Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/15BE36Skrbic

SLOJ ZA PERZISTENCIJU PODATAKA U ČISTO FUNKCIONALNOJ PROGRAMSKOJ PARADIGMI

DATA PERSISTENCE LAYER IN A PURELY FUNCTIONAL PROGRAMMING PARADIGM

Milan Škrbić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu je opisana teorija kategorija i njena primena u Haskell programskom jeziku. Pored teorijskog dela prikazana je i konkretna implementacija sloja za perzistenciju prodataka u Instagram klon aplikaciji. Aplikacija je implementirana u IHP radnom okviru.*

Ključne reči: *Funkcionalno programiranje, funkcija, funktor, monada, teorija kategorija, aplikativni funktor, produkt, koprodukt, transformacija*

Abstract – *The paper describes the category theory and its usage in the Haskell programming language. In addition to the theoretical part, a concrete implementation of the layer for data persistence in the Instagram clone application is shown. The application is implemented in the IHP framework.*

Keywords: *Functional programming, function, functor, monad, category theory, applicative functor, product, coproduct, transformation*

1. UVOD

U ovom radu prikazana je primena koncepata iz teorije kategorija u programskom jeziku Haskell, kao i implementacija aplikacije Instagram klona u IHP-u (Integrated Haskell Platform). Implementacija aplikacije se koristi radi prikazivanja primene koncepata teorije kategorija u Haskell programskom jeziku.

2. FUNKCIONALNA PROGRAMSKA PARADIGMA

Jedan od većih problema objektno orijentisanog programiranja jeste to što svaka funkcija može da ima bočne efekte koje je teško pratiti u složenijim sistemima i često se dešava da baš ovaj problem proizvede veliki broj neočekivanih ponašanja u toku izvršavanja programa.

2.1 Razlike funkcionalnog i objektno orijentisanog programiranja

U funkcionalnom programiranju, kako mu ime i sugeriše, funkcija igra veliku ulogu. Za razliku od objektno orijentisanog programiranja, u funkcionalnom programiranju svaka funkcija mora da bude čista (engl. pure function), ili drugačije rečeno matematička funkcija.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Milan Segedinac, vanr. prof.

Objektno orijentisani jezici su dobri kada je prisutan fiksni skup operacija na stvarima, a kako se kod razvija, primarno se dodaju nove stvari. To se može postići dodavanjem novih klasa koje primenjuju postojeće metode, a postojeće klase ostaju nepromenjene.
Funkcionalni jezici su dobri kada je prisutan fiksni skup stvari, a kako se kod razvija, primarno se dodaju nove operacije na postojećim stvarima. To se može postići dodavanjem novih funkcija koje računaju sa postojećim tipovima podataka, a postojeće funkcije ostaju nepromenjene [3].

3. TEORIJA KATEGORIJA I HASKELL

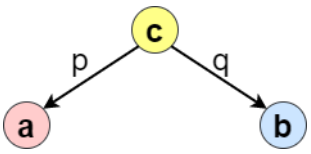
Kategorija se sastoji od objekata i strelica između tih objekata. Objekat u kategoriji je apstraktna stvar i može da se odnosi na razne pojmove. Strelice u kategoriji se drugačije nazivaju morfizmima. Konkretan primer jedne kategorije je kategorija skupova. U kategoriji skupova objekti su skupovi, dok su morfizmi funkcije koje mapiraju elemente jednog skupa na elemente drugog skupa.
Svaka kategorija mora da poštuje dva pravila, pravilo kompozicije i pravilo identiteta. Kompozicija se odnosi na to da se morfizmi spajaju tako da ako postoji strelica *f* od objekta *A* do objekta *B* i još jedna strelica *g* od objekta *B* do objekta *C*, onda mora postojati strelica, njihova kompozicija, koja ide od *A* do *C* koja se označava sa $g \circ f$ i čita se kao *„g posle f“*.
Što se tiče identiteta, u kategoriji svaki objekat mora da ima morfizam koji spaja taj objekat sa samim sobom. To znači da kada je identitet spojen sa bilo kojim drugim morfizmom koji započinje iz tog objekta, ili se završava u tom objektu, daje isti morfizam. Morfizam identiteta za objekat *A* se naziva **id**$_A$.

2.2. Tipovi i Funkcije

Funkcije u Haskellu imaju svoje tipove. Svaka funkcija se deklariše tako što se navode tipovi parametara te funkcije i odvojeni su strelicama. Poslednji tip naveden u deklaraciji funkcije ne predstavlja parametar nego povratnu vrednost funkcije. Funkcija čija deklaracija je *Integer -> Integer -> Bool*, prima 2 parametra tipa *Integer* i vraća povratnu vrednost tipa *Bool*.

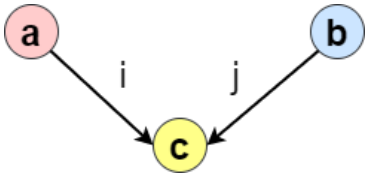
2.3. Produkti i koprodukti

Na slici 1 objekat *c* predstavlja proizvod objekata *a* i *b*. Definitivno je da ima još objekata, *c1, c2, c3...* koji imaju morfizme ka objektima *a* i *b*. Kako da znamo da je baš *c* produkt objekata *a* i *b*? Da bi *c* bio produkt objekata *a* i *b* mora da postoji unikatni morfizam *m* od svih ostalih „lažnih produkata“ ka objektu *c*.



Slika 1. *Produkt [1]*

Na slici 2 vidimo da je koprodukt dva objekta a i b objekat c sa dve injekcije takve da za bilo koji drugi objekat c1,c2,c3... sa dve injekcije postoji jedinstveni morfizam m od c do „lažnog koprodukta“ koji faktorizuje te injekcije.



Slika 2. *Koprodukt [1]*

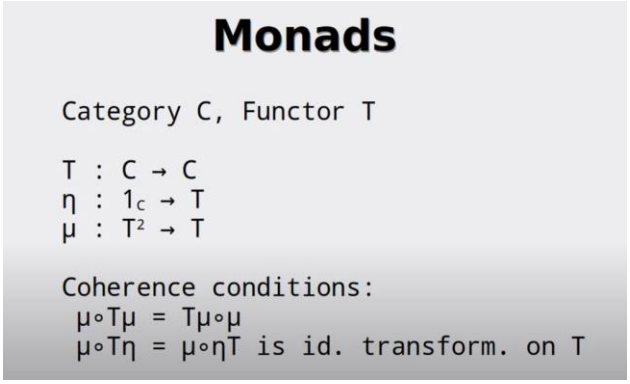
2.4. Funktori

Kada pričamo o **funktorima** želimo da podignemo nivo apstrakcije. Sada više ne gledamo morfizme između objekata u kategoriji, već gledamo morfizme između samih kategorija. Funktori predstavljaju baš to, morfizme između kategorija. Međutim, funktori ne preslikavaju samo objekte jedne kategorije u objekte druge kategorije, nego preslikavaju i morfizme između tih objekata.

2.5. Prirodne Transformacije

Prirodne transformacije (*engl. Natural transformations*) su mapiranja funktora - posebna mapiranja koja čuvaju svoju funkcionalnu prirodu.

2.6. Monade



Slika 3. *Definicija Monade [2]*

Na slici 3 možemo da vidimo konkretnu matematičku definiciju **monade**. Monadu predstavlja kategorija **C** i funktor *T*, koji mapira kategoriju **C** u samu sebe. Funktor koji mapira kategoriju u samu sebe se naziva **endofunktor**. Takođe, monadu čine i dve prirodne transformacije – μ i η, koje su veoma slične prirodnim transformacijama kod monoida. Transformacija η mapira funktor identiteta u funktor *T*, dok μ mapira dve aplikacije funktora *T* u jedan funktor *T*.

Monade kontrolišu lančanja funkcija i to podseća na imperativno programiranje. Međutim ovo nije samo još jedan način imperativnog programiranja, već prava snaga monada se ogleda u tome što nam omogućavaju kontrolisanu kompoziciju funkcija i na taj način rešavaju veliki problem funkcionalnog programiranja – kako rukovanja bočnim efektima.

3. PRIMENA KONCEPATA TEORIJE KATEGORIJA NA PRIMERU INSTAGRAM KLON APLIKACIJE

U ovom poglavlju prikazani su primeri korišćenja koncepata iz teorije kategorija u Haskellu na praktičnom primeru Instagram klon aplikacije, koja je napisana u IHP (Integrated Haskell Platform) radnom okviru. Naziv aplikacije je *„Polaroid“*.

```
39  data User' comments followsFollowers followsUsers likes posts =
40      User {id :: (Id' "users"),
41            email :: Text,
42            passwordHash :: Text,
43            lockedAt :: (Maybe UTCTime),
44            failedLoginAttempts :: Int,
45            firstName :: Text,
46            lastName :: Text,
47            pictureUrl :: (Maybe Text),
48            comments :: comments,
49            followsFollowers :: followsFollowers,
50            followsUsers :: followsUsers,
51            likes :: likes,
52            posts :: posts,
53            meta :: MetaBag
54      } deriving (Eq, Show)
```

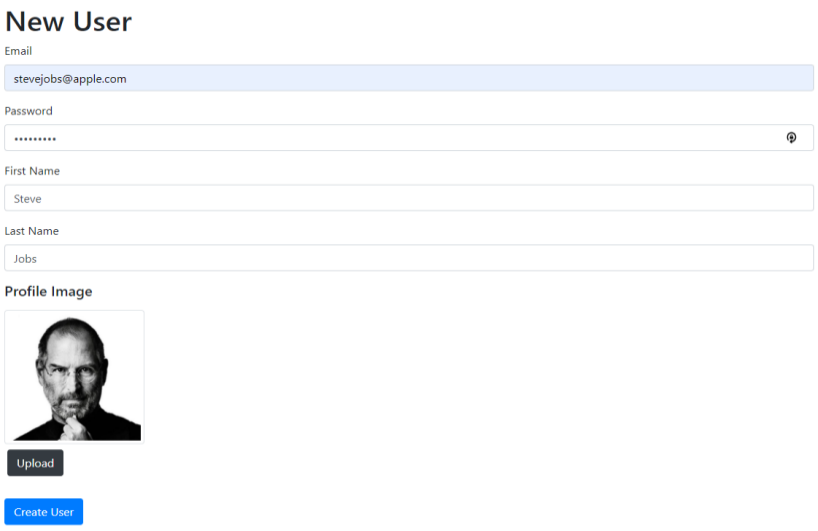
Slika 4. *User tip podataka*

Na slici 4 je dat praktičan primer *User* tipa podataka.
Sloj zadužen za upravljanje i perzistenciju podataka se nalazi u folderu *Controller*.
Na primeru aplikacije veliki broj funkcija je spojen korišćenjem **monada**, tj. „**do**“ notacije i operatora kao što je **(>>=)**.
WelcomeAction je funkcija koja, u zavisnosti od toga da li postoji ulogovani korisnik ili ne, izvlači iz baze podataka preko složenog SQL upita sve objave koje pripadaju svim korisnicima koje ulogovani korisnik prati. Objave su sortirane po vremenu i datumu objave, od najnovije do najstarije. Na sličan način, postavlja se i upit za dobavljanje korisnika koje ulogovani korisnik prati. **Lista** objava i **lista** korisnika se pomoću Haskellove funkcije **zip** spaja u listu koja sadrži ***Tuple*** elemente. Pozivom funkcije **fst** nad elementom liste dobijamo objavu, dok pozivom funkcije **snd** nad elementom liste dobijamo korisnika koji je objavio tu objavu.

4. RAZLIČITI SCENARIJI KORIŠĆENJA INSTAGRAM KLON APLIKACIJE

4.1. Registracija i login korisnika

Na slici 5 prikazana je forma za unos podataka o korisniku. Klikom na dugme „Upload“ korisniku se otvara pretraživač fajlova, gde korisnik može da izabere i postavi svoju profilnu sliku. Klikom na dugme „Create User“ se vrši validacija forme i ako je sve u redu kreira se novi korisnik u bazi podataka. Šifra korisnika se hešuje tako da je nemoguće pročitati je u bazi podataka.

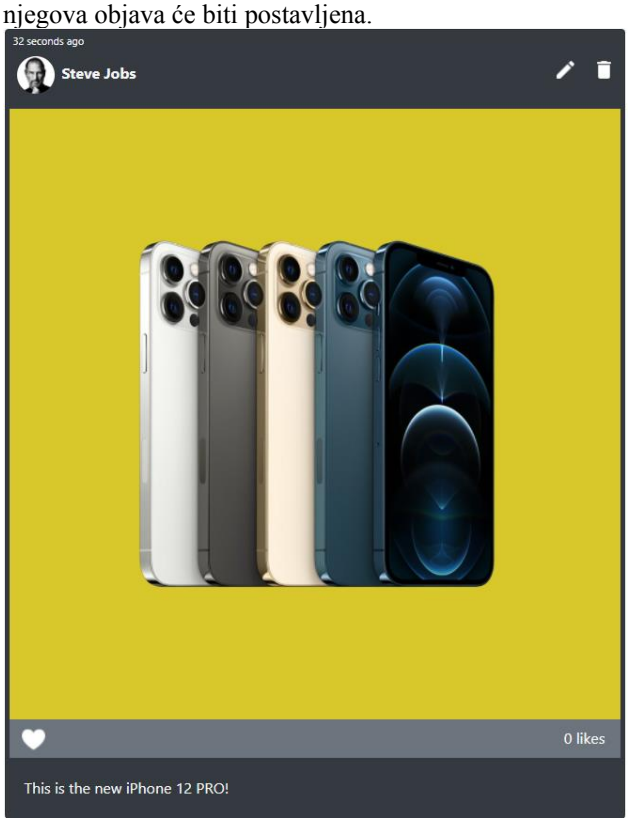


Slika 5. *Registracija korisnika*

Nakon registracije, korisnik je u mogućnosti da se uloguje u svoj nalog. Potrebno je da unese svoju e-mail adresu i šifru koju je uneo prilikom registracije. Klikom na dugme „Login“ korisnik će biti ulogovan.

4.2. Postavljanje objave

Korisnik je u mogućnosti da izabere sliku koju će se postaviti uz pomoć pretraživača fajlova, kao i da napiše željeni opis objave. Klikom na dugme „Create Post“ njegova objava će biti postavljena.

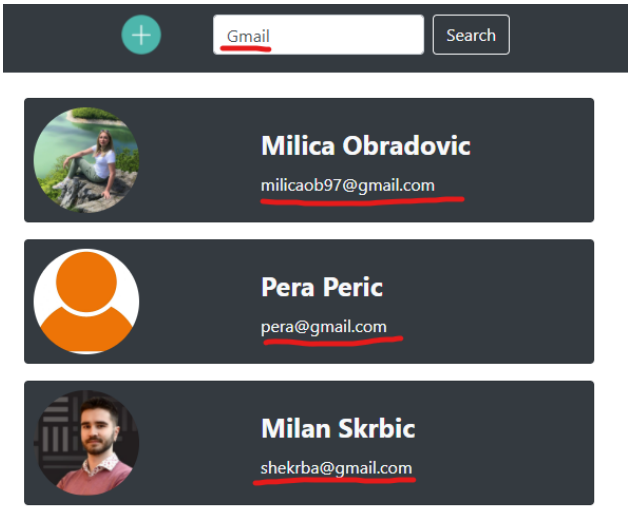


Slika 6. *Objava*

Na slici 6 prikazana je uspešno kreirana objava.

4.3. Prikaz rezultata pretrage korisnika

Moguće je uneti željeni tekst u polje za pretragu korisnika koje se nalazi u navigacionom baru. Nakon klika na „Search“ dugme prikazaće se svi korisnici koji imaju uneti tekst u imenu, prezimenu ili e-mail adresi. Prilikom pretrage korisnika nije bitno da li je uneti tekst za pretragu napisan velikim ili malim slovima (*engl. case-insensitive*).



Slika 7. *Rezultat pretrage*

Na slici 7 prikazan je rezultat pretrage korisnika na osnovu termina „Gmail“.

5. ZAKLJUČAK

Funkcionalno programiranje jeste noviji pristup rešavanju softverskih problema ali je zasnovano na teoriji koja već dugo postoji i sada napokon dobija veliku primenu u softverskom području. Sve veći broj stručnjaka se opredeljuje baš za ovaj pristup programiranju zato što vide veliki potencijal u njemu.

Međutim, funkcionalno programiranje je i dalje u veoma ranoj fazi razvoja i postoji dosta prostora za napredak i razvijanje raznih vrsta softverskih rešenja koje će pospešiti funkcionalno programiranje. Konkretno u IHP radnom okviru nije moguće korišćenje neke druge baze osim integrisane PostgreSQL baze podataka koju IHP pruža, kao ni korišćenje modernijih radnih okvira za klijentski deo aplikacije, kao što su ReactJS, Angular i VueJS.

Iako je ovaj pristup još mlad i stavljen na test, koji će u jednom trenutku dokazati da li ima poente pratiti ga ili ne, veliki broj vodećih softverskih kompanija, kao što su Facebook i Github, počeli su da ulažu u funkcionalno programiranje i određeni deo svojih softverskih rešenja su implementirali baš u Haskellu.

Funkcionalno programiranje i Haskell jesu zasnovani na velikoj apstrakciji koju pruža teorija kategorija. Baš zbog toga je teško programerima koji su navikli na imperativni pristup da pređu na funkcionalno programiranje, ali apstrakcija koju funkcionalno programiranje zahteva je trenutno neophodna da se prevaziđe veliki broj problema sa kojima su se programeri do sad susretali.

6. LITERATURA

[1] Bartosz Milewski, “ Category Theory for Programmers ” https://github.com/hmemcpy/milewski-ctfp-pdf#category-theory-for-programmers

[2] Philipp Hagenlocher, “ Haskell for Imperative Programmers ” https://www.youtube.com/watch?v=Vgu82wiiZ90&list=PLe7Ei6viL6jGp1Rfu0dil1JH1SHk9bgDV

[3] Shaistha Fathima, “ Functional Programming VS Object Oriented Programming (OOP) Which is better….? ” https://medium.com/@shaistha24/functional-programming-vs-object-oriented-programming-oop-which-is-better-82172e53a526

Kratka biografija:

Milan Škrbić rođen je u Novom Sadu 26.02.1996. god. Osnovne studije je završio na Fakultetu tehničkih nauka na smeru Računarstvo i Automatika 2019. god. Master studije je upisao iste godine na Fakultetu tehničkih nauka na smeru Računarstvo i Automatika, modul Elektronsko Poslovanje.

kontakt: shekrba@gmail.com