Otvoreno o Zatvorenju

Seminarski rad u okviru kursa
Metodologija stručnog i naučnog rada
Matematički fakultet

N. Grulović, J. Dmitrović, D. Martinović, N. Bogdanović
n.grulovic@outlook.com, jdmitrovic@gmail.com,
dunavpe@gmail.com, nadezdabogdanovic1@gmail.com

6.april 2019.

Sažetak

Clojure (*zatvorenje*) je dinamičan, homoikoničan jezik opšte namene, sa naglaskom na funkcionalno programiranje, koji objedinjuje poželjne osobine različitih programskih jezika. Cilj ovog rada je da približi čitaocu ovaj programski jezik i ukaže mu na oblasti, kao i načine njegove primene.

Ključne reči: *Clojure, funkcionalan, homoikoničan, dinamičan*

Sadržaj

1 Uvod

> „Bolje je imati 100 funkcija koje operišu nad jednom strukturom podataka, nego 10 funkcija koje operišu nad 10 različitih struktura podataka.“
>
> — *Alan J. Perlis*

Kao savremeni dijalekt Lisp programskog jezika, čiji makro-sistem poseduje, Clojure posmatra kod kao podatke nepromenljive strukture. Uspešno kombinuje pristupačnost i interaktivan razvoj, kakav se može susresti kod skriptnog jezika, sa efikasnom i robustnom infrastrukturom višenitnog programiranja.

Uprkos činjenici da je kao jezik kompajliran, on uspeva da očuva potpunu dinamičnost i time omogući da svaka osobenost podržana od strane Clojure-a bude podržana i za vreme izvršavanja. Njegov osnovni interfejs za programiranje, REPL, nas oslobađa ograničenja u smislu kompajliranja i pokretanja izvršnog koda kao jedinih opcija i daje slobodu interaktivnog pisanja programa.

Kao mlad jezik, Clojure je ređe korišćen, ako ne i slabije poznat u programerskim krugovima, uprkos velikim mogućnostima koje pruža. Često je okarakterisan, od strane svojih pristalica, kao zabavan jezik koji dopušta drugačiji pogled na samu logiku programiranja.

Ono što čini Clojure veoma omiljenim i pristupačnim, jeste veliki dijapazon šablona i modularnih biblioteka. Posedujući visok stepen unapred definisane modularnosti, on omogućava da se projekat započne odmah, a istovremeno onemoguće moduli od kojih nema trenutne koristi, čime rezultuje niskom potrošnjom resursa i visokom efikasnošću implementacije.

U ovom radu, čitaocu će najpre biti predstavljena osnovna svojstva jezika i pojašnjenja njegove sintakse, a zatim će, vođen uputstvom za instalaciju i primerom koda biti u mogućnosti da se i sam okuša u programiranju.

2 Razvoj programskog jezika Clojure

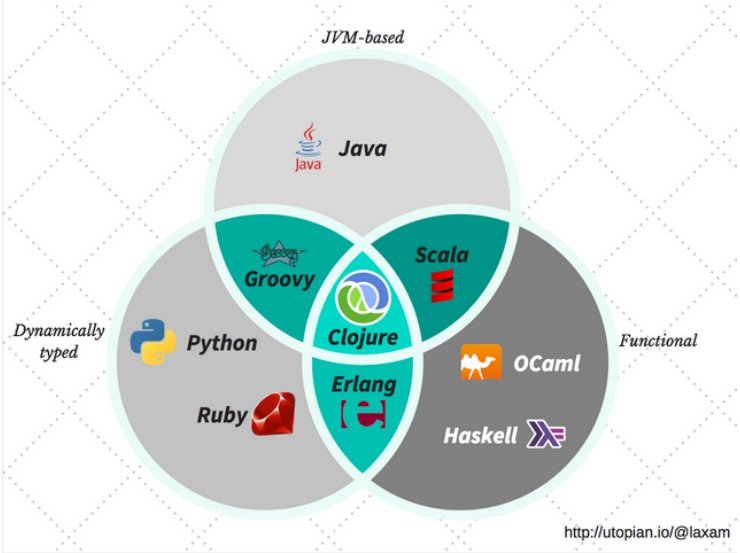
Clojure je razvijen od strane Riča Hikija(engl. *Rich Hickey*)[8], tvorca. Prva verzija je zvanično objavljena 16.10.2007. iako je jezik postojao već dve godine. U tabeli 1, hronološki su predstavljene sve objavljene verzije [8] i njihove glavne odlike.

Clojure je programersku zajednicu obogatio uvođenjem 2 novine: **transduktori** (engl.*transducers*) i **reduktori** (engl.*reducers*).Transduktori predstavljaju kompozitne algoritamske transformacije nezavisne od konteksta svojih ulaznih i izlaznih podataka. Pomoću njih je moguće izvršiti lančanje transformacija usmerenih ka jednom elementu kolekcija, kanala, strimova... „Reduktori su kombinacija reduktorske kolekcije (kolekcija koja ume da se sama redukuje) i reduktorske funkcije (“recept„ koji objašnjava šta je potrebno uraditi tokom redukcije)“[8].

Tabela 1: Objavljene verzije

Verzija	Datum izlaska	Novine
	16.10.2007.	Prvo zvanično objavljivanje
1.0	04.05.2009.	Prva stabilna verzija
1.1	31.12.2009.	Operator `future` za sinhronizaciju
1.2	19.08.2010.	Protokoli
1.3	23.09.2011.	Napredna podrška za primitive
1.4	15.04.2012.	Literali čitača
1.5	01.03.2013.	Upravljanje kolekcijama sa reduktorima
1.6	25.03.2014.	Java API, unapređeni heš algoritmi
1.7	30.06.2015.	Transformacije podataka sa transduktorima
1.8	19.01.2016.	Funkcije za niske, direktno linkovanje
1.9	08.12.2017.	Alati komandne linije
1.10	17.12.2018.	Novi strimovi za REPL, Java kompatibilnost

Primetno je na osnovu slike 1 [4], da je Clojure jezik opšte namene. Ipak njegova priroda čini ga posebno pogodnim za upravljanje i obradu podataka velikog obima. Tako se može koristiti pri radu sa skladištima podataka, u simulacijama, ili prilikom konstruisanja drveta odlučivanja, a trenutno se najviše koristi u veb programiranju. Predviđa se da će uskoro u bioinformatici, gde podaci dostižu i do 75 PB[5], naći sve veću primenu, s obzirom na razvoj kompatibilnih biblioteka za rad sa Amazonovim serverima i platformom Docker[1].

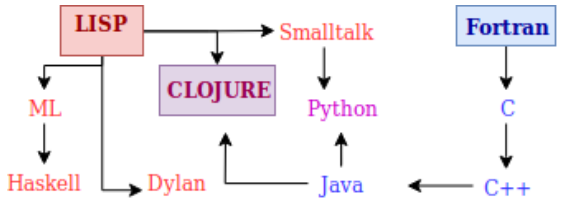


Slika 1: Odnos Clojure-a i drugih programskih jezika

Kao direktan Lisp-ov potomak, smestio se u stablu programskih jezika,

[1] AWS (Amazon Web Server) i Docker kontejneri su često korišćeni u bioinformatici, jer poboljšavaju prostornu i vremensku efikasnost programa i pomažu lakše praćenje procesuiranja i čuvanja podataka. Više o njima se može naći na [3] i [6].

kako je i prikazano na slici 2, kao „mlađi brat“ funkcionalnih jezika kao što su ML, Smalltalk i Dylan.



Slika 2: Clojure u stablu programskih jezika

3 Osnovna svojstva

Clojure je dijalekt programskog jezika Lisp. Drugim rečima, oba jezika počivaju na istim postulatima. U saglasnosti sa tim, Clojure je, pre svega, funkcionalni programski jezik, a pored toga, ima bliskosti i sa konkurentnom i reaktivnom paradigmom[8].

3.1 Platforme

Budući da je Clojure funkcionalni programski jezik, njegova veza sa programskim jezikom Java, koji je predstavnik objektno-orijentisane paradigme, može biti iznenađujuća. Clojure je dizajniran tako da bude tzv. *hosted* programski jezik - za razliku od jezika kao što su C, Python ili Haskell koji prevode svoj izvorni kôd u mašinski ili međukod, Clojure prevodi izvorni kôd u Java bajtkod koji će se potom izvršavati uz pomoć Java Virtuelne Mašine (engl. *Java Virtual Machine*). JVM je izabrana kao standardna platforma za Clojure zbog svoje portabilnosti, sigurnosti, kao i rasprostranjenosti u industriji. Kako poslednjih godina raste naklonost ka funkcionalnim jezicima, tako raste i broj platformi kompatibilnih sa ovim jezikom.

Integrisanost sa programskim jezikom Java omogućuje programskom jeziku Clojure da koristi Java klase, metode i objekte. Rič Hiki je ovu funkcionalnost nazvao *Java Interop*[9]. Moguće je i korišćenje Clojure kôda u okviru programskog jezika Java uz pomoć paketa `clojure.java.api`.

3.2 Clojure kao funkcionalni programski jezik

Kao i u svim funkcionalnim jezicima, u programskom jeziku Clojure se funkcije posmatraju kao *građani prvog reda*; drugim rečima, ne postoji ograničenje kako se funkcije mogu kreirati ili koristiti. Međutim, Clojure nije *čist* funkcionalni jezik, tako da se ne drži strogih pravila o referencijalnoj transparentnosti[7][11].

Jedna od karakteristika koju Clojure deli sa programskim jezikom Lisp jeste **homoikoničnost** (engl. *homoiconicity*). Izvorni kôd homoikoničnih programskih jezika se posmatra kao struktura podataka napisanog u tom istom programskom jeziku. Ovo svojstvo omogućava programima napisanim u programskom jeziku Clojure da manipulišu drugim Clojure programima, kao i da ih generišu.

Jedna od glavnih osobina koja sam programski jezik čini toliko popularnim je *konkurentnost*. Clojure uprošćava višenitno programiranje korišćenjem imutabilnih objekata i STM-a (Softwer Transactional Memory system), koji podržava potrebu za promenljivim stanjima u programu, kao i njihovo sinhrono i asinhrono deljenje među nitima programa [8].

Raspolažući samo sa imutabilnim objektima, Clojure na taj način rešava probleme koje donosi mutabilnost objekata. Budući da to rešenje doprinosi činjenici da se narušavaju performanse nekih operacija na strukturama podataka kao što su vektori i heš mape, te strukture su interno implementirane uz pomoć heš stabala.

Kako lokalne promenljive ne postoje u programskom jeyiku Clojure, petlje se emuliraju uz pomoć (repne) rekurzije. Mnogi funkcionalni programski jezici omogućavaju da repna rekurzija ne koristi stek okvire, time održavajući konstantnu prostornu složenost rekurzivnih poziva; Clojure nema tu mogućnost, zbog načina na koji JVM poziva funkcije. Ovaj problem se prevazilazi korišćenjem operatora `recur`[10].

3.3 Dinamičko programiranje

Prilikom programiranja u programskom jeziku Clojure, nismo ograničeni samo na kompajliranje i pokretanje izvršnog kôda - postoji i mogućnost interaktivnog pisanja programa. Iako se Clojure može ugraditi u Java kôd, osnovni interfejs za programiranje predstavlja **REPL** (***R**ead-**E**val-**P**rint-**L**oop*). REPL je konzolni interfejs gde se napisani Clojure kôd evaluira za svaku napisanu liniju. Ovakav pristup programiranju daje brz odziv na promene u programu što čini neke zadatke jednostavnijim, na primer pronalaženje i uklanjanje bagova.

4 Sintaksna struktura

Kako je Clojure dijalekt programskog jezika Lisp, i sintaksa ta dva jezika je slična. Ono što je karakteristično za ovakve jezike jeste njihova sintaksna *uniformnost*.

4.1 Literali i operatori

Literali su najjednostavniji izrazi - to su oni izrazi koji se evaluiraju u sami sebe. Neki od osnovnih literala u programskom jeziku Clojure su:

- Celi brojevi
- Brojevi u pokretnom zarezu
- Razlomljeni brojevi
- Niske
- Ključne reči

Brojevi se predstavljaju na način koji je standardan u programskim jezicima, niske se predstavljaju sa dvostrukim navodnicima (jednostruki nisu dozvoljeni), dok su ključne reči obeležene sa dvotačkom ispred reči (ali reč je navedena *bez* navodnika). Ključne reči se koriste za referisanje, na primer prilikom dohvatanja elemenata mape.

Opisan je binarni, ali se na analogan način koristi i n-arni operator. Primetimo da su zagrade *obavezne*, kao i da se operandi razdvajaju blanko karakterom (zapeta se takođe smatra blanko karakterom). Clojure svoju

uniformnost održava onemogućavajući operatorima da se navode na drugačiji način. Na primer, uobičajena infiksna notacija aritmetičkih operatora nije omogućena u programskom jeziku Clojure.

Neki od operatora su dati u narednom listingu:

```
;; Aritmeticki operatori (+, -, *, /)
(+ 12 8)
;; => 20

;; Relacioni operatori (=, not=, >, <, >=, <=)
(not= 1 2)
;; => true

;; Logicki operatori (and, or, not)
(and true false true)
;; => false
```

Listing 1: Aritmetički, relacioni i logički operatori

Dodeljivanje imena vrednostima se vrši pomoću operatora `def`:

```
(def ime_konstante operand_1)
```

U drugim programskim jezicima, vrednosti se dodeljuju promenljivama. To nije slučaj u programskom jeziku Clojure, zbog njegove imutabilne prirode.

4.2 Kontrola toka

U programskom jeziku Clojure nisu izostali mehanizmi kontrole toka. Za naredbu grananja, koristi se operator `if`:

```
(if true
    "If grana"
    "Else grana")
;; => "If grana"

(if true
    (do (println "Marko")
        "Petar"))
;; Marko
;; => "Petar"

(if nil
    "Nece biti povratna vrednost"
    "Povratna vrednost")
;; => "Povratna vrednost"
```

Listing 2: Operatori `if` i `do`

Operator `if`, ukoliko je vrednost prvog operanda `true` vraća vrednost drugog operanda; inače, vraća vrednost trećeg operanda. *Else* grana u okviru operatora `if` nije obavezna.

Treba napomenuti da se sve vrednosti tretiraju kao tačne, osim literala `nil` i `false`. Operator `and` vraća poslednju tačnu vrednost ukoliko su svi operandi tačni; inače vraća prvu netačnu vrednost. Analogno, operator `or` vraća poslednju netačnu vrednost ukoliko su svi operandi netačni; inače, vraća se vrednost prvog tačnog operanda.

Operator `do` omogućava da se izvrši više operacija u nekoj ili obe grane. Povratna vrednost ovog operatora je vrednost poslednjeg operanda.

4.3 Strukture podataka

U programskom jeziku Clojure je dostupno korišćenje tradicionalnih struktura podataka kao što su liste, vektori, mape i skupovi, prikazani u tabeli 2. Treba napomenuti da elementi ovih struktura podataka ne moraju biti istog tipa.

Tabela 2: Upravljanje strukturama podataka

Struktura	Definicija	Dohvatanje
Liste	`'(element_1 element_2 element_3)`	`nth`
Vektori	`[element_1 element_2 element_3]`	`get`
Skupovi	`#{element_1 element_2 element_3}`	`get`
Mape	`{:prva_kljucna_rec vrednost_1}`	`get`

Dodavanje elemenata u vektore i liste se vrši pomoću operatora `conj`; novi element će biti dodat na početak liste, odnosno na kraj vektora. Za proveru pripadnosti nekog elementa skupu, koristi se operator `contains?`.

4.4 Definisanje funkcija

Korišćenje funkcija je identično korišćenju operatora. Njihovo definisanje se vrši pomoću operatora `defn` na sledeći način:

```
1000 (defn zdravo
         "Vraca se \"Zdravo \" i nadovezuje ime"
1002     [ime]
         (str "Zdravo " ime "!"))
1004
     (zdravo "svete")
1006 ;; => "Zdravo svete!"

1008 (defn zdravo_svima
         [& ljudi]
1010     (map zdravo ljudi))

1012 (zdravo_svima "Ana" "Marija" "Luka")
     ;; => ("Zdravo Ana!" "Zdravo Marija!" "Zdravo Luka!")
```

Listing 3: Definisanje funkcija sa tačnim i proizvoljnim brojem parametara

Prvi operand je ime funkcije (navodi se *bez* navodnika), zatim (opciono) postavljanje deskripcije koja se može dohvatiti operatorom `doc`, treći operand je vektor koji sadrži argumente funkcije i, na kraju, povratna vrednost funkcije.

Primetimo da je moguće definisati funkciju tako da uzima proizvoljan broj parametara; to se postiže korišćenjem simbola `&` koji će poslate parametre smestiti u listu.

4.5 Makroi i navođenje

Definisanje tzv. **makroa** (engl. *macro*) se izvodi na sličan način kao i definisanje funkcija, osim što se u ovom slučaju koristi operator `defmacro`. Razlog postojanja makroa leži u homoikoničnosti programskog jezika Clojure; sav izvorni kôd se prvo prevodi u listu koja se potom evaluira.

Kako je lista i sama konstrukt kojim se može slobodno manipulisati u programskom jeziku Clojure, to znači da je moguće *proširiti* opisnu moć čitavog jezika, što se najlakše može uraditi uz pomoć makroa. Drugim rečima, makroi omogućavaju menjanje kôda pre njegove evaluacije;

kod funkcija, svaki argument mora biti evaluiran pre nego što se funkcija primeni. Ovime je omogućeno, na primer, i korišćenje infiksne notacije:

```
(defmacro infix
  [[operand_1 op operand_2]]
  (list op operand_1 operand_2))

(infix (1 + 2))
;; => 3
```

Listing 4: Definisanje makroa `infix`

Da je definisana funkcija umesto makroa, poziv funkcije `infix` bi izbacio izuzetak, jer bi se pokušalo izračunavanje vrednosti argumenta; kako broj 1 nije validan operator, funkcija ili makro, dolazi do greške.

Pored izbegavanja valuacije argumenata, poželjno je imati mehanizam izbegavanja evaluacije u okviru tela makroa. U programskom jeziku Clojure se taj mehanizam naziva **navođenje** (engl. *quoting*). Postoje dve vrste navođenja: prosto (engl. *simple quoting*), koje se navodi uz pomoć operatora `quote` ili simbola `'`, i sintaksno (engl. *syntax quoting*), za koje se koristi simbol `` ` ``. Sledeći listing će prikazati razliku između prostog i sintaksnog navođenja, kao i izvorni kod logičkog operatora `and` koji je implementiran uz pomoć makroa:

```
(list '+ 1 (inc 1))
;; => (+ 1 2)

`(+ 1 ~(inc 1))
;; => (clojure.core/+ 1 2)

(defmacro and
  ([] true)
  ([x] x)
  ([x & next]
   `(let [and# ~x]
      (if and# (and ~@next) and#))))
```

Listing 5: Navođenje

Korišćenjem simbola `'`, prosleđujemo neevaluirani operator `+` operatoru `list`, koji će napraviti listu od datih argumenata. U drugom slučaju, korišćenjem simbola `` ` `` će se primeniti na sve elemente u listi koja sledi; ali, korišćenjem operatora `~`, moguće je označiti argumente koji će se evaluirati. Budući da se ovde ne koristi operator `list`, dozvoljeno je korišćenje operatora `~`, kao i da je operator sabiranja naveden zajedno sa svojim imenskim prostorom, sintaksno navođenje se češće koristi u praksi. Primetimo da je, u oba slučaja, dobijena lista validan izraz u programskom jeziku Clojure koji se može izračunati.

Makro `and` je definisan tako da prima 0, 1 ili više argumenata: razlog za to je zato što se ovaj makro poziva rekurzivno. U slučaju 0 ili 1 argumenta implementacija je trivijalna; ali, za više argumenata situacija je malo kompleksnija. Prvo, sa `x` je nazvan prvi element liste, a sa `next` je nazvan ostatak liste. Zatim, kreira se konstanta `and#` uz pomoć operatora `let`, da ne bi došlo do višestruke evaluacije argumenta `x`. Na kraju, koristi se grananje koje će vratiti odgovarajuću vrednost. Ali, argument `next` je lista - da bi elementi unutar te liste mogli biti iskorišćeni kao argumenti ovog makroa, oni se moraju „otpakovati“ uz pomoć simbola `~@`.

5 Okruženja

Ono što čini Clojure atraktivnim je veliki spektar šablona i modularnih biblioteka. Dovoljno je univerzalan da radi na gotovo svakoj Java virtualnoj mašini, dok je dinamičan do te mere da nudi širok spektar funkcionalnosti.

Clojure najveću primenu ima u vebu, pa sa tim visok broj okruženja je vezan za sam veb.

Neki od poznatijih okruženja za Clojure su:[13]

- Luminus
- Hoplon
- Pedestal

5.1 Luminus

Luminus je mikro-okruženje(minimalan skup funkcionalnosti potrebnih za razvijanje aplikacija) čiji je cilj da obezbedi robustnu, skalabilnu i jednostavnu platformu.

Lumius funkcioniše kao vrsta šablonskong sistema, obezbeđujući ugrađene razvojne sisteme i neke podrazumevane module za brzi razvoj.
Okruženje nudi mogućnost da se moduli od kojih nema trenute koristi onemoguće što rezultuje niskom potrošnjom resursa i visokom efikasnošću same implementacije.

Tipičan radni tok pravljenja aplikacije podrzaumeva povezivanje sa REPL sistemom. Luminus omogućava dva načina povezivanja na REPL, preko Boot alata i Leiningen sistema.[14]

5.2 Hoplon

Hoplon je skup Clojure i ClojureScript biblioteka, povezanih zajedno sa Boot alatom, koji ujedinjuju dobre osobine veb platforme i predstavljaju interesantan način dizajniranja i izrade veba sa jedinstvenom stranicom.[1]

Hoplon pruža kompajler za izradu frontend veb aplikacija, i sadrži sledeće biblioteke od kojih zavisi:[12]

1. Javelin - Biblioteka za protok podataka za kontrolu stanja klijenta. Hoplon se čvrsto integriše sa Javelin bibliotekom.
2. Castra - Potpuno opremljena biblioteka za pozivanje udaljenih procedura (engl. *Remote Procedure Call*, RPC) za Clojure i ClojureScript, koje obezbeđuje okruženje servera. (opciono)

Primer prostog HTML elementa generisan pomocu Hoplon okruženja.

```
(div
  :id "great-id"
  :click #(js/alert "foo!")

  (h1 "Hello world!")
  (strong
    (em "Hoplon is pretty sweet!")))
```

Listing 6: Hoplon

```
<div id="great-id" onclick="alert('foo!');">
  <h1>Hello world!</h1>
  <strong><em>Hoplon is pretty sweet!</em></strong>
</div>
```

Listing 7: HTML

5.3 Pedestal

Pedestal je skup bilbioteka pomoću kojih se grade veb servisi i aplikacije. Radi u backendu, opslužuje HTML stranice ili odgovara na API zahteve.

Postoje mnogo alata u ovoj oblasti. Pedestal je napravljen sa dva cilja:

- Primarni cilj je da radi sa API zahtevima.
- Pedestal omogućava pravljenje interaktivnih aplikacija koje moraju da odgovore u veoma kratkom vremenu, čak i dok se odvija komunikacija sa backendom.

Okruženje omogućava pravljenje dinamičnih aplikacijia. On koristi ansihrone mogućnosti programskog jezika Clojure i Java NIO(Java API za I/O operacije niskog nivoa) paketa.[2]

Primer veb servisa koji odgovara sa „Hello, world!“, i vraća status 200(OK).

```
1000 (defn respond-hello [request]
       {:status 200 :body "Hello, world!"})
```

Listing 8: Pedestal

6 Instalacija

Clojure pruža skup alata komandne linije koji se mogu koristiti za pokretanje Clojure REPL-a, upotrebu Java i Clojure biblioteka i pokretanje Clojure programa.

6.1 Linux

1. Obezbediti da su instalirani paketi: `curl`, `rlwrap`, i `java`.
   Instalacija na sistemima koji koriste apt kao sistem za upravljanje paketima:

   ```
   $ sudo apt install curl
   $ sudo apt install rlwrap
   ```

   Ako nije instalirana Java:

   ```
   $ sudo apt install default-jre
   $ sudo apt install openjdk-11-jdk
   ```

2. Preuzeti i instalirati pomoću linux skripta za instalaciju, koji će kreirati fajlove /usr/local/bin/clj, /usr/local/bin/clojure, and /usr/local/lib/clojure:

   ```
   $ curl -O https://download.clojure.org/
   install/linux-install-1.10.0.442.sh
   $ chmod +x linux-install-1.10.0.442.sh
   $ sudo ./linux-install-1.10.0.442.sh
   ```

6.2 Windows

Trenutno postoji samo alfa verzija Clojure-a za Windows. Najpre se sa link-a preuzme poslednja verzija instalacionog fajla. Nakon pokretanja instalacije, biće ponuđene 3 moguće lokacije za instalaciju:

```
Possible install locations:
1) \\Drive\Home\Documents\WindowsPowerShell\Modules
2) C:\ProgramFiles\WindowsPowerShell\Modules
3) C:\WINDOWS\system32\WindowsPowerShell\v1.0\Modules\
Enter number of preferred install location:
```

Treba imati u vidu da pri izboru 1. opcije nisu potrebne admin privilegije, ali se kreira dodatni fajl u `\Documents`, dok za 2. i 3. opciju je potrebno imati admin privilegije.

6.3 Pokretanje

Nakon preuzimanja i instalacije potrebnih alata, REPL se pokreće pomoću komande `clj`:

```
$ clj
Clojure 1.10.0
user=>
```

Prilikom ulaska u REPL, moguće je kucati Clojure izraze i pokretati ih pritiskom na Enter. Postoji veliki broj Clojure i Java biblioteka koje nude razne funkcionalnosti. Često korišćena biblioteka je clj-time koja radi sa datumima i vremenom.

Da bi se koristila ova biblioteka potrebno je napraviti **deps.edn** fajl za deklarisanje zavisnosti:

```
{:deps
   {clj-time {:mvn/version "0.14.2"}}}
```

Za pisanje programa, potrebno je napraviti novi direktorijum i kopirati **deps.edn** fajl u odgovarajući direktorijum.

Komanda `clj` automatski traži izvorne fajlove u **src** direktorijumu pa je potrebno fajl sa ekstenzijom *.clj* sačuvati na putanji `\src\program.clj` i pokrenuti:

```
$ clj -m program
```

Pored instalacije na lokalnom računaru, REPL je moguće koristiti i u veb pregledaču pomoću servisa repl.it. Ova veb stranica omogućuje korišćenje više nezavisnih REPL interfejsa, kao i povezivanje sa Github nalogom.

7 Zaključak

Već nakon prvog upoznavanja sa programskim jezikom Clojure, primećuje se da je kao jezik jednostavan, elegantan i kako zahteva manje kodiranja nego ostali jezici. Obezbeđuje bezbednu manipulaciju konkurentnim

podacima i to na visokom nivou, kao i podršku za standardnu biblioteku za ceo JVM ekosistem. Njegova sintaksa je visoko-proširiva pomoću makroa, a podržava i čitav niz apstrakcija koje potpomažu umnogome skalabilan kod i olakšavaju njegovo refaktorisanje.

Uprkos svojim poželjnim osobinama, Clojure poseduje i određena ograničenja. Nije memorijski efikasan, ne može se koristiti za osetljive servise koji rade u realnom vremenu i njegove poruke o grešci mogu biti teške za dešifrovanje. Inicijalizacija okruženja može biti spora, a i nedostatak eksplicitnih i statičkih tipova otežava testiranje.

Kako je sam jezik još uvek u razvoju, postaje kompatibilan sa sve većim brojem platformi, te se može očekivati proširenje njegovih oblasti primene.Sve češće se, pored svoje standardne primene u veb programiranu, koristi i u sferama poput muzike, videa i bioinformatike. Kako dobija sve više pristalica i postaje sve popularniji programski jezik, očekuje se da Clojure postane dominantan u odnosu na svoje funkcionalne srodnike.

Literatura

[1] Adzerk. Hoplon docs. Online at: http://hoplon.io/.

[2] Adzerk. Pedestal. Online at: http://pedestal.io.

[3] Amazon web service. Online at: https://aws.amazon.com.

[4] Laxam Clojure community. Programming in clojure. part 1: Why clojure. Online at: https://steemit.com/utopian-io/@laxam/programming-in-clojure-part-1-why-clojure.

[5] Charles E. Cook, Ewan Birney, Guy Cochrane, Robert D. Finn, Rolf Apweiler, and Mary Todd Bergman. The European Bioinformatics Institute in 2016: Data growth and integration. *Nucleic Acids Research*, 44(D1):D20–D26, 12 2015.

[6] Docker. Online at: https://www.docker.com/why-docker.

[7] Peter Sestoft Harald Søndergaard. Referential transparency, definiteness and unfoldability. *Acta Informatica*, 1990.

[8] Rich Hickey. Clojure official website. Online at: https://clojure.org/.

[9] Daniel Higginbotham. *Clojure for the Brave and True*. No Starch Press, 2015.

[10] Carin Meier. *Living Clojure*. O'Reilly, 2015.

[11] John C. Mitchell. *Concepts in Programming Languages*. Cambridge University Press, 2002.

[12] Matthew Ratzke. Hoplon wiki. Online at: https://belitsoft.com/laravel-development-services/full-stack-framework-or-microframework-laravel-or-lumen.

[13] Kristopher Sandoval. Frameworks. *10 Frameworks For Building Web Applications In Clojure*, May 2018. https://nordicapis.com/10-frameworks-for-building-web-applications-in-clojure.

[14] Dmitri Sotnikov. Luminus. Online at: http://www.luminusweb.net/.