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: *funkcionalan, homoikoničan, Lisp, dinamičnost, konkurentnost*

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*)[11], tvorca .Lisp-a i drugih Lisp-olikih jezika, pokušaja povezivanja sa Javom. Prva verzija je zvanično objavljena 16.10.2007. iako je jezik postojao već dve godine. U tabeli 1, hronološki su predstavljene sve objavljene verzije i njihove glavne odlike.

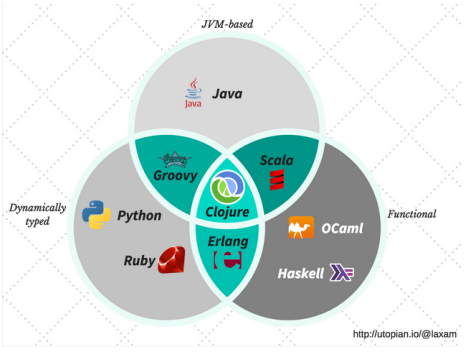
Verzija	Datum izlaska	Novine
	16.10.2007.	Prvo zvanično objavljivanje
1.0	04.05.2009.	Prva stabilna verzija
1.1	31.12.2009.	Operator `future` za sinhronizaciju
1.2	19.08.2010.	Protokoli
1.3	23.09.2011.	Napredna podrška za primitive
1.4	15.04.2012.	Literali čitača
1.5	01.03.2013.	Reduktori
1.6	25.03.2014.	Java API, unapređeni heš algoritmi
1.7	30.06.2015.	Uslovi za čitač, transduktori
1.8	19.01.2016.	Funkcije za niske, direktno linkovanje
1.9	08.12.2017.	Alati komandne linije
1.10	17.12.2018.	Izveštaji o greškama, Java kompatibilnost

Tabela 1: Objavljene verzije[7][8] [11]

Razvoju jezika umnogome je doprinela njegova zajednica, koja je aktivna od trenutka kada se on pojavio. Clojure konferencije, od kojih su najposećenije one u Sjedinjenim državama i evropske konferencije, organizuju se svake godine širom planete.

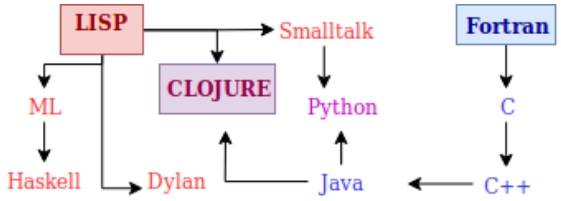
2.1 Odnos sa drugim programskim jezicima

Kao odgovor na pitanje zašto je osmislio još jedan Lisp-oliki jezik, njegov tvorac je odgovorio: „U osnovi, bio mi je potreban Lisp za funkcionalno programiranje, simbiotičan sa osnovnom Java platformom i dizajniran za konkurentnost. Nisam mogao da pronađem takav jezik, pa sam ga stvorio“[11].



Slika 1: Odnos Clojure-a i drugih programskih jezika[9]

Kao direktan Lisp-ov potomak, smestio se u stablu 2 programskih jezika kao „mlađi brat“ funkcionalnih jezika kao što su ML, Smalltalk i Dylan.



Slika 2: Clojure u stablu programskih jezika

3 Osnovna svojstva

Clojure je dijalekat 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[11].

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.

Integrisanost sa Java-om omogućuje Clojure-u i da koristi Java klase, metode i objekte. Ovu funkcionalnost Rič Hiki je nazvao *Java Interop*[12]. Moguće je i korišćenje Clojure kôda u okviru programskog jezika Java uz pomoć paketa `clojure.java.api`.

Kako poslednjih godina raste naklonost ka funkcionalnim jezicima, tako raste i broj platformi kompatibilnih sa ovim jezikom. Neke od implementacija koje koriste alternativne platforme su ClojureScript [6] omogućava integraciju sa JavaScript-om, ClojureCLR[5] sa CLR-om, clojure-py[15] sa Pythonom, Ferret[2] sa C++-om, clojerl[4] sa Erlang-om i Joker[3] interpreter i linter pisan u programskom jeziku Go.

3.2 Clojure kao funkcionalni programski jezik

Kao i u svim funkcionalnim jezicima, u Clojure-u 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[10][14].

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 Clojure-u da manipulišu drugim Clojure programima, kao i da ih generišu.

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 Clojure-u, 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`[13].

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 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.

4.1 Linux

1. Obezbediti da su instalirani paketi: `curl`, `rlwrap`, i `java`. Provera verzija pomenutih paketa se vrši putem komandi:

   ```
   $ curl -V
   $ rlwrap -v
   $ java -version
   ```

   Instalacija:

   ```
   $ 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
```

4.2 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.

4.3 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 web pregledaču pomoću servisa repl.it. Ova web stranica omogućuje korišćenje više nezavisnih REPL interfejsa, kao i povezivanje sa Github nalogom.

5 Sintaksna struktura

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

5.1 Literali i operatori

```
10
;; => 10

10.4
;; => 10.4

"MATF"
;; => "MATF"

:programming
;; => :programming

2/7
;; => 2/7
```

Listing 1: Literali

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.

Literali se koriste zajedno uz pomoć **operatora** - ugrađenih funkcija. Korišćenje operatora (i drugih definisanih funkcija) se obavlja na sledeći način:

```
(operator operand_1 operand_2)
```

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 2: Aritmetički, relacioni i logički operatori

Operator `/` se različito ponaša u zavisnosti od operanada: ukoliko su oba operanda celi brojevi, onda je rezultat izračunavanja tog izraza razlomljeni broj ili ceo broj (ako je prvi operand deljiv drugim). Ukoliko je neki od operanada broj u pokretnom zarezu, onda je i rezultat broj u pokretnom zarezu.

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 Clojure-u, zbog njegove imutabilne prirode.

5.2 Kontrola toka

U Clojure-u 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 3: 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.

5.3 Definisanje funkcija

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

```
(defn zdravo
    "Vraca se \"Zdravo \" i nadovezuje ime"
    [ime]
    (str "Zdravo " ime "!"))

(hello "svete")
; => "Zdravo svete!"

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

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

Listing 4: 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.

5.4 Strukture podataka

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

Liste su definisane na sledeći način:

```
'(element_1 element_2 element_3)
```

Na sličan način se definišu i vektori, ali se koriste simboli `[` i `]`. Operatorom `nth` se može dohvatiti n-ti element liste, dok se ista stvar za vektore postiže korišćenjem operatora `get`. Dodavanje novih elemenata u listu ili vektor se vrši pomoću operatora `conj`; novi element će biti dodat na početak liste, odnosno na kraj vektora.

Mape se definišu ili putem operatora `hash-map` ili uz pomoć vitičastih zagrada:

```
{:prva_kljucna_rec vrednost_1 :druga_kljucna_rec vrednost_2}
```

Pomenuti `get` operator će vratiti vrednost navedene ključne reči.

Skupovi su vektori čiji su svi elementi jedinstveni. Njihova definicija je:

```
#{element_1 element_2 element_3}
```

Naknadna pojavljivanja istih elemenata u okviru istog skupa biće ignorisana. Za proveru pripadnosti neke vrednosti skupu koristi se operator `contains?`.

6 Framework

Ono što čini Clojure atraktivnim je veliki spektar šablona i modularnih biblioteka. Dovoljno je univerzalan da radi na gotovo svakom JVM-u, dok je dinamičan do te mere da nudi širok spektar funkcionalnosti.

Neki od poznatijih framework-a za Clojure su:[17]

- Luminus
- Hoplon
- ClojureHomePage

6.1 Luminus

Luminus mikro-framework je zasnovan na skupu laganih(malih) biblioteka, 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 jumpstart razvoj.[18]

S obzirom da ima mogućnost za jumpstart razvoj, psotoji moduli za kreiranje specifične implementacije. U slučaju da se takvi moduli ne koriste da bi se ubrzao razvoj, moguće je da se pojave dodatni troškovi u budućnosti.

Postoji mogućnost da se projekat započne veoma brzo, a istovremeno onemoguće moduli od kojih nema trenutne koristi. Ovo rezultuje niskom potrošnjom resursa i visokom efikasnošću implementacije i to zahvaljujući tome što je modularnost unapred definisana.[17]

6.2 Hoplon

Hoplon je skup Clojure i ClojureScript biblioteka, povezanih zajedno sa Boot build alatom, koji ujedinjuju neke idiosinkrazije veb platforme i predstavljaju interesantan način dizajniranja i izrade veb stranica sa jednom stranicom.[1]

Hoplon pruža kompajler za izradu frontend veb aplikacija, i sadrži sledeće biblioteke od kojih zavisi:[16]

1. Javelin - Biblioteka za protok podataka za kontrolu stanja klijenta. Hoplon se čvrsto integrira sa Javelinom kako bi reaktivno vezao DOM elemente na grafikonu koji se nalazi u pozadini Javelinove ćelije.
2. Castra - Potpuno opremljena RPC biblioteka za Clojure i ClojureScript, obezbeđujuć i okruženje servera. (opciono)

6.3 ClojureHomePage

ClojureHomePage je jedinstven framework, koji se u potpunosti fokusira na kreiranje veb stranice koja koristi Clojure za front i backend. Podržava širok raspon veb zaglavlja, varijabli okruženja i ruta, i sposoban je generirati prilično složen HTML i CSS. Pored toga, njegova podrška za SKL funkcije znači da, dok se frontend generiše, pozadina se takođe generiše na veoma čvrsto integrisan i moćan način.[17]

7 Zaključak

Već nakon prvog upoznavanja s Clojure-om, 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, podršku za unit i generativne testove, kao i za standardnu biblioteku za ceo JVM ekosistem. Njegova sintaksa je visoko-proširiva pomoću makroa i literala čitača. Podržava čitav niz apstrakcija koje potpomažu u mnogome 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 u odnosu na konkurenciju, 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. U industriji ga koriste Apple, Atlassian, Netflix, Walmart i vladina tela kao što je NASA, a sve češće se koristi i u sferama poput muzike, videa, 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] N. Akkaya. Ferret github repozitorijum. Online at: https://github.com/nakkaya/ferret.

[3] R. Bataev. Joker github repozitorijum. Online at: https://github.com/candid82/joker.

[4] Cojure on the Erlang VM Clojerl. clojerl github repozitorijum. Online at: https://github.com/clojerl/clojerl.

[5] Clojure. Clojureclr github repozitorijum. Online at: https://github.com/clojure/clojurescript.

[6] Clojure. Clojurescript github repozitorijum. Online at: https://github.com/clojure/clojurescript.

[7] Cojure community. Clojure blog. Online at: http://clojure.blogspot.com/2009/05/clojure-10.html.

[8] Cojure community. Clojure blog. Online at: http://clojure.blogspot.com/2009/12/clojure-11-release.html.

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

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

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

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

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

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

[15] D. Raines. clojure-py github repozitorijum. Online at: https://github.com/drewr/clojure-py.

[16] Matthew Ratzke. Hoplon wiki. Online at: https://github.com/hoplon/hoplon/wiki.

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

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