

Sadržaj

1	Uvod	1
2	Teorijske osnove	2
2.1	Softver otvorenog koda	2
2.2	Sistem za praćenje verzija datoteka	3
2.3	Git	4
2.4	GitHub	5
2.4.1	Upravljanje projektima	5
2.4.2	Upravljanje timom	6
2.4.3	Pregled i komentarisanje koda	6
2.4.4	Integracija	7
2.4.5	Paketi	7
2.4.6	Bezbednost	7
2.4.7	Hostovanje sajtova	8
2.4.8	Akcije	8
2.5	Python	13
2.6	Python paketi, PyPI	14
2.6.1	Distribuiranje Python paketa	14
2.7	Poetry	14
2.7.1	Instalacija Poetry-ja	15
2.7.2	Kreiranje Python projekta koji koristi Poetry	15
2.7.3	Kreiranje, pakovanje i objavljivanje Python paketa	16
2.8	Pipenv	17
2.8.1	Instalacija Pipenv-a	17

2.8.2	Kreiranje projekta i upotreba Pipenv-a	17
2.9	Bash	19
3	Specifikacija i implementacija projekta	20
3.1	pjisp-template-name	21
3.1.1	Način imenovanja repozitorijuma	21
3.1.2	Instalacija i pokretanje	22
3.2	pjisp-diff	22
3.2.1	Instalacija i pokretanje	23
3.3	poetry-publish	23
3.3.1	Primer koraka koji koristi akciju	24
3.3.2	Upotreba akcije	25
3.4	smoke_test	25
3.5	pjisp-assignment-template	26
3.5.1	Project create	28
3.5.2	PJISP assignment	30
4	Primeri korišćenja	33
4.1	Primer kreiranja repozitorijuma	33
4.2	Primer ispravnog kreiranja zadatka	38
4.3	Primeri grešaka	39
4.3.1	Primer sa neispravnim testovima	39
4.3.2	Primer sa neispravnim izmenama datoteka	40
4.3.3	Primer sa neispravnim nazivom repozitorijuma	40
5	Diskusija i zaključci	43
5.1	Zaključci nakon upotrebe GitHub akcija	43
5.2	Dalji razvoj projekta	44
6	Literatura	45

Spisak slika

2.1	Sistem za praćenje verzija datoteka	4
3.1	Kada nema status	28
3.2	Kada se “PJISP assignment” neuspešno izvrši	28
3.3	Kada se “PJISP assignment” uspešno izvrši	28
4.1	Kreiranje repozitorijuma - šablon	33
4.2	Kreiranje repozitorijuma - popunjavanje forme	34
4.3	Generisanje repozitorijuma	34
4.4	Repozitorijum	35
4.5	<i>README.rst</i> - nema bedž	35
4.6	Prikaz tokova poslova	35
4.7	Prikaz poslova i koraka	36
4.8	Detalji izvršavanja koraka	36
4.9	Završetak izvršavanja tokova poslova	36
4.10	“PJISP assignment” - neuspešno izvršavanje posla	37
4.11	“PJISP assignment” - detalji o neuspešnom izvršavanju posla	37
4.12	<i>README.rst</i> - failing bedž	38
4.13	Prikaz tokova poslova	38
4.14	<i>README.rst</i> - passing bedž	39
4.15	Kada se neki od testova neuspešno izvrši	39
4.16	Kada se ne izmeni datoteka koju je potrebno izmeniti	40
4.17	Kreiranje repozitorijuma sa neispravnim nazivom	41
4.18	Prikaz tokova poslova	41

4.19	Kada dužina naziva repozitorijuma nije odgovarajuća	42
4.20	Kada naziv repozitorijuma nije odgovarajući	42
5.1	GitHub status, Avgust 2020	43

Skraćenice

- API** – *Application programming interface*, Interfejs za programiranje aplikacija
- CLI** – *Command line interface*, Interfejs komandne linije
- GCC** – *GNU compiler collection*, GNU kolekcija kompajlera
- PDF** – *Portable Document Format*, Portabilni format dokumenata
- URL** – *Uniform resource locator*, Jedinstveni lokator resursa
- YAML** – *YAML Ain't Markup Languag*, YAML nije jezik za označavanje

1. Uvod

Od samog početka razvoja računarstva, ideja je bila da računari i softverski alati koji se na njima koriste olakšaju ljudske aktivnosti koje zahtevaju mnogo uloženog vremena ili resursa da bi se izvršile jednom ili više puta. Ove aktivnosti su vremenom postajale sve kompleksnije, a alati koji ih rešavaju sve moćniji. Jedan od takvih alata, koji je nastao pre svega nekoliko meseci, su GitHub akcije.

Zadatak ovog rada je da kroz realan primer razvoja softvera otvorenog koda (eng. *open-source*) testira primer upotrebe Github akcija i sve prednosti i mane koje ovaj alat donosi. Projekat otvorenog koda koji se koristi u ovom primeru je *pjisp-assignment-template*. U pitanju je softverski alat koji se koristi za automatizovano kreiranja zadataka za studente korišćenjem šablona na predmetu Programski jezici i strukture podataka na Fakultetu tehničkih nauka u Novom Sadu.

Proces kreiranja zadatka upotrebom *pjisp-assignment-template* alata sastoji se iz nekoliko koraka, te postoji mogućnost da se napravi greška. Pomoću GitHub akcija, u malopre pomenuti alat se integrišu drugi alati, koji služe za vršenje provera ispravnosti nekih od koraka i koji dodatno olakšavaju proces kreiranja zadatka. Ovi alati predstavljaju programe koji se mogu instalirati i nezavisno od *pjisp-assignment-template* alata, a i sami za pripremanje izvršne verzije koriste GitHub akcije.

Ovaj rad opisuje sve prethodno pomenute alate, tehnologije koje se koriste za njihovu implementaciju i način njihove integracije u celinu upotrebom GitHub akcija. Na kraju samog rada, izvedeni su zaključci o GitHub akcijama stečeni na osnovu njihovog korišćenja u procesu izrade ovog rada.

2. Teorijske osnove

U ovom poglavlju će biti ukratko objašnjeni svi koncepti i alati koji su bili potrebni za izradu projekta o kojem ovaj rad govori. Tu grupu čini pre svega Softver otvorenog koda kao koncept pod kojim je ovaj projekat razvijen. Zatim se navode osnove karakteristike Sistema za praćenje verzija datoteka i Git kao glavni predstavnik takvih sistema. Naredno poglavlje se nadovezuje pričom o GitHub-u. Sledećih nekoliko poglavlja se bave Python programskim jezikom, njegovim paketima, repozitorijumom za te pakete i alatima za upravljanje paketima - Poetry-jem i Pipenv-om. Poslednje poglavlje opisuje Bash jezik komandne linije.

2.1 Softver otvorenog koda

Pojam otvoreni kod se odnosi na nešto što je dostupno svima za pravljenje izmena ili deljenje. Javio se kao pojam vezan za razvoj softvera (eng. *software*), ali se danas koristi i u drugim kontekstima vezanim za otvorenost i dostupnost.

Softver otvorenog koda je softver čiji izvorni kod svako može da posmatra, menja, poboljšava, ispravlja greske, dodaje nove funkcionalnosti, itd. Ovakav softver se objavljuje pod licencama otvorenog koda. Ove licence dozvoljavaju korišćenje i deljenje koda pod određenim uslovima. Najčešće se kodu može pristupiti besplatno. Nekada se licencom zahteva da se kod koristi samo u nekomercijalne svrhe ili da se navede originalni autor koda. Često licenca traži i da se izmenjeni kod deli pod istom tom licencom.

Pogodnosti pisanja i korišćenja ovakvog koda su mnogobrojne, a ovo su neke od njih:

- **Kontrola** - Korisnici softvera otvorenog koda imaju uvid u kod koji koriste, i samim tim veću kontrolu nad onim što instaliraju i koriste na svojim računarima. Takođe, mogu da menjaju

delove koda koji im se ne dopadaju.

- **Učenje** - Dostupnost koda omogućava korisnicima da analiziraju kod i time postaju bolji programeri. Takođe, pisanjem softvera otvorenog koda, korisnici ostavljaju mogućnost drugim korisnicima da vide njihov kod i da im daju sugestije/kritike. Pri pronalaženju grešaka u kodu, korisnici mogu da ih podele sa drugima i time spreče druge korisnike u pravljenju istih grešaka.
- **Bezbednost** - Mnogi smatraju softver otvorenog koda bezbednijim od softvera zatvorenog koda. Zato što je kod javan i dostupan svima za izmene, postoji mogućnost da će neko od korisnika pronaći i ispraviti propuste koje autor nije primetio. Takođe, nađene greške se brže ispravljaju pošto nema potrebe da se čeka na autora koda da ih ispravi, već to može da uradi bilo ko.
- **Stabilnost** - Softver otvorenog koda se takođe smatra stabilnijim i pouzdanijim rešenjem za projekte koji treba da se koriste dugi niz godina. Čak i ako originalni autor koda odustane od razvoja, postoji velika mogućnost da će neko od korisnika i učesnika nastaviti da ga održava.
- **Zajednica** - Oko softvera otvorenog koda često se formira zajednica korisnika i učesnika u razvoju koda. To su ljudi koji pišu, testiraju i koriste kod i zajedno razmenjuju ideje o poboljšanjima projekta i napretku.

2.2 Sistem za praćenje verzija datoteka

Sistem za praćenje verzija datoteka (eng. *version control system*) je sistem zadužen za praćenje istorije promena u datotekama. Promena koja se pamti se najčešće naziva revizija. Sve promene se uglavnom čuvaju na centralizovanom sistemu za skladištenje podataka i svi korisnici sa potrebnim privilegijama mogu da mu pristupaju, preuzmu sve nove revizije i primene ih na lokalnim datotekama, izmene lokalne datoteke i zatim objave novu reviziju koja se od tog momenta smatra najnovijom.

U ovim sistemima, čuva se celokupna istorija promena i moguće je vratiti datoteke u bilo koju od prethodnih revizija. U pogledu softvera, ovo nam omogućava da vratimo u upotrebu bilo koju prethodnu verziju, u slučaju da dođe do greške u nekoj od novijih verzija. Istorija se čuva u strukturi tipa grafa:

Graf se najčešće sastoji od glavne grane (eng. *branch*), na kojoj se nalaze glavne izmene, i pomoćnih grana na kojima se nalaze izmene za koje još nije odlučeno da li će pripadati glavnoj grani. Ako sve izmene na pomoćnim granama budu odgovarajuće, one se spajaju (eng. *merge*) sa glavnom granom i postaju deo njenog sadržaja. U slučaju praćenja verzija softvera, ovakve grane najčešće predstavljaju nove funkcionalnosti (eng. *feature*) koje se dodaju na glavnu granu projekta ili ispravljanje greške (eng. *bugfix*) koja je nastala u nekoj od prethodnih verzija.

2.3 Git

Za razliku od drugih sistema za praćenje verzija datoteka u distribuiranom režimu, Git nije softver klijent-server arhitekture. Svaki Git repozitorijum na svakom računaru čuva kompletnu istoriju promena i verzija i nije zavisen od pristupa mreži ili centralizovanom serveru. Kao takav, Git je osnova za razvoj mnogih drugih nezavisnih alata za praćenje verzija datoteka.

2.4 GitHub

GitHub [2] je servis otvorenog koda za hostovanje (eng. *hosting*) repozitorijuma sa različitim datotekama, najčešće izvornim kodovima za razna softverska rešenja. Za praćenje verzija datoteka, GitHub koristi Git. GitHub služi kao centralizovano mesto na kojem se čuvaju i razmenjuju sve izmene koje su sačuvane upotrebom Git alata.

U ovom trenutku, GitHub se smatra najpopularnijim alatom ovog tipa i trenutno broji preko 40 miliona korisnika. Stvari koje ovaj servis omogućava su:

1. Upravljanje projektima
2. Upravljanje timom
3. Pregled i komentarisanje koda
4. Integracija
5. Paketi
6. Bezbednost
7. Hostovanje
8. Akcije

Moguće mu je pristupiti preko CLI verzije, desktop verzije, ekstenzije za Visual Studio program za uređivanje datoteka (eng. *text editor*) i mobilne aplikacije.

2.4.1 Upravljanje projektima

Upravljanje projektom je proces koji je potreban u bilo kojem većem programerskom projektu. Menadžeri projekta su zaduženi da prate razvoj projekta i upućuju programere u razvoj narednih delova projekata, kao i da upravljaju raspoređivanjem vremena rada na različitim zadacima koje projekat iziskuje.

GitHub nudi mogućnost koordinacije projektom, praćenje razvoja i menjanje statusa projekta na jednom mestu. Koordinacija započinje kreiranjem zadataka (eng. *tasks*) koji traže izmene koda ili unapređenja projekta. Zadaci mogu biti u obliku greške/pitanja (eng. *issue*), komentara (eng. *comment*) ili zahteva za izmenom (eng. *pull-request*), što podrazumeva kreiranje celine koja je spremna za spajanje sa glavnim granom na projektu. Zadacima se dodeljuju osobe koje su zadužene za njih. Time će oni dobijati obaveštenja kada dođe do bilo koje izmene u projektu vezane za njihov

deo posla. Zadatacima je moguće dodeliti i prioritet i oznake (eng. *label*), što dodatno olakšava proces upravljanja projektom.

Sve ove informacije moguće je pratiti i menjati u formi tabele (eng. *board*) ili u repozitorijumu na kojem se projekat nalazi. Svaki zadatak ima unikatan URL na kojem se mogu videti sve potrebne informacije vezane za njega. Čuva se i istorija svih zadataka i izmena na projektu, kojima se stvara uvid u proces razvoja projekta. Po završetku svih zadataka, upravljanje projektom se završava i projekat se označava kao završen.

2.4.2 Upravljanje timom

Kako u timu imamo različite uloge, koje imaju različite permisije, GitHub nudi mogućnost hijerarhijskog organizovanja učesnika na projektu gde se svakoj grupi učesnika ili pojedincu dodeljuju potrebna prava.

Takođe, moguće je dodati u repozitorijum dokumente koji se tiču projekta, kao što su kodeks ponašanja (eng. *code of conduct*) i licenca (eng. *license*). Ove dokumente nije potrebno svaki put pisati iznova, već postoje unapred predefinisane verzije često korišćenih kodeksa ponašanja i licenci.

2.4.3 Pregled i komentarisanje koda

Komunikacija između učesnika na projektu doprinosi kvalitetu izrade samog projekta. Pored podele zadataka i raspoređivanja vremena za svaki zadatak, javlja se potreba i za diskutovanjem koda koji se dodaje u obliku zahteva za izmenom.

Pri kreiranju zahteva za izmenom ili revizija (eng. *commit*), GitHub nudi prikaz svih izmena u odnosu na granu sa koje su izmene krenule i na koju bi se dodale ako je sve u redu. Taj prikaz je dostupan svima koji učestvuju u izradi projekta. Moguće je, dodatno, zamoliti nekoga od učesnika da pregleda izmene pre njihovog spajanja sa odabranom granom (eng. *request a review*). Na svakoj od izmena, moguće je dodati poruku, koja otvara mogućnost diskusije sa ostalim učesnicima projekta. Po završetku, osoba koja je zadužena da pregleda izmene može da odobri ili odbije zahtev za izmenom, uz mogućnost traženja dodatnih izmena pre odobrenja.

Takođe, pri kreiranju zahteva za izmenom, GitHub automatski pregleda kod u potrazi za bezbednosnim propustima ili konfliktima sa granom sa kojom se trenutna grana spaja. Ako ih ima, GitHub obaveštava korisnika i traži izmenu pre odobrenja i spajanja koda.

2.4.4 Integracija

Pored samog pisanja aplikacije, često se javlja potreba za testiranjem i praćenjem (eng. *monitoring*) aplikacije, automatizovanim postavljanjem u produkciju (eng. *deployment*), kontinualnom integracijom (eng. *continuous integration*), proverom kvaliteta koda (eng. *code quality*) i drugim pomoćnim aktivnostima u procesu razvoja softver-a.

Pošto GitHub nije u mogućnosti da ponudi sve te aktivnosti, omogućio je proces integracije sa servisima koji ih nude. Svi ovi servisi mogu se pronaći u GitHub prodavnici (eng. *marketplace*) [3]. Jedna od novina u GitHub prodavnici su GitHub akcije koje će biti jedna od centralnih tema ovog rada, a biće opisane u jednoj od narednih sekcija.

2.4.5 Paketi

Alati poput NPM-a, Docker-a, Maven-a, Gradle-a i drugih omogućavaju kreiranje artifakta koji sadrže sve potrebne alate za pokretanje nekog softverskog rešenja. Većina ovakvih servisa ima mogućnost čuvanja artifakta na njihovim repozitorijumima.

Zbog jednostavnosti korišćenja, GitHub je kreirao alat koji objedinjuje mnoge ovakve repozitorijume u jedan repozitorijum - GitHub Packages.

2.4.6 Bezbednost

Bezbednost u softverskim rešenjima igra veoma bitnu ulogu i najčešće zavisi od mnogo faktora: programera, ljudi koji održavaju kod (eng. *maintainers*), istraživača, timova specijalizovanih za bezbednost i drugih. Da bi se postigao što veći stepen bezbednosti, potrebno je da što više različitih strana učestvuje u testiranju koda na bezbednosne propuste (eng. *vulnerabilities*).

Neke od mogućnosti koje GitHub nudi za poboljšanje bezbednosti koda koji se nalazi na njemu su:

- **Automatsko skeniranje koda** - pri dodavanju koda na repozitorijum, pokreće se automatsko skeniranje koda na poznate bezbednosne propuste i obaveštava se korisnik ako se pronađe neki od njih
- **Definisanje bezbednosnog procesa rada projekata otvorenog koda** - omogućava definisanje bezbednosnih polisa u okviru *SECURITY.MD* datoteke u repozitorijumu u kojoj se navode sve potrebne informacije o tome koje bezbednosne probleme i na koji način treba da prijave korisnici koji ih pronađu

- **Diskutovanje o uticaju bezbednosnih propusta** - kada neki od korisnika prijavi bezbednosni propust ili grešku u kodu, GitHub nudi prostor za komentarisanje i razmenu mišljenja vezanog za postavljeni problem
- **Automatsko skeniranje zavisnosti** - slično skeniranju koda, pokreće se automatsko skeniranje zavisnosti nekog projekta - odnosno svih drugih softverskih rešenja od kojih projekat zavisi. Ako neko od njih ima bezbednosni propust, korisnik se obaveštava o tome. Ovakvi problemi često budu otklonjeni u narednoj verziji rešenja od kojeg projekat zavisi, pa GitHub često nudi promenu verzije kao rešenje ovakvog problema. Ovaj proces se nastavlja za vreme životnog ciklusa projekta, i ako se u bilo kojem momentu pronađe bezbednosni propust, automatski se kreira zahtev za izmenom sa ispravkom
- **Pretraga kredencijala** - korisnicima se može desiti da slučajno upišu kredencijale za neki od eksternih servisa u kod projekta. Takvi kredencijali imaju specifičan oblik koji GitHub prepoznaje za preko 24 servisa i obaveštava korisnika ako ih pronađe

2.4.7 Hostovanje sajtova

Pored hostovanja repozitorijuma, što je glavna uloga GitHub-a, postoji i mogućnost hostovanja sajta pod GitHub domenom. Potrebno je napraviti repozitorijum sa nazivom *username.github.io* i svi dokumenti u okviru njega će biti statički hostovani pod *username.github.io* domenom.

2.4.8 Akcije

Kao što vidimo iz prethodnih sekcija, GitHub nudi mnoge mogućnosti pored njegove osnovne namene - praćenja verzija datoteka. Ipak, ako i te mogućnosti nisu dovoljne, postoji alat - GitHub Actions koji nudi mogućnost pisanja tokova posla (eng. *workflow*) i reagovanje na mnoge događaje, tako da korisnik može, u specifičnom formatu, da definiše sve potrebne reakcije na događaje i pokrije slučajeve koje ostali GitHub alati ne pokrivaju. Ove akcije najčešće podrazumevaju automatizaciju, podešavanje i izvršavanje delova softverskih rešenja nakon nekog specifičnog događaja.

2.4.8.1 Događaji

Neki od događaja na koje je moguće reagovati su:

- **create** - pokreće se izvršavanje svaki put kada se kreira nova grana ili privezak
- **delete** - pokreće se izvršavanje svaki put kada se obriše grana ili privezak

- `fork` - pokreće se izvršavanje pri račvanju novog projekta iz trenutnog projekta
- `gollum` - pokreće se izvršavanje svaki put kada se kreira ili izmeni Wiki stranica vezana za projekat
- `issues` - pokreće se izvršavanje svaki put kada se desi *issue* događaj, što podrazumeva otvaranje teme sa pitanjem, njenu izmenu, zatvaranje, brisanje, dodavanje priveska, itd.
- `issue_comment` - pokreće se izvršavanje svaki put kada se doda, izmeni ili obriše komentar na otvorenom pitanju
- `label` - pokreće se izvršavanje svaki put kada se doda, izmeni ili obriše oznaka
- `milestone` - pokreće se izvršavanje svaki put kada se doda, izmeni ili obriše prekretnica u projektu
- `project` - pokreće se izvršavanje svaki put kada se desi *project* događaj, što podrazumeva kreiranje, izmenu, zatvaranje ili brisanje projekta
- `public` - pokreće se izvršavanje svaki put kada privatni repozitorijum postane javni
- `pull_request` - pokreće se izvršavanje svaki put kada se desi *pull_request* događaj, što podrazumeva otvaranje zahteva za izmenom ili njegovu izmenu, zatvaranje, brisanje, dodeljivanje, dodavanje priveska, sinhronizaciju, traženje pregledanja, itd.
- `pull_request_review` - pokreće se izvršavanje kada se izvrši, izmeni ili obriše pregledanje zahteva za izmenom
- `pull_request_review_comment` - pokreće se izvršavanje kada se kreira, izmeni ili obriše komentar na pregledanju zahteva za izmenom
- `push` - pokreće se izvršavanje svaki put kada se doda novi kod na granu
- `registry_package` - pokreće se izvršavanje svaki put kada se objavi ili izmeni paket
- `release` - pokreće se izvršavanje svaki put kada se desi *release* događaj, što podrazumeva kreiranje, objavljivanje, izmenu ili brisanje nove verzije projekta

Takođe, moguće je zakazati izvršavanje neke akcije navođenjem tačnog datuma i vremena izvršavanja ili zakazivanjem izvršavanja na primer svake nedelje u određeno vreme. Za opisivanje vremena koristi se softverski alat *Cron*, koji je zadužen za planiranje i zakazivanje poslova na osnovu vremena (eng. *time-based job scheduler*) na Unix operativnim sistemima.

Tokovi poslova se opisuju u YAML formatu. Događaji se opisuju u okviru *on* sekcije, uz dodatak *type* i *branch* podsekcija ako ima potrebe. Na primer:

```
on:
  pull_request:
    branches:
      - master
  release:
    types:
      - created
```

2.4.8.2 Poslovi

Nakon prepoznavanja događaja, pokreće se izvršavanje jednog ili više poslova (eng. *jobs*), gde se svaki od poslova može sastojati od jednog ili više koraka (eng. *step*).

Za poslove se definiše na kojem operativnom sistemu treba da se izvrše. U ponudi su Windows Server 2019, Ubuntu 20.04, Ubuntu 18.04, Ubuntu 16.04 i macOS Catalina 10.15. Opciono se dodaje naziv posla, dodatni uslov pod kojim se izvršava, promenjive koje se preuzimaju iz okruženja (eng. *environment variables*), maksimalno vreme izvršavanja (eng. *timeout*) i drugi parametri.

2.4.8.3 Koraci

Koraci su zaduženi za pokretanje komandi, pokretanje zadataka za podešavanje (eng. *setup tasks*) ili pokretanje GitHub Akcija iz trenutnog repozitorijuma, nekog javnog repozitorijuma ili sa Docker registra (eng. *registry*). Korak ne mora da ima akciju u sebi, ali svaka akcija koja se pokreće mora da se pokreće kao izolovani korak. Svaki korak se pokreće kao zaseban proces u okruženju pokretača (eng. *runner*) i ima pristup svom okruženju i sistemu datoteka na kojem se pokreće. Ako ima potrebe, mogu postojati zasebni koraci koji se pokreću na početku i kraju posla, koji služe da pripreme okruženje pre početka izvršavanja i odrade završne aktivnosti na kraju izvršavanja posla.

Kao i posao, korak sadrži svoj unikatni identifikator, naziv, uslov izvršavanja ako postoji, ulazne parametre, promenjive koje se preuzimaju iz okruženja, maksimalno vreme izvršavanja itd. Aktivnost koja se izvršava u toku koraka može se definisati na dva načina, upotrebom jednom od sledećih rezervisanih reči:

- *uses* - u slučaju kada se koristi postojeća akcija, na ovom mestu se navodi putanja do akcije

(u trenutnom repozitorijumu, nekom javnom repozitorijumu ili na Docker registru) i verzija akcije u obliku: `path/name@version`.

- `run` - u slučaju kada se ne koristi postojeća akcija, već se definiše niz komandi u interfejsu komandne linije operativnog sistema. Moguće je imati više `run` komandi u okviru jednog koraka. Tada se svaka komanda pokreće kao odvojeni proces u jezgru operativnog sistema. Ako se pod jednom `run` sekcijom pokrene komanda u više linija koja se sastoji od više komandi u jednoj liniji, tada se sve jednolinijske komande pokreću u okviru jednog procesa. Komande se mogu pokretati u jednom od sledećih jezika komandne linije: `bash`, `python`, `sh`, `cmd`, `powershell`.

2.4.8.4 Kontekst okruženja

Na osnovu konteksta okruženja, moguće je odlučivati o tome da li će se neki korak ili posao izvršiti. Za to postoji rezervisana reč `if`. Kontekst se nalazi u nekoliko objekata koji su nosioci različitih informacija i konteksta različitih delova toka posla. Neki od najčešće korišćenih su:

- `github` - informacije o toku posla

Primer: naziv akcije, putanja do akcije, identifikator trenutnog posla koji se izvršava, koji korisnik je zaslužan za pokretanje akcije, koji korisnik je vlasnik repozitorijuma na kojem je pokrenuta akcija, grana ili privezak na kojem se desilo pokretanje akcije, itd.

- `env` - sadrži promenjive koje se preuzimaju iz okruženja podešene u okviru toka posla, posla ili koraka.

- `job` - informacije o poslu koji se trenutno izvršava

Primer: status posla, kontejner u kojem se posao izvršava, servisi koji su potrebni za izvršavanje posla, itd.

- `steps` - informacije o koraku koji se trenutno izvršava

Primer: status koraka, vrednost izlaznih promenljivih, itd.

- `runner` - informacije o pokretaču trenutnog posla

Primer: operativni sistem, putanja privremenog (eng. *temporary*) direktorijuma, itd.

- `secrets` - sadrži promenjive koje se čuvaju kao tajne

- `needs` - dozvoljava pristup promenjivama koje su izlazne promenjive prethodnih poslova

2.4.8.5 Pisanje akcija

Neke od unapred definisanih akcija nudi GitHub, dok su druge napisane od strane zajednice koja ih koristi. Svako može da napiše akciju koju koristi za sebe ili da je podeli sa drugim korisnicima. Akcija se može pokretati direktno u okviru virtuelne mašine (eng. *virtual machine*) ili u okviru Docker kontejnera. Mogu se definisati ulazni i izlazni parametri i promenjive koje se preuzimaju iz okruženja.

Pri pisanju akcije potrebno je kreirati *action.yml* ili *action.yaml* datotoku koja definiše sve informacije potrebne za izvršavanje akcije. Akcija može biti u obliku Docker kontejnera, ili opisana u JavaScript programskom jeziku. Takođe, postoje kompozitne akcije (eng. *composite run steps action*) koje omogućavaju kombinovanje nekoliko koraka u jednu akciju.

- **Docker akcije** - Docker kontejneri upakuju celokupno okruženje sa kodom GitHub akcije. Ovaj pristup čini dosta pouzdano rešenje, pošto korisnik ne mora da brine o verzijama i alatima. U kontejneru se instaliraju svi potrebni alati, što ovakav način pisanja akcija čini pogodnim za sisteme koji moraju da se prilagode specifičnom okruženju. Mana ovog pristupa je to što je potrebno dosta vremena da se kreira i pokrene kontejner pa su ove akcije sporije od JavaScript akcija.
- **JavaScript akcije** - Ove akcije se pokreću direktno na računarima pokretačima, što pojednostavljuje pisanje koda akcija i znatno ubrzava njihovo izvršavanje. Da bi se mogle izvršavati na bilo kojem operativnom sistemu, ove akcije moraju biti napisane u čistom JavaScript-u.

Akcije napisane od strane pojedinaca mogu se objaviti u GitHub prodavnici i time postati dostupne svima za korišćenje. Da bi akcija bila spremna za objavljivanje, potrebno je da se ceo kod akcije i datoteke potrebne za njeno izvršavanje nalaze u jednom repozitorijumu. Taj repozitorijum ne sme ništa drugo da sadrži. Takođe, repozitorijum mora biti javan, da bi drugi korisnici mogli da mu pristupe. Potrebno je dodati privezak na repozitorijum, koji predstavlja verziju akcije u GitHub prodavnici i zatim je objaviti.

2.4.8.6 Hostovanje pokretača

GitHub nudi mašine koje hostuju pokretače GitHub Akcija, ali i daje mogućnost hostovanja akcija na korisničkom hardveru (eng. *hardware*). Hostovanje na korisničkim mašinama daje veću slobodu korisnicima u izboru samih hardverskih komponenti, operativnog sistema i alata koji se instaliraju na tim mašinama. Mogu biti u obliku fizičkih mašina, virtuelnih mašina, kontejnera, ili računari u oblaku (eng. *cloud*).

Korisničke mašine se povezuju sa GitHub-om preko aplikacije koju GitHub nudi. Ako se ne pokrene ni jedna akcija u roku od 30 dana, prekida se veza između korisničkog računara i GitHub-a.

Problem pri hostovanju na korisničkim mašinama može biti to što je korisnik zadužen za ažuriranje verzije operativnog sistema i svih softverskih alata na njemu, dok na GitHub-ovim mašinama to radi sam GitHub. Takođe, GitHub pokreće novu, čistu instancu za svaki posao koji se izvršava, dok korisnik to ne mora da radi.

2.5 Python

Python [4] je interpretirani programski jezik opšte namene. Spada u grupu programskih jezika visokog nivoa. Dizajniran je tako da bude izrazito čitljiv, što se postiže obaveznom upotrebom belina (eng. *whitespace character*) i izbegavanjem upotrebe zagrada za razdvajanje blokova koda. Pogodan je za razvoj različitih softverskih rešenja pošto podržava više programskih paradigmi:

1. Proceduralnu
2. Objektno orijentisanu
3. Funkcionalnu
4. Strukturalnu

Takođe, prilagođen je za korišćenje u različitim okruženjima na različitim operativnim sistemima.

Python se prvi put spominje krajem 1980-tih kao naslednik ABC programskih jezika. Dizajnirao ga je Guido van Rossum i objavio prvu verziju 1991. godine. Python 2.0, koji je izdat 2000. godine je znatno unapređenje prve verzije, koje se do skoro koristilo. Poslednja velika revizija urađena je 2008. godine kada je nastao Python 3.0. Zbog toga što Python 3 nije u potpunosti kompatibilan sa prethodnim verzijama, podrška za Python 2 prestala je u januaru 2020. godine.

Iako sam jezik ima jako puno funkcionalnosti i većina stvari se može uraditi upotrebom standardne biblioteke, Python je dizajniran tako da podrži lako pravljenje novih biblioteka i lako dodavanje biblioteka u projekat. Ovo ga je učinilo pogodnim za pravljenje širokog spektra aplikacija različite namene.

2.6 Python paketi, PyPI

Python paketi su imenski prostori (eng. *namespaces*) koji u sebi sadrže druge pakete i module - delove softvera koji obavljaju konkretnu funkcionalnost. Svaki paket je direktorijum koji u sebi mora imati `__init__.py` datoteku. Ova datoteka može biti prazna, što označava da je trenutni direktorijum Python paket i da može biti korišćen kao zaseban modul. Inače, `__init__.py` datoteka čuva kod koji služi za inicijalizaciju tog paketa.

Da bi se koristile funkcionalnosti koje neki paket nudi, potrebno ga je uvezati (eng. *import*) u projekat korišćenjem rezervisane reči `import`. Da bi se uvezao pojedinačni objekat iz nekog paketa, treba naznačiti koji objekat se iz kojeg paketa uvezuje korišćenjem sledećeg Python koda:

- `from <package> import <object>`

2.6.1 Distribuiranje Python paketa

Glavna uloga Python paketa je distribuiranje biblioteka za Python programski jezik. Paket može da kreira bilo ko i da ga potom podeli sa ostatkom Python zajednice. Da bi paket bio dostupan, potrebno je kreirati arhivu sa svim potrebnim konfiguracijama za njega i objaviti je na nekom repozitorijumu za Python pakete.

Repozitorijumi su softverski alati za čuvanje i distribuiranje paketa. Postoje javni repozitorijumi, koji omogućavaju razmenu paketa sa bilo kim, i privatni, koji omogućavaju razmenu paketa u ograničenoj grupi korisnika koji imaju specijalne privilegije.

Najčešće korišćen javni repozitorijum je PyPI [5] - The Python Package Index.

2.7 Poetry

Poetry [6] je alat otvorenog koda za kreiranje Python paketa i upravljanje paketima koje Python projekat koristi (eng. *dependency management*). Pomaže korisnicima tako što umesto njih instalira i menja verzije bibliotekama od kojih projekat zavisi, a koje korisnik mora prethodno da specificira. Poetry je takođe znatno olakšao kreiranje, pakovanje i objavljivanje paketa na PyPI repozitorijumu i time omogućio lakše deljenje Python paketa sa drugim korisnicima Python programskog jezika.

Prva verzija alata objavljena je 28.02.2018. godine. Verzije programskog jezika Python koje su podržane su 2.7 i 3.4+. Ideja je da Poetry radi podjednako dobro na različitim platformama, između ostalog na Linux-u, Windows-u i macOS-u.

2.7.1 Instalacija Poetry-ja

Za razliku od drugih Python alata za upravljanje paketima od kojih projekat zavisi, umesto pip-a - instalera za Python pakete, Poetry koristi svoj specifičan način za instalaciju. Instalator doda Poetry u korisnički direktorijum, tako da može da se koristi sa bilo kojom verzijom Python-a. Omogućena je i instalacija Poetry-ja korišćenjem pip-a, ali se ne proporučuje iz dva razloga:

1. može dovesti do konflikta sa drugim sistemskim fajlovima
2. otežava održavanje konzistentnosti pri korišćenju različitih verzija Python-a i različitih virtuelnih okruženja (eng. *virtual environments*)

2.7.2 Kreiranje Python projekta koji koristi Poetry

Nakon instalacije alata, moguće je kreirati projekat koji koristi Poetry za upravljanje paketima ili dodati Poetry u postojeći projekat i time kreirati virtuelno okruženje u kojem će se izvršavati naredne akcije. U oba slučaja će se kreirati *pyproject.toml* datoteka koja čuva sve bitne informacije o projektu. U početku ova datoteka sadrži samo osnovne informacije o projektu:

```
[tool.poetry]
name = "poetry-demo"
version = "0.1.0"
description = ""
authors = ["Name Surname <email>"]

[tool.poetry.dependencies]
python = "*"

[tool.poetry.dev-dependencies]
pytest = "^3.4"
```

Informacije o samom projektu, njegovom autoru, verziji, licenci, opisu, dokumentaciji, itd. nalaze se u prvom segmentu - `[tool.poetry]`. Informacije o paketima koje projekat koristi nalaze se u naredne dve sekcije - `[tool.poetry.dependencies]` i `[tool.poetry.dev-dependencies]`, gde se u prvoj od ove dve sekcije nalaze informacije o paketima koji se koriste u produkcionom okruženju, a u drugoj informacije o paketima koji se

koriste u toku razvoja projekta. Pri instalaciji novog paketa, njegov naziv i verzija će se dopisati u jednu od ove dve sekcije, ili u obe ako je to potrebno.

Ovu datoteku je moguće ručno menjati, ali se sa njom najčešće interaguje pozivanjem poetry komandi u konzoli. Neke od tih komandi su:

- `poetry add` - dodaje novi paket u *pyproject.toml* datoteku
- `poetry remove` - briše paket iz projekta

Komande koje takođe interaguju sa *pyproject.toml* datotekom, ali je ne menjaju su:

- `poetry install` - instalira pakete definisane u *pyproject.toml* datoteci
- `poetry update` - ažurira pakete u skladu sa verzijama koje pišu u *pyproject.toml* datoteci
- `poetry lock` - zaključava pakete u projektu
- `poetry check` - proverava ispravnost *pyproject.toml* datoteke

Za pokretanje komandi u virtuelnom okruženju opisanom u *pyproject.toml* datoteci koristi se komanda:

- `poetry run`

Za sve ostale detalje najbolje je pogledati dokumentaciju pomoću komande:

- `poetry help`

2.7.3 Kreiranje, pakovanje i objavljivanje Python paketa

Da bi se Python projekat mogao objaviti, potrebno je kreirati arhivu sa svim potrebnim konfiguracijama za njega. To se postiže upotrebom komande:

- `poetry build`

Nakon toga, paket je spreman za objavljivanje. U projektu se pojavi novi direktorijum sa nazivom `dist` i u njemu se nalaze dve datoteke:

1. `poetry-demo-0.1.0-py2.py3-none-any.whl`

2. `poetry-demo-0.1.0.tar.gz`

Repozitorijum na kojem se paket objavljuje se podešava pomoću komande:

- `poetry config`

Paket se objavljuje pozivanjem komande:

- `poetry publish`

U tom momentu, paket postaje dostupan na izabranom repozitorijumu i svako ko ima pristup repozitorijumu može da instalira paket. Najčešće se paketi objavljuju na PyPI repozitorijumu.

2.8 Pipenv

Pipenv [7] je alat za upravljanje paketima koje Python projekat koristi. Zadužen je da automatski kreira i upravlja virtuelnim okruženjem i da dodaje i briše pakete iz *Pipfile* datoteke svaki put kada se instalira ili deinstalira neki Python paket. *Pipfile* datoteka čuva informacije o paketima i verzijama paketa koje se koriste u okviru virtuelnog okruženja. Pipenv na osnovu ove datoteke zaključava verzije paketa u projektu u okviru *Pipfile.lock* datoteke i time omogućava upotrebu istih zavisnih paketa u bilo kojem okruženju.

Pipenv alat je prilagođen da radi na različitim platformama, između ostalog na Linuxu, Windowsu, MacOS-u i BSD-u.

2.8.1 Instalacija Pipenv-a

Pipenv se, kao i većina drugih Python alata, instalira korišćenjem pip instalera za Python pakete kucanjem sledeće komande u konzoli:

```
pip install pipenv
```

2.8.2 Kreiranje projekta i upotreba Pipenv-a

Nakon instalacije alata, moguće je kreirati projekat koji koristi Pipenv za upravljanje paketima ili dodati Pipenv u postojeći projekat i time kreirati virtuelno okruženje u kojem će se izvršavati naredne akcije. Ovo se postiže pozivanjem komande:

```
pipenv install
```

u direktorijumu u kojem se nalazi ili će se nalaziti projekat čijim paketima Pipenv treba da upravlja. Tada će se kreirati *Pipfile* datoteka koja čuva sve bitne informacije o projektu. Ova datoteka u početku izgleda ovako:

```
[[source]]
name = "pypi"
url = "https://pypi.org/simple"
verify_ssl = true

[dev-packages]

[packages]

[requires]
python_version = "3.8"
```

Informacije o samom projektu nalaze se u prvom segmentu - `[[source]]`. Informacije o paketima koje projekat koristi nalaze se u naredne dve sekcije - `[dev-packages]` i `[packages]`, gde se u prvoj od ove dve sekcije nalaze informacije o paketima koji se koriste u toku razvoja projekta, a u drugoj informacije o paketima koji se koriste u produkcionom okruženju. Pri instalaciji novog paketa, njegov naziv i verzija će se dopisati u jednu od ove dve sekcije, ili u obe ako je to potrebno. Poslednja sekcija - `[requires]` specificira verziju Python-a koju je potrebno instalirati.

Ovu datoteku je moguće ručno menjati, ali se sa njom najčešće interaguje pozivanjem `pipenv` komandi u konzoli. Neke od tih komandi su:

- `pipenv install` - ako se navede naziv paketa, dodaje taj paket u virtuelno okruženje i *Pipfile* datoteku
- `pipenv uninstall` - deinstalira paket iz virtuelnog okruženja i brše ga iz *Pipfile* datoteke

Za kreiranje *Pipfile.lock* datoteke koristi se komanda:

```
pipenv lock
```

Pokretanje komandi u virtuelnom okruženju opisanom u *Pipfile* datoteci može se uraditi na dva načina:

- `pipenv shell` - kreira konzolu u virtuelnom okruženju i onda se naredne komande pozivaju u toj konzoli
- `pipenv run` - očekuje komandu kao argument i pokreće je u virtuelnom okruženju

2.9 Bash

GNU Bash [8] ili češće nazvan samo Bash (Bourne Again SHell) je Unix interpreter i jezik komandne linije koji je napravljen za GNU projekat kombinujući korisne funkcionalnosti iz Korn i Csh interpretera.

Objavljen je 1989. godine i od tada se koristi kao uobičajena konzola za prijavljivanje na većini Linux distribucija i svim verzijama macOS-a pre macOS Catalina verzije. U skorije vreme je dostupan i za Windows pomoću Windows podsistema za Linux (eng. *Windows Subsystem for Linux*).

Pokreće se kao interaktivni proces unutar terminala u kojem korisnik kuca komande koje pokreću određene akcije. Takođe, može da pokreće i komande koje se zadaju u obliku tekstualne datoteke.

3. Specifikacija i implementacija projekta

Projekat o kojem ovaj rad govori sastoji se iz grupe alata koje je bilo potrebno napisati i povezati da bi se dodale nove funkcionalnosti u postojeći projekat - *pjisp-assignment-template* [9]. Pomenuti alat služi za automatizovano kreiranje studentskih zadataka, njihovo puštanje u učionicama u kojima studenti rade zadatke i, na kraju, pregledanje zadataka. Prvenstveno je namenjen da nastavnom osoblju olakša ceo proces od kreiranja do pregledanja kolokvijuma ili zadataka koje studenti rade.

Zadatak nastavnog osoblja je pre svega da iz repozitorijuma projekta koji je šablonski (eng. *template*) repozitorijum kreira svoj repozitorijum sa zadatkom. Važno je da svoj repozitorijum nazove u skladu sa predefinisanim načinom imenovanja. Nakon toga se u repozitorijumu kreiraju svi potrebni materijali koji su potrebni za generisanje studentskog zadatka. Zadatak nastavnog osoblja je da pokrene generisanje zadatka za jedan od predefinisanih testova na predmetu Programski jezici i strukture podataka. Jedan od alata o kojem ovaj rad govori - *pjisp-template-name* dodaje mogućnost automatizovanog kreiranja šablona za zadatak za određeni test na osnovu naziva repozitorijuma koji autor zadatka navede, uz proveru ispravnosti naziva.

Po završetku pisanja zadatka i njegovog testiranja, nastavnik ili saradnik je dužan da obavesti drugog nastavnika o tome, da bi on mogao da proveriti ispravnost zadatka i odobri korišćenje zadatka u nastavi, naravno ako je dobar i pogodan za studente. Kao olakšicu, autor zadatka može da pokrene alat za testiranje zadatka i proveriti ispravnost. Projekat o kojem ovaj rad govori uvodi isto to testiranje i nakon objavljivanja nove verzije zadatka na GitHub-u. Kao dodatnu proveru, jedan od alata iz ovog projekta - *pjisp-diff* proverava da li su izmenjene sve potrebne datoteke, da bi zadatak bio kompletan, odnosno da bi imao korektan tekst zadatka, primer rešenja i testove. Ako svi ovi alati vrate rezultat pozitivnog ishoda, u GitHub repozitorijumu se pojavljuje oznaka da su svi uslovi ispunjeni i da je zadatak spreman za pregledanje. U suprotnom, stoji oznaka da

zadatak ne ispunjava sve potrebne uslove.

Naredna poglavlja opisuju alate koji su korišćeni da bi se postigla opisana unapređenja *pjisp-assignment-template* alata, a poslednje poglavlje opisuje njihovu integraciju u jednu celinu.

3.1 pjisp-template-name

Alat *pjisp-template-name* [10] je aplikacija koja služi za proveru ispravnosti naziva repozitorijuma. Napisana je u obliku konzolne aplikacije otvorenog koda u programskom jeziku Python.

3.1.1 Način imenovanja repozitorijuma

Za imenovanje repozitorijuma koristi se predefinisani šablon, koji izgleda ovako:

```
pjisp-{SCHOOL_YEAR}-{COURSE_ID}-{TEST_ID}-{GROUP_ID}
```

gde su:

- SCHOOL_YEAR - školska godina, na primer 2019
- COURSE_ID - identifikator kursa (E214 za zimski semestar ili E111 za letnji)
- TEST_ID - identifikator testa (T12, T34 ili SOV)
- GROUP_ID - identifikator grupe studenata, na primer G10

Alat pre svega proverava da li je dužina naziva ispravna, odnosno da li naziv ima sve celine odvojene znakom "-". Ako nema, ispisuje se poruka *"Repository name length not valid"* i program izlazi sa izlaznim kodom 1. U suprotnom, program nastavlja sa izvršavanjem.

Nakon provere dužine, proverava se ispravnost ostalih tokena, prema malopre opisanom načinu davanja imena. Ako bilo gde dođe do greške, ispisuje se poruka *"Repository name not valid. Error on <token>"* i izlazi se sa izlaznim kodom 1. U ovom slučaju, <token> predstavlja jednu od celina odvojenih znakom "-". Ako naziv u potpunosti ispunjava šablon, kao izlazna vrednost vraća se identifikator testa, da bi na osnovu toga mogle da se generišu datoteke potrebne za kreiranje tog testa.

3.1.2 Instalacija i pokretanje

Ovaj alat se instalira kao bilo koji drugi Python paket. Potrebno je u konzolnoj liniji pokrenuti komandu:

```
pip install pjisp-template-name
```

i nakon toga je alat spreman za upotrebu. Pokreće se pozivanjem komande:

```
pjisp_template_name <template_name>
```

3.2 pjisp-diff

Alat *pjisp-diff* [11] je aplikacija koja služi kao pomoćni alat nastavnom osoblju koji proverava da li su izmenili sve potrebne datoteke pri kreiranju zadataka za studente. Napisana je u obliku konzolne aplikacije otvorenog koda u programskom jeziku Python.

Pored provere da li su promenjene sve potrebne datoteke, ovaj alat proverava i da li su ostale nepromenjene sve pomoćne datoteke koje nije dozvoljeno menjati. Datoteke koje je potrebno menjati zavise od ulaznog parametra, šablona na osnovu kojeg se kreira zadatak. Datoteke koje je potrebno uvek menjati su:

- *assignment_solution.c*
- *assignment.rst*

Ako je vrednost ulaznog parametra jednaka “T12”, potrebno je menjati:

- *fixtures/stdio-numbers.yaml*

Ako je šablon vrednost ulaznog parametra jednaka “T34” ili “SOV”, potrebno je menjati:

- *fixtures/file-error-input-not-readable.yaml*
- *fixtures/file-error-output-not-writable.yaml*
- *fixtures/fixtures/file-text.yaml*

Datoteka koju nije dozvoljeno menjati je:

- `assignment_notes.rst`

Ako su po završetku programa izmenjene sve potrebne datoteke i nisu izmenjene one koje ne smeju da se menjaju, alat vraća 0 kao vrednost izlaznog koda. U suprotnom, alat vraća vrednost 1 uz jednu ili više poruka u obliku: “*Please change the <filename> file.*” ili “*Please do not change the <filename> file.*”. U ovom slučaju, <filename> predstavlja naziv datoteke koja nije izmenjena, a treba da bude ili je izmenjena, a ne treba da bude menjana.

3.2.1 Instalacija i pokretanje

Ovaj alat se instalira kao bilo koji drugi Python paket. Potrebno je u konzolnoj liniji pokrenuti komandu:

```
pip install pjisp-diff
```

i nakon toga je alat spreman za upotrebu. Pokreće se pozivanjem komande:

```
pjisp_diff <template>
```

gde <template> predstavlja identifikator testa i može biti T12, T34 ili SOV.

3.3 poetry-publish

Alati koji su opisani u prethodna 2 poglavlja napravljeni su tako da mogu biti instalirani i korišćeni kao Python paketi. Oba paketa su dostupna na PyPI repozitorijumu Python paketa. Za postavljanje na repozitorijum, korišćen je alat Poetry. Pošto je potrebno objaviti novu verziju paketa na repozitorijumu nakon svake značajne izmene u samom paketu, korišćena je *poetry-publish* [12] GitHub akcija koja pri svakom novom objavljivanju verzije alata na GitHub-u, kreira i novu verziju Python paketa i postavlja je na PyPI repozitorijum.

Akcija *poetry-publish* je kreirana od strane autora ovog rada pre nego što se razvila ideja za projektom koji je u ovom radu opisan, ali je znatno olakšala kreiranje malopre pomenutih paketa, tako da je imala veoma važnu ulogu i u ovom projektu. Akcija je projekat otvorenog koda i dostupna je za preuzimanje u GitHub prodavnici [13]. Dostupna je pod BSD 3-Clause licencom.

Poetry-publish je GitHub akcija koja služi za kreiranje paketa i njihovo objavljivanje na Python repozitorijumu upotrebom alata Poetry. Kreirana je u obliku Docker akcije. U Docker kontejneru se instaliraju svi potrebni alati za korišćenje Python-a i zatim se pokrenu Poetry komande za kreiranje i objavljivanje paketa:

- `poetry build` - da kreira Python paket
- `poetry publish` - da objavi paket na repozitorijumu koji je prethodno podešen

Paramtri koje ova akcija očekuje su:

- **`python_version`** - verzija Python-a koja se instalira u kontejneru i koristi za kreiranje paketa. Ako se ne navede, podrazumeva se da se koristi najnovija verzija. Za bolje performanse, poželjno je koristiti predefinisano - najnoviju verziju.
- **`poetry_version`** - verzija Poetry-a koja se instalira u kontejneru i koristi za kreiranje paketa. Ako se ne navede, podrazumeva se da se koristi najnovija verzija. Navodi se u PIP sintaksi za specifikaciju verzija
- **`pypi_token`** - jedini obavezan parametar. Služi kao API token za autentifikaciju pri objavljivanju paketa na PyPI.
- **`repository_name`** - naziv repozitorijuma na kojem se objavljuje paket. Ako se ne navede, podrazumeva se da se objavljuje na PyPI repozitorijumu. Potrebno je izmeniti pri objavljivanju na PyPI test repozitorijumu ili na privatnom repozitorijumu.
- **`repository_url`** - adresa repozitorijuma na kojem se objavljuje paket. Ako se ne navede, podrazumeva se da se objavljuje na PyPI repozitorijumu. Potrebno je izmeniti pri objavljivanju na PyPI test repozitorijumu ili na privatnom repozitorijumu.

Takođe, potrebno je specificirati *pyproject.toml* datoteku na osnovu koje će se kreirati Python paket. Ova datoteka treba da se nalazi u korenskom direktorijumu repozitorijuma koji koristi ovu akciju.

3.3.1 Primer koraka koji koristi akciju

```
- name: Build and publish to pypi
uses: JRubics/poetry-publish@v1
with:
  python_version: '3.7.1'
  poetry_version: '==1.0.5' # (PIP version specifier syntax)
  pypi_token: ${ secrets.PYPI_TOKEN }
  repository_name: 'testpypi'
  repository_url: 'https://test.pypi.org/legacy/'
```

3.3.2 Upotreba akcije

U *pjisp-template-name* i *pjisp-diff* akcija je iskorišćena na sledeći način:

```
name: Python package
on:
  push:
    tags:
      - 'v*.*.*'
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Change version in pyproject.toml
        run: |
          REF=$(echo ${github.ref} | sed "s#\(\refs/tags/\)\?v\?##")
          sed -i "s/^version=\.\.\/version=\"$REF\"/" pyproject.toml
      - name: Build and publish to pypi
        uses: JRubics/poetry-publish@v1.1
        with:
          pypi_token: ${secrets.PYPI_TOKEN}
```

Ova akcija se izvršava samo kada se napravi novi privezak u repozitorijumu u obliku *v*.*.**. Sastoji se od jednog posla koji se pokreće na poslednjoj verziji Ubuntu operativnog sistema i 3 koraka. Prvi korak preuzima sadržaj repozitorijuma. Drugi korak menja verziju u *pyproject.toml* datoteci na verziju priveska koji je prouzrokovao pokretanje ove akcije. Poslednji korak koristi *poetry-publish* akciju za objavljivanje nove verzije paketa. Svi ulazni parametri koriste predefinisanu verziju, samo je *pypi_token* parametar postavljen na vrednost PyPI tokena koji se preuzima iz GitHub-ovog konteksta okruženja *secrets*.

3.4 smoke_test

Još jedan alat otvorenog koda koji je korišćen i izmenjen u ovom radu je *smoke_test* [14]. To je konzolna aplikacija i API koji služi za testiranje studentskih zadataka na dim (eng. *smoke testing*).

Ovaj alat se koristi i za testiranje zadataka koje nastavno osoblje piše kao primer tačno urađenog zadatka i samim tim se koristi u okviru *pjisp-assignment-template* alata.

Zadatak koji ovaj alat očekuje treba da bude napisan u programskom jeziku C ili nekom jeziku od kojeg je GCC programski prevodilac sposoban da napravi izvršnu datoteku. Zadatak se prvobitno kompajlira pomoću GCC-a. Nakon toga zadatak se testira na slučajeve korišćenja opisane u *.yaml* datotekama koje nastavno osoblje priprema prilikom kreiranja zadatka. Oblik pojedinačnog test slučaja je uvezani par datoteka koje sadrže, respektivno, ulazne vrednosti i očekivani izlazni oblik, koji je rešenje zadatka dužno da ispoštuje. Svaki slučaj korišćenja vraća rezultat onoga za šta je bio zadužen da testira u obliku očekivane vrednosti i vrednosti koja se dobije evaluacijom studentskog rešenja. Rezultati svih slučajeva na koje se zadatak testira se grupišu u jedan krajnji rezultat koji se prikazuje korisniku.

Rezultat je u ljudski čitljivom formatu, ali je izlazni kod u svakom slučaju, bilo pri pozitivnoj ili negativnoj evaluaciji studentskog rešenja uvek bio 0. Da bi *smoke_test* alat mogao da se upotrebi u okviru GitHub akcije koja proverava ispravnost rešenja koje nastavno osoblje kreira, i da bi se izvršavanje akcije prekinulo u slučaju negativne evaluacije studentskog rešenja, izmenjen je izlazni kod na vrednost 1 ako ponuđeno rešenje ne ispunjava sve uslove navedene u *.yaml* datotekama i ako je prosleđen argument *-e* kao argument komandne linije. Kada GitHub akcija u bilo kojem koraku dobije rezultat različit od 0, smatra se da taj korak nije uspešno izvršen i ne prelazi se na naredne korake. U slučaju *pjisp-assignment-template* alata, to znači da autor zadatka nije dobro napisao primer ispravnog rešenja i da mora da ga ispravi.

3.5 *pjisp-assignment-template*

Alat *pjisp-assignment-template* nudi veliki broj funkcionalnosti za kreiranje i proveru ispravnosti kreiranog zadatka za studente. Ove funkcionalnosti su u velikoj meri učestvovala u dodavanju GitHub tokova poslova čije funkcionalnosti su opisane na samom početku poglavlja. Ipak, neke od funkcionalnosti je trebalo malo izmeniti i bilo je potrebno kreirati nekoliko novih.

Postojeće funkcionalnosti koje alat nudi su:

- *help* - za prikazivanje poruke koja objašnjava ostale funkcionalnosti
- *init* - za generisanje datoteka specifičnih za pojedinačne testove
- *test-solution* - za proveru ispravnosti primera rešenja
- *assignment-clean* - za brisanje svih generisanih datoteka vezanih za studentski zadatak

- *assignment-build* - za generisanje PDF-a sa tekstom zadatka
- *assignment-view* - za prikazivanje PDF-a sa tekstom zadatka
- *assignment-pack* - za pakovanje studentskog zadatka
- *assignment* - za kreiranje, prikazivanje i pakovanje studentskog zadatka
- *extract-exams* - za raspakivanje (eng. *extract*) studentskih zadataka iz ispitnih datoteka
- *examine* - za pregledanje studentskog zadatka

Novosti koje uvodi ovaj projekat su:

- *assignment-diff* - za proveru da li su izmenjene sve datoteke koje je potrebno menjati i da li su ostale iste sve datoteke koje se ne smeju menjati
- *assignment-check* - za proveru izmena datoteka i proveru ispravnosti primera rešenja
- *get-template* - za pronalaženje naziva testa na osnovu naziva repozitorijuma i proveru ispravnosti naziva repozitorijuma

Takođe, uvedene su sitne izmene u neke od postojećih funkcionalnosti:

- *init* - dodatno, nakon generisanja datoteka na osnovu izabranog identifikatora testa, kreira se *.template* datoteka u koju se smešta identifikator za kasniju upotrebu
- *test-solution* - dodato je da pre pokretanja funkcionalnosti zahteva da postoji *assignment_solution.c* datoteka
- *assignment-build* - dodato je da pre pokretanja funkcionalnosti zahteva da postoji *assignment_solution.c* datoteka
- *assignment-pack* - dodato je da se pre pakovanja, pored provere ispravnosti primera rešenja, proveru da li su izmenjene sve datoteke koje je potrebno menjati i da li su ostale iste sve datoteke koje se ne smeju menjati

Kao zavisnosti u projektu dodati su *pjisp-template-name* i *pjisp-diff* korišćenjem Pipenv alata. *Get-template* funkcionalnost poziva *pjisp-template-name* sa nazivom repozitorijuma kao ulaznim parametrom i identifikatorom testa kao povratnom vrednošću. *Assignment-diff* poziva *pjisp-diff* sa identifikatorom testa kao ulaznim parametrom.

Nakon dodavanja malopre pomenutih izmena u kod *pjisp-assignment-template* alata, uvedene su sve funkcionalnosti potrebne za kreiranje GitHub tokova poslova čije funkcionalnosti su opisane na početku trenutnog poglavlja. Kreirane su dve nove datoteke na mestu koje je predviđeno za pisanje GitHub tokova poslova - *.github/workflows*. Tokovi podataka nazvani su “Project create” i “PJISP assignment” i nalaze se u *init-repo.yml* i *test-solution.yml* datotekama respektivno.

3.5.1 Project create

“Project create” je tok poslova koji se pokreće, kako mu i samo ime kaže, izazvan događajem kreiranja projekta, odnosno nakon pravljenja repozitorijuma na osnovu šablon repozitorijuma *pjisp-assignment-template*. Uloga ovog toka posla je da postavi inicijalno stanje projekta. Na osnovu naziva repozitorijuma, zaključuje se na koji test se repozitorijum odnosi i onda se na osnovu identifikatora testa kreiraju datoteke namenjene tom testu. Identifikator testa može biti T12, T34 ili SOV. U *README.rst* datoteku, koja sadrži sve bitne informacije o projektu i datotekama u njemu, postavlja bedž (eng. *badge*) koji daje informacije o tome da li je zadatak kreiran kako treba, a menja se na osnovu rezultata drugog toka poslova - “PJISP assignment”.

Bedž može biti u jednom od tri stanja:



Slika 3.1: Kada nema status



Slika 3.2: Kada se “PJISP assignment” neuspešno izvrši



Slika 3.3: Kada se “PJISP assignment” uspešno izvrši

Kod toka posla izgleda ovako:

```
name: Project create
on:
  create
jobs:
```

```

build:
runs-on: ubuntu-latest
steps:
- uses: actions/checkout@v2
- name: Set up Python 2.7
  uses: actions/setup-python@v1
  with:
    python-version: 2.7
- name: Install pipenv
  uses: dschep/install-pipenv-action@v1
- name: init_template
  run: |
    REPO=$(echo ${GITHUB_REPOSITORY} | cut -d "/" -f 2)
    pipenv install
    TEMPLATE=$(pipenv run make get-template repo_name=$REPO)
    pipenv run make init template=$TEMPLATE
- name: add_badge
  run: |
    REPO=${GITHUB_REPOSITORY}
    echo "|Actions Status|
    .. |Actions Status| \image:: \
      https://github.com/ \
      $REPO/workflows/PJISP%20assignment/badge.svg
      :alt: CPython build status on GitHub Actions
      :target: https://github.com/$REPO/actions
    " | cat - README.rst > README
    mv README README.rst
- name: push_changes
  run: |
    git config --global user.email "action@github.com"
    git config --global user.name "github"
    git add .
    git commit -m "Init template"
    git push

```

Ovaj tok posla u sebi ima jedan posao koji se sastoji od šest koraka. Posao se pokreće na

Ubuntu operativnom sistemu na najnovijoj verziji. Prvi korak preuzima sadržaj repozitorijuma. Drugi korak instalira Python u verziji 2.7 u okruženju pokretača. Treći korak je zadužen za instaliranje Pipenv-a, alata za upravljanje Python paketima koji *pjisp-assignment-template* koristi. Ova tri koraka izvršavaju već postojeće akcije iz GitHub prodavnice, prva 2 koriste zvanične GitHub-ove akcije, dok treći koristi akciju koju je neki od korisnika napisao i objavio.

Četvrti korak je zadužen za inicijalizaciju datoteka na osnovu naziva repozitorijuma. Ovaj, kao i naredna dva koraka spada u grupu koraka koji ne koriste postojeće akcije, nego navode niz komandi koje je potrebno izvršiti. Jezik komandne linije koji je ovde korišćen je *bash*. Naziv repozitorijuma dobija se iz promenjive `github.repository` koja se preuzima iz konteksta okruženja. Ova promenjiva čuva informaciju o repozitorijumu na kojem se akcija pokrenula u obliku `<username>/<repo_name>`. Da bi se preuzeo naziv repozitorijuma, iz ove promenjive se preuzima samo deo nakon kose crte. Nakon toga se instaliraju zavisnosti korišćenjem Pipenv alata i pokreće se *get-template* funkcionalnost. Ova funkcionalnost proverava ispravnost naziva repozitorijuma i na osnovu njega pronalazi identifikator testa. Ako naziv repozitorijuma nije ispravan, ovde se prekida izvršavanje ovog posla i izvršavanje toka posla se smatra neuspešnim. Ako je naziv repozitorijuma ispravan, poziva se *init* funkcionalnost sa identifikatorom testa kao ulaznim parametrom.

Peti korak služi za kreiranje bedža i njegovo dodavanje u *README.rst* datoteku. Naziv repozitorijuma se ponovo dobavlja iz `github.repository` promenjive konteksta okruženja. Na osnovu njega se kreira bedž koji prati status izvršavanja “PJISP assignment” toka posla i dodaje se na sam početak *README.rst* datoteke.

Poslednji, šesti korak je zadužen da postavi sve novonastale izmene na GitHub repozitorijum. Za tu aktivnost je potrebno podesiti email i korisničko ime korisnika koji postavlja ove izmene. U tu svrhu se koristi predefinisani korisnik radi razlikovanja sadržaja koji je pravi korisnik postavio i onog koji je GitHub akcija kreirala. Nakon postavljanja ovih izmena, sve nove datoteke nalaze se na GitHub repozitorijumu i bedž je vidljiv na vrhu *README.rst* datoteke.

Po izvršavanju celog ovog toka poslova, repozitorijum je spreman za preuzimanje i kreiranje studentskog zadatka. Bedž u početku ima *failing* vrednost, pošto zadatak nije spreman za studente i očekuje se od nastavnog osoblja da ga kreira.

3.5.2 PJISP assignment

“PJISP assignment” tok posla se pokreće na događaj push ili pull-request na glavnoj - master grani. Zadužen je da svaki put nakon izmene koda na repozitorijumu proveri da li je repozitorijum spreman za pregledanje od strane nastavnika i davanje studentima u vidu zadatka ili

kolokvijuma.

Kod toka posla izgleda ovako:

```
name: PJISP assignment
on:
  push:
    branches: [ master ]
  pull_request:
    branches: [ master ]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Set up Python 2.7
        if: ${{!contains(github.repository, 'pjisp-assignment-template')}}
        uses: actions/setup-python@v1
        with:
          python-version: 2.7
      - name: Install pipenv
        if: ${{!contains(github.repository, 'pjisp-assignment-template')}}
        uses: dschep/install-pipenv-action@v1
      - name: Run test
        if: ${{!contains(github.repository, 'pjisp-assignment-template')}}
        run: |
          pipenv install
          pipenv run make assignment-check
```

Ovaj tok posla u sebi ima jedan posao koji se sastoji od četiri koraka. Posao se pokreće na Ubuntu operativnom sistemu na najnovijoj verziji. Prvi korak preuzima sadržaj repozitorijuma. Drugi korak instalira Python u verziji 2.7 u okruženju pokretača. Treći korak je zadužen za instaliranje Pipenv-a, alata za upravljanje Python paketima koji *pjisp-assignment-template* koristi. Ova tri koraka izvršavaju već postojeće akcije iz GitHub prodavnice, prva 2 koriste zvanične GitHub-ove akcije, dok treći koristi akciju koju je neki od korisnika napisao i objavio. Četvrti korak ne sadrži ni jednu akciju, već izvršava komande napisane u *bash* jeziku komandne linije.

Svi koraci osim prvog imaju uslov pod kojim se izvršavaju, a to je da u svom nazivu ne sadrže

“pjisp-assignment-template”. Ovaj uslov osigurava se na *pjisp-assignment-template* repozitorijumu neće pokretati ovi koraci, pošto je on samo šablonski repozitorijum i nikada neće sadržati konkretan zadatak za studente, pa samim tim nema smisla testirati njegovu ispravnost.

Poslednji, četvrti korak je zadužen za pokretanje testiranja zadatka. Prvo se instaliraju zavisnosti projekta korišćenjem Pipenv alata, a onda se pokreće *assignment-check* funkcionalnost. Ova funkcionalnost prvo pokreće *assignment-diff*. *Assignment-diff* proverava da li su izmenjene sve datoteke koje potrebno menjati i da li su ostale nepromenjene sve datoteke koje ne bi trebalo menjati. Ako se uspešno izvrši, prelazi se na *test-solution* funkcionalnost. Ako se ne izvrši uspešno, onda se izvršavanje koraka, a zatim i celog toka posla prekida i kreiranje zadatka se smatra neuspešnim. *Test-solution* funkcionalnost ispituje ispravnost primera rešenja na osnovu testova opisanih u *.yaml* datotekama. Ako se barem jedan test ne izvrši ispravno, prekida se izvršavanje koraka, a zatim i celog toka posla i primer rešenja zadatka se smatra neispravnim. Potrebno je da se svi testovi uspešno izvrše da bi se zadatak smatrao ispravnim. Ako se u bilo kojem momentu prekine izvršavanje toka posla, bedž se postavlja na *failing* vrednost. Ako se ceo tok posla uspešno završi, bedž se postavlja na *passing* vrednost, što znači da je zadatak ispravan i spreman za pregledanje ili rešavanje od strane studenata.

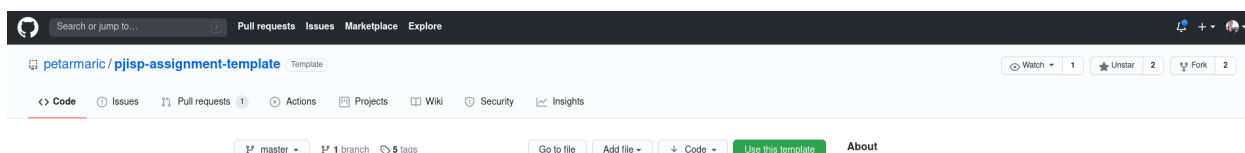
4. Primeri korišćenja

Ovo poglavlje prikazuje primer korišćenja *pjisp-assignment-template* alata. Akcenat je stavljen na akcije koje se izvršavaju prilikom kreiranja ili izmene projekta. U njima se vide funkcionalnosti koje uvodi projekat o kojem ovaj rad govori.

U narednim sekcijama biće prikazan proces kreiranja repozitorijuma, a zatim nekoliko primera kreiranja zadatka. Prvo će biti prikazan ispravan način kreiranja zadatka, a zatim nekoliko primera grešaka koje nastavno osoblje može napraviti, a koje će alati primetiti i omogućiti autoru zadatka da ih otkloni uz ponovnu proveru ispravnosti nakon toga.

4.1 Primer kreiranja repozitorijuma

Za kreiranje repozitorijuma sa zadatkom, potrebno je otići na <https://github.com/petarmaric/pjisp-assignment-template> URL i pritisnuti zeleno dugme - “*Use this template*”



Slika 4.1: Kreiranje repozitorijuma - šablon

Nakon toga se korisniku prikazuje novi prozor, sa formom za kreiranje repozitorijuma iz šablona:

Create a new repository from pjjsp-assignment-template

The new repository will start with the same files and folders as [petarmaric/pjjsp-assignment-template](#).

Owner *

JRubics

Repository name *

pjjsp-2020-E214-T12-G7

Great repository names are short and memorable. Need inspiration? How about **crispy-bassoon**?

Description (optional)

☐ Public

Anyone on the internet can see this repository. You choose who can commit.

☒ Private

You choose who can see and commit to this repository.

☐ Include all branches

Copy all branches from petarmaric/pjjsp-assignment-template and not just master.

Create repository from template

Slika 4.2: Kreiranje repozitorijuma - popunjavanje forme

Od korisnika se očekuje da unese naziv repozitorijuma i da repozitorijum učini privatnim. Nakon pritiska na dugme *“Create repository from template”* započinje generisanje repozitorijuma:

Generating your repository...

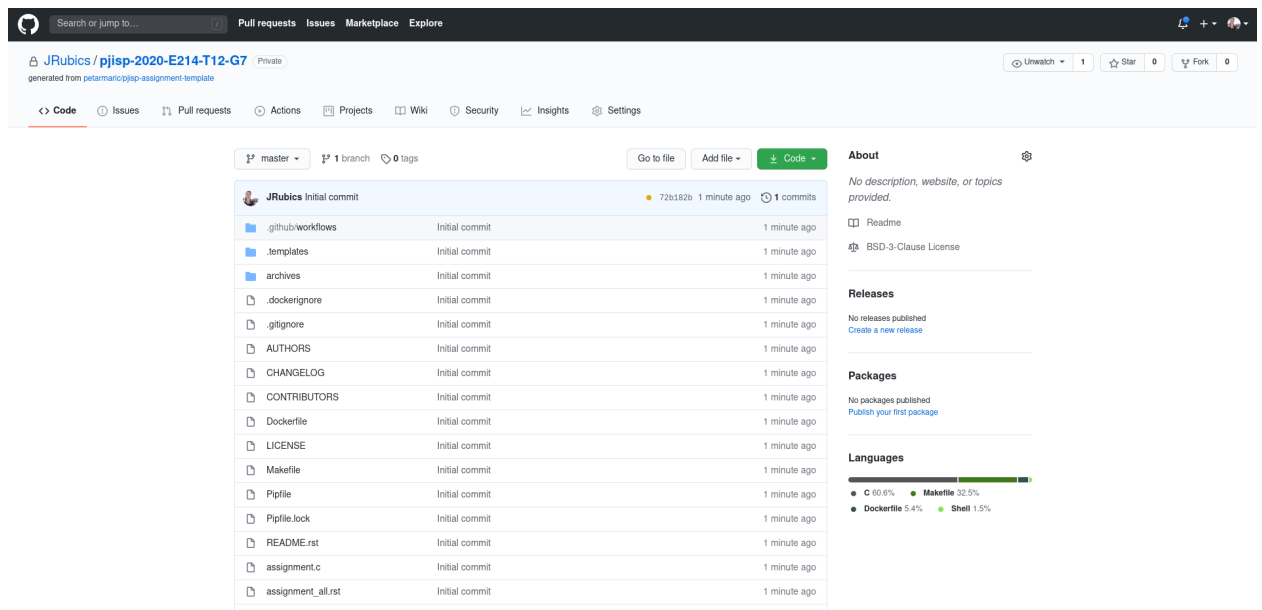
It should only take a few seconds.

Refresh



Slika 4.3: Generisanje repozitorijuma

Po završetku kreiranja repozitorijuma, na korisnikovom nalogu se pojavljuje repozitorijum sa unetim nazivom i svim potrebnim datotekama za kreiranje zadatka za studente:



Slika 4.4: Repozitorijum

Između ostalog, generiše se i *README.rst* datoteka na čijem početku se za sada ne nalazi ni jedan bedž:

README.rst

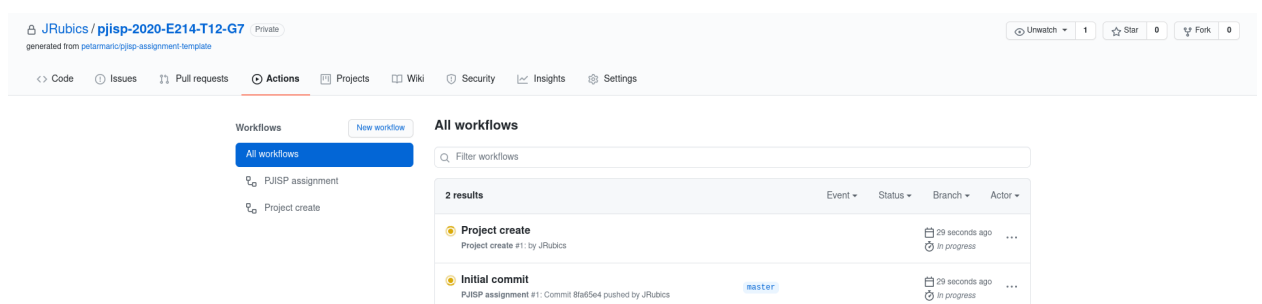
About

An automated workflow for creating assignments for our [ACS](#) students, their deployment and examination.

Slika 4.5: *README.rst* - nema bedž

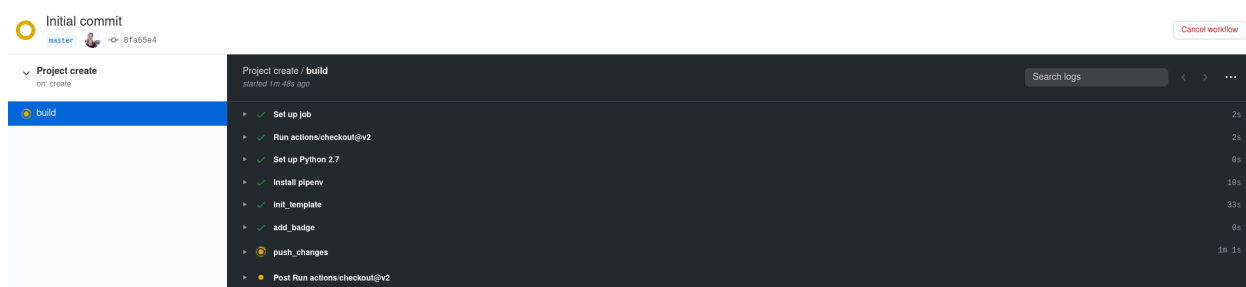
Uporedno sa kreiranjem repozitorijuma, u odeljku *Actions* se pokreću dva GitHub toka poslova:

1. **Project create** - pri kreiranju repozitorijuma
2. **PJISP Assignment** - pri inicijalnoj izmeni repozitorijuma (što je u ovom slučaju kreiranje)



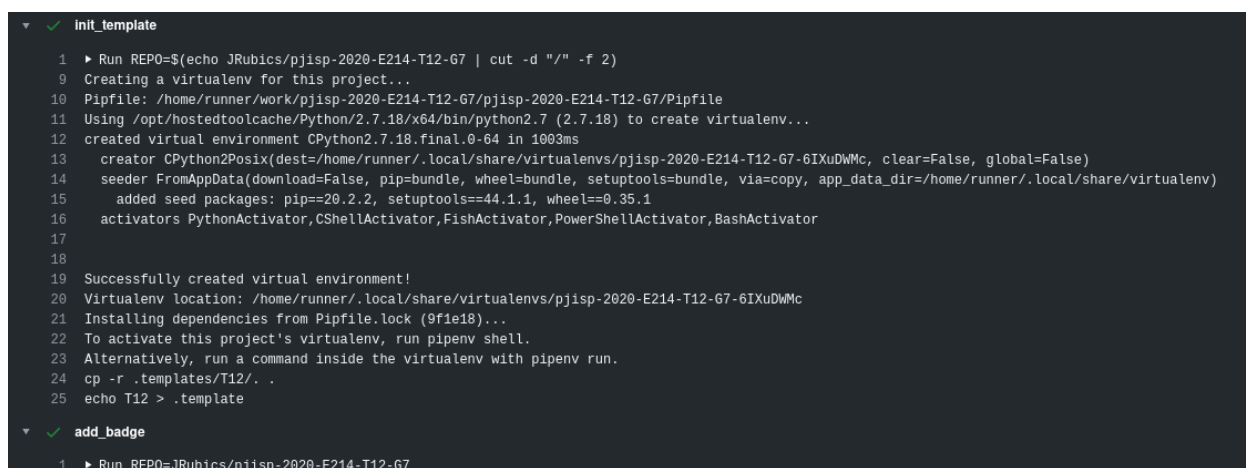
Slika 4.6: Prikaz tokova poslova

Svakom od tokova poslova moguće je pojedinačno pristupiti i pogledati njegov spisak poslova i koraka:



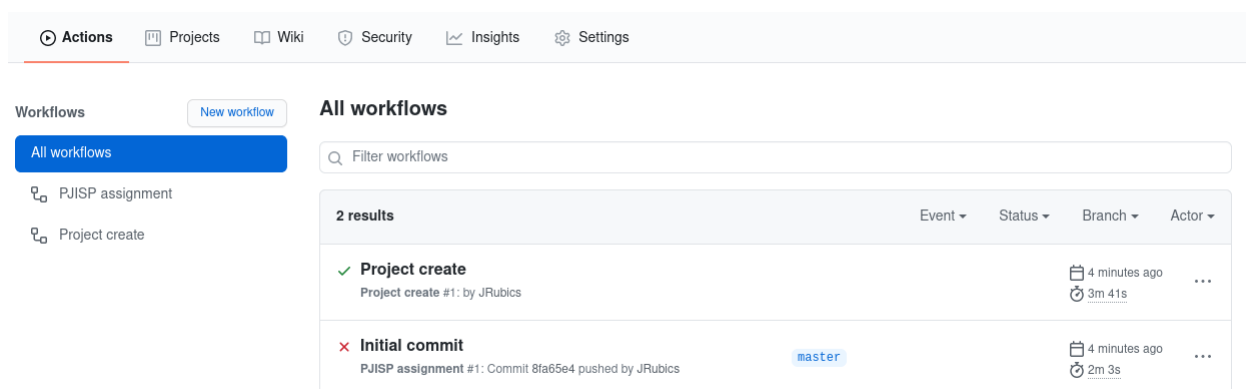
Slika 4.7: Prikaz poslova i koraka

Pritiskom mišem na neki od koraka, prikazuju se detalji njegovog izvršavanja:



Slika 4.8: Detalji izvršavanja koraka

Nakon završetka izvršavanja, odeljak *Actions* će imati jedan uspešno i jedan neuspešno završen tok poslova:

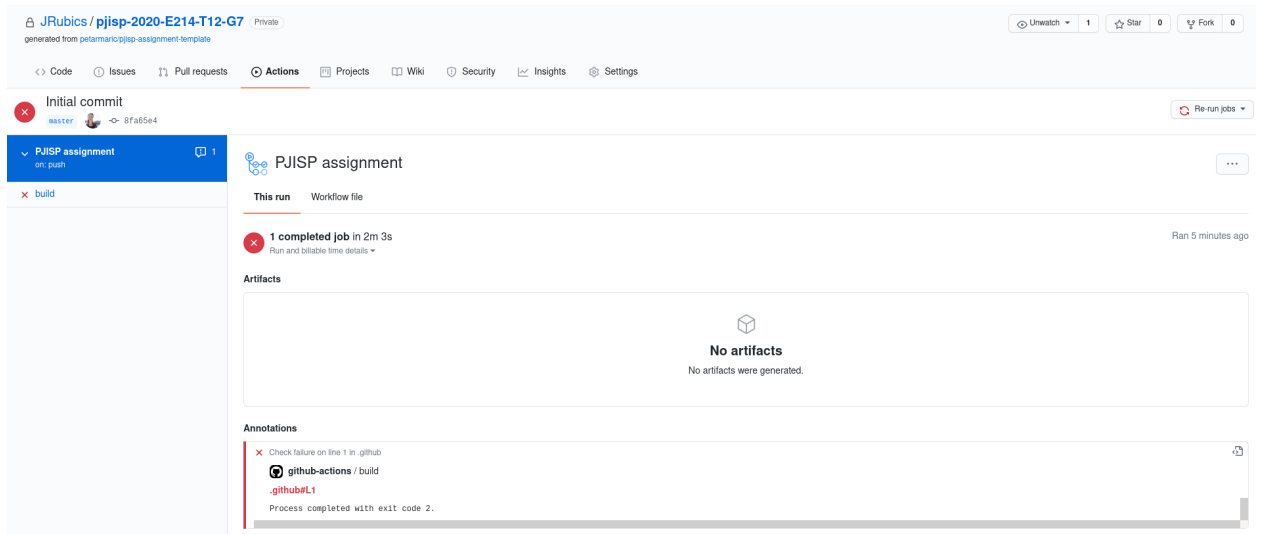


Slika 4.9: Završetak izvršavanja tokova poslova

Uspešno završen tok poslova je onaj koji se pokreće pri kreiranju repozitorijuma. Neuspešno

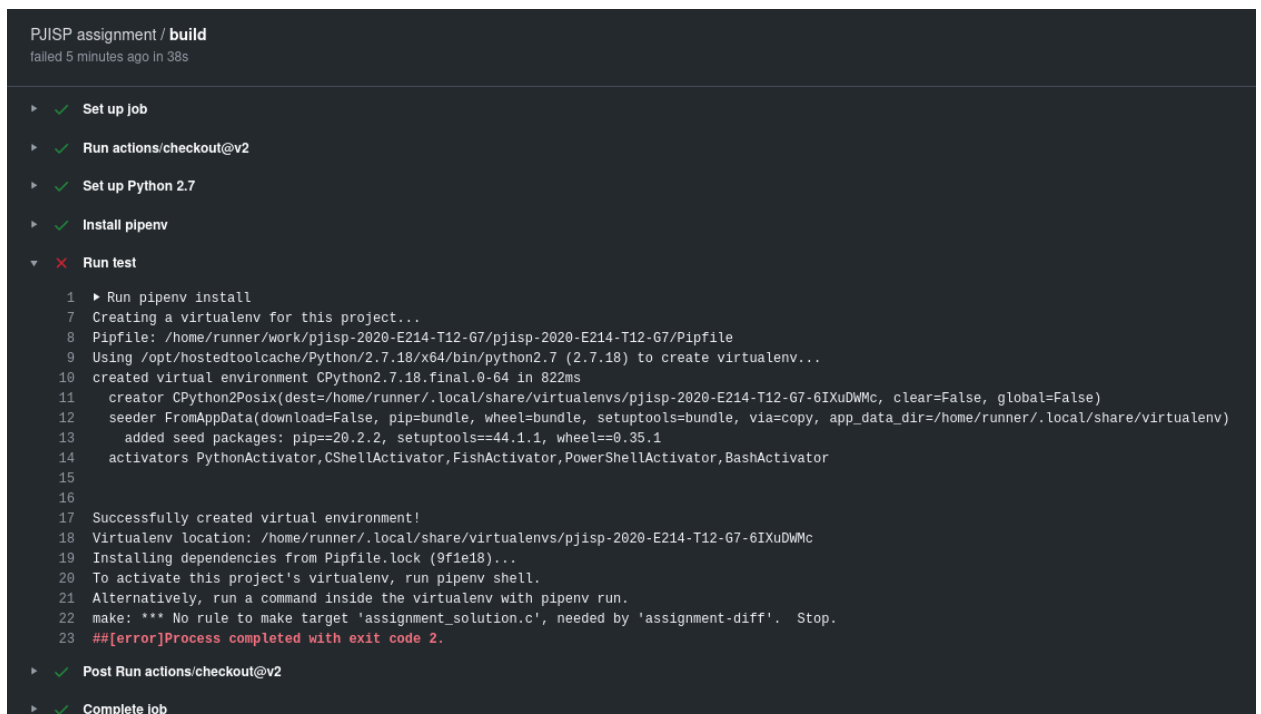
završen tok posla je onaj koji testira ispravnost zadatka. Ovakav rezultat je i očekivan, pošto zaduženi iz nastavnog osoblja još uvek nije kreirao zadatak.

Odeljak *PJISP Assignment* toka poslova prikazuje koji posao se neuspešno izvršio:



Slika 4.10: “PJISP assignment” - neuspešno izvršavanje posla

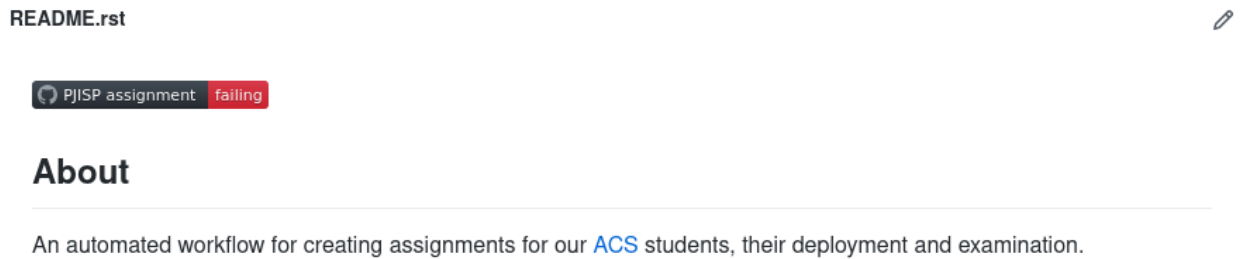
Pritiskom mišem na neuspešno izvršen posao dobijaju se detalji o koraku koji jeouzrokovao neuspešno izvršavanje:



Slika 4.11: “PJISP assignment” - detalji o neuspešnom izvršavanju posla

U ovom slučaju, to je nedostatak *assignment_solution.c* datoteke.

Pri izvršavanju *Project create* toka posla, u *README.rst* datoteku je dodat bedž na vrh. On za sada ima *failing* vrednost zbog neuspešnog izvršavanja *PJISP Assignment* toka poslova:



Slika 4.12: *README.rst* - failing bedž

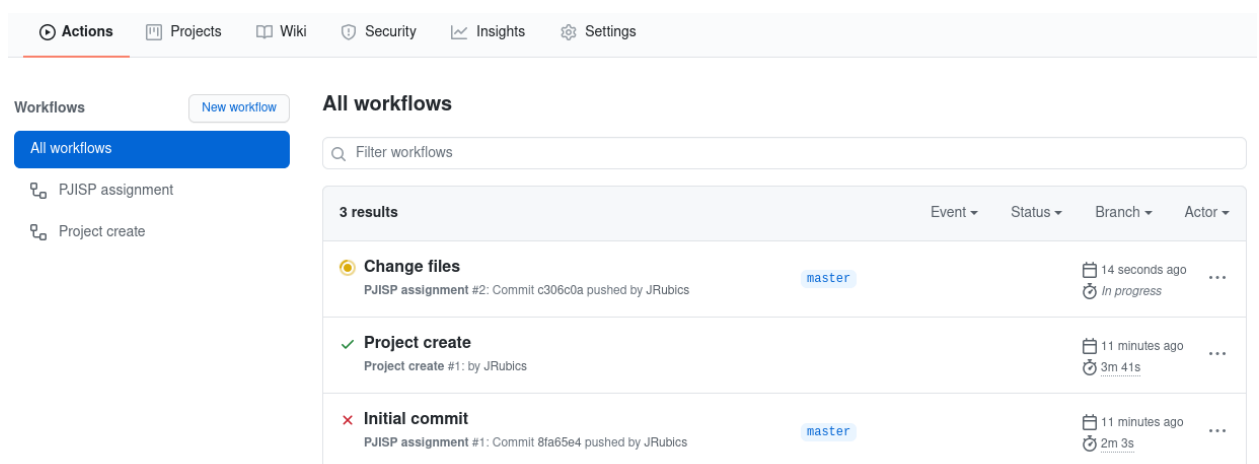
Dodate su i datoteke specifične za T12 test:

- *assignment_solution.c*
- *assignment.rst*
- *assignment_notes.rst*
- *fixtures/stdio-numbers.yaml*

Ovim je završeno izvršavanje tokova poslova koji se pokreću pri kreiranju repozitorijuma, i on je spreman za pravljenje zadatka.

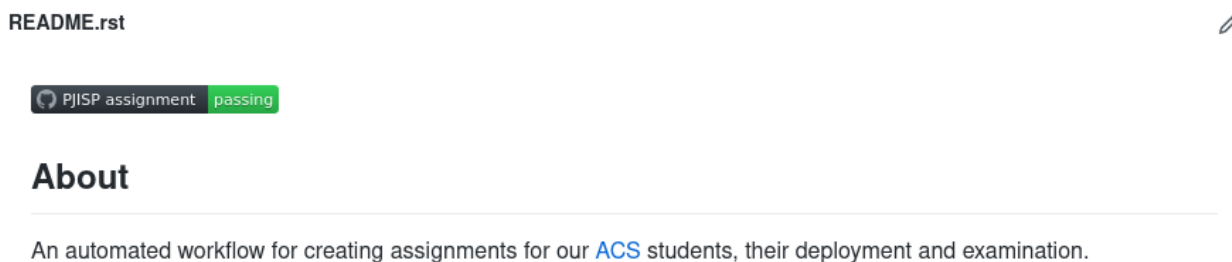
4.2 Primer ispravnog kreiranja zadatka

Kada autor zadatka izmeni sve potrebne datoteke i doda novu izmenu na GitHub, ponovo se pokreće *PJISP Assignment* tok poslova:



Slika 4.13: Prikaz tokova poslova

Pri uspešnom izvršavanju ovog toka poslova, bedž na početku *README.rst* datoteke dobija *passing* vrednost:



Slika 4.14: *README.rst* - passing bedž

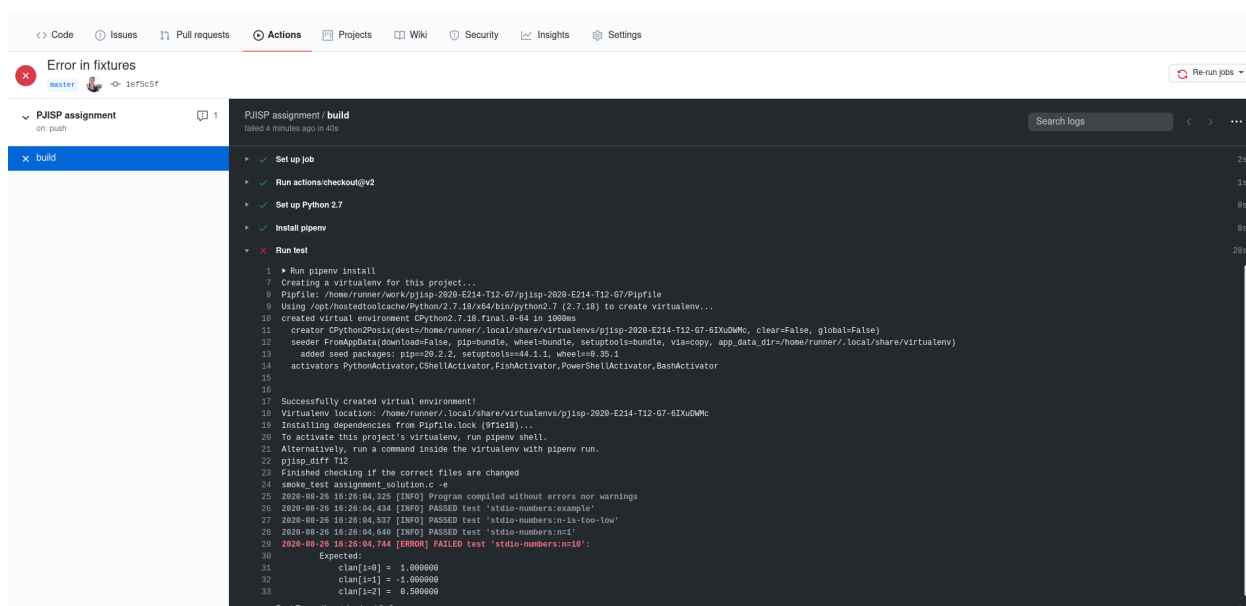
Ovim se uspešno završava kreiranje zadatka i on je spreman za pregledanje od strane nekog od nastavnika.

4.3 Primeri grešaka

Ova sekcija prikazuje po jedan primer za svaku od čestih grešaka koje nastavno osoblje može da napravi. Bilo koja slična greška, izazvaće isto ponašanje programa.

4.3.1 Primer sa neispravnim testovima

Ako se pri ponovnom pokretanju *PJISP Assignment* toka poslova neki od testova ne izvrši uspešno, prikaz detalja o poslu će napisati razlog neuspešnog izvršavanja testa, a samim tim i toka posla:

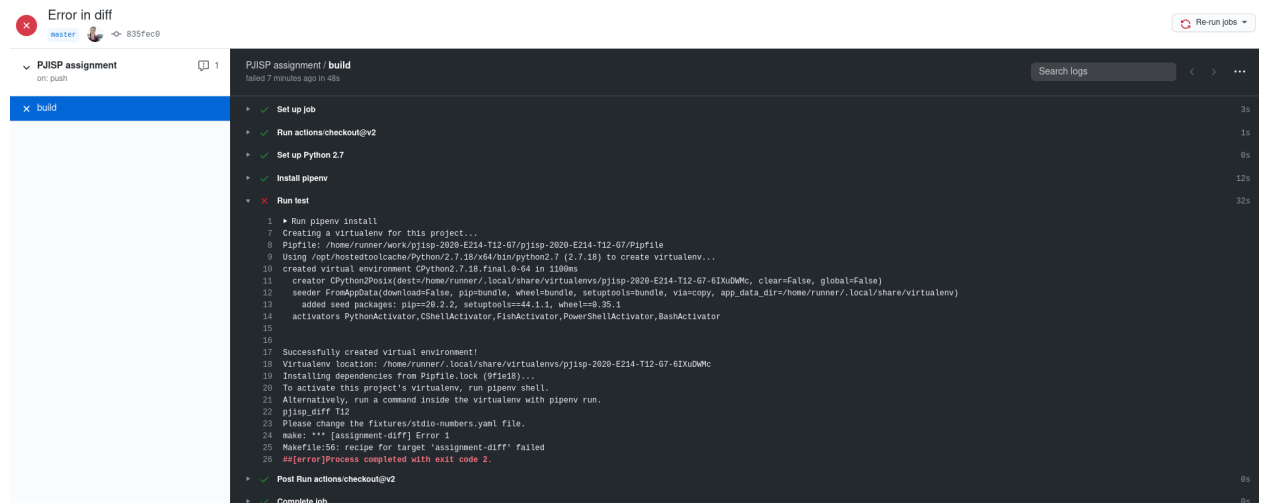


Slika 4.15: Kada se neki od testova neuspešno izvrši

U ovom primeru, test `stdio-numbers:n=10` imao je neočekivan izlaz i zato je dalje izvršavanje toka posla obustavljeno. Zbog toga, bedž na početku *README.rst* datoteke dobija *failing* vrednost.

4.3.2 Primer sa neispravnim izmenama datoteka

Ako se pri ponovnom pokretanju *PJISP Assignment* toka poslova neka od datoteka koje je potrebno izmeniti ne izmeni ili se izmeni neka od datoteka koje se ne smeju menjati, prikaz detalja o poslu će obavestiti korisnika koja izmena nije odgovarajuća i obustaviće izvršavanje toka posla:



Slika 4.16: Kada se ne izmeni datoteka koju je potrebno izmeniti

U ovom primeru nije izmenjena datoteka *fixtures/stdio-numbers.yaml* i zbog toga se prikazuje poruka “Please change the fixtures/stdio-numbers.yaml file.” i vraća se izlazni kod 1, a zatim se tok poslova prekida sa izlaznim kodom 2. Zbog toga, bedž na početku *README.rst* datoteke dobija *failing* vrednost.


4.3.3 Primer sa neispravnim nazivom repozitorijuma

Pri kreiranju repozitorija, korisnik je u mogućnosti da unese neispravan naziv repozitorijuma i nakon toga pokrene kreiranje repozitorijuma:

Create a new repository from pjjsp-assignment-template

The new repository will start with the same files and folders as [petarmaric/pjjsp-assignment-template](#).

Owner * **Repository name ***

 JRubics / test ✓

Great repository names are short and memorable. Need inspiration? How about **animated-journey**?

Description (optional)

☐ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☒ **Private**
You choose who can see and commit to this repository.

☐ **Include all branches**
Copy all branches from petarmaric/pjjsp-assignment-template and not just master.

Create repository from template

Slika 4.17: Kreiranje repozitorijuma sa neispravnim nazivom

U tom slučaju će se *Project create* tok podataka neuspešno izvršiti:

JRubics / test Private
generated from petarmaric/pjjsp-assignment-template

Unwatch 1 Star 0 Fork 0

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Workflows [New workflow](#)

All workflows

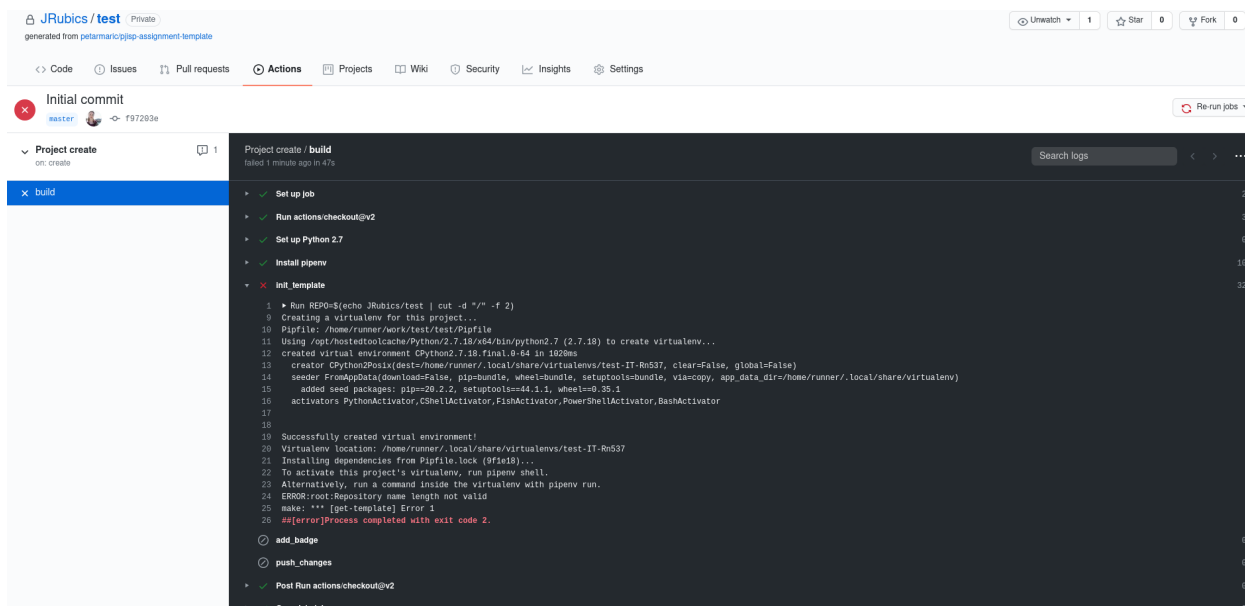
Filter workflows

2 results

	Event	Status	Branch	Actor
Project create	Project create #1: by JRubics			
Initial commit	PJJSP assignment #1: Commit 6099eca pushed by JRubics		master	

Slika 4.18: Prikaz tokova poslova

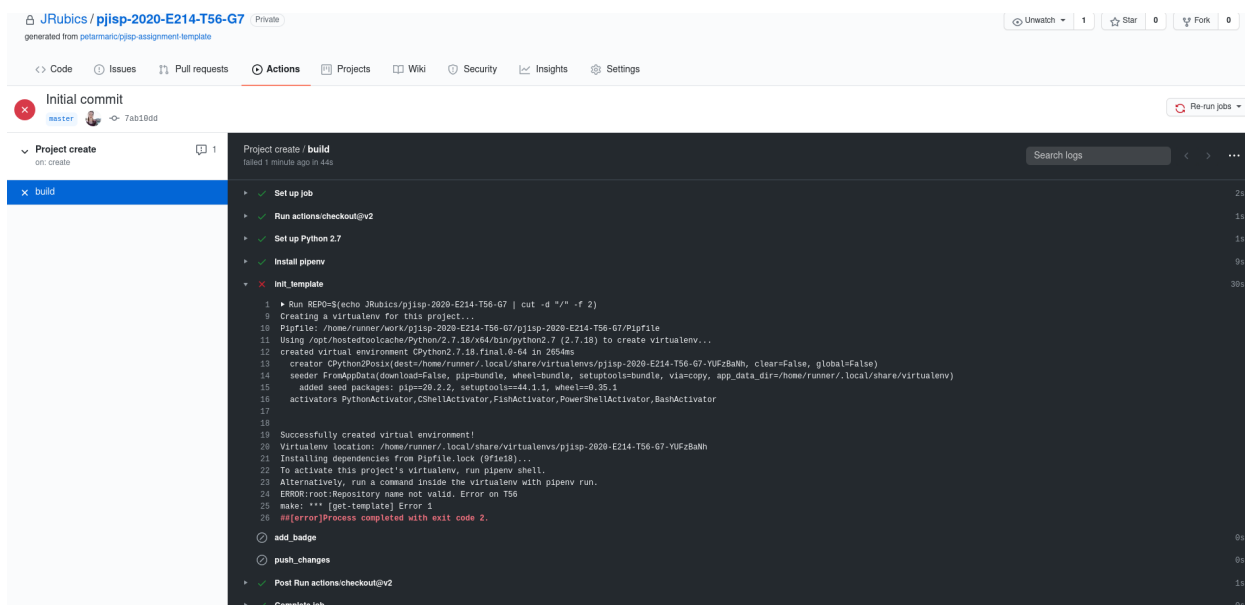
Prikaz detalja o koracima će dati korisniku razlog neuspešnog izvršavanja:



Slika 4.19: Kada dužina naziva repozitorijuma nije odgovarajuća

U ovom slučaju, to je neodgovarajuća dužina naziva repozitorijuma. Zbog toga, izvršavanje ovog toka posla se zaustavlja i preostala dva koraka posla u kojem je došlo do problema se neće izvršiti.

Ako je ipak dužina naziva dobra, ali neki od delova naziva ne odgovara šablonu davanja naziva repozitorijumu, ispisuje se poruka o tome i obustavlja se izvršavanje toka posla:



Slika 4.20: Kada naziv repozitorijuma nije odgovarajući

U ovom slučaju, problem je u tome što je kao naziv testa navedeno T56, iako su T12, T34 ili SOV jedine moguće opcije. Zbog toga korisnik dobija poruku “*Repository name not valid. Error on T56*”.

5. Diskusija i zaključci

U ovom radu je predstavljen celokupan proces razvoja i unapređenja projekata otvorenog koda. Ideja je krenula od postojećeg projekta - *pjisp-assignment-template* i u njega su se dodavale dodatne funkcionalnosti (*pjisp-diff* i *pjisp-template-name*). Svaka od tih funkcionalnosti je projekat za sebe, koji je odvojeno razvijan, a zatim integrisan sa malopre pomenutim projektom upotrebom GitHub tokova poslova i akcija. Jedan od postojećih Github akcija - *poetry-publish* takođe igra važnu ulogu u razvoju projekata koji su nastali kao unapređenje *pjisp-assignment-template* alata.

5.1 Zaključci nakon upotrebe GitHub akcija

Iz prethodnog pasusa vidimo da se mnoge nove funkcionalnosti u malopre pomenutom projektu oslanjaju na funkcionalnosti koje GitHub Actions servis nudi. Ovaj servis je dosta mlad, nastao je u novembru 2019. godine i to sa sobom donosi neke probleme. Još uvek nije u potpunosti pouzdan, pošto su neke od njegovih funkcionalnosti povremeno nedostupne [15]. U trenutku pisanja ovog rada, funkcionalnosti su bile uglavnom dostupne:



Slika 5.1: GitHub status, Avgust 2020

Svaki podeok na vremenskoj liniji (eng. *timeline*) predstavlja jedan dan od proteklih 90 dana. Žute linije predstavljaju dane kada Akcije nisu bile dostupne manje od jednog sata. Narandžasta boja označava nedostupnost funkcionalnosti između jednog i dva sata, a crvena više od 2 sata.

Takođe, zbog brzog razvoja softvera, dokumentacija nije u svakom momentu usklađena sa

alatom, pa to nekada može predstavljati problem za programere koji žele da koriste nove funkcionalnosti alata čim se one pojave.

I pored pomenutih problema, za potrebe ovakvog projekta, GitHub akcije donese mnoge beneficije i znatno olakšavaju razvoj i korišćenje svih prethodno pomenutih alata. Integracija alata je bila veoma jednostavna, pošto se šablon repozitorijum već nalazi na GitHub-u i onda nije bilo potrebno uvoditi dodatne alate u proces integracije.

Funkcionalnosti koje ove akcije uvode mogu znatno da smanje učestalost grešaka koje nastavno osoblje pravi prilikom kreiranja zadataka za studente. Olakšavaju i procenu spremnosti zadataka za davanje studentima, pošto bedž na početku *README.rst* datoteke u repozitorijumu zadatka već naznačava da je zadatak prošao sve testove koje alat trenutno pokreće nad njim i da je spreman za pregledanje od strane drugog nastavnika.

5.2 Dalji razvoj projekta

Dalji razvoj ovog projekta bi pre svega obuhvatao praćenje razvoja GitHub akcija i implementaciju novih funkcionalnosti iz njih u projekat. Vremenom se mnoge funkcionalnosti menjaju i dodaju, i samim tim i korisničke akcije treba da se menjaju i poboljšavaju. Time će performanse i pouzdanost ovog projekta rasti.

Još neke stvari koje bi se mogle dodati bi bile dodatne provere prilikom kreiranja repozitorijuma ili dodavanja koda na repozitorijum. Ipak, svaki ovakav dodatak zahteva dodatno vreme izvršavanja toka posla. Zato, treba odrediti pravi odnos između benefita koji dodatne provere donose i vremena koje je potrebno za njihovo izvršavanje.

6. Literatura

- [1] Git. *Git*. URL: <https://git-scm.com>.
- [2] Inc GitHub. *Github*. URL: <https://github.com>.
- [3] Inc GitHub. *Github marketplace*. URL: <https://github.com/marketplace>.
- [4] Python Software Foundation. *Python documentation*. URL: <https://www.python.org>.
- [5] Python Software Foundation. *The Python Package Index*. URL: <https://pypi.org>.
- [6] Poetry. *Poetry documentation*. URL: <https://python-poetry.org>.
- [7] Python Packaging Authority. *Pipenv*. URL: <https://pipenv.pypa.io/en/latest>.
- [8] Brian Fox. *GNU Bash*. URL: <https://www.gnu.org/software/bash>.
- [9] Petar Marić. *pjisp-assignment-template*. URL: <https://github.com/petarmaric/pjisp-assignment-template>.
- [10] Jelena Dokić. *pjisp-template-name*. URL: <https://github.com/JRubics/pjisp-template-name>.
- [11] Jelena Dokić. *pjisp-diff*. URL: <https://github.com/JRubics/pjisp-diff>.
- [12] Jelena Dokić. *poetry-publish*. URL: <https://github.com/JRubics/poetry-publish>.
- [13] Jelena Dokić. *publish-python-poetry-package*. URL: <https://github.com/marketplace/actions/publish-python-poetry-package>.
- [14] Petar Marić. *smoke_test*. URL: https://github.com/petarmaric/smoke_test.
- [15] Inc GitHub. *GitHub status*. URL: <https://www.githubstatus.com>.