Proširenja Jupyter notebook razvojne okoline

Goran, Alković

Undergraduate thesis / Završni rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:987871

Rights / Prava: Attribution-NoDerivs 3.0 Unported/Imenovanje-Bez prerada 3.0

Download date / Datum preuzimanja: **2024-03-29**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

zir.nsk.hr

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Goran Alković

PROŠIRENJA JUPYTER NOTEBOOK RAZVOJNE OKOLINE

ZAVRŠNI RAD

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Goran Alković

Matični broj: 43953/15–R

Studij: Informacijski sustavi

PROŠIRENJA JUPYTER NOTEBOOK RAZVOJNE OKOLINE

ZAVRŠNI RAD

Mentor :

Doc. dr. sc. Marcel Maretić

Varaždin, rujan 2018.

Goran Alković

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Cilj ovog rada je ukratko opisati proces izrade proširenja za razvojnu okolinu *Jupyter*. Prvo će biti ukratko biti objašnjena razvojna okolina *Jupyter* i njene mogućnosti. Nakon toga će biti opisano kako se *Jupyter* instalira, koje su njegove glavne komponente (jezgre) i kako ih instalirati i mijenjati te proces instalacije proširenja (eng. *extension*), upravljanja proširenjima i njihove uloge. Detaljnije će se opisati sama proširenja - njihovu strukturu, načine kako ih kreirati i implementirati. Navest će se i opisati neka od jednostavnijih i popularnijih proširenja (njihova uloga i struktura). Izradit će se jednostavno proširenje uz objašnjenje procesa rada i prikaz izvornog koda.

Ključne riječi: python; jupyter; jupyterlab; ekstenzija; proširenje; javascript; jezgra; typescript

Sadržaj

1. Uvod 1

2. O JupyterLab razvojnoj okolini 2
2.1. Povijest 2
2.2. JupyterLab 3
2.3. Instalacija JupyterLab-a 3
2.3.1. Instalacija uz *Anaconda* paket softvera 4
2.3.2. Instalacija pomoću `conda` sustava 4
2.3.3. Instalacija pomoću `pip` sustava 5
2.4. Mogućnosti JupyterLab razvojne okoline 5
2.4.1. Konzola 5
2.4.2. Terminal 8
2.4.3. Uređivač teksta 8
2.4.4. Bilježnice (*Notebooks*) 10
2.5. *JupyterHub* 12

3. Jezgre u JupyterLab-u 13
3.1. Dostupne jezgre 13
3.2. Instalacija jezgre 14

4. Proširenja u JupyterLab-u 16
4.1. Instalacija proširenja 16
4.2. Struktura proširenja 17
4.3. Izrada proširenja 18
4.4. Proširenje *Code snippet manager* 19
4.4.1. Umetanje isječka kôda 20
4.4.2. Upravljanje isječcima kôda 20

5. Zaključak 23

Popis literature 25

Popis slika 26

Prilog 1: Izvorni kôd proširenja - *TypeScript* 27

Prilog 2: Izvorni kôd proširenja - *CSS* 43

1. Uvod

Biti programer danas nije lako, ali to želi biti svatko. Radi se s praktički beskonačnim količinama podataka, postoji mnoštvo programskih jezika i pripadajućih alata, a taj broj se sve brže povećava.

S obzirom na velik broj jezika postoji podjela s obzirom na područje kojem su najprikladniji, pa je na primjer JavaScript najprikladniji razvoju za web, a Kotlin razvoju Android aplikacija. Naravno, neki jezici se mogu "provući" kroz više područja, na primjer C(++).

Python je jedan od jezika koji nalazi svoje primjene u više područja. Svojom jednostavnom sintaksom u kratko vrijeme je postao jedan od popularnijih prvih jezika za nove programere, ali i favorit među "programerima starog kova". Od strojnog učenja i web aplikacija pa sve do automatizacije uređaja u pametnim kućama na platformama poput *Raspberry Pi*-ja ovaj jezik je jedan od svestranijih objektno-orijentiranih jezika. [1]

Razvijen od strane Guido van Rossum-a svijetu je predstavljen 1991. godine. Trenutna verzija je 3.7.0. Uz standardnu instalaciju Pythona dobivamo i *IDLE*, jednostavno razvojno okružje za Python. Ne nudi velik set značajki, ali je za početnike i nezahtjevne korisnike dovoljan. [2]

Uz *IDLE*, pisanje programa je moguće u bilo kojem uređivaču teksta, ali se program mora izvršavati u konzoli (*Powershell* na Windows platformi, *Terminal* na Linux i Mac platformama). Brzo se pojavilo i mnoštvo drugih razvojnih okružja (*eng.* IDE, Integrated Development Environment), kao i proširenja za postojeća. Neka od njih su PyCharm, Visual Studio Code, Python tools for Visual Studio i drugi. Ova razvojna okružja olakašavaju rad i pomažu programerima u otkrivanju grešaka, testiranju koda i izradi sučelja.

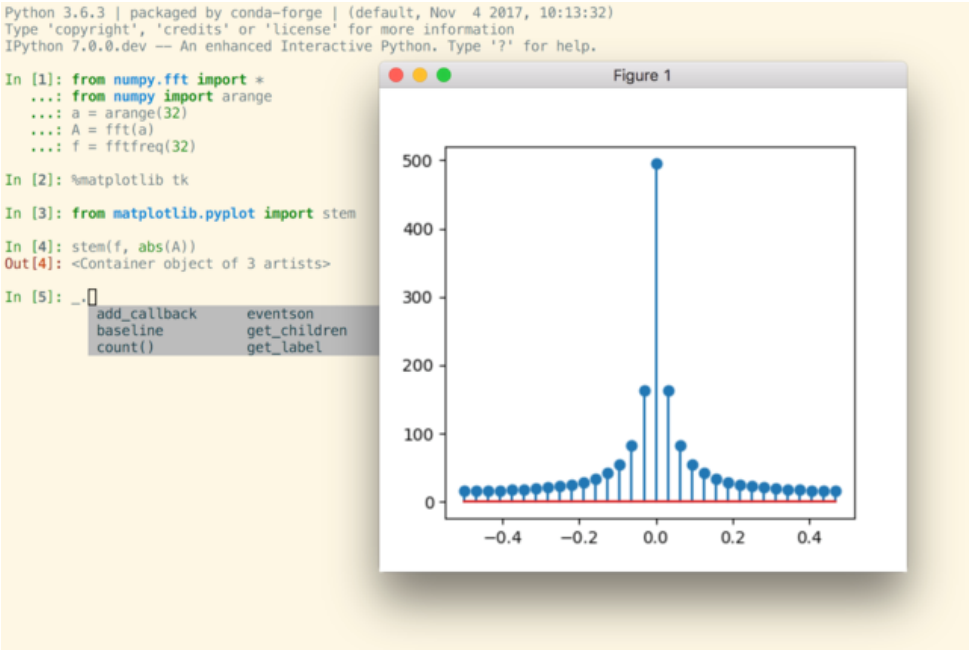
Nakon što je Python postao vrlo popularan među *podatkovnim znanstvenicima* (*eng.* data scientists) i statističarima, počele su se razvijati biblioteke i alati za uvoz, analizu i prikaz velikih količina podataka. Jedan od alata koji je popularizirao Python za data science je **Jupyter**.

U ovom radu će biti izrađeno proširenje za posljednju iteraciju *projekta Jupyter* i nasljednika Jupyter Notebooka - **JupyterLab** (više u sljedećem poglavlju). Iako je ime promijenjeno iz *Jupyter Notebook* u *JupyterLab*, sva funkcionalnost je prenešena te su dodane mnoge nove značajke i integracije. Kako će *JupyterLab* službeno zamijeniti *Jupyter Notebook* vrlo skoro, nije bilo previše smisla pisati proširenje za sustav star četiri godine koji će uskoro biti zamijenjen.

2. O JupyterLab razvojnoj okolini

2.1. Povijest

2011. Fernando Perez je nakon puno godina razvoja izdao *IPython*. IPython je proširio Pythonovo sučelje u naredbenom retku (*eng.* command prompt/command line, CLI - Command Line Interface) mogućnostima izlaza u bojama, prikaza grafikona i bogatih grafičkih elemenata (vidi Sliku 1). [3]



Slika 1: Sučelje IPythona u naredbenom retku (Izvor: [4])

Nakon što je IPython postao popularan, počelo se dodavati sve više značajki. Jedna od njih je bila i novi program - ***IPython Notebook***. Dobivši ideju od alata *Sage*, autori IPython Notebooka su napravili sustav gdje se sve radi u *bilježnicama* (*eng.* notebook). Svaka od bilježnica može sadržavati programski kôd ili tekst/multimediju u *Markdown* zapisu, a sve je smješteno u tzv. *ćelije*, kojih može biti više u jednoj bilježnici. [3]

Zapis podataka bilježnica vrši se u `.ipynb` datoteke, koje su po strukturi bazirane na JSON-u (JSON - *JavaScript Object Notation*, uz XML jedan od najpopularnijih načina zapisa podataka danas [5]).

IPython Notebook je vrlo brzo doživio popularnost unutar zajednice, pa je razvoj nastavio. Projekt je počeo dobivati pomoć od velikih tvrtki koje su donirale mnoštvo novca za nastavak razvoja i napredovanje IPython Notebooka. Na početku je IPython Notebook, kako i ime kaže, samo dolazio s podrškom za Python 2, ubrzo je dodana i podrška za verziju 3.

2014. godine je osnovana neprofitna tvrtka *Project Jupyter* (u daljnjem tekstu *Jupyter*) s ciljem da olakša pristup u područje data science-a i napravi izvrstan alat za data scientiste. Cilj je bio da sve što naprave bude besplatno i otvorenog koda (*eng.* open source). [6]

Za bazu i prvu verziju programa tvrtke Jupyter uzet je IPython Notebook, koji je sada preimenovan u **Jupyter Notebook** jer se uz osnovnu Python jezgru moglo instalirati i jezgre (*eng.* kernel, više u poglavlju 3) za druge programske jezike. [7]

Malo poslije je izdan i sustav *Jupyter Hub*, koji je omogućio mrežnu instalaciju i distribuciju Jupyter bilježnica unutar tvrtki, obrazovnih institucija i sl. [8]

2.2. JupyterLab

U veljači 2018. godine predstavljena je sljedeća generacija Jupyter Notebooka - **JupyterLab**. Razvijen uz pomoć zajednice i po zahtjevima zajednice, JupyterLab je napisan "iz nule" s ciljem da se iskustvo Jupyter Notebooka nadogradi novim, brzim i jednostavnim sučeljem te poboljšanim sustavom proširenja. [9]

Novo je sučelje puno jednostavnije, bolje prati trenutne *web development* standarde i nudi mnoštvo fleksibilnosti - npr. svaki prozor se može premjestiti po želji korisnika, poput *klasičnih desktop* razvojnih okružja. Svaki element sučelja je zasebno proširenje, što poboljšava stabilnost i brzinu, te olakšava ažuriranja. Glavni jezik za izradu proširenja je sada *TypeScript* (više u poglavlju 4). [10]

Očuvana je kompatibilnost s prethodnim verzijama Jupyter Notebooka i IPython Notebooka, pa se sve prijašnje bilježnice mogu lako prenijeti u JupyterLab.

Proširenja zahtijevaju izmjene za razliku od Jupyter Notebooka, ali promjene nisu ogromne. Sva ugrađena proširenja iz Jupyter Notebooka su dostupna i za JupyterLab.

Trenutna verzija JupyterLab-a je 0.34.3. *Prva* verzija (1.0) će izaći u trećem kvartalu 2018. Sustav je sada na razini stabilnosti Jupyter Notebooka i spreman za svakodnevno korištenje te je većina API-ja finalizirana (API - *Application Programming Interface*, sučelje prema aplikaciji koje se može koristiti za izradu proširenja i alata za komunikaciju nekog alata s drugim alatima).

Opisi konfiguracije, instalacije, korištenja i izrade proširenja u ovom radu će biti bazirani na sustavu JupyterLab.

2.3. Instalacija JupyterLab-a

Preduvjeti za instalaciju JupyterLaba su:

- Moderni web preglednik (Google Chrome, Mozilla Firefox, Apple Safari, Microsoft Edge 16 ili noviji, ...)
- Jupyter Notebook 4.3 ili noviji

Internet Explorer nije podržan zbog korištenja modernih web tehnologija poput *CSS varijabli*, koje u tom pregledniku nisu podržane.

Postoje četiri načina za instalaciju JupyterLab-a:

1. Instalacija kao dio *Anaconda* paketa softvera
2. Pomoću `conda` sustava
3. Pomoću `pip` sustava
4. Kao okruženje `pipenv` sustavom

(prema [11])

2.3.1. Instalacija uz *Anaconda* paket softvera

Anaconda Cloud (dalje u tekstu *Anaconda*) je paket softvera prvotno namijenjen *data scientistima*. Baziran na `conda` sustavu instalacije paketa razvijenom od strane iste tvrtke, omogućava laku instalaciju na svim platformama (Windows, Linux, MacOS) pomoću grafičkog sučelja. [12]

Osnovna instalacija Anaconde sadrži sljedeće programe:

- Jupyter Notebook
- JupyterLab
- *Spyder* - razvojno okružje za Python namijenjeno znanstvenicima
- *R Studio* - glavno okružje za razvoj u R programskog jeziku
- *glueviz* - softver za naprednu vizualizaciju podataka
- *Visual Studio Code* - napredno razvojno okružje za mnoštvo programskih jezika

Uz ove programe instalira se i mnoštvo Python biblioteka poput *NumPy*-ja, *SciPy*-ja, *Numba*-e i sl. koje su posebno korisne znanstvenicima, matematičarima i sl. [12]

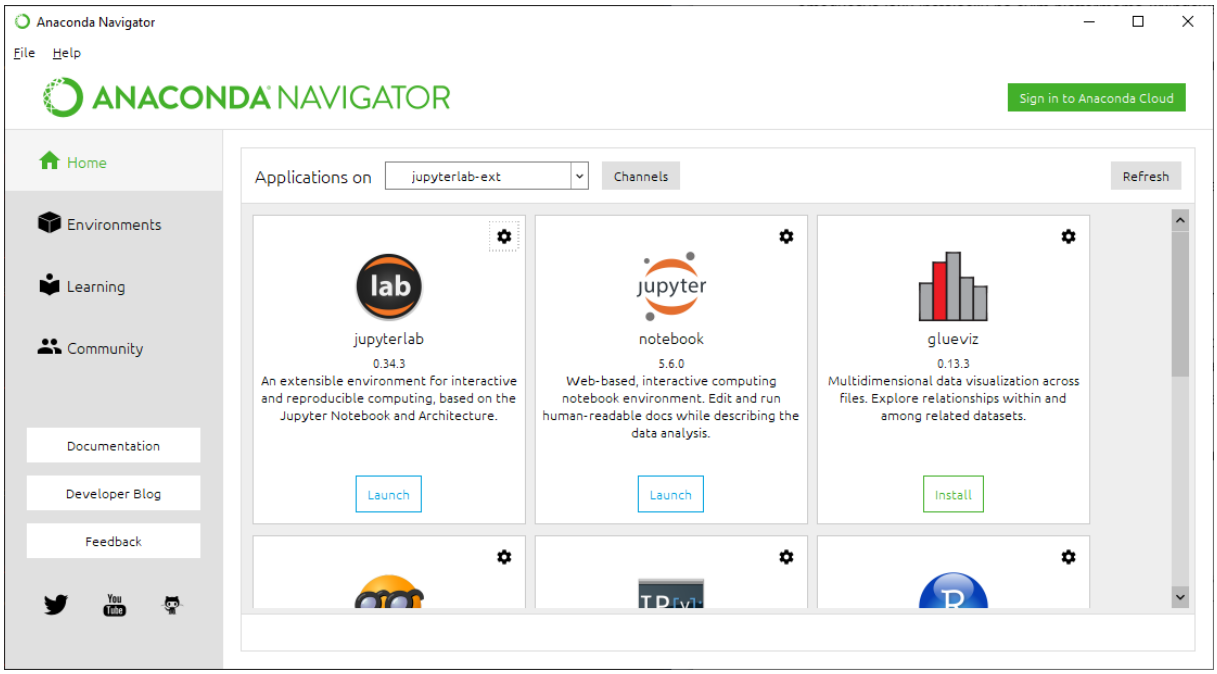
Instaliraju se i biblioteke za strojno učenje poput *TensorFlow*-a, što čini Anacondu odličnim paketom softvera za ljude koji ulaze u svijet strojnog učenja. [12]

Sve se upravlja pomoću programa *Anaconda Navigator* (vidi sliku 2).

Anaconda se može preuzeti na službenim web stranicama Anaconda projekta (`https://www.anaconda.com/download/`)

2.3.2. Instalacija pomoću `conda` sustava

Ako koristimo već spomenuti `conda` sustav za instalaciju paketa dovoljno je pokrenuti naredbu `conda install -c conda-forge jupyterlab` za instalaciju. [11]



Slika 2: Sučelje *Anaconda Navigatora* u operacijskom sustavu Windows 10 (vlastita izrada)

2.3.3. Instalacija pomoću `pip` sustava

Python uz svoju instalaciju instalira i `pip` - Pythonov softver za upravljanje paketima. Preko njega možemo instalirati JupyterLab pokretanjem naredbe `pip install jupyterlab`. [11]

2.4. Mogućnosti JupyterLab razvojne okoline

JupyterLab nudi četiri glavna modula:

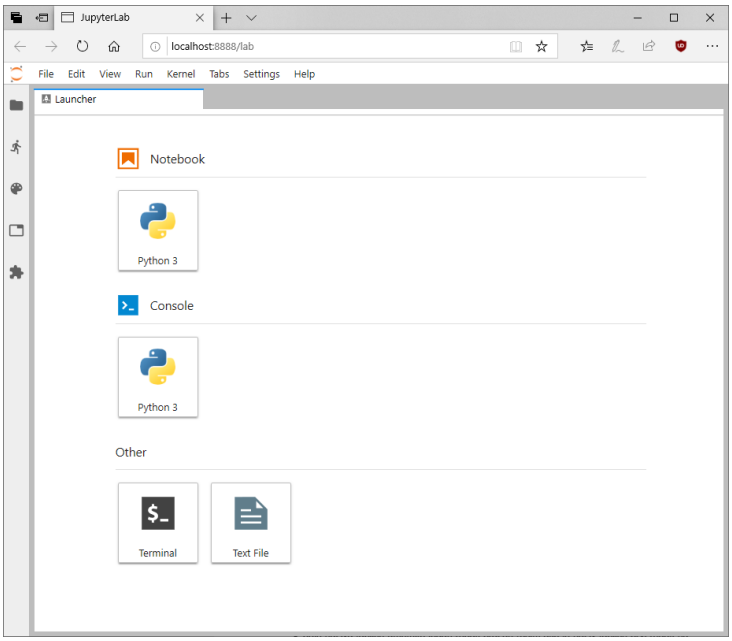
- Konzola (*eng.* Console)
- Bilježnice (*eng.* Notebooks)
- Pristup terminalu/naredbenom retku
- Uređivač teksta (*eng.* Text editor)

2.4.1. Konzola

Konzola omogućava izvršavanje naredbi jezika odabrane jezgre, kao što bi to inače radili u naredbenom retku. Omogućava brz i interaktivan razvoj jednostavnih programa. [13]

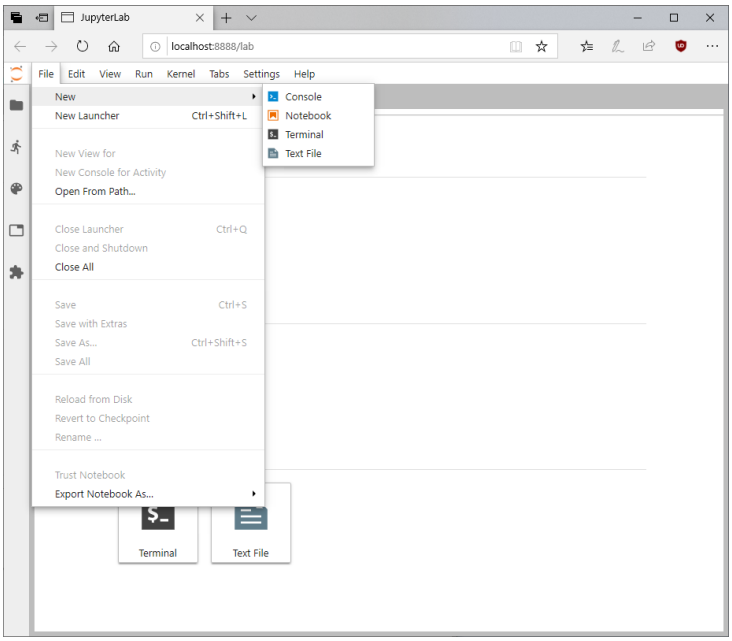
Konzola se može pokrenuti na više načina:

- Pokretanjem iz *Launcher prozora* klikom na željeni jezik pod *Console*



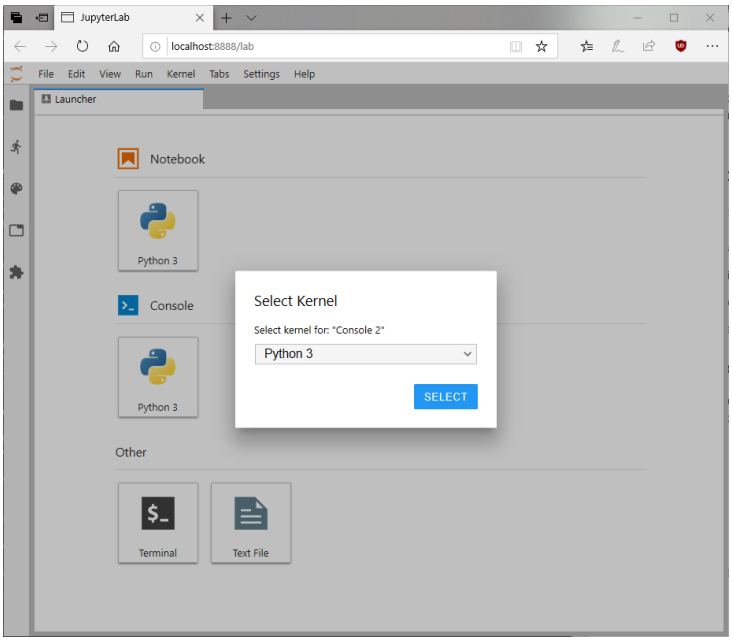
Slika 3: *Launcher* prozor JupyterLab-a (vlastita izrada)

- Korištenjem izbornika `File`, naredbom `New`



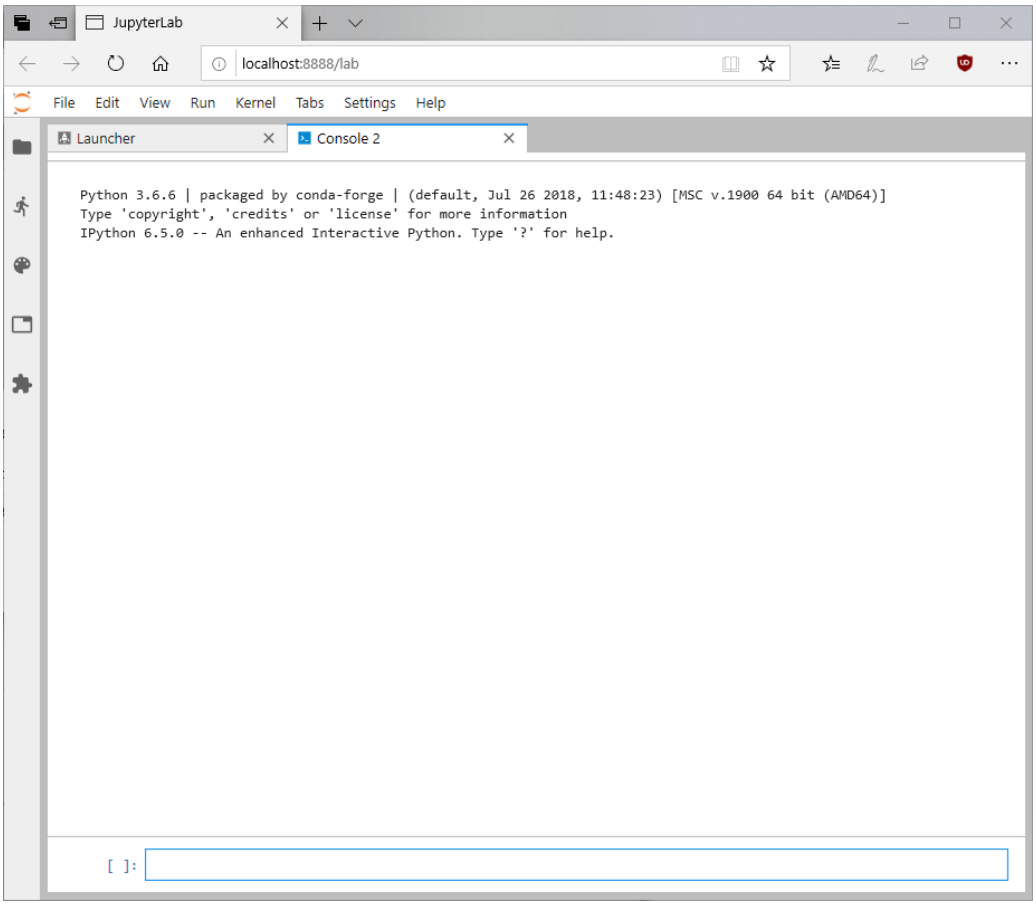
Slika 4: Izbornik `File > New` (vlastita izrada)

Zatim odabirom jezgre u dijaloškom okviru:



Slika 5: Prozor za odabir *jezgre* (vlastita izrada)

Nakon pokretanja, prozor konzole izgleda kao na Slici 6. U središnjem dijelu prozora vidimo izlaz *konzole*, a na dnu je polje za unos naredbi (naznačeno simbolom `[ ]:`)

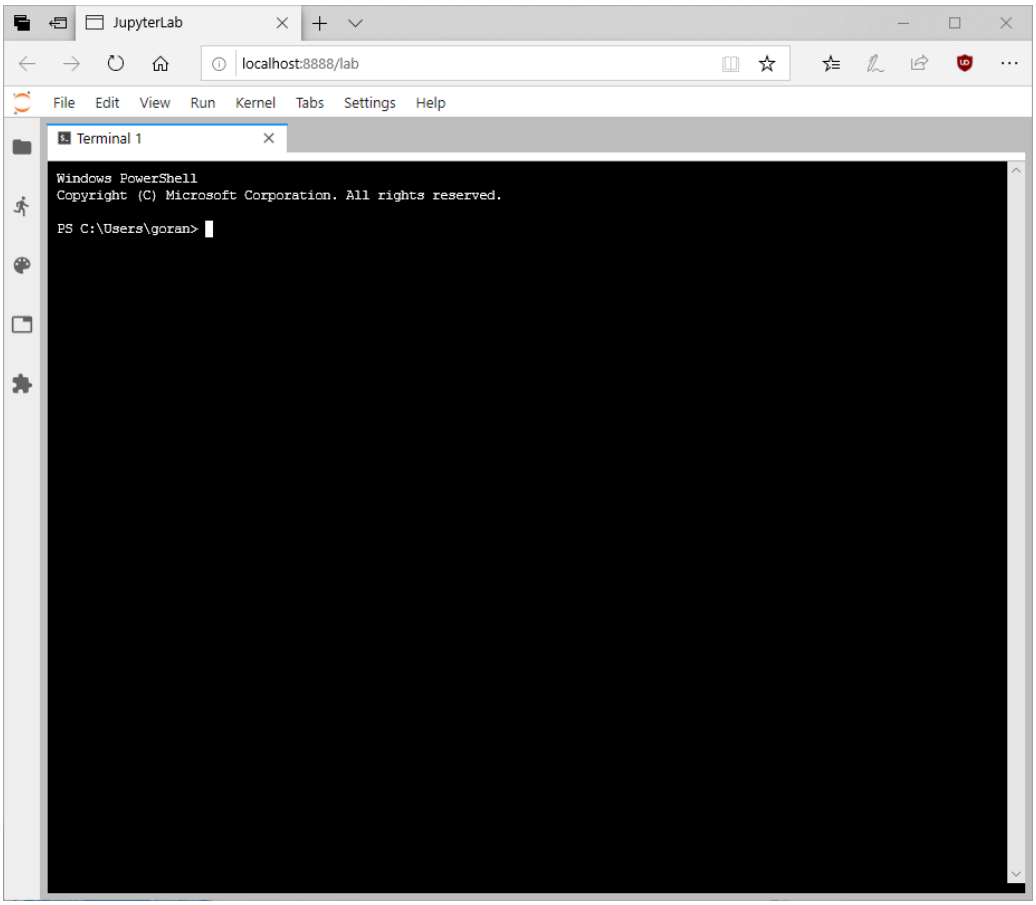


Slika 6: Prozor za odabir *jezgre* (vlastita izrada)

2.4.2. Terminal

Terminal omogućava pristup sustavskom naredbenom retku (npr. `bash` na Linuxu ili `PowerShell` na Windowsu). Kako je terminal pokrenut na poslužitelju koji pokreće JupyterLab, prava pristupa i struktura direktorija ovise o računalu, odnosno korisničkom računu na kojem je poslužitelj za JupyterLab pokrenut. Za većinu korisnika je to lokalno računalo. [14]

Terminal se pokreće na isti način kao i Konzola, preko *Launcher* sučelja ili izbornika `File`. Izgled sučelja terminala na operacijskom sustavu Windows 10 vidljivo je na Slici 7.

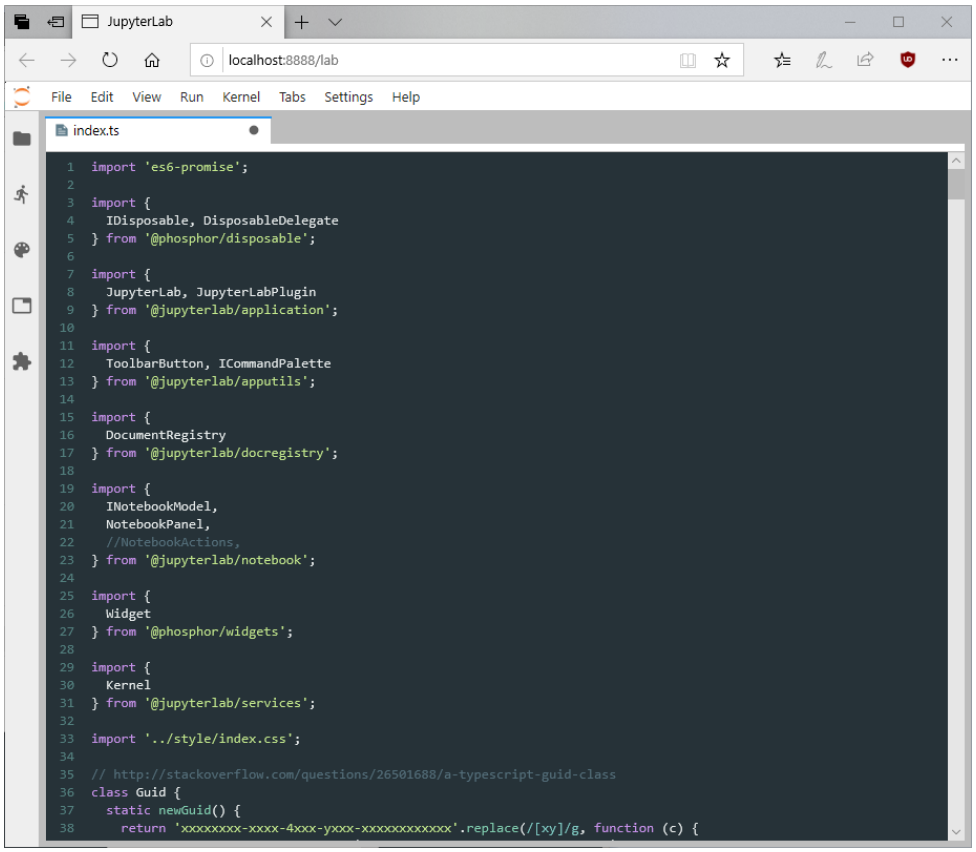


Slika 7: Prozor *Terminal* (vlastita izrada)

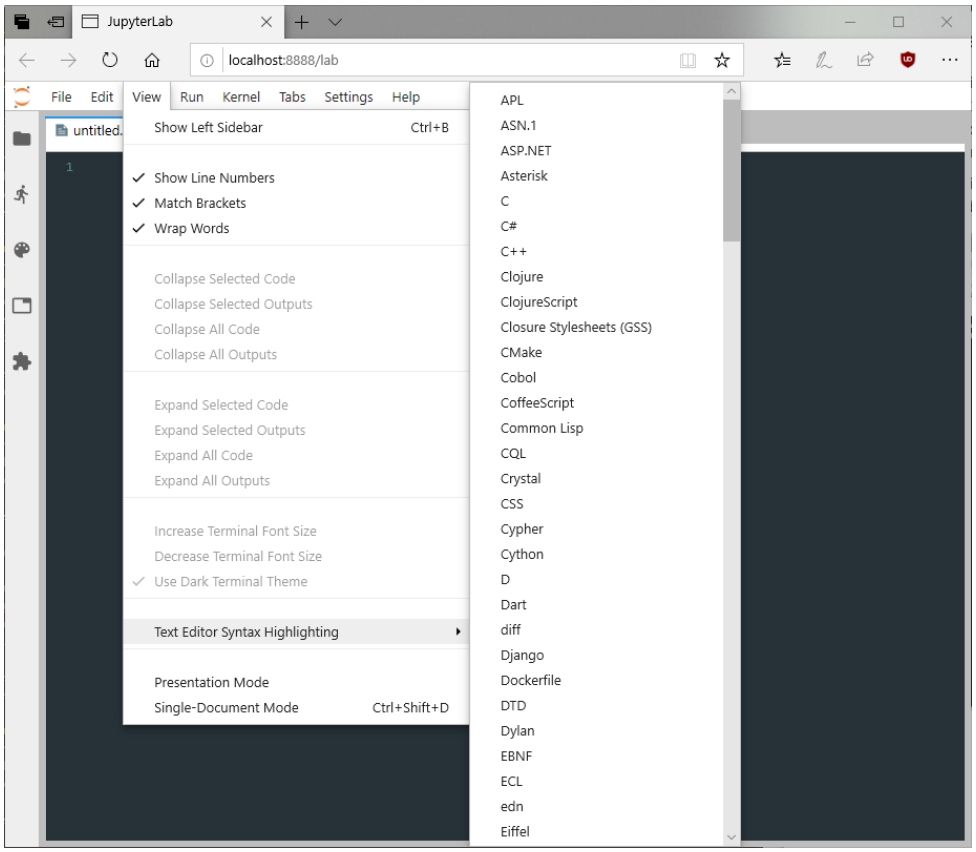
2.4.3. Uređivač teksta

Prvotno predstavljen u Jupyter Notebook-u, JupyterLab nudi novi tekstualni uređivač. Podržava sve današnje formate datoteka i jezike, može se konfigurirati da koristi različite tipkovne prečace, različite sheme boja i sl. [15]

Uređivač teksta se također pokreće preko *Launcher* sučelja ili izbornika `File`. Također, ako se iz preglednika datoteka otvori bilo koja kompatibilna datoteka otvorit će se u uređivaču teksta. Na Slici 8 vidljivo je sučelje uređivača teksta, a na Slici 9 vidljiv je način ručne promjene trenutnog jezika iz izbornika *View*. U izborniku *Settings* se nalaze ostale opcije uređivača, poput sheme boja.



Slika 8: Prozor uređivača teksta s otvorenom datotekom (vlastita izrada)



Slika 9: Postavljanje teme uređivača teksta (vlastita izrada)

2.4.4. Bilježnice (*Notebooks*)

Bilježnice su najbitniji dio JupyterLab-a, bez njih sustav ne bi bio toliko popularan. To su dokumenti koji na jedno mjesto stavljaju kôd koji se može interkativno izvršavati i pridružuju mu slike, bogato oblikovan tekst (koristeći *Markdown*), interkativne vizualizacije, LaTeX jednadžbe i još mnogo toga. [16]

Slično ostalim modulima, bilježnice se mogu stvoriti na više načina: iz prozora *Launcher*, iz izbornika `File` ili otvaranjem postojeće bilježnice iz preglednika datoteka. Izgled prozora bilježnice vidljiv je na Slici 10.

Format, izgled i tipkovnički prečaci su preneseni iz Jupyter Notebooka. JupyterLab dodaje mnoštvo novih značajki, a neke od njih su:

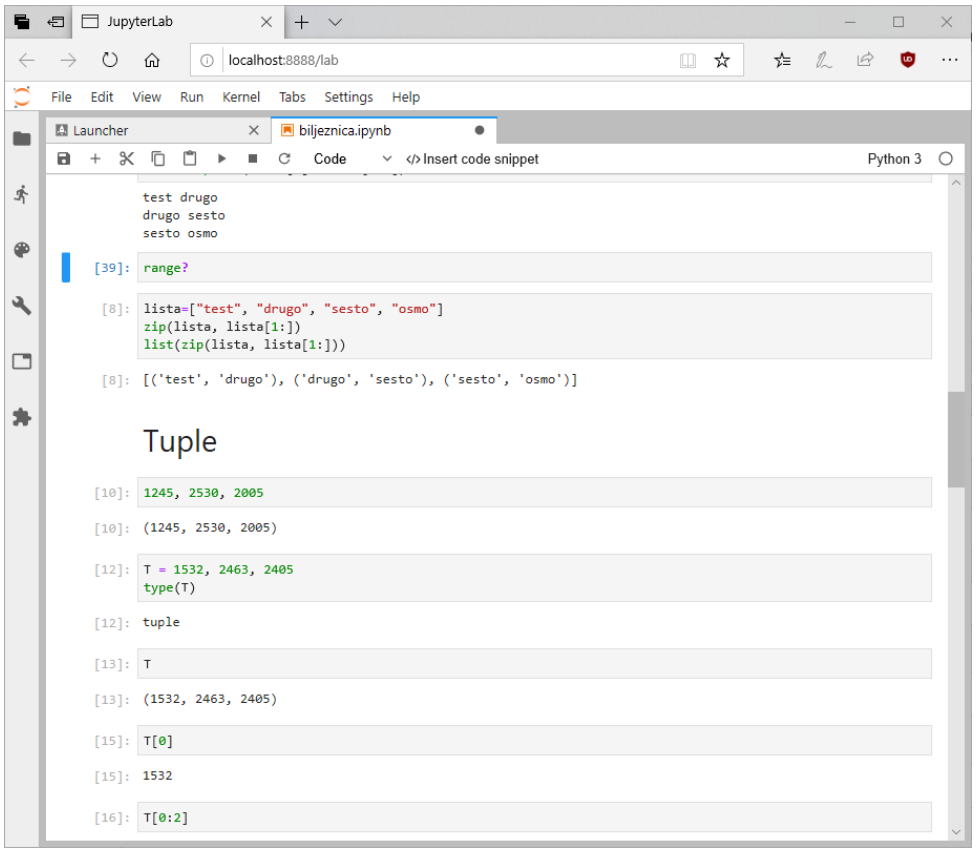
- Ćelije se mogu premještati unutar bilježnice klikom, držanjem i pomicanjem miša
- Za brzo kopiranje između bilježnica dovoljno je odvući ćeliju iz jedne bilježnice u drugu
- Može se otvoriti više prozora iste bilježnice, ti će prozori biti sinkronizirani
- Ćelije s kôdom i njihovi izlazi se mogu smanjiti plavim gumbom pokraj ćelije ili iz izbornika *View*
- Može se omogućiti *scrollanje* sadržaja izlaza ćelije ako postane predug
- Izlaz ćelije se može odvojiti u drugi prozor i bit će sinkroniziran s ćelijom
- Poboljšano je dopunjavanje teksta, dostupno pritiskom gumba `Tab`
- Za lakše prepoznavanje elemenata tipkama `Shift` i `Tab` se mogu uključiti zaslonski opisi (*eng.* tooltip)
- Uz bilježnicu se može pridružiti konzola koja može sekvencijalno bilježiti sve učinjene radnje

(prema [16])

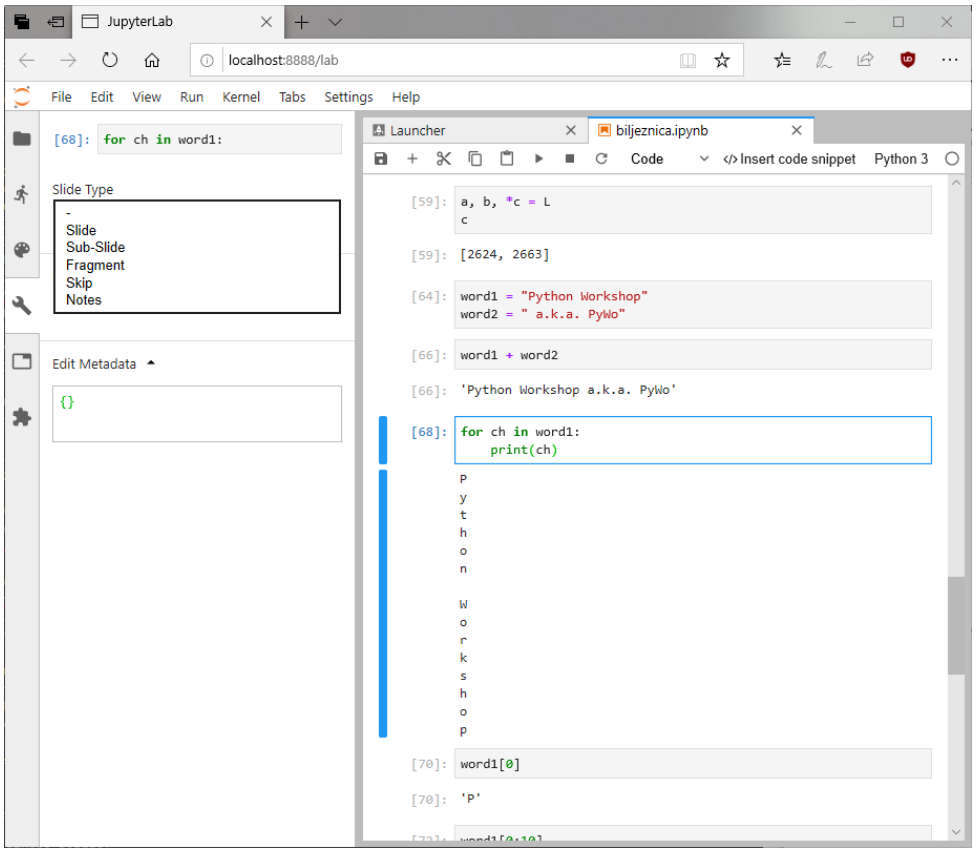
Kako su bilježnice vezane uz jednu *jezgru*, u jednoj bilježnici se može nalaziti samo kôd jednog programskog jezika. U budućnosti je planirano miješanje programskih jezika unutar jedne bilježnice.

Ovakva struktura je savršena za izradu lekcija, prezentacija i slično. Kako je sve napisano u web tehnologijama jednom napisana bilježnica se može spremiti kao web stranica ili objaviti na webu, gdje postoje preglednici `.ipynb` formata, koji omoguću interakciju s postojećim sadržajem bilježnice.

Vrlo korisna značajka prebačena iz Jupyter Notebooka je i alat za izradu prezentacija pomoću `reveal.js` biblioteke. U postavkama ćelije se određene ćelije (vidi Sliku 11) mogu odrediti kao slajdovi, pa se pri izvozu dobije web stranica s uključenim kontrolama koje služe za hijerarhijsku navigaciju kroz prezentaciju. Kôd također može biti uključen.



Slika 10: Sučelje bilježnice (vlastita izrada)



Slika 11: Mogućnosti za oblikovanje ćelije kao slajda (vlastita izrada)

2.5. *JupyterHub*

S rastom popularnosti Jupyter Notebooka ljudi su pronašli kreativne načine za korištenje programa. Jupyter se pokazao vrlo koristan i popularan u obrazovnim ustanovama i tvrtkama. Međutim, tu je nastao problem. Što ako više osoba želi paralelno raditi na jednom *notebook*-u? Tadašnji sustav to nije omogućavao, jer bi sve stvorene varijable i zauzeta memorija bili zajednički za sve. [17]

Razvojni tim Jupytera je zato krenuo u razvoj nove verzije *Jupyter Notebook*-a, tada nazvanoj *Jupyter Multi-User* i *Jupyter Classroom*, koji su omogućavali da više ljudi radi na jednoj bilježnici, a da svaka osoba ima odvojene varijable i dio memorijskog prostora.

S vremenom su *Multi-user* i *Classroom* verzije spojene u jedan proizvod - *JupyterHub*. Sustav radi na sličnom principu kao i *obični* Jupyter, na poslužitelju se pokreću *jezgre* i sve potrebno za web sučelje. Razlika je da se u *JupyterHub*-u može dodati više korisnika, svaki sa svojim podacima za prijavu, koji se mogu promijeniti od strane administratora. Ostatak sustava je identičan Jupyter Notebook-u, odnosno JupyterLab-u.

JupyterHub se može instalirati na više načina, a neki od njih su:

- instalacija na poslužitelj kao klasična aplikacija
- instalacija na *virtualni uređaj* pomoću softvera *The Littlest JupyterHub*
- instalacija korištenjem *Docker* sustava

(prema [8])

Docker je sustav napravljen za laku instalaciju, upravljanje i distribuciju softvera koji treba raditi jednostavne zadatke odvojeno od ostatka "glavnog" sustava. Na primjer, na jednom poslužitelju možemo imati pokrenut Docker *kontejner* na kojem se pokreće *JupyterHub* poslužitelj za jedan fakultet, u drugoj instanci (odnosno *kontejneru*) *JupyterHub* za drugi fakultet i sl. [18]

3. Jezgre u JupyterLab-u

Jezgre (*eng.* kernel) su "mozak" svake bilježnice, terminala ili konzole u JupyterLab-u. One pokreću sav kôd koji želimo izvršiti i šalju rezultate nazad da bi ih se moglo prikazati. JupyterLab omogućava da se bilo koja datoteka poveže s bilo kojom jezgrom. [19]

Kako je i prije spomenuto, trenutno **jedan dokument može biti povezan uz samo jednu jezgru**, a u budućnosti se planira omogućavanje korištenja više jezgri u jednom dokumentu.

3.1. Dostupne jezgre

Svaka instalacija JupyterLab-a dolazi uz Python 3 jezgru. Python 2 nije kompatibilan s JupyterLab-om niti Jupyter Notebook-om.

Popis svih dostupnih jezgri i poveznica za preuzimanje dostupan je na službenoj GitHub stranici Jupyter projekta (`https://github.com/jupyter/jupyter/wiki/Jupyter-kernels`), a neke od poznatijih su:

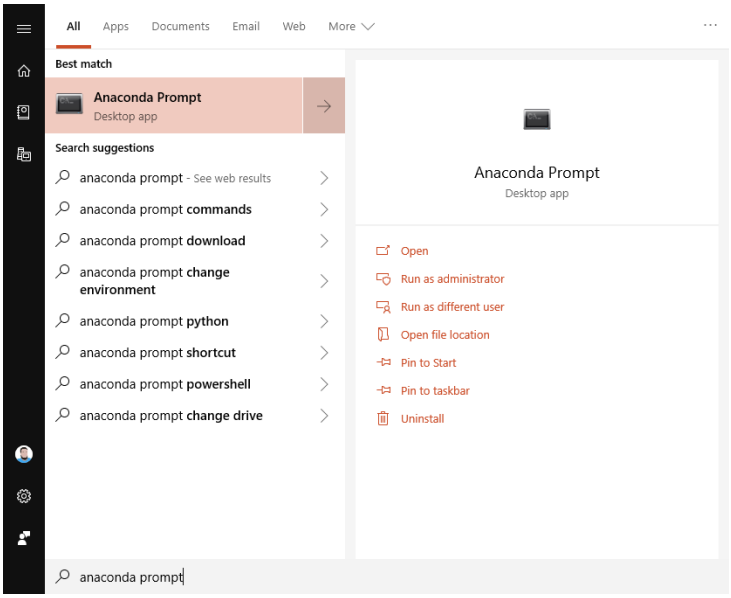
- Julia
- Ruby
- NodeJS
- C#
- F#
- C++
- R (instalirano uz Anaconda distribuciju)
- Go
- PHP
- MatLab
- Lua
- Wolfram Mathematica
- Kotlin
- TypeScript
- MicroPython (za programiranje *IoT* uređaja)
- mnoge druge...

3.2. Instalacija jezgre

Uz svaku jezgru dolaze upute za instalaciju (vidi prijašnje poglavlje). Ovdje će biti prikazana instalacija jezgre `IJavaScript` koja omogućuje podršku za programski jezik `JavaScript`. Često se procedure razlikuju za različite operacijske sustave i distribucije JupyterLab-a – ovdje će biti prikazana instalacija na Windows operacijskom sustavu uz Anaconda distribuciju.

Instalacija je jednostavna:

1. Otvorimo *Anaconda Prompt*



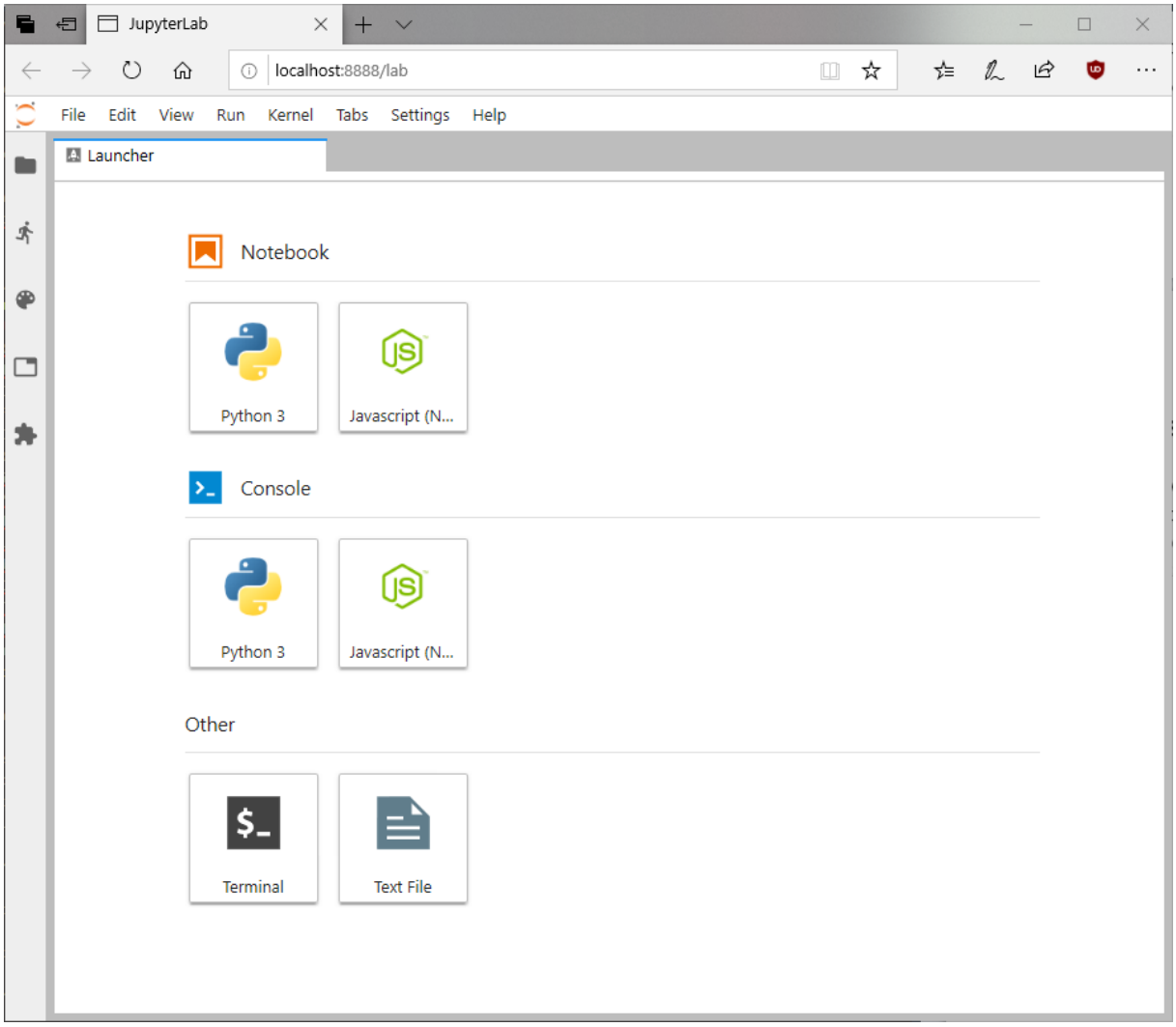
Slika 12: Otvaranje *Anaconda prompt* programa (vlastita izrada)

Slika 13: Prozor *Anaconda prompt*-a (vlastita izrada)

2. Pokrenemo naredbu `conda install nodejs`, pričekamo kraj instalacije, s `Y` potvrdimo pitanja

3. Pokrenemo naredbu `npm install -g ijavascript`
4. Pokrenemo naredbu `ijsinstall`
5. Ako imamo JupyterLab otvoren, osvježimo stranicu, ako ne pokrenemo JupyterLab

Nakon uspješne instalacije možemo vidjeti novu jezgru u *Launcher* prozoru (Slika 14).



Slika 14: Instalirana `JavaScript` jezgra (vlastita izrada)

4. Proširenja u JupyterLab-u

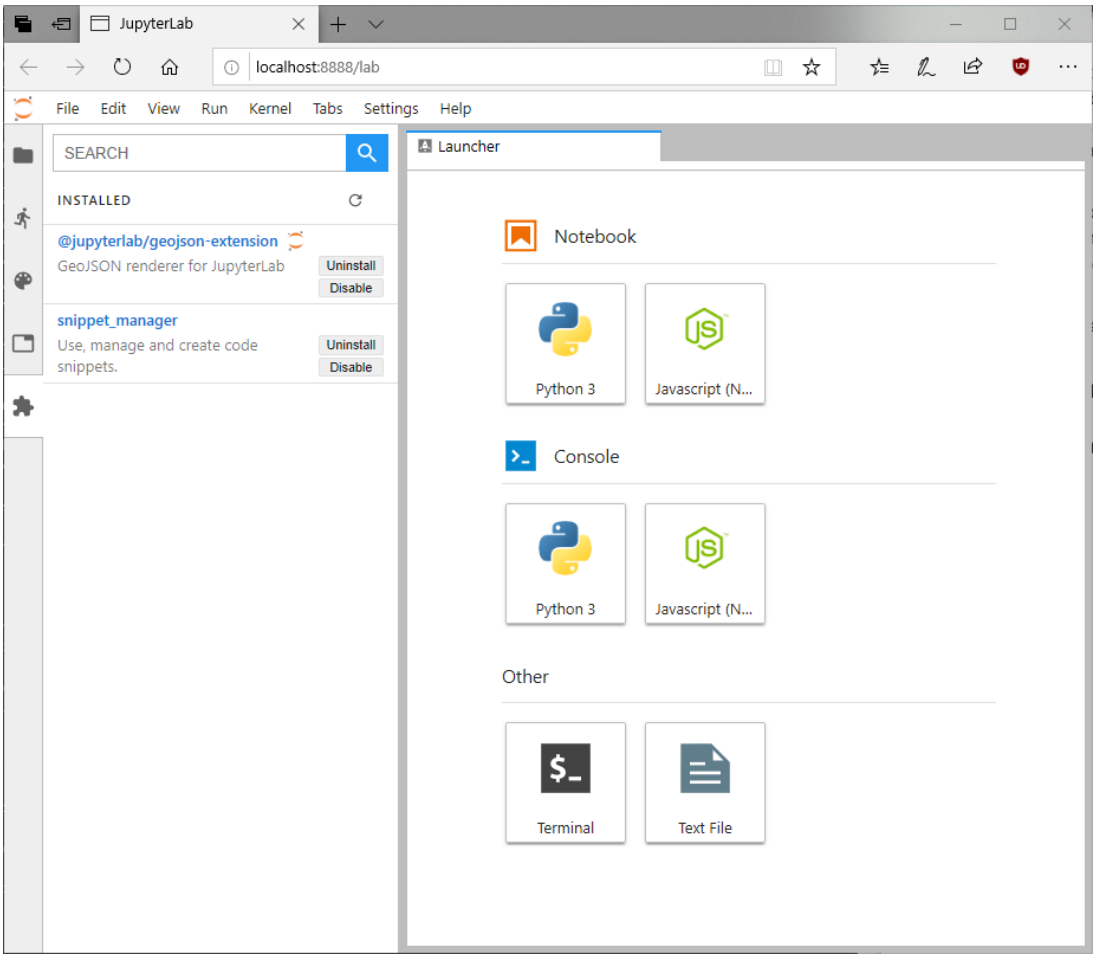
JupyterLab je stvoren da bude proširiv. Mogu se instalirati nove sheme boja, uređivači, preglednici datoteka, proširenja ćelija itd. Sâm JupyterLab je napravljen kao skup proširenja, pa su traka izbornika, bočna traka, prozori, uređivači, kontekstni izbornici i slični elementi svi implementirani kao zasebna proširenja. To olakšava ažuriranje, otkrivanje grešaka i distribuciju. [20]

Za komunikaciju između proširenja stvoren je sustav *poruka* (*eng.* messages). Sučelje se izvodi pomoću biblioteke *Phosphor.JS*, ali se za razvoj proširenja mogu koristiti i *čisti* HTML, CSS i JavaScript. [21]

Stiliziranje cijelog sučelja je lako jer je su za sve korištene *CSS varijable* i stilovi bazirani na principima *Google Material Design*-a, poboljšavajući konzistentnost. [21]

4.1. Instalacija proširenja

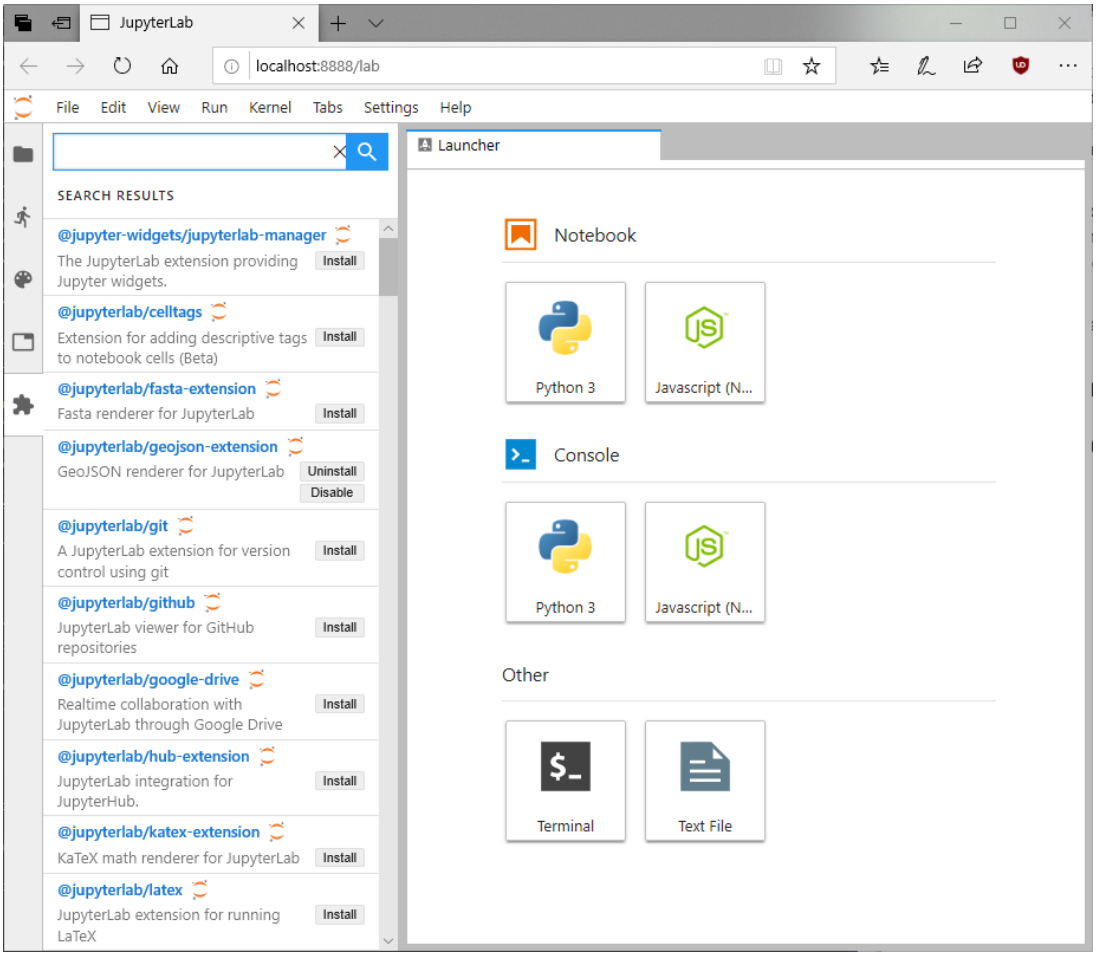
Proširenja je jednostavno instalirati. U lijevom oknu kliknemo na gumb *Extensions* (ikona *puzzle*), pa će se otvoriti popis trenutno instaliranih proširenja (Slika 15).



Slika 15: Popis instaliranih proširenja (vlastita izrada)

Ovdje možemo deinstalirati ili onemogućiti trenutno instalirana proširenja klikom na gumbe *Uninstall*, odnosno *Disable* uz proširenja.

Pri vrhu se nalazi traka za pretraživanje. Ako želimo vidjeti sva proširenja, kao pojam za pretragu upišemo razmak, a ako znamo ime ili dio imena proširenja upišemo ga. Pritisnemo `Enter` ili kliknemo na gumb za pretragu. Sada će se izlistati rezultati pretraživanja (Slika 16).



Slika 16: Prikaz rezultata pretrage (ovdje popis svih proširenja) (vlastita izrada)

Proširenje instaliramo klikom na gumb *Install*. Pričekamo par sekundi dok se preuzme i instalira. Neka proširenja zahtijevaju dodatne potvrde ili alate i prikazat će dijaloške okvire potvrde, koje treba potvrditi.

Proširenja se također mogu instalirati preko naredbenog retka.

4.2. Struktura proširenja

Kako su proširenja u JupyterLab-u bazirana na `npm` (*Node Package Manager*) paketima, sva proširenja imaju istu bazičnu strukturu (Slika 17).

Slika 17: Osnovna struktura mape proširenja (vlastita izrada)

Mapa `lib` sadrži automatski generirane datoteke i ne bi se smjela uređivati.

Mapa `node_modules` sadrži *NodeJS* module koji se koriste pri izradi proširenja. Sadržaj se ne smije ručno uređivati, već preko naredbi po potrebi dodavati ili uklanjati *pakete*.

Mapa `src` sadrži datoteku `index.ts` koja sadrži osnovnu logiku proširenja.

Mapa `styles` sadrži stilske datoteke vezane uz proširenje, automatski se generira datoteka `index.css` koja se može uređivati.

Datoteke `package.json` i `tsconfig.json` su automatski generirane i u pravilu se ne bi smjele uređivati.

U mapi se može nalaziti još datoteka koje može dodati razvojno okružje ili klijent za verzioniranje, ali su ovdje prikazane osnovne datoteke. Ostale datoteke neće smetati proširenju.

4.3. Izrada proširenja

JupyterLab je olakšao proces izrade proširenja. S par komandi se postavi mapa s potrebnim datotekama i pokrene instanca JupyterLab-a za testiranje proširenja. [22]

Za pisanje proširenja se koristi jezik *TypeScript*, koji je baziran na *JavaScript*u, ali je između ostaloga stroži pri određivanju i korištenju tipova podataka.

Osnovni koraci za izradu proširenja (u ovom primjeru sve naredbe pokrenuti u *Anaconda Prompt*-u):

1. Postaviti okružje za razvoj proširenja (`conda create -n jupyterlab-ext nodejs jupyterlab cookiecutter git -c conda-forge`)

2. Aktivirati okružje (`conda activate jupyterlab-ext`)
3. Inicijalizacija projekta (`cookiecutter https://github.com/jupyterlab/extension-cookiecutter-ts`)
4. Unos imena proširenja i autora (pratiti korake na zaslonu)
5. Instalacija upravitelja paketima (`jlpm install`)
6. Prvotna instalacija proširenja (`jupyter labextension install . --no-build`)
7. **U drugoj komandnoj liniji** pokrenuti instancu JupyterLab-a (`conda activate jupyterlab-ext`, zatim `jupyter lab -watch`)
8. **U prvoj komandnoj liniji** pokrenuti automatsku kompilaciju (`jlpm run watch`) koja će pri svakom spremanju izvršiti provjeru grešaka i kompilaciju. Osvježavanjem stranice u pregledniku ćemo odmah moći vidjeti promjene.

(prema [22])

Nakon ovih koraka u razvojnom okružju po želji možemo otvoriti mapu proširenja (mapa naziva kao što je uneseno u koraku **4.**, najčešće u *korisničkoj mapi*).

Sada smo spremni za izradu našeg proširenja.

4.4. Proširenje *Code snippet manager*

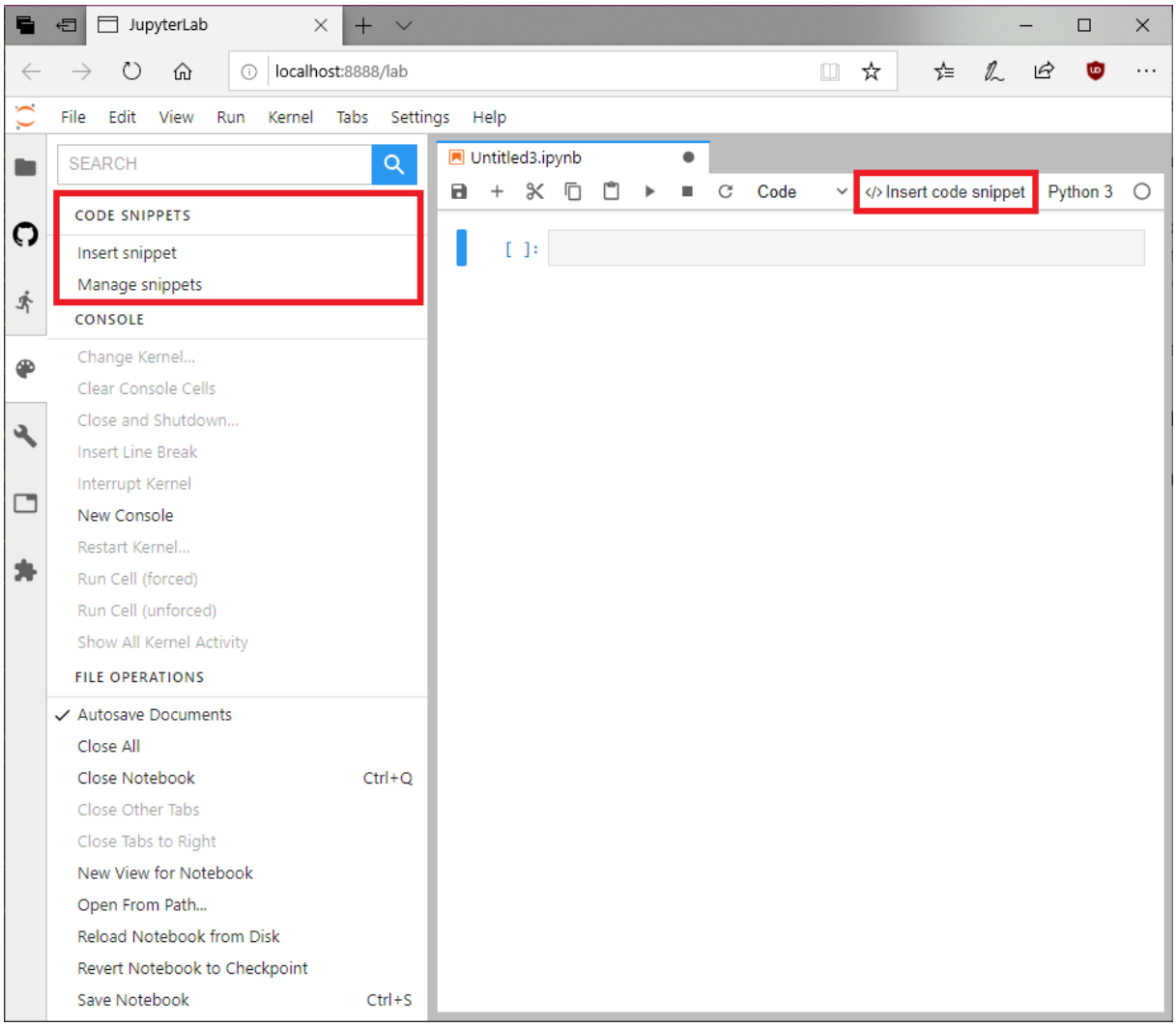
Kao primjer izrađeno je proširenje *Code snippet manager* koje korisniku omogućava kreiranje, spremanje i poslije umetanje isječaka kôda. Nekad se nađu dijelovi kôda koje trebamo u više projekata ili postoje proširenja koja zahtijevaju određenu inicijalizaciju koja nije lako pamtiva. Svi isječci se spremaju u `JSON` datoteku, pa se proširenja mogu lako prenositi između računala.

Proširenje ima dva glavna sučelja:

- Sučelje za umetanje isječaka
- Sučelje za upravljanje isječcima (dodavanje, uređivanje, brisanje)

Nakon instalacije proširenja u alatnu traku svake bilježnice dodaje se gumb *Insert code snippet* koji služi za umetanje isječka kôda. Također, u paletu naredbi (*eng.* Command palette) s lijeve strane dodaju se mogućnosti *Insert snippet* (ista funkcionalnost kao i gumb u alatnoj traci) i *Manage code snippets* (otvara sučelje za upravljanje isječcima kôda) (vidi Sliku 18).

Izvorni kôd proširenja vidljiv je u Prilogu 1, a CSS kôd u Prilogu 2.



Slika 18: Gumbi koje dodaje proširenje (vlastita izrada)

4.4.1. Umetanje isječka kôda

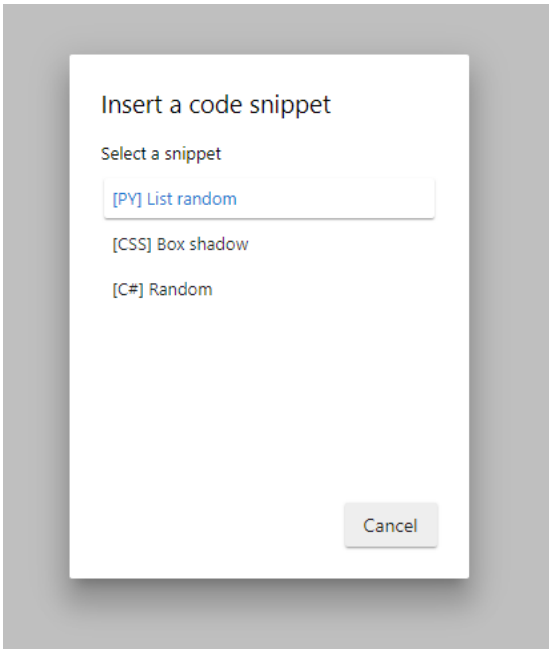
Klikom na gumb *Insert code snippet* na alatnoj traci ili gumba *Insert snippet* u paleti naredbi otvori se dijaloški okvir kao na slici 19.

Nakon klika na naziv isječka u trenutnu ćeliju na poziciju kursora se dodaje odabrani isječak. Klikom na gumb *Cancel* dijaloški okvir se zatvara.

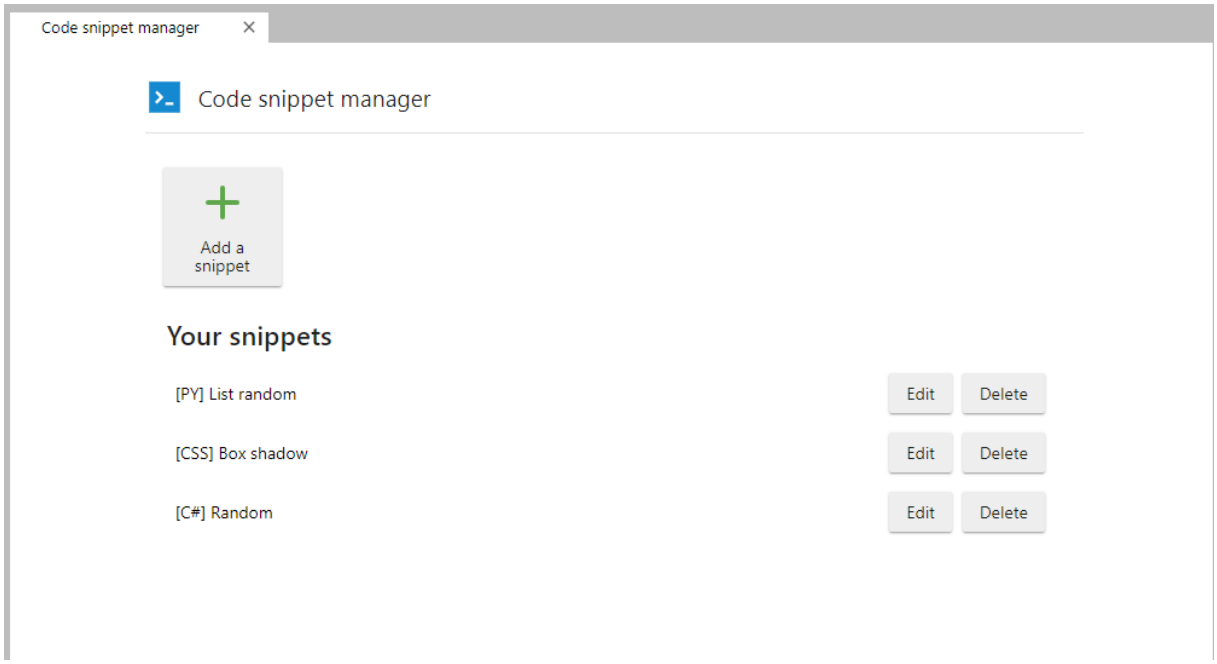
4.4.2. Upravljanje isječcima kôda

Klikom na gumb *Manage code snippets* otvara se nova kartica *Code snippet manager* (Slika 20).

Klikom na gumb *Add a snippet* otvara se dijaloški okvir *New code snippet* (vidi Sliku 21) za dodavanje novog isječka kôda. Potrebno je unijeti naziv i sâm isječak. Klikom na gumb *Add* isječak se zapisuje u datoteku. Klikom na gumb *Cancel* dijaloški okvir se zatvara.

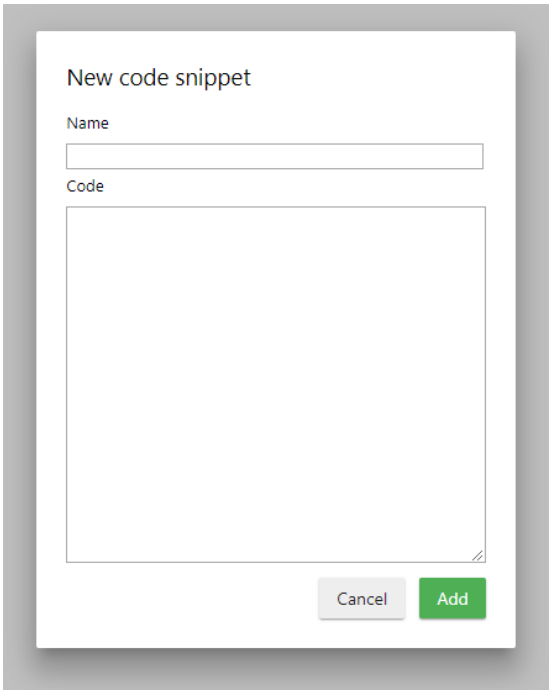


Slika 19: Dijaloški okvir za odabir isječka za umetanje (vlastita izrada)



Slika 20: Kartica *Code snippet manager* (vlastita izrada)

Uz svaki isječak na popisu *Your snippets* pojavljuju se gumbi *Delete* (brisanje isječka) i *Edit* za uređivanje trenutnog isječka. Dijaloški okvir za uređivanje je sličan onom za dodavanje.



Slika 21: Dijaloški okvir za dodavanje novog isječka kôda (vlastita izrada)

5. Zaključak

Kao što je Python uveo mnoge u svijet programiranja, Project Jupyter je revolucionirao i olakšao ulaz u područja koja su danas iznimno bitna poput *data science*-a, strojnog učenja i neuronskih mreža.

Od početaka kao IPython, preko IPython Notebooka i Jupyter Notebooka, JupyterLab je danas moderno, brzo, fleksibilno i lako proširivo razvojno okružje napisano prema najnovijim standardima web programiranja.

Sustav jezgri omogućava proširivanje osnovnog sustava (koji podržava Python 3) mogućnostima izvršavanja kôda mnogih drugih programskih jezika, poput C/C++-a, C#-a, R-a i mnogih drugih.

Sustavom proširenja se rad, izgled i funkcionalnost JupyterLab-a može znatno poboljšati, čineći ga jednim od najpraktičnijih razvojnih okružja danas. Jednostavnom strukturom proširenja i relativno jednostavnom procesu razvoja proširenja očekujem da će se broj proširenja do izlaska verzije 1.0 znatno povećati.

Kao primjer izrađeno je proširenje *Code snippet manager* koje omogućava izradu, umetanje i upravljanje isječcima kôda, što je vrlo korisna značajka u današnjim, sve kompleksnijim strukturama programa.

Popis literature

[1] (2018). Python Beginner's guide, Python Software Foundation, adresa: https://wiki.python.org/moin/BeginnersGuide/Overview (pogledano 14. 7. 2018).

[2] (2018). General Python FAQ, Python Software Foundation, adresa: https://docs.python.org/3/faq/general.html#why-was-python-created-in-the-first-place (pogledano 14. 7. 2018).

[3] F. Perez. (8. siječnja 2012). The IPython notebook: a historical retrospective, adresa: http://blog.fperez.org/2012/01/ipython-notebook-historical.html (pogledano 15. 7. 2018).

[4] Mbussonn, *IPython terminal*, 11. travnja 2018. adresa: https://upload.wikimedia.org/wikipedia/commons/thumb/d/de/IPython-6.x-screenshot-osx.png/640px-IPython-6.x-screenshot-osx.png (pogledano 1. 8. 2018).

[5] (). JSON - about, adresa: https://www.json.org/ (pogledano 1. 8. 2018).

[6] (2018). Project Jupyter - About, Project Jupyter, adresa: http://jupyter.org/about (pogledano 14. 7. 2018).

[7] (2018). Project Jupyter, Wikipedia, adresa: https://en.wikipedia.org/wiki/Project_Jupyter (pogledano 16. 7. 2018).

[8] (2018). Jupyter Hub, Project Jupyter, adresa: http://jupyter.org/hub (pogledano 30. 8. 2018).

[9] (20. veljače 2018). Jupyter Blog, Project Jupyter, adresa: https://blog.jupyter.org/jupyterlab-is-ready-for-users-5a6f039b8906 (pogledano 4. 8. 2018).

[10] (2018). JupyterLab - Overview, Project Jupyter, adresa: https://jupyterlab.readthedocs.io/en/stable/getting_started/overview.html (pogledano 4. 8. 2018).

[11] (2018). JupyterLab - Installation, Project Jupyter, adresa: https://jupyterlab.readthedocs.io/en/stable/getting_started/installation.html (pogledano 5. 8. 2018).

[12] (2018). About Anaconda Cloud, Anaconda, adresa: https://anaconda.org/about (pogledano 6. 8. 2018).

[13] (2018). Code Consoles, Project Jupyter, adresa: https://jupyterlab.readthedocs.io/en/stable/user/code_console.html (pogledano 6. 8. 2018).

[14] (2018). Terminals, Project Jupyter, adresa: https://jupyterlab.readthedocs.io/en/stable/user/terminal.html (pogledano 6. 8. 2018).

[15] (2018). Text Editor, Project Jupyter, adresa: https://jupyterlab.readthedocs.io/en/stable/user/file_editor.html (pogledano 6. 8. 2018).

[16] (2018). Text Editor, Project Jupyter, adresa: https://jupyterlab.readthedocs.io/en/stable/user/notebook.html (pogledano 6. 8. 2018).

[17] D. Toomey, *Learning Jupyter*. Packt Publishing, 2016.

[18] Wikipedia. (2018). Docker (software), adresa: https://en.wikipedia.org/wiki/Docker_(software) (pogledano 30. 8. 2018).

[19] (2018). Documents and Kernels, Project Jupyter, adresa: https://jupyterlab.readthedocs.io/en/stable/user/documents_kernels.html (pogledano 7. 8. 2018).

[20] (2018). Extensions, Project Jupyter, adresa: https://jupyterlab.readthedocs.io/en/stable/user/extensions.html (pogledano 12. 8. 2018).

[21] (2018). Extension Developer Guide, Project Jupyter, adresa: https://jupyterlab.readthedocs.io/en/stable/developer/extension_dev.html (pogledano 14. 8. 2018).

[22] (2018). Let's Make an xkcd JupyterLab Extension, Project Jupyter, adresa: https://jupyterlab.readthedocs.io/en/stable/developer/xkcd_extension_tutorial.html (pogledano 18. 8. 2018).

Popis slika

1. Sučelje IPythona u naredbenom retku (Izvor: [4]) 2
2. Sučelje *Anaconda Navigator*a u operacijskom sustavu Windows 10 (vlastita izrada) 5
3. *Launcher* prozor JupyterLab-a (vlastita izrada) 6
4. Izbornik `File > New` (vlastita izrada) 6
5. Prozor za odabir *jezgre* (vlastita izrada) 7
6. Prozor za odabir *jezgre* (vlastita izrada) 7
7. Prozor *Terminal* (vlastita izrada) 8
8. Prozor uređivača teksta s otvorenom datotekom (vlastita izrada) 9
9. Postavljanje teme uređivača teksta (vlastita izrada) 9
10. Sučelje bilježnice (vlastita izrada) 11
11. Mogućnosti za oblikovanje ćelije kao slajda (vlastita izrada) 11
12. Otvaranje *Anaconda prompt* programa (vlastita izrada) 14
13. Prozor *Anaconda prompt*-a (vlastita izrada) 14
14. Instalirana `JavaScript` jezgra (vlastita izrada) 15
15. Popis instaliranih proširenja (vlastita izrada) 16
16. Prikaz rezultata pretrage (ovdje popis svih proširenja) (vlastita izrada) 17
17. Osnovna struktura mape proširenja (vlastita izrada) 18
18. Gumbi koje dodaje proširenje (vlastita izrada) 20
19. Dijaloški okvir za odabir isječka za umetanje (vlastita izrada) 21
20. Kartica *Code snippet manager* (vlastita izrada) 21
21. Dijaloški okvir za dodavanje novog isječka kôda (vlastita izrada) 22

Prilog 1: Izvorni kôd proširenja - *TypeScript*

```
import 'es6-promise';

import {
  IDisposable, DisposableDelegate
} from '@phosphor/disposable';

import {
  JupyterLab, JupyterLabPlugin
} from '@jupyterlab/application';

import {
  ToolbarButton, ICommandPalette
} from '@jupyterlab/apputils';

import {
  DocumentRegistry
} from '@jupyterlab/docregistry';

import {
  INotebookModel,
  NotebookPanel,
} from '@jupyterlab/notebook';

import {
  Widget
} from '@phosphor/widgets';

import {
  Kernel
} from '@jupyterlab/services';

import '../style/index.css';

// http://stackoverflow.com/questions/26501688/a-typescript-guid-class
class Guid {
  static newGuid() {
    return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function (c) {
      const r = Math.random() * 16 | 0, v = c === 'x' ? r : (r & 0x3 | 0x8);
      return v.toString(16);
    });
  }
}

// Snippet class
class Snippet {
```

```
  name: string;
  code: string;
  id: string;
}

var currentNotebook: NotebookPanel;

// Toolbar button extension
export
  class ButtonExtension implements DocumentRegistry.IWidgetExtension<NotebookPanel,
   ↪  INotebookModel> {

  createNew(panel: NotebookPanel, context:
   ↪  DocumentRegistry.IContext<INotebookModel>): IDisposable {
    currentNotebook = panel;
    let button = new ToolbarButton({
      className: 'smgr-button',
      iconClassName: 'fa fa-code',
      onClick: insertCodeSnippet,
      tooltip: 'Insert a code snippet',
      label: 'Insert code snippet'
    });

    panel.toolbar.insertItem(9, 'insrtSnpt', button);
    return new DisposableDelegate(() => {
      button.dispose();
    });
  }
}

// Plugin functions

function insertCodeSnippet() {

  if (snippetList.length > 0) {
    showInsertSnippetDialog();
  } else {
    // Load snippets
    const code = [
      'filePath = "C:\\snippets.json"',
      'f = open(filePath, "a")',
      'f.close()',
      'with open(filePath, "r") as fp:',
      ' lines = fp.read()',
      ' print(lines)'
    ].join('\n');

    Kernel.startNew().then(kernel => {

      let future = kernel.requestExecute({ code });

      // Load data into memory
      future.onIOPub = msg => {
```

```
        if (msg.content.name == 'stdout') {
          for (var item of JSON.parse(msg.content.text.toString())) {
            if (new String(item.name).length > 0) {
              let newSnipp = new Snippet;
              newSnipp.name = item.name;
              newSnipp.code = item.code;
              newSnipp.id = item.id;

              snippetList.push(newSnipp);
            }
          }
        }
      };

      // Do stuff with data
      future.done.then(() => {
        showInsertSnippetDialog();

        kernel.shutdown();
      });
    });
  }

}

function showInsertSnippetDialog() {
  let dialog = document.createElement('div');
  dialog.classList.add("p-Widget");
  dialog.classList.add("jp-Dialog");
  dialog.id = "addnew-dialog";

  document.getElementsByClassName("jp-MainAreaWidget")[0].appendChild(dialog);

  //
  let dialogContent = document.createElement('div');
  dialogContent.classList.add("p-Widget");
  dialogContent.classList.add("p-Panel");
  dialogContent.classList.add("jp-Dialog-content");

  dialog.appendChild(dialogContent);

  // Dialog header
  let dialogHeader = document.createElement('span');
  dialogHeader.classList.add("p-Widget");
  dialogHeader.classList.add("jp-Dialog-header");
  dialogHeader.innerText = "Insert a code snippet";

  dialogContent.appendChild(dialogHeader);

  // Dialog body
  let dialogBody = document.createElement('div');
  dialogBody.classList.add("p-Widget");
  dialogBody.classList.add("jp-Dialog-body");
```

```
dialogContent.appendChild(dialogBody);

//
let newSnippetNameLabel = document.createElement('label');
newSnippetNameLabel.innerText = "Select a snippet";

dialogBody.appendChild(newSnippetNameLabel);

let snippetContainer = document.createElement('div');
snippetContainer.className = "snippet-container";

dialogBody.appendChild(snippetContainer);

for (var snippet of snippetList) {
  let snippetItem = document.createElement("button");
  snippetItem.innerText = snippet.name;
  snippetItem.className = "snippet-list-item";
  snippetItem.id = snippet.id;
  snippetItem.addEventListener('click', () => {
    let activeEditor = currentNotebook.content.activeCell.inputArea.editor;
    let snippetCode = (snippetList.find(x => x.id == snippetItem.id)).code;


    ↪  activeEditor.model.value.insert(activeEditor.getOffsetAt(activeEditor.getCursorPositior
    ↪  snippetCode);


    ↪  document.getElementsByClassName("jp-MainAreaWidget")[0].removeChild(document.getElement
  });
  snippetContainer.appendChild(snippetItem);
}

// Dialog footer
let dialogFooter = document.createElement('div');
dialogFooter.classList.add("p-Widget");
dialogFooter.classList.add("jp-Dialog-footer");

dialogContent.appendChild(dialogFooter);

//
let cancelBtn = document.createElement('button');
cancelBtn.classList.add("c-btn");
cancelBtn.classList.add("c-btn-fill");
cancelBtn.innerText = "Cancel";
cancelBtn.setAttribute('style', 'margin: auto 0 0 0;')
cancelBtn.addEventListener('click', () => {

   ↪  document.getElementsByClassName("jp-MainAreaWidget")[0].removeChild(document.getElementBy
});

dialogFooter.appendChild(cancelBtn);
```

```
}


var snippetList: any[] = new Array<any>();
var widget: Widget;

function manageCodeSnippets() {
  snippetList.length = 0;

  if (!widget) {
    widget = new Widget();
    widget.id = 'smgr-manage';
    widget.title.label = 'Code snippet manager';
    widget.title.closable = true;
    snippetList.length = 0;
  }

  // Outer div
  let outerBody = document.createElement('div');
  outerBody.className = "jp-Launcher-body";
  outerBody.id = "widget-body";

  widget.node.appendChild(outerBody);

  // Main container
  let contentWrapper = document.createElement('div');
  contentWrapper.className = "jp-Launcher-content";
  outerBody.appendChild(contentWrapper);

  // Header
  let sectionHeader = document.createElement('div');
  sectionHeader.className = "jp-Launcher-sectionHeader";

  contentWrapper.appendChild(sectionHeader);

  let sectionHeaderIcon = document.createElement('div');
  sectionHeaderIcon.classList.add("jp-CodeConsoleIcon");
  sectionHeaderIcon.classList.add("jp-Launcher-sectionIcon");
  sectionHeaderIcon.classList.add("jp-Launcher-icon");

  sectionHeader.appendChild(sectionHeaderIcon);

  let sectionHeaderTitle = document.createElement('h2');
  sectionHeaderTitle.className = "jp-Launcher-sectionTitle";
  sectionHeaderTitle.innerText = "Code snippet manager";

  sectionHeader.appendChild(sectionHeaderTitle);

  // Content

  let snippetMgrContainer = document.createElement('div');
  snippetMgrContainer.className = "snippet-manager";
```

```
contentWrapper.appendChild(snippetMgrContainer);

let addNewSnippetBtn = document.createElement('button');
addNewSnippetBtn.classList.add("c-btn");
addNewSnippetBtn.classList.add("c-btn-fill");
addNewSnippetBtn.classList.add("c-btn-square");
addNewSnippetBtn.classList.add("c-btn-icon-text");

addNewSnippetBtn.addEventListener('click', () => {

  let dialog = document.createElement('div');
  dialog.classList.add("p-Widget");
  dialog.classList.add("jp-Dialog");
  dialog.id = "addnew-dialog";

  widget.node.appendChild(dialog);

  //
  let dialogContent = document.createElement('div');
  dialogContent.classList.add("p-Widget");
  dialogContent.classList.add("p-Panel");
  dialogContent.classList.add("jp-Dialog-content");

  dialog.appendChild(dialogContent);

  // Dialog header
  let dialogHeader = document.createElement('span');
  dialogHeader.classList.add("p-Widget");
  dialogHeader.classList.add("jp-Dialog-header");
  dialogHeader.innerText = "New code snippet";

  dialogContent.appendChild(dialogHeader);

  // Dialog body
  let dialogBody = document.createElement('div');
  dialogBody.classList.add("p-Widget");
  dialogBody.classList.add("jp-Dialog-body");

  dialogContent.appendChild(dialogBody);

  //
  let newSnippetNameLabel = document.createElement('label');
  newSnippetNameLabel.innerText = "Name";

  dialogBody.appendChild(newSnippetNameLabel);

  //
  let newSnippetName = document.createElement('input');
  newSnippetName.id = "newSnippetName";
  newSnippetName.setAttribute('type', 'text');
  newSnippetName.setAttribute('style', 'width: 25em');

  dialogBody.appendChild(newSnippetName);
```

```
//
let newSnippetContentsLabel = document.createElement('label');
newSnippetContentsLabel.innerText = "Code";

dialogBody.appendChild(newSnippetContentsLabel);

//
let newSnippetContents = document.createElement('textarea');
newSnippetContents.id = "newSnippetCode";
newSnippetContents.setAttribute('rows', '20');
newSnippetContents.setAttribute('style', 'width: 25em');

dialogBody.appendChild(newSnippetContents);

// Dialog footer
let dialogFooter = document.createElement('div');
dialogFooter.classList.add("p-Widget");
dialogFooter.classList.add("jp-Dialog-footer");

dialogContent.appendChild(dialogFooter);

//
let cancelBtn = document.createElement('button');
cancelBtn.classList.add("c-btn");
cancelBtn.classList.add("c-btn-fill");
cancelBtn.innerText = "Cancel";
cancelBtn.setAttribute('style', 'margin: auto 12px 0 0;')
cancelBtn.addEventListener('click', () => {
  widget.node.removeChild(document.getElementById("addnew-dialog"));
});

dialogFooter.appendChild(cancelBtn);

//
let addBtn = document.createElement('button');
addBtn.classList.add("c-btn");
addBtn.classList.add("c-btn-fill-success");
addBtn.innerText = "Add";
addBtn.setAttribute('style', 'margin: auto 0 0 0;');
addBtn.addEventListener('click', () => {
  let name =
   ↪  (<HTMLInputElement>document.getElementById('newSnippetName')).value;
  let snippetCode =
   ↪  (<HTMLInputElement>document.getElementById('newSnippetCode')).value;
  let id = Guid.newGuid();

  if (name.length < 1) {
    alert("Name is required");
    return;
  }

  if (snippetCode.length < 1) {
```

```
      alert("Code is required");
      return;
    }

    let newSnippet = new Snippet;
    newSnippet.name = name;
    newSnippet.code = snippetCode;
    newSnippet.id = id;

    snippetList.push(newSnippet);

    let outJson = JSON.stringify(snippetList)

    // Save snippets
    const code = [
      'import json',
      'with open("C:\\snippets.json", "w+") as outfile:',
      ' resJson = ' + outJson,
      ' json.dump(resJson, outfile, indent=4)'
    ].join('\n');

    Kernel.startNew().then(kernel => {
      let future = kernel.requestExecute({ code });

      future.done.then(() => {
        kernel.shutdown();

        widget.node.removeChild(document.getElementById("addnew-dialog"));
      });
    });

    widget.node.removeChild(document.getElementById("widget-body"));

    setTimeout(() => {

      manageCodeSnippets();
    }, 1000);

  });


  dialogFooter.appendChild(addBtn);
});

let addNewSnippetBtnDiv = document.createElement('div');

addNewSnippetBtn.appendChild(addNewSnippetBtnDiv);

let letaddNewSnippetBtnIcon = document.createElement('span');
letaddNewSnippetBtnIcon.className = "btn-icon";
letaddNewSnippetBtnIcon.innerHTML =
  `
```

```
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" x="0px" y="0px"
↪  width="24px" height="24px"
   xmlns:xml="http://www.w3.org/XML/1998/namespace" xml:space="preserve"
↪  enable-background="new 0 0 24 24"
   xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1" id="add-snippet-icon">
   <g id="Bounding_Boxes">
     <g id="ui_x5F_spec_x5F_header_copy_3" display="none">
     </g>
     <path fill="none" d="M 0 0 h 24 v 24 H 0 V 0 Z" />
   </g>
   <g id="Rounded_1_">
     <g id="ui_x5F_spec_x5F_header_copy_6" display="none">
     </g>
     <path d="M 18 13 h -5 v 5 c 0 0.55 -0.45 1 -1 1 h 0 c -0.55 0 -1 -0.45 -1 -1 v
↪  -5 H 6 c -0.55 0 -1 -0.45 -1 -1 v 0 c 0 -0.55 0.45 -1 1 -1 h 5 V 6 c 0 -0.55
↪  0.45 -1 1 -1 h 0 c 0.55 0 1 0.45 1 1 v 5 h 5 c 0.55 0 1 0.45 1 1 v 0 C 19 12.55
↪  18.55 13 18 13 Z" />
   </g>
 </svg>
 `;

 addNewSnippetBtnDiv.appendChild(letaddNewSnippetBtnIcon);

 let letaddNewSnippetBtnLabel = document.createElement('span');
 letaddNewSnippetBtnLabel.className = "btn-label";
 letaddNewSnippetBtnLabel.innerText = "Add a snippet";

 addNewSnippetBtnDiv.appendChild(letaddNewSnippetBtnLabel);

 snippetMgrContainer.appendChild(addNewSnippetBtn);

 let tableOuter = document.createElement('table');

 snippetMgrContainer.appendChild(tableOuter);

 let tableHead = document.createElement('thead');

 tableOuter.appendChild(tableHead);

 let tableHeadRow = document.createElement('tr');

 tableHead.appendChild(tableHeadRow);

 let tableHeadRowTh = document.createElement('th');
 tableHeadRowTh.setAttribute('colspan', '3');

 tableHeadRow.appendChild(tableHeadRowTh);

 let tableHeadRowH3 = document.createElement('h3');
 tableHeadRowH3.innerText = "Your snippets";

 tableHeadRowTh.appendChild(tableHeadRowH3);
```

```
  let tableBody = document.createElement('tbody');
  tableBody.id = "snippet-list-big";

  tableOuter.appendChild(tableBody);

  // Load snippets
  const code = [
    'filePath = "C:\\snippets.json"',
    'f = open(filePath, "a")',
    'f.close()',
    'with open(filePath, "r") as fp:',
    ' lines = fp.read()',
    ' print(lines)'
  ].join('\n');

  Kernel.startNew().then(kernel => {

    let future = kernel.requestExecute({ code });

    // Load data into memory
    future.onIOPub = msg => {
      if (msg.content.name == 'stdout') {
        for (var item of JSON.parse(msg.content.text.toString())) {
          if (new String(item.name).length > 0) {
            let newSnipp = new Snippet;
            newSnipp.name = item.name;
            newSnipp.code = item.code;
            newSnipp.id = item.id;

            snippetList.push(newSnipp);
          }
        }
      }
    };

    // Do stuff with data
    future.done.then(() => {
      showAllSnippets();

      kernel.shutdown();
    });
  });

  //

  if (!widget.isAttached) {
    // Attach the widget to the main work area if it's not there
    mainApp.shell.addToMainArea(widget);
  }

  // Activate the widget
  mainApp.shell.activateById(widget.id);
}
```

```
function showAllSnippets() {

  if (snippetList.length > 0) {

    document.getElementById('snippet-list-big').innerHTML = "";

    for (var item of snippetList) {

      let row = document.createElement('tr');

      //
      let name = document.createElement('td');
      name.innerText = item.name;

      row.appendChild(name);

      //
      let editTd = document.createElement('td');
      editTd.id = "item-name";

      let editBtn = document.createElement('button');
      editBtn.classList.add("c-btn");
      editBtn.classList.add("c-btn-fill");
      editBtn.id = item.id;
      editBtn.innerText = "Edit";
      editBtn.addEventListener('click', () => {

        let found = snippetList.find(x => x.id == editBtn.id);
        let foundIndex = snippetList.indexOf(found);

        let dialog = document.createElement('div');
        dialog.classList.add("p-Widget");
        dialog.classList.add("jp-Dialog");
        dialog.id = "addnew-dialog";

        widget.node.appendChild(dialog);

        //
        let dialogContent = document.createElement('div');
        dialogContent.classList.add("p-Widget");
        dialogContent.classList.add("p-Panel");
        dialogContent.classList.add("jp-Dialog-content");

        dialog.appendChild(dialogContent);

        // Dialog header
        let dialogHeader = document.createElement('span');
        dialogHeader.classList.add("p-Widget");
        dialogHeader.classList.add("jp-Dialog-header");
        dialogHeader.innerText = "Edit code snippet";

        dialogContent.appendChild(dialogHeader);
```

```
// Dialog body
let dialogBody = document.createElement('div');
dialogBody.classList.add("p-Widget");
dialogBody.classList.add("jp-Dialog-body");

dialogContent.appendChild(dialogBody);

//
let newSnippetNameLabel = document.createElement('label');
newSnippetNameLabel.innerText = "Name";

dialogBody.appendChild(newSnippetNameLabel);

//
let newSnippetName = document.createElement('input');
newSnippetName.id = "newSnippetName";
newSnippetName.setAttribute('type', 'text');
newSnippetName.setAttribute('style', 'width: 25em');
newSnippetName.value = found.name;

dialogBody.appendChild(newSnippetName);

//
let newSnippetContentsLabel = document.createElement('label');
newSnippetContentsLabel.innerText = "Code";

dialogBody.appendChild(newSnippetContentsLabel);

//
let newSnippetContents = document.createElement('textarea');
newSnippetContents.id = "newSnippetCode";
newSnippetContents.setAttribute('rows', '20');
newSnippetContents.setAttribute('style', 'width: 25em');
newSnippetContents.value = found.code;

dialogBody.appendChild(newSnippetContents);

// Dialog footer
let dialogFooter = document.createElement('div');
dialogFooter.classList.add("p-Widget");
dialogFooter.classList.add("jp-Dialog-footer");

dialogContent.appendChild(dialogFooter);

//
let cancelBtn = document.createElement('button');
cancelBtn.classList.add("c-btn");
cancelBtn.classList.add("c-btn-fill");
cancelBtn.innerText = "Cancel";
cancelBtn.setAttribute('style', 'margin: auto 12px 0 0;')
cancelBtn.addEventListener('click', () => {
  widget.node.removeChild(document.getElementById("addnew-dialog"));
```

```
});

dialogFooter.appendChild(cancelBtn);

//
let addBtn = document.createElement('button');
addBtn.classList.add("c-btn");
addBtn.classList.add("c-btn-fill-info");
addBtn.innerText = "Update";
addBtn.setAttribute('style', 'margin: auto 0 0 0;');
addBtn.addEventListener('click', () => {
  let name =
   ↪  (<HTMLInputElement>document.getElementById('newSnippetName')).value;
  let snippetCode =
   ↪  (<HTMLInputElement>document.getElementById('newSnippetCode')).value;

  if (name.length < 1) {
    alert("Name is required");
    return;
  }

  if (snippetCode.length < 1) {
    alert("Code is required");
    return;
  }

  snippetList[foundIndex].name = name;
  snippetList[foundIndex].code = snippetCode;

  let outJson = JSON.stringify(snippetList)

  // Save snippets
  const code = [
    'import json',
    'with open("C:\\snippets.json", "w+") as outfile:',
    ' resJson = ' + outJson,
    ' json.dump(resJson, outfile, indent=4)'
  ].join('\n');

  Kernel.startNew().then(kernel => {
    let future = kernel.requestExecute({ code });

    future.done.then(() => {
      kernel.shutdown();

      widget.node.removeChild(document.getElementById("addnew-dialog"));
    });
  });

  widget.node.removeChild(document.getElementById("widget-body"));

  setTimeout(() => {
```

```
            manageCodeSnippets();
          }, 1000);

        });


        dialogFooter.appendChild(addBtn);
      });

      editTd.appendChild(editBtn);

      row.appendChild(editTd);

      //
      let deleteTd = document.createElement('td');
      deleteTd.id = "item-name";

      let deleteBtn = document.createElement('button');
      deleteBtn.classList.add("c-btn");
      deleteBtn.classList.add("c-btn-fill");
      deleteBtn.innerText = "Delete";
      deleteBtn.id = item.id;
      deleteBtn.addEventListener('click', () => { deleteSnippet(deleteBtn.id); });

      deleteTd.appendChild(deleteBtn);

      row.appendChild(deleteTd);

      document.getElementById('snippet-list-big').appendChild(row);
    }

  }
  else {
    alert("Empty!");
  }
}

function deleteSnippet(id: string) {
  let found = snippetList.find(x => x.id == id);
  let foundIndex = snippetList.indexOf(found);

  snippetList.splice(foundIndex, 1);

  setTimeout(() => {
    saveJsonAndReload();
  }, 500);
}

function saveJsonAndReload() {
  let outJson = JSON.stringify(snippetList)

  const code = [
    'import json',
```

```
    'with open("C:\\snippets.json", "w+") as outfile:',
    ' resJson = ' + outJson,
    ' json.dump(resJson, outfile, indent=4)'
  ].join('\n');

  Kernel.startNew().then(kernel => {
    let future = kernel.requestExecute({ code });

    future.done.then(() => {
      kernel.shutdown();
    });
  });

  widget.node.removeChild(document.getElementById("widget-body"));

  setTimeout(() => {
    manageCodeSnippets();
  }, 1000);
}

function addPalleteItem(app: JupyterLab, palette: ICommandPalette, label: string,
 ↪  func: () => void, cmnd: string, categoryName: string = 'Code snippets'): void {
  const command: string = cmnd;
  app.commands.addCommand(command, {
    label: label,
    execute: func
  });

  palette.addItem({ command, category: categoryName });
}

// Plugin activation
var mainApp: JupyterLab;

function activate(app: JupyterLab, palette: ICommandPalette): void {
  console.log("Code snippets plugin activated");

  // Add toolbar button
  app.docRegistry.addWidgetExtension('Notebook', new ButtonExtension());

  // Add commands
  addPalleteItem(app, palette, 'Insert snippet', insertCodeSnippet, 'smgr:insert');

  mainApp = app;
  addPalleteItem(app, palette, 'Manage snippets', manageCodeSnippets,
   ↪  'smgr:manage');
}

// Extension init
const extension: JupyterLabPlugin<void> = {
  id: 'snippet_manager',
  autoStart: true,
  activate: activate,
```

```
  requires: [ICommandPalette]
};

export default extension;
```

Prilog 2: Izvorni kôd proširenja - *CSS*

```css
.snippet-manager {
    padding: 20px;
    margin: 0;
    display: flex;
    flex-direction: column;
}

.c-btn {
    background: none;
    border: 0;
    margin: 0 auto 0 0;
    padding: 0.5em 1em;
    box-shadow: var(--jp-elevation-z1);
    font-family: var(--jp-ui-font-family);
    font-size: var(--jp-content-font-size1);
    transition: 200ms all cubic-bezier(0.4, 0.0, 0.2, 1);
    cursor: pointer;
}

.c-btn:hover {
    box-shadow: var(--jp-elevation-z4);
}

.c-btn-fill {
    background: var(--jp-layout-color2);
    color: var(--jp-ui-font-color1);
}

.c-btn-fill-primary {
    background: var(--jp-brand-color1);
    color: var(--jp-ui-inverse-font-color1);
}

.c-btn-fill-warn {
    background: var(--jp-warn-color1);
    color: var(--jp-ui-inverse-font-color1);
}

.c-btn-fill-error {
    background: var(--jp-error-color1);
    color: var(--jp-ui-inverse-font-color1);
}

.c-btn-fill-success {
    background: var(--jp-success-color1);
```

```
    color: var(--jp-ui-inverse-font-color1);
}

.c-btn-fill-info {
    background: var(--jp-info-color1);
    color: var(--jp-ui-inverse-font-color1);
}

.c-btn-fill-accent {
    background: var(--jp-accent-color1);
    color: var(--jp-ui-inverse-font-color1);
}

.c-btn-square {
    width: 7em;
    height: 7em;
    padding: 0.5em;
}

.c-btn-icon-text div {
    display: grid;
    grid-template-rows: 1fr auto;
    width: 100%;
    height: 100%;
    align-content: center;
    justify-content: center;
}

.btn-icon {
    margin: auto;
}

.btn-label {
    padding: 0.2em;
    font-size: var(--jp-content-font-size1);
    line-height: 1.1;
    max-height: 2.5em;
    text-overflow: ellipsis;
}

#add-snippet-icon {
    transform: scale(2);
    fill: #5da748;
}

#contents {
    display: flex;
    flex-direction: column;
}

#contents h4 {
    margin: 0;
    padding: 0;
```

```
    cursor: pointer;
}

.snippet-manager table {
    width: 100%;
}

#snippet-list-big tr {
    display: grid;
    grid-template-columns: 1fr auto auto;
    grid-column-gap: 0.5em;
    padding: 0.5em;
    align-content: center;
    transition: 200ms all cubic-bezier(0.4, 0.0, 0.2, 1);
    font-size: var(--jp-content-font-size1);
    box-shadow: var(--jp-elevation-z0);
}

#snippet-list-big td {
    margin: auto 0;
}

#snippet-list-big th {
    text-align: left;
}

.snippet-manager h3 {
    font-size: 1.4em;
    font-weight: var(--jp-content-heading-font-weight);
    color: var(--jp-ui-font-color1);
    box-sizing: border-box;
    text-align: left;
    margin: 1em 0 0.25em 0
}

.snippet-container {
    height: 18em;
    overflow-y: scroll;
    display: flex;
    flex-direction: column;
}

.snippet-list-item {
    border: none;
    margin: 2px;
    padding: 0.5em;
    font-size: var(--jp-ui-font-size1);
    font-family: var(--jp-ui-font-family);
    box-shadow: var(--jp-elevation-z0);
    transition: 200ms all cubic-bezier(0.4, 0.0, 0.2, 1);
    background: transparent;
    color: var(--jp-ui-font-color1);
    cursor: pointer;
```

```
    text-align: left;
}

.snippet-list-item:hover {
    box-shadow: var(--jp-elevation-z1);
    color: var(--jp-brand-color0);
}
```