



CS324 - SKRIPTING JEZICI

Python radna okruženja

Lekcija 13

PRIRUČNIK ZA STUDENTE

CS324 - SKRIPTING JEZICI

Lekcija 13

PYTHON RADNA OKRUŽENJA

- ✓ Python radna okruženja
- ✓ Poglavlje 1: Uvod u virtuelna okruženja
- ✓ Poglavlje 2: Mikroservisna arhitektura aplikacije
- ✓ Poglavlje 3: Prednosti mikroservisne arhitekture
- ✓ Poglavlje 4: Nedostaci mikroservisne arhitekture
- ✓ Poglavlje 5: Flask mikroservis sa razvoj veb aplikacija
- ✓ Poglavlje 6: Pokazne vežbe #13
- ✓ Poglavlje 7: Individualne vežbe #13
- ✓ Poglavlje 8: Domaći zadatak
- ✓ Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

▼ Uvod

UVOD

Uvod u lekciju #13

Do sada je bilo reči o Python kao jeziku za razne inženjerske i naučne svrhe, ali naravno, python se može koristiti i u komercijalnom programiranju.

Zbog toga uvodimo Flask.

Flask predstavlja jedan lagani WSGI, odnosno gateway interfejs ka veb serveru. WSGI jeste pecifikacija koja opcije kako veb server komunicira sa veb aplikacijama, i kako se veb aplikacije mogu povezati da isprociraju jedan zahtev.

Flask je jedan framework za web aplikacije. Projektovanje je tako da se može brzo i lako postaviti, ali da je istovremeno skaliranje na kompleksne aplikacije takođe jednostavno.

Trenutno je jedan od najpopularnijih framework-a za veb aplikacija.

Flask je specifičan da daje preporuke, ali ne zahteva nikakve zavisnosti niti specifičnu strukturu odnosno layout projekta. Ostavljeno je na programeru da izabere alate i biblioteke koje želi da koristi.

Flask podržava mnoge dodatke jer je aktivan razvoj samog *framework*-a, tako da je dodavanje novih funkcionalnosti lako.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

▼ Poglavlje 1

Uvod u virtuelna okruženja

POTREBA ZA VIRTUELNIM OKRUŽENJEM

Odvajanje virtuelnog okruženja od sistemskog omogućuje se migracija Python interpretera

Nekada je potrebno izdvojiti radno okruženje od sistemskog okruženja, i to se postiže pomoću paketa venv.

Paket **venv** - **Virtual Environment** pravi virtuelno okruženje za Python u kojem nema naknadno instaliranih paketa, već su dostupni samo sistemski paketi.

Odvajanje virtuelnog okruženja od sistemskog omogućuje se migracija Python interpretera, a da se pri tom ne javljaju greške zbog nepostojećih paketa na drugom računaru.

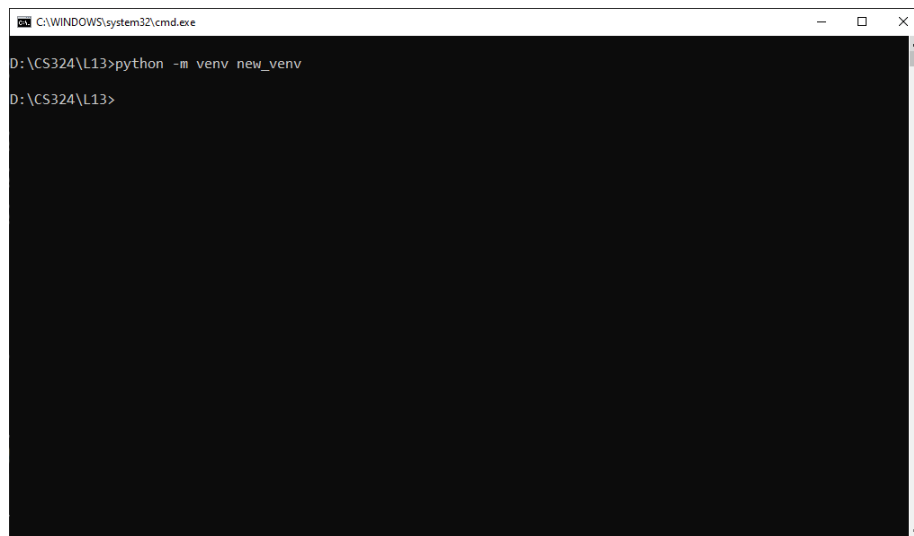
Prilikom kreacije virtuelnog okruženja, svaki put se pravi novi Python interpreter sa podrazumevanim paketima (**pip** i **setuptools**), i treba unutar tog okruženja instalirati ostale pakete koji su neophodni na Python projektu.

Kreacija virtuelnog okruženja

Da bi se napravilo novo virtuelno okruženje, potrebno je u terminalu doći do direktorijuma u kojem želite napraviti okruženje, i ukucati sledeću komandu:

```
python -m venv new_venv
```

U ovom primeru opcija **-m** znači da se pokreće modul **venv**, a **new_venv** biće novi direktorijum unutar radnog direktorijuma gde će biti smešteno virtuelno okruženje.

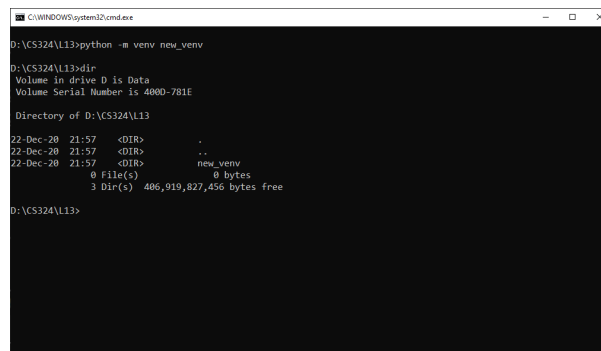


Slika 1.1 Pravljenje direktorijuma za virtuelno okruženje. [Izvor: Autor]

AKTIVIRANJE VIRTUELNOG OKRUŽENJA

Jednom kada se virtuelno okruženje napravi, unutar novog direktorijuma pojaviće se dodatne datoteke

Jednom kada se virtuelno okruženje napravi, unutar novog direktorijuma pojaviće se dodatne datoteke.



Slika 1.2 Novi direktorijum sa virtuelnim okruženjem. [Izvor: Autor]

Ove datoteke zapravo predstavljaju zaseban Python interpreter, i biće iste verzije kao i sistemski interpreter, tačnije biće iste verzije kao i interpreter koji je napravio novo virtuelno okruženje.

Slika 1.3 Sadržaj direktorijuma Scripts unutar virtuelnog okruženja. [Izvor: Autor]

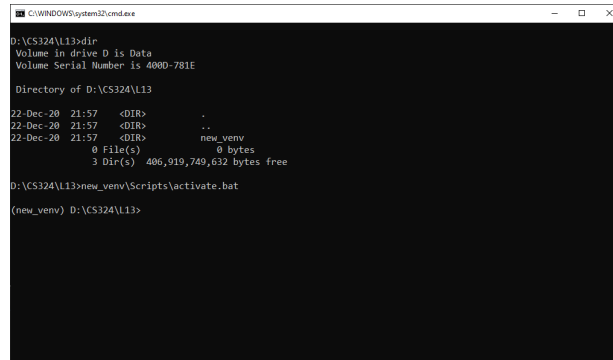
Aktiviranje virtuelnog okruženja

Novo virtuelno okruženje aktivira se iz terminala pozivanjem datoteke **activate.bat** u direktorijumu **Scripts**.

```
new_venv\Scripts\activate.bat
```

Virtuelno okruženje je aktivirano ukoliko pre *prompt*-a u zagradama piše ime samog virtuelnog okruženja, u primeru biće:

```
(new_venv)
```



Slika 1.4 Aktiviranje virtuelnog okruženja. [Izvor: Autor]

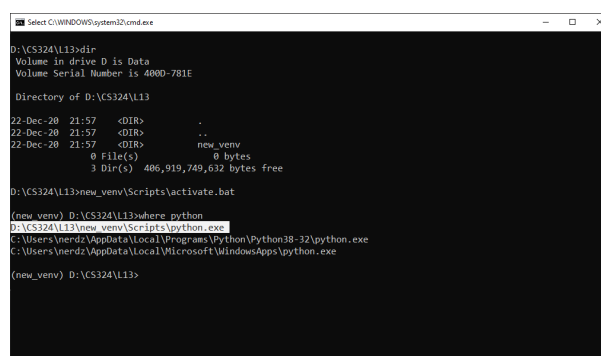
RAD I DEAKTIVIRANJE VIRTUELNOG OKRUŽENJA

Jednom kada se aktivira virtuelno okruženje, Python interpreter se ponaša nezavisno od sistemskog interpretera.

Jednom kada se aktivira virtuelno okruženje, Python interpreter se ponaša nezavisno od sistemskog interpretera.

Provera interpretera se može uraditi komandom u terminalu:

```
where python
```



Slika 1.5 Putanja do aktivnog Python interpretera. [Izvor: Autor]

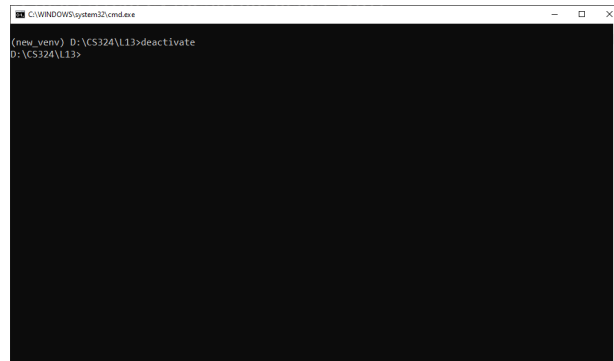
Svi paketi i moduli koji su prethodno bili instalirani neće biti dostupni, već se moraju naknadno instalirati.

Slika 1.6 Pip list komanda u novom virtuelnom okruženju. [Izvor: Autor]

Deaktiviranje virtuelnog okruženja

Vraćanje u sistemski Python interpreter se vrši komandom

```
deactivate
```



Slika 1.7 Deaktiviranje virtuelnog okruženja. [Izvor: Autor]

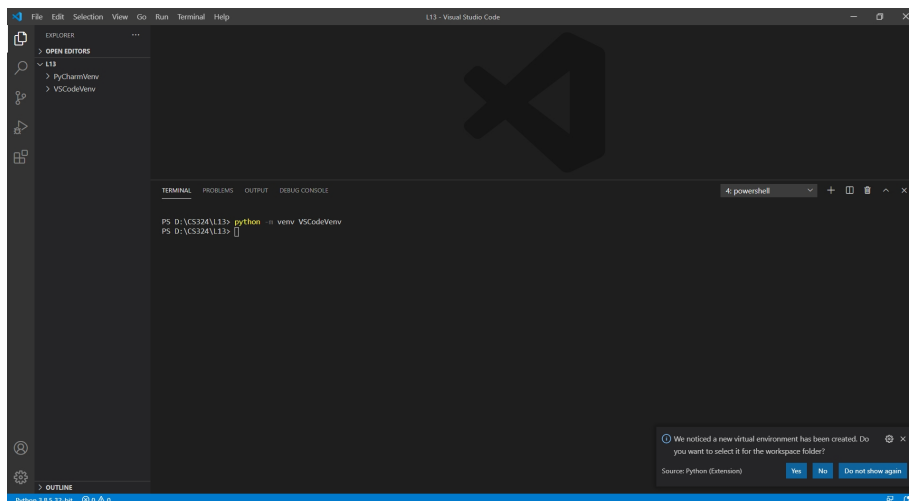
Prilikom ponovnog aktiviranja virtuelnog okruženja, ostaće instalacije paketa koji su instalirani unutar tog okruženja.

AKTIVIRANJE VIRTUELNOG OKRUŽENJA U VISUAL STUDIO CODE

Aktiviranje virtuelnog okruženja u Visual Studio Code-u je jednostavno kao i preko terminala.

U **Visual Studio Code** IDE-u, kreacija virtuelnog okruženja je identična kao pri radu sa terminalom. **Visual Studio Code** odmah će prepoznati da je u pitanju novi interpreter i pitaće korisnika da li želi da se prebaci na novokreirani interpreter.

Nakon toga, potrebno je aktivirati i putem terminala, ali treba aktivirati datotekom `activate.ps1`, pošto je podrazumevani terminal **PowerShell**, a ne običan **Command Prompt**.



Slika 1.8 Kreiranje virtuelnog okruženja u VSCode. [Izvor: Autor]

Napomena:

Ukoliko se ne može aktivirati novo okruženje, potrebno je omogućiti pokretanje .ps1 skripti u Power Shell-u. Treba pokrenuti Power Shell sa administratorskim privilegijama i ukucati sledeće:

```
set-executionpolicy remotesigned
```

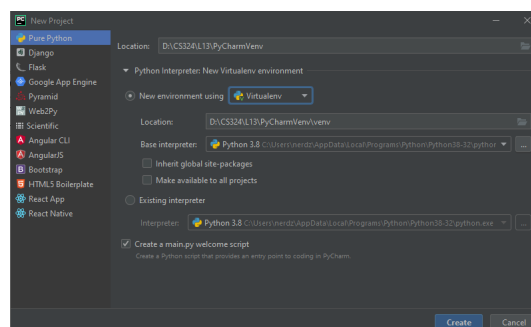
Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

AKTIVIRANJE VIRTUELNOG OKRUŽENJA U PYCHARM

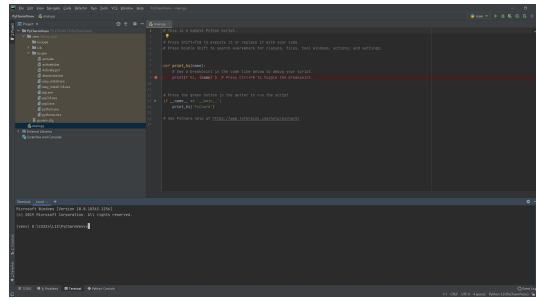
Pycharm podrazumevano nudi opciju kreacije virtuelnog okruženja prilikom pravljenja novog projekta.

Py Charm IDE prilikom pokretanja nudi da se napravi novi projekat sa postojećim Python interpreterom, ili sa kreacijom novog interpretera u novom virtuelnom okruženju.

Ukoliko se napravi virtuelno okruženje, ono se odmah aktivira.



Slika 1.9 Odabir sistemskog interpretera ili kreiranje virtuelnog okruženja. [Izvor: Autor]



Slika 1.10 Novo virtuelno okruženje je automatski aktivirano. [Izvor: Autor]

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

▼ Poglavlje 2

Mikroservisna arhitektura aplikacije

UVOD U MIKROSERVISE - SOA

Osnovni princip SOA jeste organizacija aplikacije u diskretne jedinice funkcionalnosti, kojima se može daljinski pristupiti, i koje se nezavisno ažuriraju.

Mikroservisna arhitektura

Pošto ne postoji zvanični standard za mikroservise, postoji mnoštvo definicija za njih.

Pojedinci često pominju servisno-orijentisanu arhitekturu (en. [Service-Oriented Architecture](#), [SOA](#)) kada pokušavaju da objasne šta su mikroservisi

Wikipedia:

SOA prethodi mikroservisima - njen osnovni princip je ideja da se organizuju aplikacije u diskretne jedinice funkcionalnosti kojima se može daljinski pristupiti da se na njih deluje i koje se nezavisno ažuriraju.

Iako SOA jasno ističe da servisi treba da budu samostalni procesi, nije rečeno koji protokoli treba da se upotrebe za te procese za međusobnu interakciju i ostaje prilično nejasna u pogledu načina raspoređivanja i organizovanja aplikacije.

U SOA Manifestu (<http://www.soa-manifesto.org>), koji je publikovan na Vebu 2009. godine, stručnjaci koji su ga potpisali čak ne spominju da li servisi međusobno vrše interakciju pomoću mreže.

SOA servisi mogu da komuniciraju pomoću međuprocenke komunikacije (en. [Inter-Process Communication](#), IPC), koristeći priključke (*sokete*, en. [sockets](#)) na istoj mašini, korišćenjem *deljene memorije* i *indirektnih redova*

poruka ili, čak, pomoću *udaljenih poziva procedura* (en. [Remote Procedure Calls](#), RPC). Opcije su razne i na kraju SOA može da bude *sve i svašta*, sve dok ne pokrenete ceo kod aplikacije u jednom procesu.

Međutim, uobičajeno je reći da su mikroservisi specijalizacija SOA ciljeva, koji su počeli da se pojavljuju poslednjih godina, zato što ispunjavaju neke SOA ciljeve, kao što je izgradnja aplikacija pomoću samostalnih komponenata koje međusobno komuniciraju.

Ako želimo da formulišemo kompletnu definiciju šta su mikroservisi, najbolji način je da prvo pogledamo kako je izgrađena većina softvera.

MONOLITNA ARHITEKTURA

Najpre su veb aplikacije pratile monolitnu arhitekturu.

Pre objašnjenja mikroservisa, potrebno je objasniti pojam monolitne arhitekture (en. **monolithic application architecture**)

Primer:

Upotrebićemo jedan veoma jednostavan primer tradicionalne monolitne aplikacije: veb sajt za rezervaciju hotela. Osim statičnog HTML sadržaja, veb sajt ima funkcije za rezervacije, koje će omogućiti korisnicima da rezervišu hotele u bilo kom gradu na svetu. Korisnici mogu da pretražuju

hotele, pa da rezervišu smeštaj u njima, koristeći kreditnu karticu.

Kada korisnik izvrši pretragu na veb sajtu hotela, aplikacija prolazi kroz sledeće korake:

1. Pokreće dva SQL upita za baze podataka hotela.
2. HTTP zahtev za servis partnera je kreiran za dodavanje više hotela u listu.
3. Generisana je stranica sa rezultatima upotrebom mehanizma HTML šablona.

Od te tačke, kada korisnik pronađe odgovarajući hotel i klikne na njega da bi u njemu rezervisao smeštaj, aplikacija izvršava sledeće korake:

1. Korisnik je kreiran u bazi podataka ako je potrebno i treba da mu se proveriti identitet.
2. Plaćanje se izvršava interakcijom sa veb servisom banke.
3. Aplikacija snima detalje o plaćanju u bazi podataka zbog pravnih razloga.
4. Potvrda prijema je generisana pomoću PDF generatora.
5. E-mail rekapitulacije je poslat korisniku pomoću e-mail servisa.
6. E-mail u vezi rezervacije je prosleđen nezavisnom hotelu pomoću e-mail servisa.
7. Unos baze podataka je dodat za praćenje rezervacije.

Ovaj proces je, naravno, pojednostavljeni model, ali je prilično realističan.

Aplikacija vrši interakciju sa bazom podataka koja sadrži informacije o hotelu, detalje o rezervaciji i naplati, informacije o korisniku i tako dalje. Takođe vrši interakciju sa eksternim servisima za slanje e-mailova, izvršavanje plaćanja i dodavanje naziva za više hotela.

U **LAMP** (Linux-Apache-MySQL-PHP) arhitekturi svaki ulazni zahtev generiše niz SQL upita u bazi podataka i nekoliko mrežnih poziva ka eksternim servisima, a zatim server generiše HTML odgovor, koristeći mehanizam šablona.

OPIS MONOLITNE ARHITEKTURE

Najveća prednost je što se cela aplikacija nalazi u jednoj osnovi koda.

Ovo je tipična monolitna aplikacija, koja ima mnogo očiglednih prednosti.

Najveća prednost je što se cela aplikacija nalazi u **jednoj osnovi koda**, pa, kada započne proces kodiranja projekta, sve postaje jednostavnije. Izgradnja dobrog testa je jednostavna i

kod može da se organizuje na jasan i strukturirani način unutar njegove osnove. Skladištenje svih podataka u jednu bazu podataka takođe pojednostavljuje razvoj aplikacije. Mogu da se podešavaju model podataka i način na koji će ih kod zatražiti.

Raspoređivanje je, takođe, jednostavno: možemo da označimo osnovu koda, da izgradimo paket i da ga negde pokrenemo. Da bismo skalirali aplikaciju, možemo da pokrenemo nekoliko instanci aplikacije za rezervacije i nekoliko baza podataka sa postavljenim mehanizmima

za repliciranje. Ako aplikacija ostane mala, ovaj model funkcioniše dobro i jedan tim ga može jednostavno održavati.

Međutim, projekti obično rastu i postaju veći nego što je prvobitno planirano. Ako je cela aplikacija u jednoj osnovi koda, to nam izaziva neke ozbiljne probleme u radu.

Na primer, ako treba da izvršimo promene čišćenja koje su velikog obima, kao što je menjanje bankovnog servisa ili sloja baze podataka, cela aplikacija postaje veoma nestabilna. Ove promene su veoma značajne u „životu“ projekta i zahtevaju mnogo dodatnih testiranja za raspoređivanje nove verzije.

Male promene mogu, takođe, generisati kolateralnu štetu, zato što različiti delovi sistema imaju različite zahteve za vreme ispravnog rada i za stabilnost. Postavljanje procesa za naplatu i rezervaciju u rizičnu situaciju, zato što je funkcija koja kreira PDF srušila server, malo je problematično.

Nekontrolisan rast je još jedan problem. Aplikacija dobija nove funkcije i ako programeri napuštaju projekat i priključuju se projektu, organizacija koda može da postane neuredna, a testovi sporiji. Ovaj rast se, obično, završava sa špageti osnovom koda, koja je veoma teška za održavanje, a neuredne baze podataka zahtevaju komplikovane planove migracije uvek kada neki od programera preradi model podataka.

Veliki projekti softvera obično zastarevaju za dve godine, a zatim polako počinju da se pretvaraju u nerazumljivu zbrku, koja je veoma teška za održavanje. A to se ne dešava zato što su programeri loši, već zato što, dok raste kompleksnost, sve manje ljudi razume implikacije veoma malih promena, pa pokušavaju da rade u izolaciji u jednom uglu osnove koda,

tako da, kada pogledate prikaz od 10.000 stopa projekta, vidite haos.

PREDNOSTI I MANE MONOLITNE ARHITEKTURE

Najveća mana monolitne arhitekture jeste loše skaliranje pri rastu aplikacije.

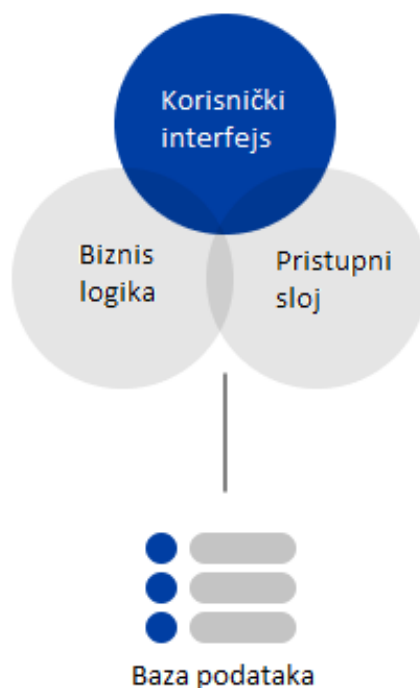
Sledeće tačke rezimiraju prednosti i mane monolitnog pristupa:

- Započinjanje projekta kao monolitnog je jednostavno i verovatno predstavlja najbolji pristup.
- Centralizovana baza podataka pojednostavljuje dizajn i organizaciju podataka.
- Raspoređivanje jedne aplikacije je jednostavno.
- Svaka promena u kodu može da utiče na nevezane funkcije. Kada se nešto ošteti, cela aplikacija može biti oštećena.
- Kako osnova koda raste, teže je da se održava čista i da bude pod kontrolom.

Očigledno rešenje je da razdvojimo aplikaciju u posebne delove, čak i ako će se rezultirajući kod i dalje pokretati u jednom procesu. Programeri to rešavaju izgradnjom svojih aplikacija, koristeći eksterne biblioteke i radne okvire.

Izgradnja veb aplikacije u Pythonu, ako koristimo radni okvir kao što je Flask, omogućava da se fokusiramo na poslovnu logiku i pojednostavljuje eksternalizaciju koda u Flask ekstenzije i male Python pakete. A razdvajanje koda u male pakete je često dobra ideja za kontrolisanje rasta aplikacije.

Monolitna arhitektura

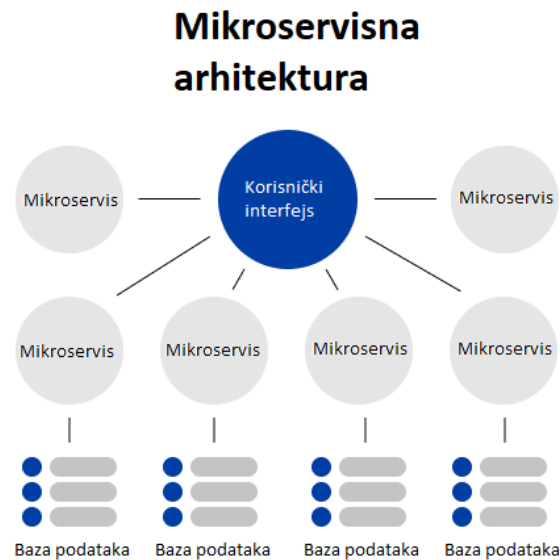


Slika 2.1 Monolitna arhitektura. [Izvor: Autor]

MIKROSERVISNA ARHITEKTURA

Umesto da imamo jednu aplikaciju koja je za sve odgovorna, razdvojićemo je na više različitih mikroservisa.

Ako želimo da izgradimo istu aplikaciju upotrebom mikroservisa, organizovaćemo kod u nekoliko posebnih komponentata koje se pokreću u posebnim procesima. Umesto da imamo jednu aplikaciju koja je za sve odgovorna, razdvojićemo je na više različitih mikroservisa, kao što je prikazano na sledećem dijagramu.



Slika 2.2 Mikroservisna arhitektura. [Izvor: Autor]

Primer:

U primeru sa hotelom, prebacili smo deo složenosti na kraju smo dobili sledeće samostalne komponente:

1. Booking UI - servis čeonog prikaza koji generiše veb korisnički interfejs i vrši interakciju sa svim drugim mikroservisima,
2. PDF reporting service - veoma jednostavan servis koji kreira PDF-ove za potvrde ili bilo koji drugi dokument za određeni šablon ili neke podatke,
3. Search - servis koji može da bude ispitan za dobijanje liste hotela za određeni naziv grada. Ovaj servis ima sopstvenu bazu podataka.
4. Payments - servis koji vrši interakciju sa nezavisnim bankovnim servisom i upravlja bazom podataka naplate. Takođe šalje e-mail o uspešnom plaćanju.
5. Reservations - Skladišti rezervacije i generiše PDF-ove.
6. Users - Skladišti korisničke informacije i vrši interakciju sa korisnicima pomoću e-maila.
7. Authentication - servis koji vraća tokene provere identiteta koje svaki mikroservis može da upotrebi za proveru identiteta kada poziva druge.

DEFINICIJA MIKROSERVISA

Mikroservis je jednostavna aplikacija koja obezbeđuje suženu i jasno definisanu listu funkcija

Ovi mikroservisi, zajedno sa nekoliko eksternih servisa, kao što je e-mail servis, obezbeđuju skup funkcija sličan monolitnoj aplikaciji. U ovom projektu svaka komponenta komunicira upotrebom [HTTP](#) protokola, a funkcije su dostupne pomoću [RESTful veb servisa](#). Ne postoji

centralizovana baza podataka, jer svaki mikroservis interno koristi sopstvene strukture podataka, a podaci koji ulaze ili izlaze koriste jezički nezavisan format, kao što je *JSON*. Mikroservis može da upotrebi i formate *XML* i *YAML*, sve dok oni mogu da se proizvede i upotrebe u svakom jeziku i da „putuju“ kroz *HTTP* zahteve i odgovore.

Servis Booking UI je unekoliko poseban, zato što generiše korisnički interfejs (en. *User Interface, UI*). U zavisnosti od radnog okvira čeonog prikaza koji je upotrebljen za izgradnju korisničkog interfejsa, izlaz servisa Booking UI može da bude mešavina *HTML*-a i *JSON*-a, ili, čak, čist *JSON* ako interfejs koristi statičnu alatku zasnovanu na JavaScript jeziku na strani klijenta za generisanje interfejsa direktno u pretraživaču.

Za razliku od ovog konkretnog slučaja korisničkog interfejsa, veb aplikacija koja je projektovana kao mikroservis je kompozicija od nekoliko mikroservisa koji mogu da vrše interakciju međusobno kroz *HTTP* da bi obezbedili ceo sistem.

U tom kontekstu mikroservisi su logičke jedinice koje se fokusiraju na veoma određene zadatke. Evo i cele definicije:

Mikroservis je jednostavna aplikacija koja obezbeđuje suženu i jasno definisanu listu funkcija. To je komponenta sa jednom odgovornošću, koja može da bude razvijena i raspoređena samostalno.

Ova definicija ne spominje *HTTP* ili *JSON*, zato što možete da razmotrite mali servis zasnovan na *UDP*-u koji, na primer, razmenjuje binarne podatke kao mikroservis.

Međutim, u našem slučaju u ovoj knjizi svi mikroservisi su samo jednostavne veb aplikacije koje koriste *HTTP* protokol i koriste i proizvode *JSON* kada to nije *UI*.

▼ Poglavlje 3

Prednosti mikroservisne arhitekture

KARAKTERISTIKE MIKROSERVISA

Mikroservisi su najčešće procesi koji komuniciraju putem mreže da izvrše zadatke koristeći protokole koji nisu vezani za određeni tip tehnologije.

Karakteristika mikroservisa uključuju:

- Mikroservisi su najčešće procesi koji komuniciraju putem mreže da izvrše zadatke koristeći protokole koji nisu vezani za određeni tip tehnologije poput HTTP-a [1 - 3]. Ipak, servisi mogu da koriste i druge mehanizme inter-procesne komunikacije poput deljene memorije [4]. Servisi, takođe, mogu biti pokrenuti u okviru istog procesa, na primer, OSGi paketi.
- Servisi u mikroservis arhitekturi se mogu posebno isporučivati [5].
- Servisi su lako zamenjivi.
- Servisi su organizovani oko funkcionalnosti, npr. front-end korisnički interfejs, preporuke, logistika, naplata, itd.
- Servisi se mogu implementirati korišćenjem različitih programskih jezika, baza, hardverskih i softverskih okruženja, u zavisnosti šta od toga najviše odgovara za implementaciju.
- Servisi su mali veličinom, omogućena je razmena poruka, povezani su sadržajem, nezavisno razvijani, decentralizovani, kao i bildovani i izdavani pomoću automatskih procesa.

Osobine aplikacije čija je arhitektura zasnovana na mikroservisima:

- Prirodno se nameće modularna struktura.
- Je proces kontinualne isporuke softvera. Izmena malog dela aplikacije zahteva ponovno bildovanje i isporuku samo jednog ili malog broja servisa.
- Pridržava se principa poput sitno granuliranog interfejsa (za servise koji se mogu nezavisno izdavati), poslovno orijentisan razvoj, arhitekture IDEAL računarstva u oblaku, poliglot programiranje i postojanost, isporuka jednostavnih kontejnera, decentralizovana kontinualna isporuka, kao i DevOps sa holističkim nadzorom servisa [6].
- Obezbeđuje karakteristike koje doprinose skalabilnosti [7-9].

PREDNOSTI MIKROSERVISNE ARHITEKTURE

Mikroservisi pružaju razdvajanje dužnosti i bolje skaliranje u odnosu na monolitne arhitekture.



Slika 3.1 Apstrakcija mikroservisne arhitekture. [Izvor: Wikipedia]

Iako arhitektura mikroservisa izgleda komplikovanije nego njen monolitni suparnik, prednosti su višestruke. Ona pruža sledeće:

- razdvajanje dužnosti,
- manje projekte za rad,
- više opcija za skaliranje i raspoređivanje.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

RAZDVAJANJE DUŽNOSTI

Posebni timovi mogu da nezavisno razviju svaki mikroservis.

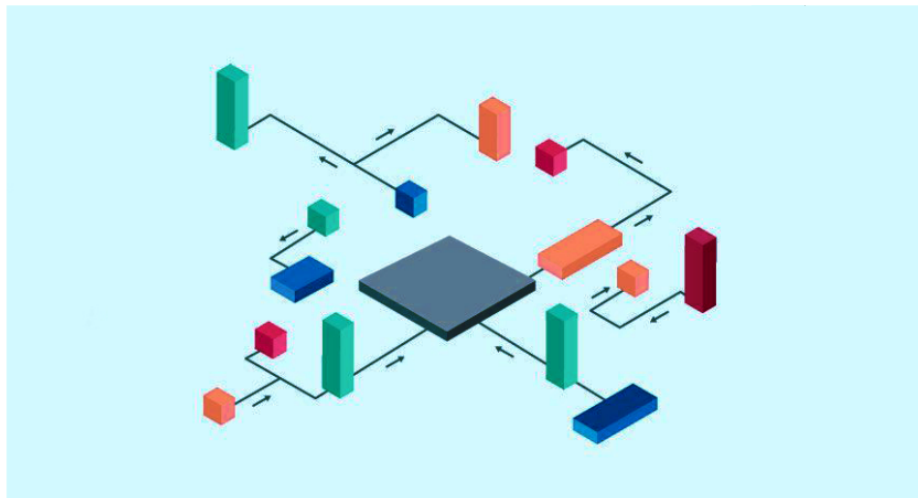
Pre svega, posebni timovi mogu da nezavisno razviju svaki mikroservis. Na primer, izgradnja servisa za rezervacije može da bude potpuno samostalan projekat. Tim koji je zadužen za ovaj servis može da ga kreira u bilo kom programskom jeziku, koristeći bilo koju bazu podataka, sve dok ima dobro dokumentovan HTTP API.

To takođe znači da je evolucija aplikacije više pod kontrolom, nego u slučaju monolitne aplikacije. Na primer, ako sistem za plaćanje promeni pozadinske akcije sa bankom, uticaj je lokalizovan unutar konkretnog servisa, a ostatak aplikacije ostaje stabilan i, verovatno, netaknut.

Ovo labavo povezivanje poboljšava brzinu projekta, jer se, na nivou servisa, primenjuje filozofija slična principu jedne odgovornosti.

Princip jedne odgovornosti je definisao Robert Martin da bi objasnio da treba da postoji samo jedan razlog da se klasa promeni; drugim rečima, svaka klasa treba da obezbedi jednu, dobro definisanu funkciju.

Primenjeno na mikroservis, to znači da želimo da se uverimo da se svaki mikroservis fokusira na jednu ulogu.

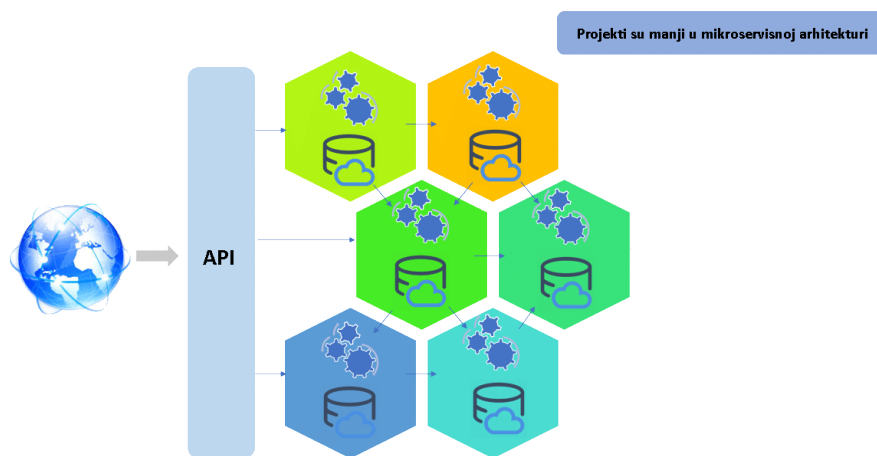


Slika 3.2 Apstrakcija razdvajanje dužnosti kod mikroservisa. [Izvor: wikipedia]

MANJI PROJEKTI

Velika prednost mikroservisne arhitekture jeste razdvajanje složenosti projekta.

Druga prednost je razdvajanje složenosti projekta. Kada dodamo funkciju u aplikaciju, kao što je pisanje PDF izveštaja, čak i ako je kreirate jasno, osnovni kod će biti veći, komplikovaniji i ponekad sporiji. Izgradnja te funkcije u posebnoj aplikaciji sprečava pojavu ovog problema i olakšava pisanje bilo kojom alatom.



Slika 3.3 Obim pojedinačnih projekata je manji. [Izvor: Autor]

Možemo da je prerađujemo često, da skratimo ciklus izdanja i da ostanemo u toku. Rast aplikacije ostaje pod kontrolom. Upotreba manjeg projekta takođe smanjuje rizik kada poboljšavamo aplikaciju: ako tim želi da isproba najnoviji programski jezik ili radni okvir, njegovi članovi mogu brzo da kreiraju

prototip koji implementira isti API mikroservisa, da ga isprobaju i da odluče da li će da ga zadrže ili ne.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

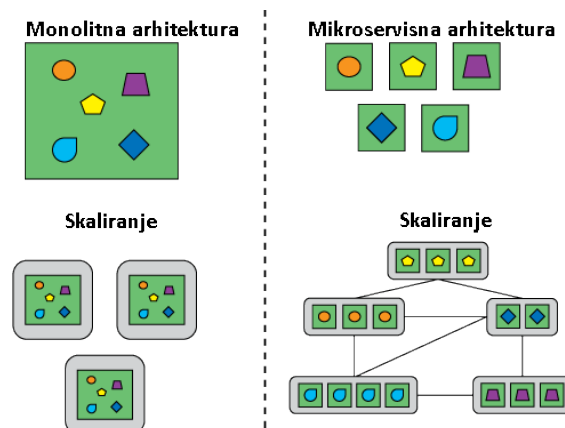
SKALIRANJE I RAZVOJ

Arhitektura mikroservisa je korisna u rešavanju mnoštva problema koji se mogu javiti kada aplikacija počne da raste.

Na kraju, razdvajanje aplikacije na komponente olakšava skaliranje, u zavisnosti od ograničenja. Recimo da počinje da se svakodnevno javlja sve više korisnika koji rezervišu hotele, pa generisanje PDF-a počinje da „zagreva“ CPU-ove. Možete da rasporedite taj konkretan

mikroservis na neke servere koji imaju veće CPU-ove.

Još jedan tipičan primer su mikroservisi koji koriste mnogo RAM-a – na primer, oni koji vrše interakciju sa bazama podataka u memoriji, kao što su [Redis](#) ili [Memcache](#). Može da se podesi njihovo raspoređivanje na servere sa manje CPU-a i mnogo više RAM-a.



Slika 3.4 Skaliranje u monolitnoj i mikroservisnoj arhitekturi. [Izvor: Autor]

Prema tome, možemo da rezimiramo prednosti mikroservisa na sledeći način:

- Tim može da razvije svaki mikroservis nezavisno i da upotrebi bilo koji tehnološki stek. On može da definiše prilagođeni ciklus izdanja. Sve što treba da definiše je jezički nezavisan HTTP API.
- Programeri rastavljaju složenost aplikacije na logičke komponente. Svaki mikroservis se fokusira da izvršava dobro jedan zadatak.
- Pošto su mikroservisi samostalne aplikacije, postoji bolja kontrola za raspoređivanje koja olakšava skaliranje.

Arhitektura mikroservisa je korisna u rešavanju mnoštva problema koji se mogu javiti kada aplikacija počne da raste. Međutim, potrebno je da se čuvamo nekih novih problema koje oni uvode u aplikaciju.

▼ Poglavlje 4

Nedostaci mikroservisne arhitekture

MANE MIKROSERVISNE ARHITEKTURE

Razvoj veb aplikacija pomoću mikroservisa ima mnogo prednosti, ali ima i mana.

Kao što je ranije napomenuto, izgradnja aplikacije pomoću mikroservisa ima mnogo prednosti, ali nije savršena.

Potrebno je da obratimo pažnju na sledeće probleme sa kojima se možemo suočiti kada kodiramo mikroservise:

- nelogično razdvajanje,
- više mrežne interakcije,
- skladištenje i deljenje podataka,
- problemi kompatibilnosti,
- testiranje.

Ovi problemi će detaljno biti opisani u sledećim odeljcima.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

NELOGIČNO RAZDVAJANJE

Razvoj veb aplikacije bazirana na mikroservisnoj arhitekturi može imati nelogično razdvajanje.

Prvi problem arhitekture mikroservisa je kako su oni dizajnirani. Ne postoji način da tim može da kreira savršenu arhitekturu mikroservisa u prvom pokušaju.

Neki mikroservisi, kao što je PDF generator, predstavljaju očigledan primer. Međutim, čim se suočimo sa poslovnom logikom, postoji opasnost da će se kod pomeriti pre nego što jasno vidite kako da ga razdvojite u odgovarajući skup mikroservisa.

Projekat treba da „sazri“ u ciklusima isprobavanja-i-neuspeha. A dodavanje i uklanjanje mikroservisa može da bude teže od prerade monolitne aplikacije.

Ovaj problem možemo da ublažimo izbegavanjem razdvajanja aplikacije u mikroservise ako ono nije evidentno.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Ako postoji bilo kakva sumnja da razdvajanje ima smisla, zadržavanje koda u istoj aplikaciji je bezbedno. Uvek je lakše razdvojiti kod u novi mikroservis kasnije od spajanja nazad u dva mikroservisa u istoj osnovi koda kada se ispostavi da je odluka bila loša.

Na primer, ako uvek treba da rasporedimo dva mikroservisa zajedno ili ako jedna promena u mikroservisu utiče na model podataka drugog mikroservisa, možda nismo dobro razdvojili aplikaciju i možda bi ta dva servisa trebalo da ponovo budu spojena.

VIŠE MREŽNIH INTERAKCIJA

Broj mrežnih interakcija za izgradnju aplikacije jeste veći u odnosu na monolitnu arhitekturu.

Drugi problem je količina mrežnih interakcija koje su dodate za izgradnju iste aplikacije. U monolitnoj verziji, čak i ako kod postane neuredan, sve se dešava u istom procesu i možemo da pošaljemo nazad rezultat, bez potrebe da pozivamo previše pozadinskih servisa za izgradnju aktuelnog odgovora.

To zahteva da se obrati pažnja na način kako je pozvan pozadinski servis i nameće pitanja:

- Šta se dešava kada Booking UI ne može da komunicira sa PDF servisom za izveštaje, zbog razdvojene mreže ili laganog servisa?
- Da li Booking UI poziva druge servise sinhrono ili asinhrono?
- Kako će to uticati na vreme odgovora?

Potrebno je da imamo jaku strategiju da bismo mogli da odgovorimo na sva ova pitanja.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

SKLADIŠTENJE I DELJENJE PODATAKA

Efikan mikroservis treba da bude nezavisan od drugih mikroservisa i ne bi trebalo da deli bazu podataka.

Problematici su i skladištenje i deljenje podataka. Efikan mikroservis treba da bude nezavisan od drugih mikroservisa i ne bi trebalo da deli bazu podataka.

Šta to znači za našu aplikaciju za rezervacije hotela?

To nameće pitanja, kao, na primer:

- Da li koristimo iste korisničke ID-ove u svim bazama podataka ili imamo nezavisne ID-ove u svakom servisu i čuvamo ih kao skrivene detalje implementacije?
- Kada je korisnik dodat u sistem, da li repliciramo neke informacije u drugim bazama podataka servisa pomoću strategija, kao što je „pumpanje“ podataka, ili je to preterivanje?
- Kako da rukujemo uklanjanjem podataka?

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

TESTIRANJE

Aplikacija razvijena preko mikroservisa ima mnogo gradivnih blokova, i samim tim mnogo više mogućnosti da nešto pođe naopako.

Na kraju, kada budemo želeli da izvršimo testove s kraja na kraj i da rasporedimo celu aplikaciju, suočićemo se sa mnogo gradivnih blokova. Potrebno je da imamo robustan i agilan proces raspoređivanja da bismo bili efikasni.

Potrebno je da se malo poigramo celom aplikacijom kada je razvijamo. Ne možemo u potpunosti da testiramo kod ako imamo samo jedan deo slagalice.

Srećom, postoje mnogi alati koje olakšavaju raspoređivanje aplikacija koje su građene pomoću nekoliko komponenata. A svi ti alati pomogli su u uspešnosti i prihvatanju mikroservisa, ali i obrnuto - mikroservisi su pomogli prihvatanje alata.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

▼ Poglavlje 5

Flask mikroservis sa razvoj veb aplikacija

UVOD U FLASK

Flask predstavlja mikroservisno okruženje za razvoj veb aplikacija u Python programskom jeziku.



Slika 5.1 Flask logo. [Izvor: <https://flask.palletsprojects.com/en/2.0.x>]

Python programski jezik poseduje mnogo paketa za razvoj veb aplikacija, koji se nazivaju i radnim okvirima (en. **frameworks**). Najpoznatiji su Django, Pyramid, Tornado, i Flask. U nastavku lekcije i u sledećim lekcijama obradiće se Flask paket.

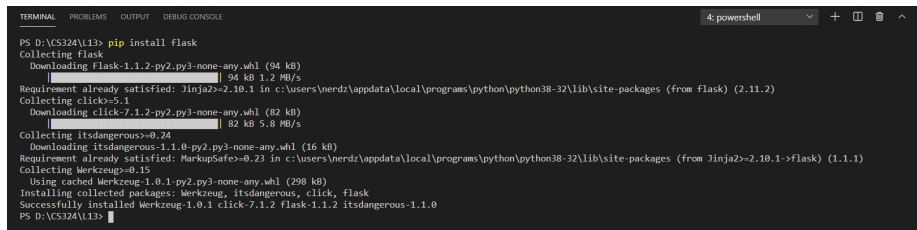
Flask predstavlja mikroservisno okruženje za razvoj veb aplikacija u Python programskom jeziku. Flask, budući da je mikroservis, ne zahteva dodatne biblioteke ili alate.

Instalacija

Kao i ostali paket, Flask se instalira preko pip-a:

```
pip install flask
```

Kada se instalira paket, instaliraće se i Jinja2 paket koji dolazi uz Flask i predstavlja **endžin** (en. **engine**) za šablone veb aplikacija u Python-u. Jinja2 notacija koristiće se u sklopu Flask aplikacija.



Slika 5.2 Instalacija Flask paketa. [Izvor: Autor]

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

MINIMALNA FLASK APLIKACIJA

Potrebno je postaviti flask promenljivu, i pokrenuti flask iz terminala.

Minimalna flask aplikacija se može naći na veb stranici Flask projekta, na **Quickstart** stranici.

Ovaj kod, kao i njegova proširenja, koristiće se za opisivanje paketa.

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'
```

Objašnjenje koda

Najpre je izvršen uvoz klase **Flask** iz paketa **flask**. Ona je napravljena instanca klase sa imenom **__name__**. Ova promenljiva je specijalna Python promenljiva i odnosi se na ime skripte. Podrazumevano ime prilikom pokretanja aplikacije jeste **__main__**,

U četvrtom redu koda, napravljen je dekorator **.route()** sa parametrom **('/')** koja govori o početnoj ruti veb aplikacije.

Definiše se funkcija **hello_world()** koja će samo vratiti string **"Hello, World!"**

Kada se pokrene ovaj program, ne javljaju se greške, ali se ništa i ne dešava.

Potrebno je najpre podesiti Flask promenljivu u terminalu, u putanji gde se nalazi Python datoteka (ovde nazvana **hello.py**) ukucati:

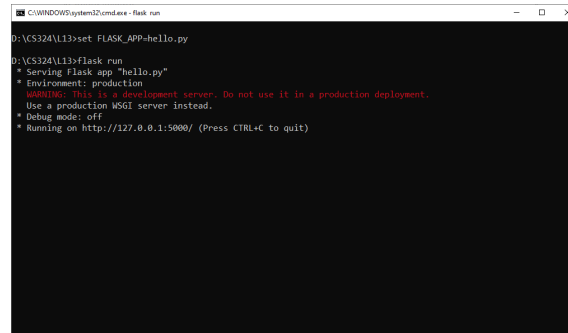
```
# windows
set FLASK_APP=hello.py
# power shell
$env:FLASK_APP = "hello"

# mac/linux
export FLASK_APP=hello.py
```


Nakon toga, treba pokrenuti Flask komandom

```
flask run
```

Sada se pokrenula flask aplikacija, što se može videti na samom terminalu.

A screenshot of a Windows command prompt window titled 'C:\WINDOWS\system32\cmd.exe - flask run'. The prompt shows the user has navigated to 'D:\CS324\L13' and entered 'set FLASK_APP=hello.py'. After typing 'flask run', the output shows: 'Serving Flask app "hello.py"', 'Environment: production', a warning 'WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.', 'Debug mode: off', and 'Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)'.

```
C:\WINDOWS\system32\cmd.exe - flask run
D:\CS324\L13>set FLASK_APP=hello.py
D:\CS324\L13>flask run
 * Serving Flask app "hello.py"
 * 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)
```

Slika 5.3 Pokretanje flask aplikacije. [Izvor: Autor]

VIDEO OBJAŠNJENJE ZA MINIMALNU FLASK APLIKACIJU

Sledi video objašnjenje minimalne Hello World flask aplikacije

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

▼ Poglavlje 6

Pokazne vežbe #13

ZADATAK #1

Pokazne vežbe rade se okvirno 45 min

Zadatak #1 (45 min)

Napraviti novo virtuelno okruženje u Python-u pod nazivom venv.

Instalirati paket flask.

Napraviti Flask aplikaciju sa rutama za stranicu "static", koja se nalazi u funkciji *prva_strana* u kojoj treba ukucati sledeći HTML kod:

```
<!DOCTYPE html>
  <html lang="en">
  <head>

    <!-- Declared Vars To Go Here -->

    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Metadata -->
    <meta name="description" content="">
    <meta name="author" content="">

    <link rel="icon" href="mysource_files/favicon.ico">

    <!-- Page Name and Site Name -->
    <title>Page Name - Squiz Matrix HTML Example</title>

    <!-- CSS -->
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/
bootstrap.min.css" rel="stylesheet">
    <link href="mysource_files/style.css" rel="stylesheet">

  </head>

  <body>

    <div class="container">
```

```
<header class="header clearfix" style="background-color: #ffffff">

    <!-- Main Menu -->
    <nav>
    <ul class="nav nav-pills pull-right">
        <li class="active"><a href="#">Home</a></li>
        <li><a href="#">About</a></li>
        <li><a href="#">Contact</a></li>
    </ul>
    </nav>

    <!-- Site Name -->
    <h1 class="h3 text-muted">Site Name</h1>

    <!-- Breadcrumbs -->
    <ol class="breadcrumb">
    <li><a href="#">Home</a></li>
    <li><a href="#">Level 1</a></li>
    <li class="active">Level 2</li>
    </ol>

</header>

<div class="page-heading">

    <!-- Page Heading -->
    <h1>Page Heading</h1>

</div>

<div class="row">

    <div class="col-sm-3">

        <!-- Sub Navigation -->
        <ul class="nav nav-pills nav-stacked">
            <li><a href="#">Level 2</a></li>
            <li class="active"><a href="#">Level 2</a>
            <ul>
                <li><a href="#">Level 3</a></li>
                <li><a href="#">Level 3</a></li>
                <li><a href="#">Level 3</a></li>
            </ul>
            </li>
            <li><a href="#">Level 2</a></li>
        </ul>

    </div>

    <div class="col-sm-6">

        <div class="page-contents">
```

```

        <!-- Design Body -->
        <h2>Sub Heading</h2>
        <p>Donec id elit non mi porta gravida at eget metus. Maecenas
faucibus mollis interdum.</p>
        <h4>Sub Heading</h4>
        <p>Morbi leo risus, porta ac consectetur ac, vestibulum at
eros. Cras mattis consectetur purus sit amet fermentum.</p>
        <h4>Sub Heading</h4>
        <p>Maecenas sed diam eget risus varius blandit sit amet non
magna.</p>

    </div>

</div>

<div class="col-sm-3">

    <!-- Login Section -->
    <h2>Login</h2>

    <!-- Search Section -->
    <h2>Search</h2>

    <!-- Nested Right Column Content -->

</div>

</div>

<footer class="footer">
    <p class="pull-right">
        <!-- Last Updated Design Area-->
        Last Updated: Wednesday, January 6, 2016
    </p>
    <p>&copy; 2016 Company, Inc.</p>
</footer>

</div> <!-- /container -->

</body>
</html>

```

Napomena:

HTML kod za stranicu static se nalazi u posebnoj datoteci. Treba uvesti funkciju **prva_strana** u glavnu datoteku.

Rešenje:

U terminalu treba uraditi sledeće:

```

C:\python -m venv venv
(venv) C:\pip install flask

```

Datoteka **PV01.py**:

```
from flask import Flask
from PV01_strana01 import prva_strana
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

@app.route('/static')
def page01():
    content = prva_strana()
    return content
```

Datoteka **PV01_strana01.py**:

```
def prva_strana():
    content = """
    <!DOCTYPE html>
    <html lang="en">
    <head>

        <!-- Declared Vars To Go Here -->

        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">

        <!-- Metadata -->
        <meta name="description" content="">
        <meta name="author" content="">

        <link rel="icon" href="mysource_files/favicon.ico">

        <!-- Page Name and Site Name -->
        <title>Page Name - Squiz Matrix HTML Example</title>

        <!-- CSS -->
        <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/
bootstrap.min.css" rel="stylesheet">
        <link href="mysource_files/style.css" rel="stylesheet">

    </head>

    <body>

        <div class="container">

            <header class="header clearfix" style="background-color: #ffffff">

                <!-- Main Menu -->
                <nav>
```

```

        <ul class="nav nav-pills pull-right">
            <li class="active"><a href="#">Home</a></li>
            <li><a href="#">About</a></li>
            <li><a href="#">Contact</a></li>
        </ul>
    </nav>

    <!-- Site Name -->
    <h1 class="h3 text-muted">Site Name</h1>

    <!-- Breadcrumbs -->
    <ol class="breadcrumb">
        <li><a href="#">Home</a></li>
        <li><a href="#">Level 1</a></li>
        <li class="active">Level 2</li>
    </ol>

</header>

<div class="page-heading">

    <!-- Page Heading -->
    <h1>Page Heading</h1>

</div>

<div class="row">

    <div class="col-sm-3">

        <!-- Sub Navigation -->
        <ul class="nav nav-pills nav-stacked">
            <li><a href="#">Level 2</a></li>
            <li class="active"><a href="#">Level 2</a>
                <ul>
                    <li><a href="#">Level 3</a></li>
                    <li><a href="#">Level 3</a></li>
                    <li><a href="#">Level 3</a></li>
                </ul>
            </li>
            <li><a href="#">Level 2</a></li>
        </ul>

    </div>

    <div class="col-sm-6">

        <div class="page-contents">

            <!-- Design Body -->
            <h2>Sub Heading</h2>
            <p>Donec id elit non mi porta gravida at eget metus. Maecenas
faucibus mollis interdum.</p>

```

```

        <h4>Sub Heading</h4>
        <p>Morbi leo risus, porta ac consectetur ac, vestibulum at
eros. Cras mattis consectetur purus sit amet fermentum.</p>
        <h4>Sub Heading</h4>
        <p>Maecenas sed diam eget risus varius blandit sit amet non
magna.</p>

    </div>

</div>

<div class="col-sm-3">

    <!-- Login Section -->
    <h2>Login</h2>

    <!-- Search Section -->
    <h2>Search</h2>

    <!-- Nested Right Column Content -->

</div>

</div>

<footer class="footer">
    <p class="pull-right">
        <!-- Last Updated Design Area-->
        Last Updated: Wednesday, January 6, 2016
    </p>
    <p>&copy; 2016 Company, Inc.</p>
</footer>

</div> <!-- /container -->

</body>
</html>
"""
return content

```

VIDEO OBJAŠNJENJE ZA ZADATAK #1

Sledi video objašnjenje za zadatak #1

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

▼ Poglavlje 7

Individualne vežbe #13

INDIVIDUALNE VEŽBE

Individualne vežbe #13 rade se okvirno 45 minuta.

Individualne vežbe #13 sastoje se iz dva do tri zadatka.

Vreme za izradu svih zadataka okvirno je 90 minuta.

Studenti u dogovoru sa predmetnim nastavnikom i asistentom dobijaju zadatke individualnih vežbi.

▼ Poglavlje 8

Domaći zadatak

DOMAĆI ZADATAK #13

Domaći zadatak #13 okvirno se radi 3h

Domaći zadatak #13 daje se u dogovoru sa predmetnim nastavnikom i/ili asistentom.

Predaja domaćeg zadatka:

Tradicionalni studenti:

Domaći zadatak treba dostaviti najkasnije 7 dana nakon predavanja, za 100% poena. Nakon toga poeni se umanjuju za 50%.

Internet studenti:

Domaći zadatak treba dostaviti najkasnije 10 dana pred polaganje ispita. Domaći zadaci se brane!

Domaći zadatak poslati dr Nemanji Zdravkoviću: nemanja.zdravkovic@metropolitan.ac.rs

Obavezno koristiti uputstvo za izradu domaćeg zadatka.

Uz .doc dokument (koji treba sadržati i screenshot svakog urađenog zadatka kao i komentare za zadatak), poslati i izvorne i dodatne datoteke.

▼ Poglavlje 9

Zaključak

ZAKLJUČAK

Zaključak lekcije #13

Rezime:

U ovoj lekciji najpre je objašnjeno kako napraviti virtuelno okruženje u Python-u i na taj način postaviti nezavistan Python interpreter u odnosu na sistemski.

Nakon toga, objašnjene su monolitne i mikroservisne arhitekture, sa naglaskom na prednosti mikroservisa.

Obrađen je Flask paket, i kako napraviti jednostavnu Flask aplikaciju sa statičkim HTML stranicama.

Literatura:

- Tarek Ziad, Python - Razboj mikroservisa, Kompjuter biblioteka, 2017. (prevod iste knjige Packt Publishing-a)
- David Beazley, Brian Jones, Python Cookbook: Recipes for Mastering Python 3, 3rd edition, O'Reilly Press, 2013.
- Mark Lutz, Learning Python, 5th Edition, O'Reilly Press, 2013.
- Andrew Bird, Lau Cher Han, et. al, The Python Workshop, Packt Publishing, 2019.
- Al Sweigart, Automate the boring stuff with Python, 2nd Edition, No Starch Press, 2020.

DODATNE REFERENCE

Dodatne reference za mikroservise

[1] Martin Fowler. „Microservices”

[2] Newman, Sam. Building Microservices, O'Reilly Media. ISBN 978-1491950357.

[3] Wolff, Eberhard. Microservices: Flexible Software Architectures

[4] „Micro-services for performance”. Vanilla Java

[5] Nadareishvili, I., Mitra, R., McLarty, M., Amundsen, M., Microservice Architecture: Aligning Principles, Practices, and Culture, O'Reilly 2016.

[6] „IFS: Microservices Resources and Positions”. hsr.ch

- [7] Nicola Dragoni, Schahram Dustdar, Stephan T. Larsen, Manuel Mazzara. „Microservices: Migration of a Mission Critical System”
- [8] Nicola Dragoni, Ivan Lanese, Stephan Thordal Larsen, Manuel Mazzara, Ruslan Mustafin, Larisa Safina. „Microservices: How To Make Your Application Scale” (PDF)
- [9] Manuel Mazzara, Kevin Khanda, Ruslan Mustafin, Victor Rivera, Larisa Safina, Alberto Sillitti. „Microservices Science and Engineering”