**FLASK FRAMEWORK PRE PYTHON**

Metodiky s príkladmi

Vypracovali:

Tomáš Halgaš

Dominika Molitorisová

Svetlana Vojtková

**Obsah**

[Flask úvod 3](#_Toc101520522)

[Nastavenie prostredia 3](#_Toc101520523)

[Nastavenie virtuálneho prostredia 4](#_Toc101520524)

[Príklad – Hello world! 5](#_Toc101520525)

[Šablóny vo Flask (templates) 7](#_Toc101520526)

[Štruktúra priečinku projektu 7](#_Toc101520527)

[Statické súbory 7](#_Toc101520528)

[Príklad – Vykresľovanie šablón 7](#_Toc101520529)

[Príklad – Šablóny zobrazujúce obsah premenných 8](#_Toc101520530)

[Príklad – Šablóny s presmerovaním medzi stránkami 11](#_Toc101520531)

[Príklad – Pridanie štýlov 13](#_Toc101520532)

[Requirements.txt 14](#_Toc101520533)

[Deployment flask aplikácie na linuxový server 15](#_Toc101520534)

[Inštalácia Gunicorn 16](#_Toc101520535)

[Prístupujte k flask aplikácii využitím Gunicorn 16](#_Toc101520536)

[Použitie Gunicorn ako systemd služby 17](#_Toc101520537)

[Konfigurácia Nginx 19](#_Toc101520538)

[Inštalácia nginx balíka 19](#_Toc101520539)

[Konfigurácia nginx 19](#_Toc101520540)

[Záver 20](#_Toc101520541)

[Deployment flask aplikácie využitím Heroku 20](#_Toc101520542)

[Heroku 20](#_Toc101520543)

[Postup deploymentu s Git 21](#_Toc101520544)

[Postup deploymentu využitím GitHubu 24](#_Toc101520545)

# Flask úvod

Flask je využívaný na vytváranie webových aplikácií. Je to mikro framework pre Python založený na Werkzeug WSGI softvérového vybavenia a nástroja šablón Jinga2.

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 aplikácií aj pri zvýšenej komplexnosti.

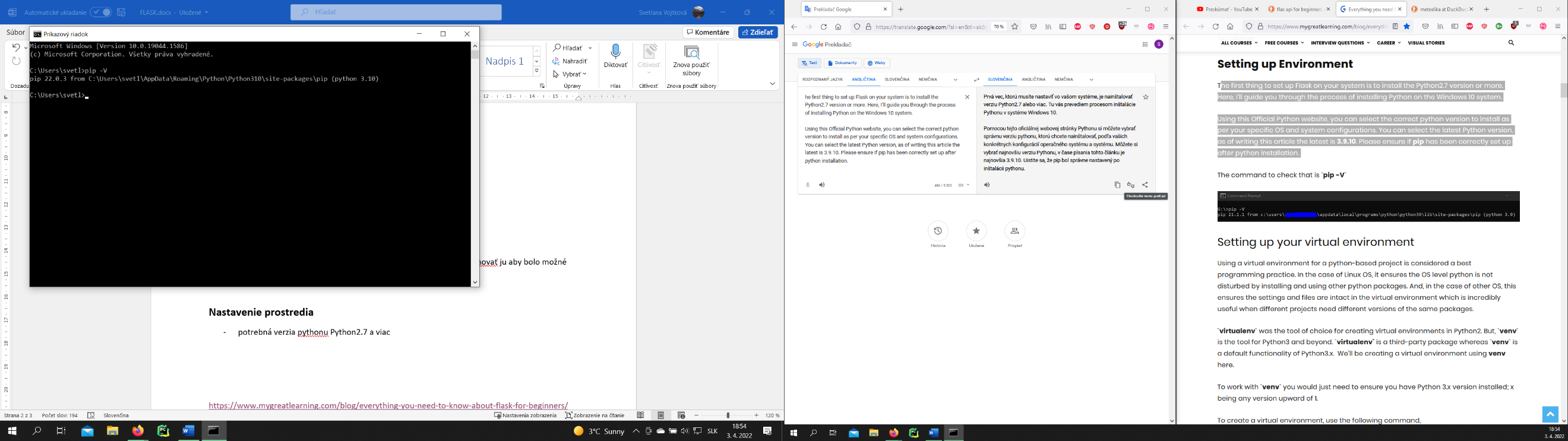
## Nastavenie prostredia

* 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 naištalované správne



## 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 .*



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 potrebne napísať celú cestu do cieľového adresára.

Obrázok, na ktorom je text

Automaticky generovaný popis

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.*

Obrázok, na ktorom je text

Automaticky generovaný popis

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

Obrázok, na ktorom je text

Automaticky generovaný popis

## 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):

Obrázok, na ktorom je text, obrazovka, snímka obrazovky

Automaticky generovaný popis

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

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

Obrázok, na ktorom je text

Automaticky generovaný popis

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.

Obrázok, na ktorom je text

Automaticky generovaný popis

**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!““ – dekóder trasy, umožní Flasku nasmerovať prevádzku stránky na funkciu (hello). Vykoná sa všetko, čo sa nachádza vo funkcií 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 errory a zobrazí ich. Stále keď je vykonaná zmena, aplikácia sa automaticky reštartne a zmeny sú hneď vykonané.

Obrázok, na ktorom je text, obrazovka, snímka obrazovky

Automaticky generovaný popis

# 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.

**Obrázok, na ktorom je text

Automaticky generovaný popis**

## 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 zadaný. 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.

**Obrázok, na ktorom je text

Automaticky generovaný popisObrázok, na ktorom je text, snímka obrazovky, monitor, obrazovka

Automaticky generovaný popis**

**Obrázok, na ktorom je text

Automaticky generovaný popis**

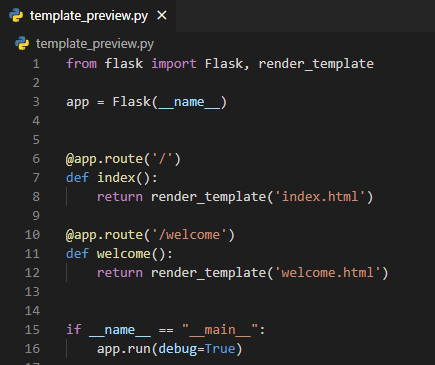
Po vykonaní príkazu *$ python template\_preview.py* sa zobrazí obsah HTML súboru, v našom prípade na url 127.0.0.1:5000. (vid obrazok Vitajte)

**Obrázok, na ktorom je text

Automaticky generovaný popis**

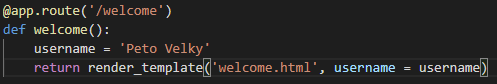
## Príklad – Šablóny zobrazujúce obsah premenných

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*.

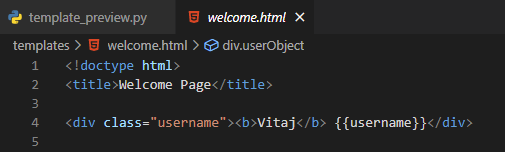


Flask šablóny nám umožňujú zobrazovať na stránke aj obsahy premenných. Pre ich využitie je potrebné pridať do HTML kódu odkaz na premennú umiestnenú v {{ … }}. Premennú, ktorú chceme posunúť HTML súboru uvedieme ako parameter metódy *render\_template()*. Príklad:

* Python súbor:



* HTML súbor:



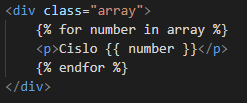
Okrem takýchto typov premenných môžeme v HTML zobraziť aj obsah definovaného poľa (napr. čísla). HTML súboru ho pošleme rovnakým spôsobom ako atribút v metóde *render\_template()*. Avšak v HTML súbore chceme prvky poľa vypisovať po jednom – každé číslo vo vlastnom elemente. Na to je potrebné použiť slučku, 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 vyzerať nasledovne:



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 %}

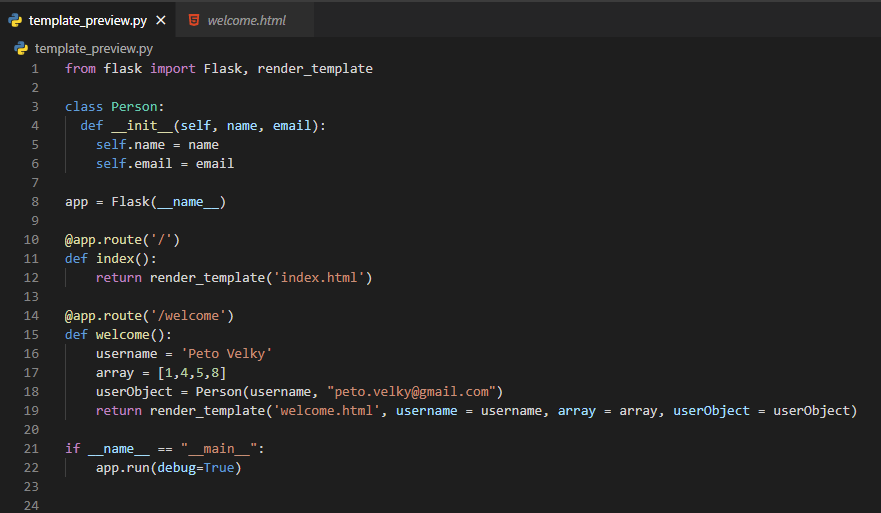
<p> {{ a }} je vacsie ako 2 </p>

{% else %}

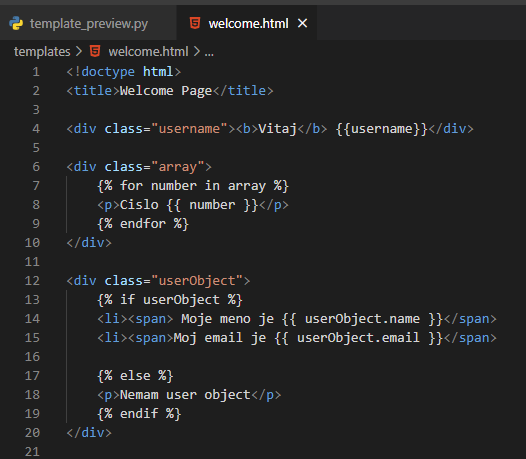
<p> {{ a }} je mensie ako 2 </p>

{% endif %}

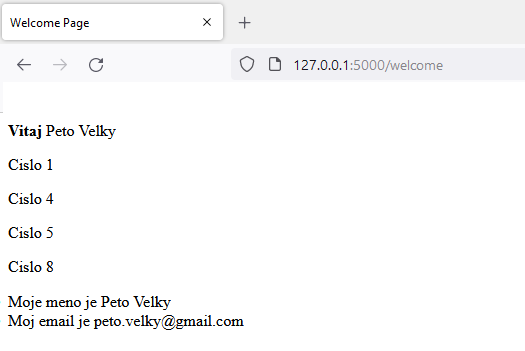
V našom príklade môžeme podmienky zobraziť pre vypísanie objektu užívateľa (tvorí ho jeho meno a email) – premenná *userObject*. Následne by celý Python súbor vyzeral nasledovne:



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:



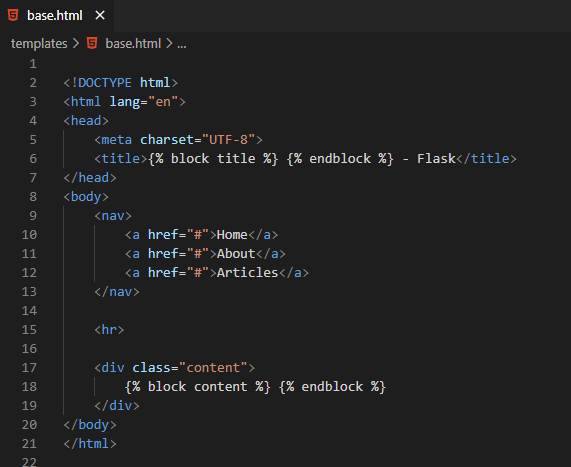
Po spustení skriptu príkazom *$ python template\_preview.py* sa nami vytvorený obsah zobrazí na URL <http://127.0.0.1:5000/welcome>. Ukážka je na nasledujúcom obrázku.



## 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.

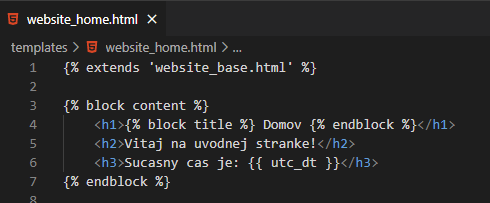
V úlohe chceme vytvoriť webstránku s 3 podstránkami, pričom všetky stránky budú mať rovnakú štruktúru a meniť sa bude len obsah hlavnej časti. Flask umožňuje vytvorenie **base šablóny**, v ktorej sa definujú štruktúra stránky. Obsah súboru website\_*base.html* (tiež vytvorený v priečinku *templates*)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



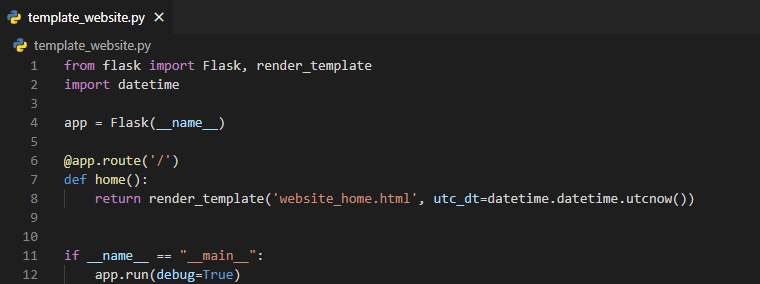
Dôležité je pochopiť tieto 2 časti Jinja šablónovacieho systému, kt. budú v ďalších šablónach menené:

* {% 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:



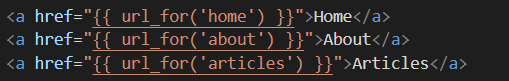
V riadku 6 si chceme zobraziť aktuálny čas pomocou premennej *utc\_dt*. Obsah premennej si pošleme z Python súboru (potrebné vytvoriť nový skript *template\_website.py*). V ňom potrebujeme získať knižnicu **datetime** pomocou *import datetime*. Následne premennú *utc\_dt* definujeme ako *datetime.datetime.utcnow()*, čo predstavuje aktuálny čas. Nový Python súbor vyzerá nasledovne:



Spustení príkazu *$ python template\_website.py* sa hlavná stránka nachádza na URL [http://127.0.0.1:5000/](http://127.0.0.1:5000/welcome). Ukážka je na nasledujúcom obrázku.



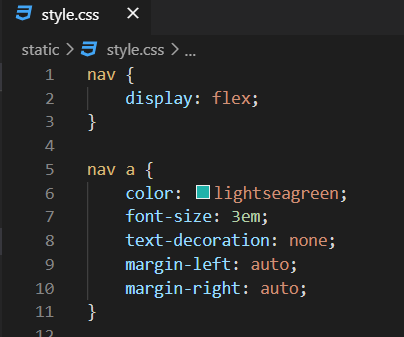
Podobným spôsobom si vytvoríme aj šablóny pre stránky **About** a **Articles** (nová šablóna a metóda). Obe HTML šablóny extenduju *base.html* a menia obsah hlavnej časti. Ak si po pridaní týchto dvoch stránok vyskúšame prekliknutie z našej úvodnej stránky na sekciu About alebo Articles (viď. obrázok 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:



**Poznámka:** Ak je použite 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 sústredí 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:



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.',

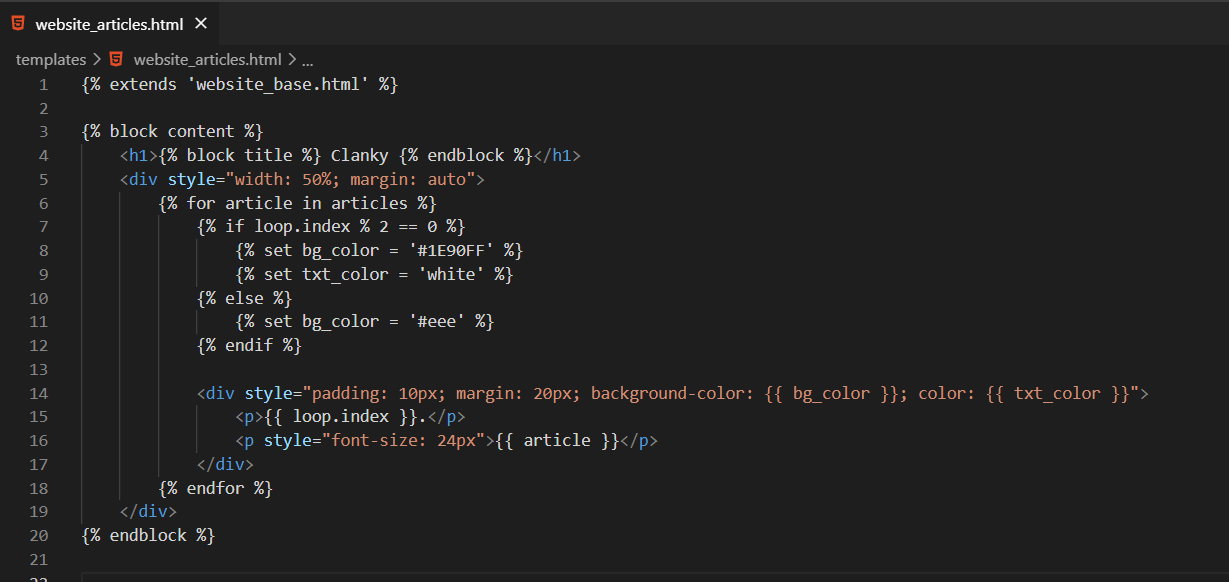
                'Treti clanok.',

                'Stvrty clanok.'

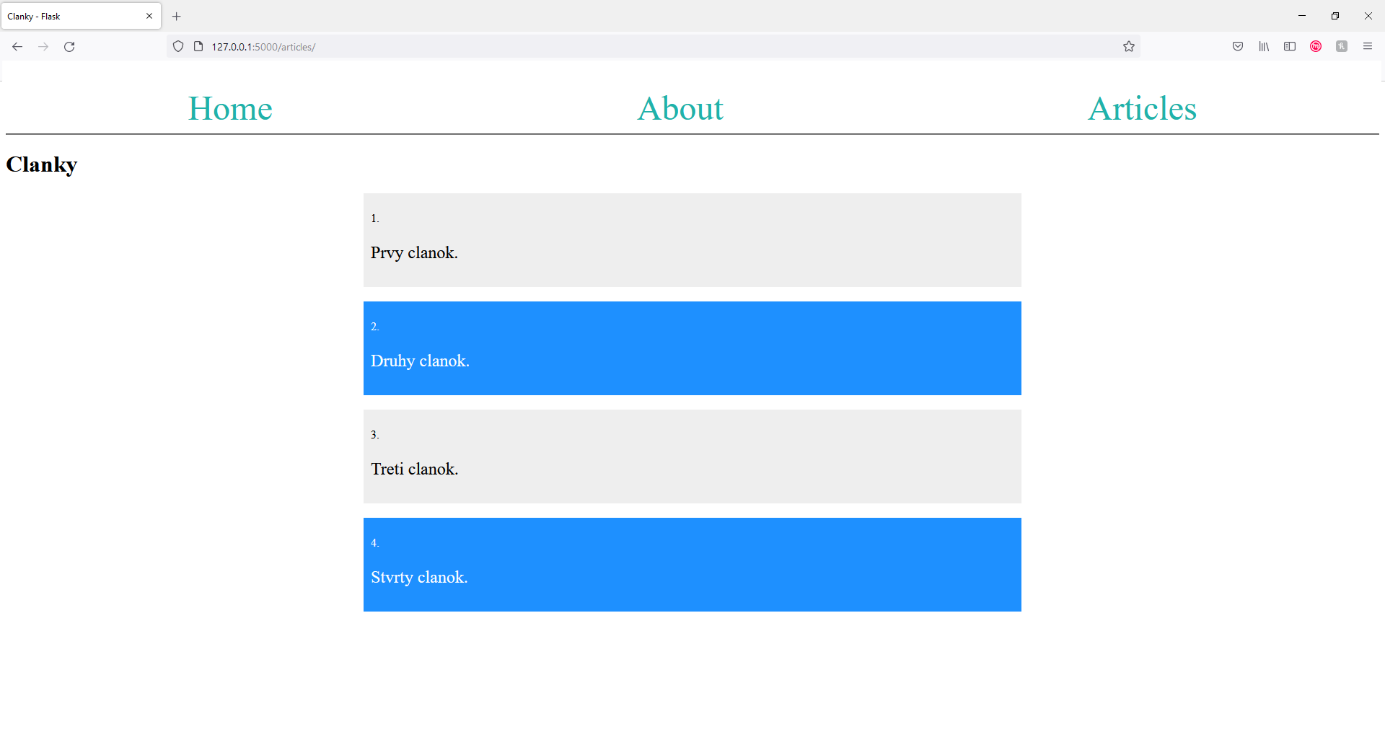
                ]

    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 docieliť takýmto zápisom:



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.



Obrázok, na ktorom je text

Automaticky generovaný popis

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](file:///D:\Stiahnuté%20súbory\Nasadzovanie%20webových%20aplikácií%20v%20Pythone%20sa%20opiera%20o%20WSGI,%20čo%20je%20štandardné%20pythonové%20rozhranie%20pre%20komunikáciu%20medzi%20webovou%20aplikáciou%20a%20webovým%20serverom%20definované%20v%20PEP%20333). 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 nezaujíma 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***

Obrázok, na ktorom je text

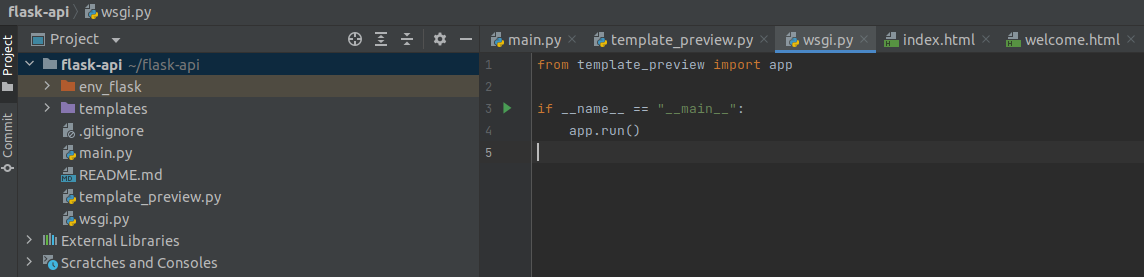
Automaticky generovaný popis

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()



## Prístupujte 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 spustite 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.

Obrázok, na ktorom je text

Automaticky generovaný popis

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.



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:

Obrázok, na ktorom je text

Automaticky generovaný popis

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. Zadajte 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. Náš 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

Obrázok, na ktorom je text

Automaticky generovaný popis

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

sudo systemctl status my-server.service

Obrázok, na ktorom je text

Automaticky generovaný popis

## Konfigurácia Nginx

### Inštalácia nginx balíka

Nginx si nainštalujeme príkazom:

sudo apt install nginx



### Konfigurácia nginx

Potom prejdite do adresára nginx:

cd /etc/nginx/

Obrázok, na ktorom je text

Automaticky generovaný popis

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

Obrázok, na ktorom je text

Automaticky generovaný popis

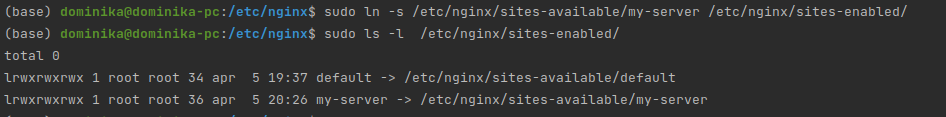
Potom spustite príkaz sudo nginx -t aby ste sa uistili, že konfiguračný súbor je ok.

Obrázok, na ktorom je text

Automaticky generovaný popis

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

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



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

Poznámka, tento príkaz nám vyhadzoval errory. Spustením týchto 4 príkazov sa to nejak opravilo a fungovalo to bez problémov ([užitočný link](https://www.codetd.com/en/article/13069344)):

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](https://www.heroku.com/home) je kontajnerová cloudová platforma ako služba (PaaS – Platform as a service). Je často využívaná developérmi 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, spotrebúva dynohodiny. Keď je nečinná, tak aplikácia prestáva spotrebúvať 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](https://www.heroku.com/free).

## Postup deploymentu s Git

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

* Git ([postup](https://git-scm.com/book/en/v2/Getting-Started-Installing-Git))
* Heroku CLI ([postup](https://devcenter.heroku.com/articles/))

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.

Obrázok, na ktorom je text

Automaticky generovaný popis

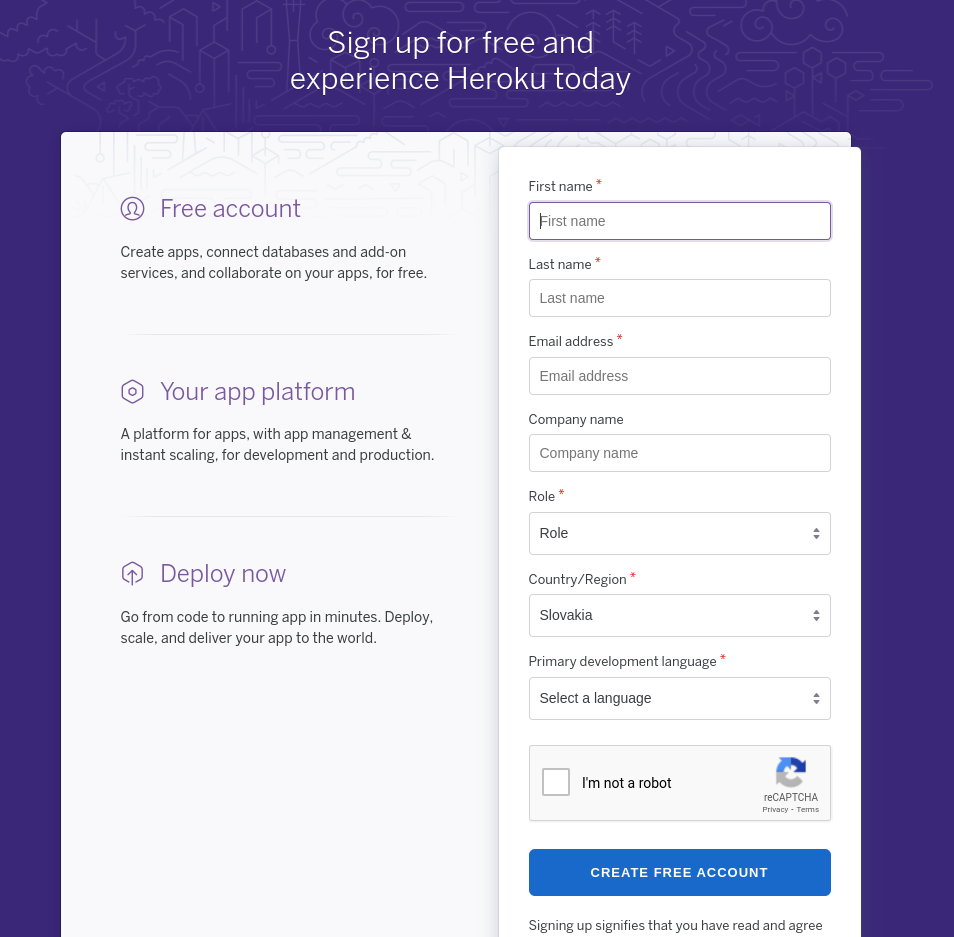
1. Zmeny v súboroch nášho projektu budeme trackovať využitím Git-u (populárny systém na správu verzií). Takže ako prvý krok by ste mali pre svoj projekt vytvoriť Git repozitár. Môžete to dosiahnuť vykonaním nasledujúceho príkazu v adresári vášho projektu: ***git init***.

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

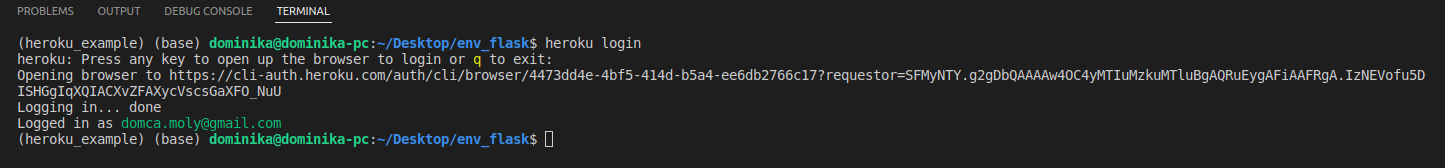
***echo \_\_pycache\_\_ >> .gitignore***

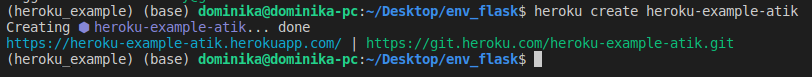
Potom zadáme príkazy ***git add .*** a následne ***git commit –m „Fisrt commit“*** . Teraz sledujete kód svojej aplikácie v miestnom repozitári Git. Zatiaľ neexistuje na žiadnych vzdialených serveroch.

1. Vytvorte si Heroku účet, ktorý je zdarma. Po vyplnení požadovaných údajov a po potvrdení vašej emailovej adresy budete môcť začať využívať Heroku.



1. Heroku CLI (command line interface), ktorý slúži na vytváranie a spravovanie Heroku aplikácii cez terminál (ak potrebujete pomoc s inštaláciou, pozrite si [dokumentáciu](https://devcenter.heroku.com/articles/heroku-cli#install-the-heroku-cli)). Prihláste 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.



1. 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 prázdnym Git repozitárom. Ak spustíte tento príkaz z koreňového adresára vašej aplikácie, prázdny Heroku Git repozitár sa automaticky nastaví ako remote pre vaše lokálne úložisko. 
2. Pomocou príkazu ***git remote -v*** si môžete potvrdiť, že pre vašu aplikáciu bolo nastavené remote s názvom heroku.

Obrázok, na ktorom je text

Automaticky generovaný popis

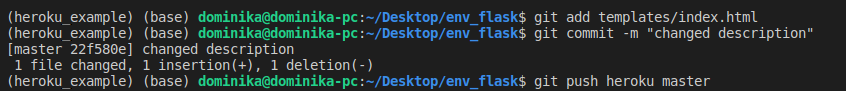
1. Ak chcete nasadiť svoju aplikáciu do Heroku, použite príkaz git push na pushnutie kódu z main vetvy lokálneho repozitára do vášho heroku remote: ***git push heroku mainObrázok, na ktorom je text

   Automaticky generovaný popis***
2. Zadajte príkaz ***heroku open***. Otvorí sa vám webový prehliadač s vašou aplikáciou.

Obrázok, na ktorom je text

Automaticky generovaný popis

**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.



Po znovunačítaní už vidíme našu aplikáciu aj so zmeneným textom.

Obrázok, na ktorom je text

Automaticky generovaný popis

## 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

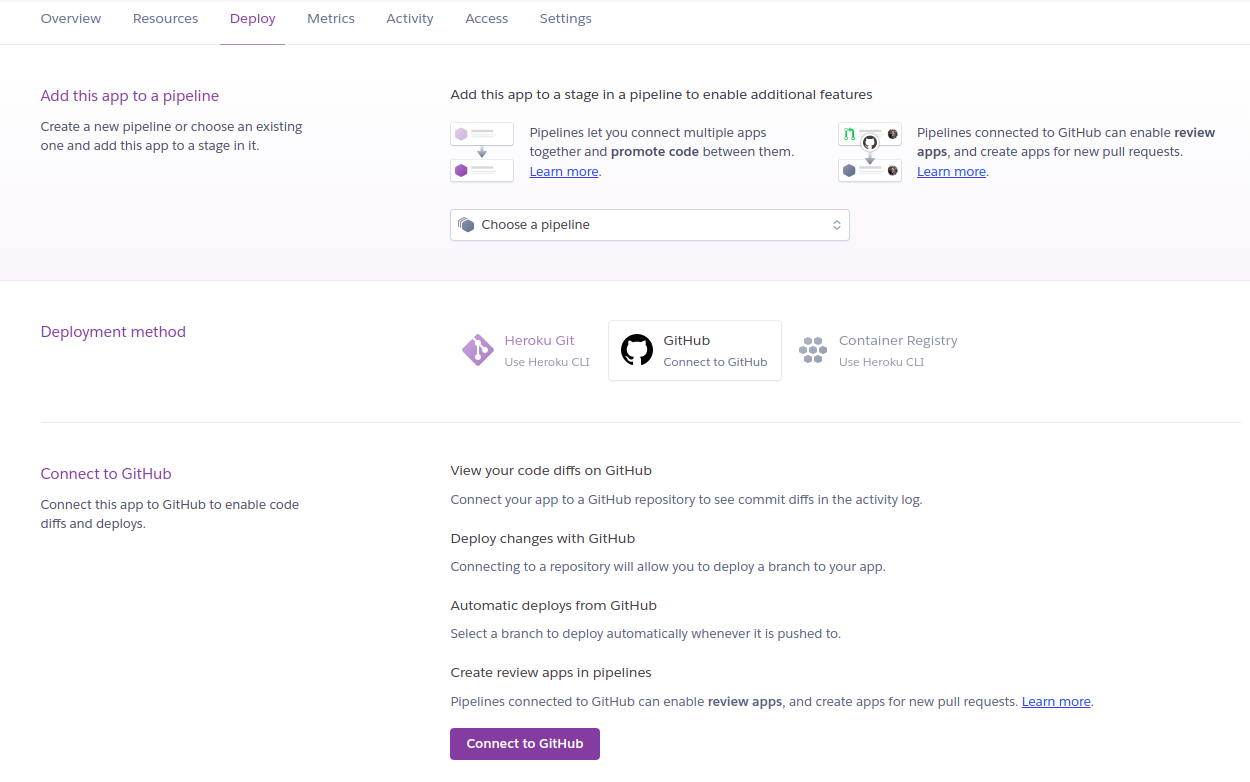
My sme si vytvorili [GitHub repo](https://github.com/dominikaxx/flask_app) s názvom flask\_app, kde sa nachádza naša príkladná flask aplikácia. Pracujeme v priečinku, kde sa nachádza naša Flask aplikácia. Uistite sa, že pracujete vo virtuálnom prostredí. My pracujeme vo virtuálnom prostredí a názvom 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.
4. 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 použite nasledujúce príkazy: ***echo heroku\_example > .gitignore*** a  ***echo \_\_pycache\_\_ >> .gitignore***
5. Teraz tieto zmeny commitnite a pushnite do vášho Github repozitára.
6. Následne choďte na [Heroku](https://dashboard.heroku.com/new-app), prihláste sa a vytvorte novú aplikáciu.

Obrázok, na ktorom je text

Automaticky generovaný popis

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



1. 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](https://stackoverflow.com/questions/71892543/heroku-and-github-items-could-not-be-retrieved-)).