využitím *{% set atribut = hodnota %}* nastavíme premenné. Tie následne použijeme v štýle daného elementu. Výsledok je zobrazený na ďalšom obrázku.



## Requirements.txt

Tento súbor pozostáva z knižníc a ich špecifických verzií, ktoré je potrebné nainštalovať pre správne fungovanie projektu. Použitím príkazu *\$ pip freeze* sa nám zobrazia všetky doteraz nainštalované knižnice a ich verzie v prostredí, v ktorom sa nachádzame. Ak príkaz má tvar *\$ pip freeze > requirements.txt*, tieto balíčky budú zapísané vo vygenerovanom súbore *requirements.txt*.

```
C:\Users\svetl\PycharmProjects\env_flask> pip freeze > requirements.txt
```

```
requirements.txt X
requirements.txt
1 click==8.1.2
2 colorama==0.4.4
3 Flask==2.1.1
4 itsdangerous==2.1.2
5 Jinja2==3.1.1
6 MarkupSafe==2.1.1
7 Werkzeug==2.1.1
8
```

Príklad výstupu requirements.txt

#### Zdroje:

<https://www.geeksforgeeks.org/flask-creating-first-simple-application/>

<https://www.mygreatlearning.com/blog/everything-you-need-to-know-about-flask-for-beginners/>

<https://www.digitalocean.com/community/tutorials/how-to-use-templates-in-a-flask-application>

## Deployment flask aplikácie na linuxový server

Po naprogramovaní funkčnej aplikácie ju môžeme nasadiť do produkcie. Vstavaný Flask server **nie je vhodný do produkcie**, ale iba na vývoj a debuggovanie, pretože nie je dobre škálovateľný. Pre nasadenie využijeme **nginx** a **gunicorn**. V tejto metodike si teda prejdeme proces nasadenia flask aplikácie na linuxový server (predpokladáme, že už máte linuxový server). Použijeme gunicorn ako WSGI server na komunikáciu s našou flask aplikáciou a Nginx ako proxy server medzi serverom gunicorn a klientom.

Nasadzovanie webových aplikácií v Pythone sa opiera o **WSGI**, čo je štandardné pythonové rozhranie pre komunikáciu medzi webovou aplikáciou a webovým serverom definované v [PEP 333](#). Prevažná väčšina webových frameworkov v Pythone toto rozhranie implementuje priamo, prípadne na tento účel obsahuje wrapper.

Je teda jedno, či používate **Flask**, Pyramid, Django, Bottle alebo Falcon, vašu aplikáciu vždy predstavuje application objekt, ktorý sa navonok chová rovnako. Webové frameworky implementujú aplikačnú časť WSGI. Rovnako existujú webové servery, ktoré implementujú serverovú časť WSGI, napríklad **Gunicorn** alebo mod\_wsgi pre httpd (Apache). Tieto servery vedia pracovať s application objektom a nezaujímajú ich, v akom frameworku je aplikácia napísaná.

My sme konkrétne mali nasledujúce prostredie:

- OS: Ubuntu 20.04.3 LTS
- Python verzia: 3.8.10
- Flask verzia: 2.1.1
- Gunicorn verzia: 20.1.0
- Nginx verzia: nginx/1.18.0 (Ubuntu)

**Poznámka:** pred inštaláciou ktoréhokoľvek z balíkov sa uistite, že pracujete vo vytvorenom virtuálnom prostredí (postup pre vytvorenie venv je uvedený vyššie), aby ste mali oddelené rôzne verzie balíkov. Po aktivácii virtuálneho prostredia sme pripravení nainštalovať balíky, ktoré potrebujeme.

## Inštalácia Gunicorn

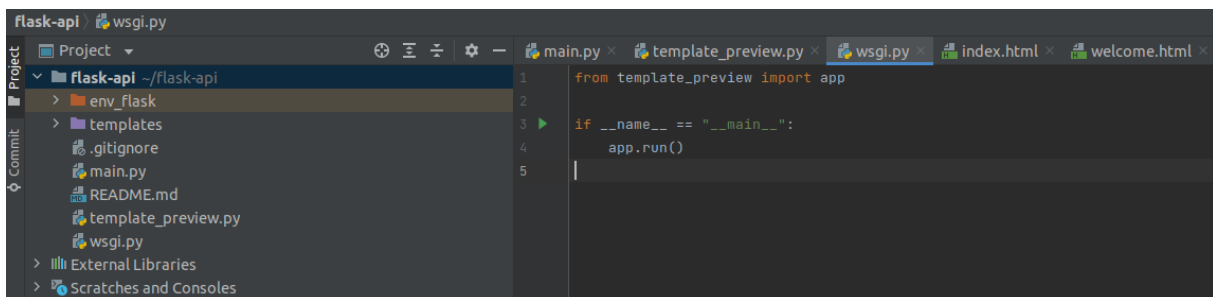
Najnovšiu verziu Gunicorn sme nainštalovali príkazom: ***pip install gunicorn***

```
(env_flask) (base) dominika@dominika-pc:~/flask-api$ pip install gunicorn
Collecting gunicorn
  Using cached gunicorn-20.1.0-py3-none-any.whl (79 kB)
Requirement already satisfied: setuptools>=3.0 in ./env_flask/lib/python3.8/site-packages (from gunicorn) (60.2.0)
Installing collected packages: gunicorn
Successfully installed gunicorn-20.1.0
```

Najprv musíme vytvoriť python súbor, ktorý bude gunicorn používať ako vstupný bod do našej aplikácie, pomenujeme ho wsgi.py:

```
from template_preview import app
```

```
if __name__ == "__main__":
    app.run()
```



## Prístupujete k flask aplikácii využitím Gunicorn

Gunicorn ponúka veľa možností príkazového riadka, ktoré možno použiť na vyladenie výkonu servera tak, aby zodpovedal vašim potrebám. Najčastejšie používané možnosti sú:

- -w na určenie počtu workrov, ktoré server použije
- --bind ktoré špecifikujú rozhranie a port, na ktorý sa má server viazať (0.0.0.0 bude verejná IP adresa vášho servera)

Teraz môžeme otestovať gunicorn server a zistiť, či dokáže spustiť aplikáciu flask. Pomocou nasledujúceho príkazu spustíte server gunicorn so 4 pracovníkmi -w (počet pracovníkov môžete zvýšiť alebo znížiť v závislosti od špecifikácií vášho servera). Je potrebné zadať rozhranie a port, na ktorý sa má server viazať pomocou voľby príkazového riadka --bind:

```
gunicorn -w 4 --bind 0.0.0.0:8000 wsgi:app
```

Ak uvidíte podobný output ako na tomto obrázku, znamená to, že váš server beží.



```
(env_flask) (base) dominika@dominika-pc:~/flask-api$ gunicorn -w 4 --bind 0.0.0.0:8000 wsgi:app
[2022-04-04 21:35:37 +0200] [30613] [INFO] Starting gunicorn 20.1.0
[2022-04-04 21:35:37 +0200] [30613] [INFO] Listening at: http://0.0.0.0:8000 (30613)
[2022-04-04 21:35:37 +0200] [30613] [INFO] Using worker: sync
[2022-04-04 21:35:37 +0200] [30615] [INFO] Booting worker with pid: 30615
[2022-04-04 21:35:37 +0200] [30616] [INFO] Booting worker with pid: 30616
[2022-04-04 21:35:37 +0200] [30617] [INFO] Booting worker with pid: 30617
[2022-04-04 21:35:37 +0200] [30618] [INFO] Booting worker with pid: 30618
```

Ako je vidieť vo vyššie uvedenom obrázku, server gunicorn počúva na porte 8000 localhost a 4 pracovníci začali, každý s iným ID procesu (PID). Ak navštívite 0.0.0.0:8000/, uvidíte koreňový adresár vašej webovej stránky (rovnako ako s webovou aplikáciou python na porte 5000 predtým).

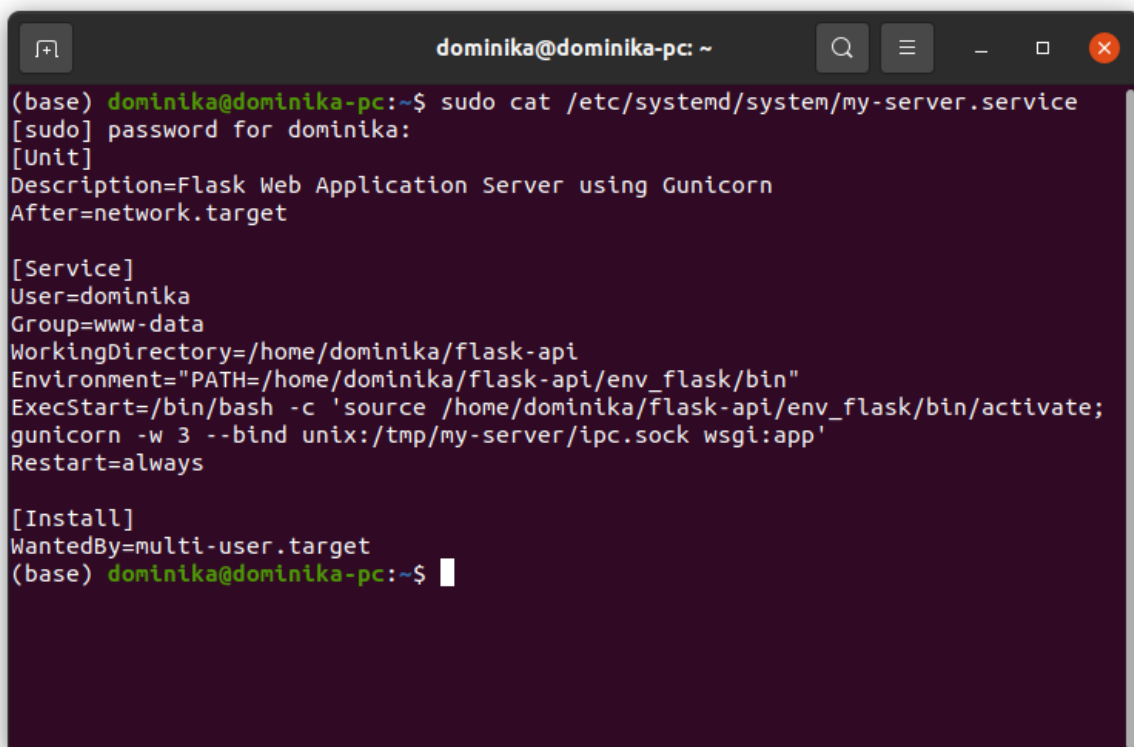
## Použitie Gunicorn ako systemd služby

Teraz musíme premeniť gunicorn na službu, aby sa spúšťala pri automatickom štarte servera a ak z nejakého dôvodu zlyhá, systemd ju reštartuje.

Najprv odídeme z virtuálneho prostredia použitím príkazu deactivate.

```
(env_flask) (base) dominika@dominika-pc:~/flask-api$ deactivate
(base) dominika@dominika-pc:~/flask-api$
```

Teraz vytvoríme službu s ľubovoľným názvom, my sme ju nazvali my-server.service. Službu môžete nazvať ako chcete, ale **nezabudnite pridať .service na koniec**. Obsah my-server.service je nasledovný:



```
(base) dominika@dominika-pc:~$ sudo cat /etc/systemd/system/my-server.service
[sudo] password for dominika:
[Unit]
Description=Flask Web Application Server using Gunicorn
After=network.target

[Service]
User=dominika
Group=www-data
WorkingDirectory=/home/dominika/flask-api
Environment="PATH=/home/dominika/flask-api/env_flask/bin"
ExecStart=/bin/bash -c 'source /home/dominika/flask-api/env_flask/bin/activate;
gunicorn -w 3 --bind unix:/tmp/my-server/ipc.sock wsgi:app'
Restart=always

[Install]
WantedBy=multi-user.target
(base) dominika@dominika-pc:~$
```

Vy si upravte hodnoty User, WorkingDirectory, Environment a ExecStart podľa vášho prostredia. Obsah súboru my-server.service nájdete aj na našom gite. Súbor má nasledujúce časti:

- **[Unit]** sekcia sa používa na špecifikáciu metadát a závislostí. Taktiež obsahuje popis našej služby. Taktiež sme nastavili, že táto služba sa spustí až po spustení sieťovej služby (network.target).
- V **[Service]** sekcii určíme používateľa a skupinu, pod ktorou chceme proces spustiť. Zadať svoje bežné používateľské meno (v mojom prípade dominika) a ako skupinu uveďte www-data, aby Nginx mohol komunikovať s procesmi Gunicorn. Ďalej nastavte WorkingDirectory tak, aby odkazoval na adresár, v ktorom je vaša flask aplikácia. Nastavte premennú prostredia PATH, aby init systém vedel, že spustiteľné súbory pre proces sa nachádzajú vo vašom virtuálnom prostredí. ExecStart obsahuje príkaz, ktorý sa použije na spustenie služby, v tomto prípade sme použili príkaz bash na aktiváciu virtuálneho prostredia a spustenie gunicornu s 3 workrami. Naš gunicorn server naviažeme (binding) na unix:/tmp/my-server/ipc.sock. Ide o soket na serveri, ktorý gunicorn používa na IPC (interpersonal communication). Nginx použije tento soket na komunikáciu s gunicornom. Socket je vytvorený pri spustení príkazu a odstránený, keď je proces z akéhokoľvek dôvodu zabitý a zakaždým je iný, preto sme ho umiestnili do adresára /tmp.
- **[Install]** sekcia hovorí systemd s čím má túto službu prepojiť, ak jej povolíte spustenie pri štarte. My chceme, aby sa služba spustila, keď bude spustený bežný multi-usr systém.

Pred spustením služby musíme najprv vytvoriť my-server/directory v /tmp (vy si môžete adresár aj gunicorn IPC súbor pomenovať ako chcete), príkazom:

```
mkdir /tmp/my-server
```

Teraz povolíme (enable) a spustíme službu. Tento príkaz umožní automatické reštartovanie služby po reboote a tiež spustí službu pre aktuálnu reláciu:

```
sudo systemctl enable my-server --now
```

```
Terminal: Local x + v
(base) dominika@dominika-pc:~/flask-api$ sudo systemctl enable my-server --now
[sudo] password for dominika:
(base) dominika@dominika-pc:~/flask-api$
```

Skontrolujte stav svojej služby a uistite sa, že je aktívna a spustená bez errorov pomocou príkazu:

```
sudo systemctl status my-server.service
```

```
(base) dominika@dominika-pc:~/flask-api$ sudo systemctl status my-server.service
● my-server.service - Flask Web Application Server using Gunicorn
   Loaded: loaded (/etc/systemd/system/my-server.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2022-04-05 19:31:10 CEST; 2s ago
     Main PID: 18256 (gunicorn)
        Tasks: 4 (limit: 19004)
       Memory: 58.5M
     CGroup: /system.slice/my-server.service
             └─18256 /home/dominika/flask-api/env_flask/bin/python /home/dominika/flask-api/env_flask/bin/gunicorn -w 3 --bind unix:/tmp/my-server/ipc.sock wsgi:app
               └─18258 /home/dominika/flask-api/env_flask/bin/python /home/dominika/flask-api/env_flask/bin/gunicorn -w 3 --bind unix:/tmp/my-server/ipc.sock wsgi:app
                 └─18259 /home/dominika/flask-api/env_flask/bin/python /home/dominika/flask-api/env_flask/bin/gunicorn -w 3 --bind unix:/tmp/my-server/ipc.sock wsgi:app
                   └─18260 /home/dominika/flask-api/env_flask/bin/python /home/dominika/flask-api/env_flask/bin/gunicorn -w 3 --bind unix:/tmp/my-server/ipc.sock wsgi:app

apr 05 19:31:10 dominika-pc systemd[1]: Started Flask Web Application Server using Gunicorn.
apr 05 19:31:10 dominika-pc gunicorn[18256]: [2022-04-05 19:31:10 +0200] [18256] [INFO] Starting gunicorn 20.1.0
apr 05 19:31:10 dominika-pc gunicorn[18256]: [2022-04-05 19:31:10 +0200] [18256] [INFO] Listening at: unix:/tmp/my-server/ipc.sock (18256)
apr 05 19:31:10 dominika-pc gunicorn[18256]: [2022-04-05 19:31:10 +0200] [18256] [INFO] Using worker: sync
apr 05 19:31:10 dominika-pc gunicorn[18258]: [2022-04-05 19:31:10 +0200] [18258] [INFO] Booting worker with pid: 18258
apr 05 19:31:10 dominika-pc gunicorn[18259]: [2022-04-05 19:31:10 +0200] [18259] [INFO] Booting worker with pid: 18259
apr 05 19:31:10 dominika-pc gunicorn[18260]: [2022-04-05 19:31:10 +0200] [18260] [INFO] Booting worker with pid: 18260
(base) dominika@dominika-pc:~/flask-api$
```

## Konfigurácia Nginx

### Inštalácia nginx balíka

Nginx si nainštalujeme príkazom:

```
sudo apt install nginx
```

```
(base) dominika@dominika-pc:~/flask-api$ sudo apt install nginx
```

### Konfigurácia nginx

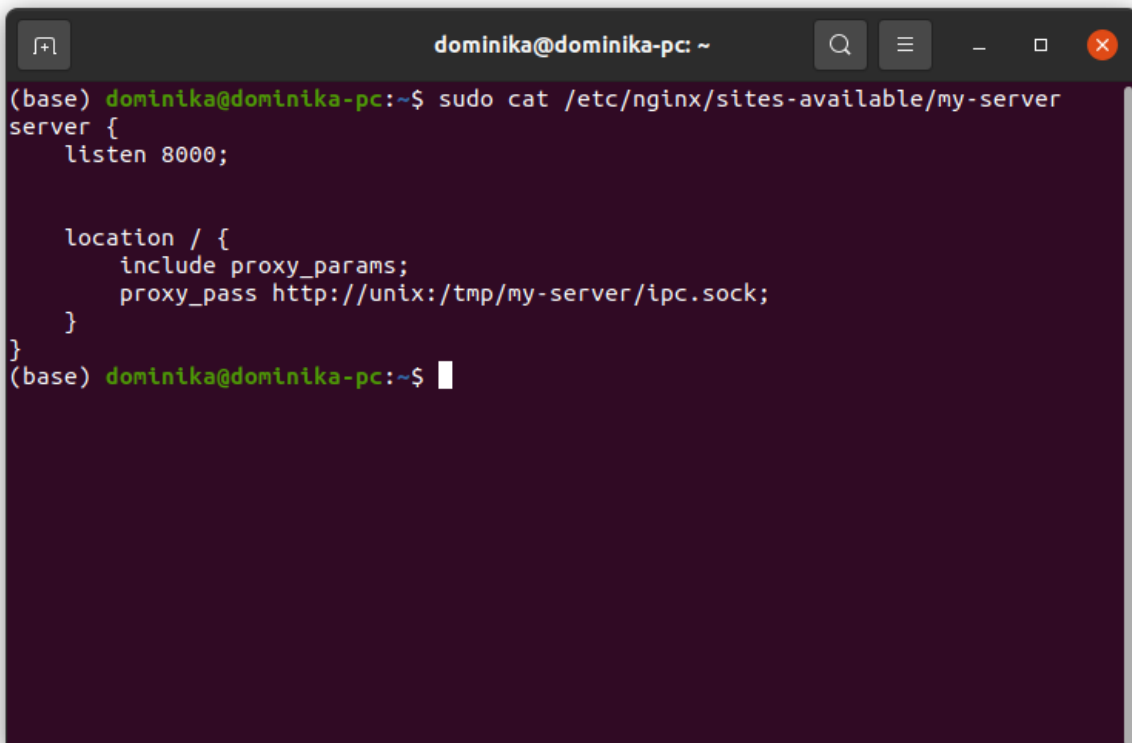
Potom prejdite do adresára nginx:

```
cd /etc/nginx/
```

```
(base) dominika@dominika-pc:~/flask-api$ cd /etc/nginx/  
(base) dominika@dominika-pc:/etc/nginx$
```

Tento adresár obsahuje všetky súbory súvisiace s nginx. Teraz musíme vytvoriť konfiguračný súbor, vďaka ktorému bude nginx fungovať ako proxy pre našu aplikáciu flask. Hlavným konfiguračným súborom je súbor s názvom nginx.conf, podľa konvencie sa tohto súboru nedotýkajú vývojári ani správcovia systému. Nové konfiguračné súbory sa vytvárajú v adresári sites-available/ a potom sa prepoja so /sites-enabled/ adresárom. Vytvoríme si teda nový súbor v adresári sites-available/:

```
sudo nano sites-available/my-server
```



The screenshot shows a terminal window titled 'dominika@dominika-pc: ~'. The user has executed the command 'sudo cat /etc/nginx/sites-available/my-server' to create a new configuration file. The output shows the following content being written to the file:

```
server {  
    listen 8000;  
  
    location / {  
        include proxy_params;  
        proxy_pass http://unix:/tmp/my-server/ipc.sock;  
    }  
}
```

The prompt '(base) dominika@dominika-pc:~\$' is visible at the bottom of the terminal, indicating the command has been executed successfully.

Potom spustíte príkaz `sudo nginx -t` aby ste sa uistili, že konfiguračný súbor je ok.

```
(base) dominika@dominika-pc:/etc/nginx$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

Vytvorte symbolický odkaz na tento súbor do sites-enabled adresára:

```
sudo ln -s /etc/nginx/sites-available/my-server /etc/nginx/sites-enabled/
```

```
(base) dominika@dominika-pc:/etc/nginx$ sudo ln -s /etc/nginx/sites-available/my-server /etc/nginx/sites-enabled/
(base) dominika@dominika-pc:/etc/nginx$ sudo ls -l /etc/nginx/sites-enabled/
total 0
lrwxrwxrwx 1 root root 34 apr  5 19:37 default -> /etc/nginx/sites-available/default
lrwxrwxrwx 1 root root 36 apr  5 20:26 my-server -> /etc/nginx/sites-available/my-server
```

Ak je všetko v poriadku, spustíte príkaz `sudo nginx -s reload`.

Poznámka, tento príkaz nám vyhadzoval error. Spustením týchto 4 príkazov sa to nejak opravilo a fungovalo to bez problémov ([užitočný link](#)):

```
systemctl daemon-reload
```

```
sudo nginx -s reload
```

```
systemctl restart nginx
```

```
systemctl status nginx.service
```

## Záver

Prešli sme si procesom nasadenia flask aplikácie pomocou gunicorn a nginx. Teraz by ste mali byť schopní otestovať Nginx s Gunicorn na `http://127.0.0.1:8000/` v akomkoľvek webovom prehliadači. Systemd je nastavený.

## Deployment flask aplikácie využitím Heroku

### Heroku

[Heroku](#) je kontajnerová cloudová platforma ako služba (PaaS – Platform as a service). Je často využívaná developermi na nasadenie, správu a škálovanie aplikácií. Heroku je plne spravovaná platforma, takže vy sa môžete sústrediť na vývoj aplikácie a nemusíte sa starať o údržbu servrov, hardvér a ani infraštruktúru.

Aplikácie sú nasadené na dynos, dynos „srdce Heroku“ sú linuxové kontajnere, do ktorých sa nasadzujú aplikácie. Keď sa zaregistrujete do Heroku, automaticky získate niekoľko bezplatných dyno hodín, ktoré môžete použiť pre svoje aplikácie. Keď je vaša aplikácia spustená, spotrebuje dynohodiny. Keď je nečinná, tak aplikácia prestáva spotrebovávať dynohodiny. Overením svojho účtu získate 1 000 bezplatných dyno hodín. Účet si overíte tak, že zadáte svoju kreditnú kartu. Neoverené účty získajú 550 hodín zadarmo. Ak nebudete využívať platenú službu, nebudú vám účtované žiadne poplatky. Overenie účtu poskytuje aj ďalšie výhody vrátane spustenia viac ako 5 bezplatných aplikácií, ako aj bezplatné vlastné názvy domén. Určite si však preštudujte podmienky a cenník na [tomto odkaze](#).

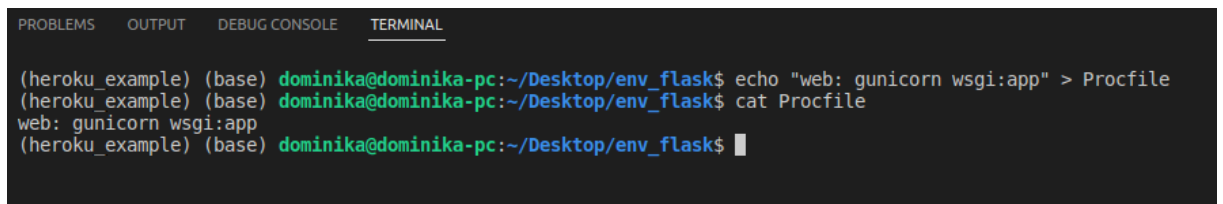
## Postup deploymentu s Git

Heroku spravuje nasadzovanie aplikácií pomocou Git, populárneho systému na správu verzií. Prerekvizity:

- Git ([postup](#))
- Heroku CLI ([postup](#))

Pracujeme v priečinku, kde sa nachádza naša Flask aplikácia. Uistite sa, že pracujete vo virtuálnom prostredí. My pracujeme vo virtuálnom prostredí a názvom heroku\_example, ktoré sme si aktivovali týmto príkazom: **source heroku\_example/bin/activate**

1. Nainštalujte si gunicorn príkazom : **pip install gunicorn**.
2. Vytvorte (aktualizujte) súbor requirements.txt so zoznamom závislostí projektu príkazom: **pip freeze > requirements.txt**. Pri nasadzovaní aplikácie sa súbor requirements.txt použije na to, aby ste Heroku informovali, ktoré balíky musia byť nainštalované, aby sa spustil kód vašej aplikácie.
3. Vytvorte súbor s názvom Procfile v koreňovom adresári projektu. Tento súbor povie Heroku, ako spustiť aplikáciu (my ako entry point do našej aplikácie využijeme wsgi.py). Môžete ho vytvoriť spustením nasledujúceho príkazu: `echo "web: gunicorn wsgi:app" > Procfile`. Názov súboru musí začínať veľkým písmenom. Tento súbor hovorí Heroku, aby slúžil vašej aplikácii pomocou servra Gunicorn.



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
(heroku_example) (base) dominika@dominika-pc:~/Desktop/env_flask$ echo "web: gunicorn wsgi:app" > Procfile
(heroku_example) (base) dominika@dominika-pc:~/Desktop/env_flask$ cat Procfile
web: gunicorn wsgi:app
(heroku_example) (base) dominika@dominika-pc:~/Desktop/env_flask$
```

4. Zmeny v súboroch nášho projektu budeme trackovať využitím Git-u (populárny systém na správu verzií). Takže ako prvý krok by ste mali pre svoj projekt vytvoriť Git repozitár. Môžete to dosiahnuť vykonaním nasledujúceho príkazu v adresári vášho projektu: **git init**. Uvedený príkaz inicializuje repozitár, ktorý sa použije na trackovanie súborov projektu. Metadáta repozitáru sú uložené v skrytom adresári s názvom .git/. Niektoré priečinky by ste **nemali** zahrnúť do vášho Git repozitára, napríklad heroku\_example/ a \_\_pycache\_\_/. Gitu môžete povedať, aby ich ignoroval, vytvorením súboru s názvom .gitignore. Na vytvorenie tohto súboru použite nasledujúce príkazy: **echo heroku\_example > .gitignore** a **echo \_\_pycache\_\_ >> .gitignore**. Potom zadáme príkazy **git add .** a následne **git commit -m „First commit“**. Teraz sledujete kód svojej aplikácie v miestnom repozitári Git. Zatiaľ neexistuje na žiadnych vzdialených serveroch.
5. Vytvorte si Heroku účet, ktorý je zdarma. Po vyplnení požadovaných údajov a po potvrdení vašej emailovej adresy budete môcť začať využívať Heroku.

# Sign up for free and experience Heroku today

## Free account

Create apps, connect databases and add-on services, and collaborate on your apps, for free.

## Your app platform

A platform for apps, with app management & instant scaling, for development and production.

## Deploy now

Go from code to running app in minutes. Deploy, scale, and deliver your app to the world.

First name \*

Last name \*


Email address \*

Company name

Role \*

Country/Region \*

Primary development language \*

☐ I'm not a robot
 

[Privacy](#) - [Terms](#)

**CREATE FREE ACCOUNT**

Signing up signifies that you have read and agree

- Heroku CLI (command line interface), ktorý slúži na vytváranie a spravovanie Heroku aplikácií cez terminál (ak potrebujete pomoc s inštaláciou, pozrite si [dokumentáciu](#)). Prihlásite sa do Heroku CLI príkazom **heroku login**. Tým sa otvorí webová stránka s tlačidlom na dokončenie procesu prihlásenia. Po prihlásení ste pripravení začať používať Heroku CLI na správu aplikácií.

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
(heroku_example) (base) dominika@dominika-pc:~/Desktop/env_flask$ heroku login
heroku: Press any key to open up the browser to login or q to exit:
Opening browser to https://cli-auth.heroku.com/auth/cli/browser/4473dd4e-4bf5-414d-b5a4-ee6db2766c17?requestor=SFMyNTY.g2gDbQAAAAw4OC4yMTIuMTkuMTluBQAQRuEygAFIAAFRgA.IzNEVofu5D
ISHGgIqXQIACKVZFAXycVscs6aXF0.Nuu
Logging in... done
Logged in as domca.moly@gmail.com
(heroku_example) (base) dominika@dominika-pc:~/Desktop/env_flask$

```

- Git remotes sú verzie vášho repozitára, ktoré žijú na iných serveroch. Aplikáciu nasadíte tak, že jej kód pushnete do špeciálneho remote hosteného v Heroku, ktorý je spojený s vašou aplikáciou. Aplikáciu môžete vytvoriť v Heroku spustením nasledujúceho príkazu: **heroku create nazov-appky**. Tento príkaz vytvorí novú prázdnu aplikáciu na Heroku spolu s pridruženým prázdny Git repozitárom. Ak spustíte tento príkaz z koreňového adresára vašej

aplikácie, prázdny Heroku Git repozitár sa automaticky nastaví ako remote pre vaše lokálne úložisko.

```
(heroku_example) (base) dominika@dominika-pc:~/Desktop/env_flask$ heroku create heroku-example-atic
Creating heroku-example-atic... done
https://heroku-example-atic.herokuapp.com/ | https://git.heroku.com/heroku-example-atic.git
(heroku_example) (base) dominika@dominika-pc:~/Desktop/env_flask$
```

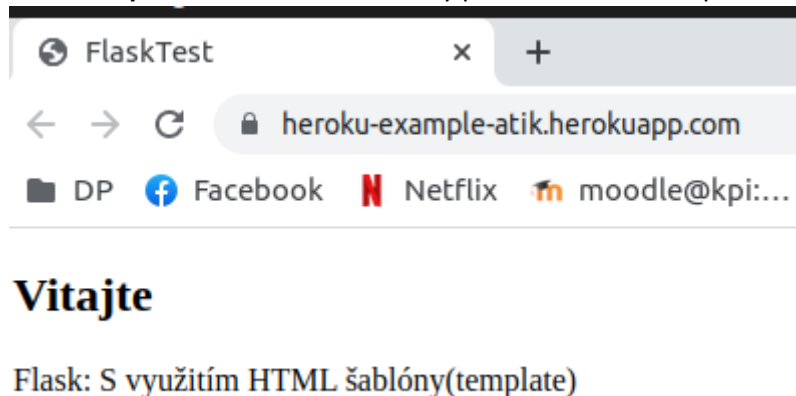
8. Pomocou príkazu **git remote -v** si môžete potvrdiť, že pre vašu aplikáciu bolo nastavené remote s názvom heroku.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
(heroku_example) (base) dominika@dominika-pc:~/Desktop/env_flask$ git remote -v
heroku https://git.heroku.com/heroku-example-atic.git (fetch)
heroku https://git.heroku.com/heroku-example-atic.git (push)
(heroku_example) (base) dominika@dominika-pc:~/Desktop/env_flask$
```

9. Ak chcete nasadiť svoju aplikáciu do Heroku, použite príkaz git push na pushnutie kódu z main vetvy lokálneho repozitára do vášho heroku remote: **git push heroku main**

```
(heroku_example) (base) dominika@dominika-pc:~/Desktop/env_flask$ git push heroku master
Enumerating objects: 946, done.
Counting objects: 100% (946/946), done.
Delta compression using up to 8 threads
Compressing objects: 100% (917/917), done.
Writing objects: 100% (946/946), 3.12 MiB | 932.00 KiB/s, done.
Total 946 (delta 52), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: ----> Building on the Heroku-20 stack
remote: ----> Determining which buildpack to use for this app
remote: ----> Python app detected
remote: ----> No Python version was specified. Using the buildpack default: python-3.10.4
remote: To use a different version, see: https://devcenter.heroku.com/articles/python-runtimes
remote: ----> Installing python-3.10.4
remote: ----> Installing pip 22.0.4, setuptools 60.10.0 and wheel 0.37.1
remote: ----> Installing SQLite3
remote: ----> Installing requirements with pip
remote: Collecting click==8.1.2
remote: Downloading click-8.1.2-py3-none-any.whl (96 kB)
remote: Collecting Flask==2.1.1
remote: Downloading Flask-2.1.1-py3-none-any.whl (95 kB)
remote: Collecting gunicorn==20.1.0
remote: Downloading gunicorn-20.1.0-py3-none-any.whl (79 kB)
remote: Collecting importlib-metadata==4.11.3
remote: Downloading importlib_metadata-4.11.3-py3-none-any.whl (18 kB)
remote: Collecting itsdangerous==2.1.2
remote: Downloading itsdangerous-2.1.2-py3-none-any.whl (15 kB)
remote: Collecting Jinja2==3.1.1
remote: Downloading Jinja2-3.1.1-py3-none-any.whl (132 kB)
remote: Collecting MarkupSafe==2.1.1
remote: Downloading MarkupSafe-2.1.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (25 kB)
remote: Collecting Werkzeug==2.1.1
remote: Downloading Werkzeug-2.1.1-py3-none-any.whl (224 kB)
remote: Collecting zipp==3.8.0
remote: Downloading zipp-3.8.0-py3-none-any.whl (5.4 kB)
remote: Installing collected packages: zipp, Werkzeug, MarkupSafe, itsdangerous, gunicorn, click, Jinja2, importlib-metadata, Flask
remote: Successfully installed Flask-2.1.1 Jinja2-3.1.1 MarkupSafe-2.1.1 Werkzeug-2.1.1 click-8.1.2 gunicorn-20.1.0 importlib-metadata-4.11.3 itsdangerous-2.1.2 zipp-3.8.0
remote: ----> Discovering process types
remote: Procfile declares types -> web
remote: ----> Compressing...
remote: Done: 65.3M
remote: ----> Launching...
remote: Released v3
remote: https://heroku-example-atic.herokuapp.com/ deployed to Heroku
remote: Verifying deploy... done.
To https://git.heroku.com/heroku-example-atic.git
* (new branch) master -> master
(heroku_example) (base) dominika@dominika-pc:~/Desktop/env_flask$
```

10. Zadajte príkaz **heroku open**. Otvorí sa vám webový prehliadač s vašou aplikáciou.

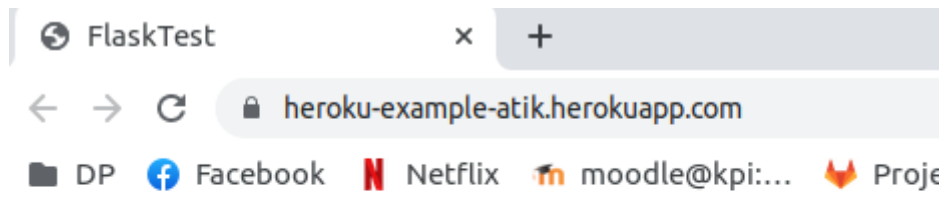




**Poznámka:** Ako redeploynúť aplikáciu po zmene v kóde? Dajme tomu, že sme zmeny vykonali v index.html. Spustením nasledujúcich príkazov našu aplikáciu znovu buildneme a deployneme. Tento postup opakujeme vždy, keď chceme nasadiť novú verziu aplikácie.

```
(heroku_example) (base) dominika@dominika-pc:~/Desktop/env_flask$ git add templates/index.html
(heroku_example) (base) dominika@dominika-pc:~/Desktop/env_flask$ git commit -m "changed description"
[master 22f580e] changed description
1 file changed, 1 insertion(+), 1 deletion(-)
(heroku_example) (base) dominika@dominika-pc:~/Desktop/env_flask$ git push heroku master
```

Po znovunačítaní už vidíme našu aplikáciu aj so zmeneným textom.



## Vitajte

Flask: TOTO JE NOVÝ FLASK S využitím HTML šablóny(template)

## Postup deploymentu využitím GitHubu

Integrácia Heroku s GitHub uľahčuje nasadenie kódu z GitHubu do aplikácie bežiacej na Heroku. Keď je integrácia GitHub nakonfigurovaná pre Heroku aplikáciu, Heroku dokáže automaticky buildovať a releasovať na základe pushov do konkrétneho GitHub repozitára.

Prerekvizity:

- vytvorený GitHub repozitár pre váš projekt

My sme si vytvorili [GitHub repo](#) s názvom flask\_app, kde sa nachádza naša príkladná flask aplikácia. Pracujeme v priečinku, kde sa nachádza naša Flask aplikácia. Uistite sa, že pracujete vo virtuálnom prostredí. My pracujeme vo virtuálnom prostredí a názvom heroku\_exe, ktoré sme si aktivovali týmto príkazom: **source heroku\_ex/bin/activate**

1. Nainštalujte si gunicorn príkazom : **pip install gunicorn.**
2. Vytvorte (aktualizujte) súbor requirements.txt so zoznamom závislostí projektu príkazom: **pip freeze > requirements.txt.** Pri nasadzovaní aplikácie sa súbor requirements.txt použije na to, aby ste Heroku informovali, ktoré balíky musia byť nainštalované, aby sa spustil kód vašej aplikácie.
3. Vytvorte súbor s názvom Procfile v koreňovom adresári projektu. Tento súbor povie Heroku, ako spustiť aplikáciu (my ako entry point do našej aplikácie využijeme wsgi.py). Môžete ho vytvoriť spustením nasledujúceho príkazu: echo "web: gunicorn wsgi:app" > **Procfile**. Názov súboru musí začínať veľkým písmenom. Tento súbor hovorí Heroku, aby slúžil vašej aplikácii pomocou servra Gunicorn.



- Niektoré priečinky by ste **nemali** zahrnúť do vášho GitHub repozitára, napríklad `heroku_example/` a `__pycache__/`. Gitu môžete povedať, aby ich ignoroval, vytvorením súboru s názvom `.gitignore`. Na vytvorenie tohto súboru použijete nasledujúce príkazy: **`echo heroku_example > .gitignore`** a **`echo __pycache__ >> .gitignore`**
- Teraz tieto zmeny commitnite a pushnite do vášho Github repozitára.
- Následne choďte na Heroku, prihláste sa a vytvorte novú aplikáciu.

Create New App

App name

flask-example-atic

flask-example-atic is available

Choose a region

Europe

Add to pipeline...

Create app

- Choďte na záložku Deploy a vyberte prepojenie na váš GitHub repozitár.

Overview

Resources

Deploy

Metrics

Activity

Access

Settings

Add this app to a pipeline

Create a new pipeline or choose an existing one and add this app to a stage in it.

Add this app to a stage in a pipeline to enable additional features

Pipelines let you connect multiple apps together and **promote code** between them. [Learn more.](#)

Pipelines connected to GitHub can enable **review apps**, and create apps for new pull requests. [Learn more.](#)

Choose a pipeline

Deployment method

Heroku Git

Use Heroku CLI

GitHub

Connect to GitHub

Container Registry

Use Heroku CLI

Connect to GitHub

Connect this app to GitHub to enable code diffs and deploys.

View your code diffs on GitHub

Connect your app to a GitHub repository to see commit diffs in the activity log.

Deploy changes with GitHub

Connecting to a repository will allow you to deploy a branch to your app.

Automatic deploys from GitHub

Select a branch to deploy automatically whenever it is pushed to.

Create review apps in pipelines

Pipelines connected to GitHub can enable **review apps**, and create apps for new pull requests. [Learn more.](#)

Connect to GitHub

- Vyberte možnosť pre automatický deployment vašej aplikácie. To umožní, aby bol váš kód automaticky nasadený vždy, keď vykonáte zmeny vo vybranej vetve (napr. main).

Poznámka: neboli sme schopní ukázať túto možnosť, pretože aktuálne (12.4.2022) má Heroku problém s integráciou s GitHubom ([issue](#)).