Sadržaj

Predgovor

Ovim putem želim se zahvaliti svome mentoru dr.sc. Goranu Klepcu koji me je uveo u svijet podatkovne znanosti i pružio mi priliku da radim na ovom radu, svojoj obitelji na velikoj potpori tijekom studiranja, te svim svojim učiteljima i profesorima koji su me vodili kroz moje dosadašnje obrazovanje, pogotovo dr.sc Janu Šnajderu.

Sažetak

Ovaj rad istražuje prednosti i primjenu funkcijske programske paradigme kroz izradu pametne kuharice koja svojim korisnicima predlaže nove recepte ovisno o njihovim prethodno zabilježenim odabirima te nudi mogućnost generiranja tjednog jelovnika i liste za kupovinu koji su optimizirani na način da se postigne maksimalna iskoristivost sastojaka.

Ključne riječi: haskell, optimizacija, strojno učenje, postgresql, internet aplikacija, obrada podataka

1 Uvod

Često se dešava da ne znamo što bi htjeli za ručak. U dućanu obično bivamo inspirirani na licu mjesta pa znamo neplanirano kupiti i više namirnica no što nam treba. Takav pristup obično vodi do određene količine sastojaka koji ostaju neiskorišteni te propadaju u frižideru i na kraju budu bačeni.

Pošto većina ljudi ne želi svoje vrijeme trošiti na planiranje optimalnog ručka i liste za kupovinu kao ni na smišljanje jelovnika, ideja je bila napraviti aplikaciju koja će to odraditi umjesto njih.

Osim same aplikacije, cilj ovog projekta je istražiti primjenu čistog funkcijskog programskog jezika Haskell kao bolje alternative za razvoj softvera.

Ideja je proizašla iz nezadovoljstva nakon višegodišnjeg korištenja objektno orijentiranih programskih jezika te pojavom elemenata funkcijskog programiranja u popularnim programskim jezicima kao što su C# i Java.

Današnje interakcije ljudi i različitih područja ljudskog djelovanja postaju sve složenije te važnost informacijske tehnologije kao glavnog vezivnog tkiva i kanala za razmjenu informacija postaje sve veća.

Iz tog razloga pojavljuju se sve složeniji sustavi i primjene programskih rješenja nad kojima se mora brzo iterirati uz maksimalnu sigurnost i kvalitetu. Takva dinamika zahtjeva efektivne alate za upravljanje kompleksnošću i tu se objektno orijentirani pristup pokazuje sve manje poželjnim izborom, dok se funkcijski pristup čini sve pogodnijim zbog svojih urođenih svojstava koja će biti izložena u nastavku.

2 Haskell

Haskell je funkcijski programski jezik sa mnogo značajki kojima ga se može opisati. Neke od najvažnijih značajki su čistoća, referencijalna transparentnost, lijenost, nepromjenjivost podataka i jak tipski sustav.

2.1 Čistoća

Čistoća je svojstvo jezika koje kaže da funkcija **uvijek** mora vraćati vrijednost te da osim vraćanja te vrijednosti ne smije imati nikakvih sporednih efekata, kao na primjer ispis na ekran, pisanje u bazu podataka ili lansiranje nuklearnih raketa.

Ovo svojstvo je izrazito korisno, pogotovo kod konkurentnog koda gdje postoji mogućnost da koristimo neku funkciju koja vraća određenu vrijednost ali u pozadini mijenja neku varijablu koje mi nismo svjesni što može utjecati na rezultat neke druge funkcije koja također čita i piše u tu istu varijablu.

Koliko god težili korištenju čistih funkcija na kraju je ipak potrebno pisati u bazu podataka ili mijenjati neko stanje. U tu svrhu koriste se monade. One su vrlo elegantno rješenje za ovaj problem te ujedno čuvaju čistoću jezika bez da moramo žrtvovat takve *sporedne* akcije.

Razlika je u tome što monade zahtijevaju da su sve takve sporedne akcije eksplicitne, odnosno da su vidljive iz samog povratnog tipa podatka. Dok u nekom prljavom jeziku možemo imati neku funkciju koja ima tip povratne vrijednosti `Int` ali ujedno mijenja neku globalnu varijablu `String x` u Haskellu takva funkcija mora eksplicitno reći da mutira neko stanje, tako da bi u tom slučaju tip povratne vrijednosti bio `State String Int` gdje `State` jasno daje do znanja da funkcija utječe na neko *stanje* koje je tipa `String` te ima povratnu vrijednost tipa `Int`. Također, prilikom pozivanja takve funkcije moramo predati početno stanje kao jedan od argumenata.

Ovakvim pristupom dali smo puno više korisnih informacija programeru kao i kompajleru koji na osnovu toga mogu donositi puno bolje zaključke o dijelovima koda i njihovom međudjelovanju.

2.2 Referencijalna transparentnost

Referencijalna transparentnost nam govori da čiste funkcije uvijek vraćaju istu vrijednost za iste argumente tako da ukoliko imamo funkcije `f(a, b) = a + b` i `g(a, b) = f(a,b) + f(b,a)` možemo slobodno zamijeniti `f(a,b)` u funkciji `g` sa `a + b + a + b` što kompajler može dalje optimizirati i pretvoriti u `2*a + 2*b`. Dakle, sigurni smo da će ponašanje funkcije ostati isto, samo će se razlikovati u performansama.

2.3 Lijenost

Radi referencijalne transparentnosti, funkcije mogu biti izvršene u bilo kojem trenutku i uvijek dati isti rezultat. Iz tog razloga je moguće odgoditi evaluaciju izraza za kasnije što nam otvara mogućnost rada sa vrlo apstraktnim elementima kao što su beskonačne liste ili *bottom* vrijednosti.

2.4 Nepromjenjivost podataka

U Haskellu ne postoje varijable. Jednom kada smo definirali neku vrijednost ona ostaje zauvijek ne promijenjena. Umjesto mutiranja originalne vrijednosti ili strukture radi se kopija.

2.5 Jak tipski sustav

Haskell ima vrlo jak i izrazit tipski sustav. Postoje razne stvari koje nam omogućava, poput programiranja na razini tipa, automatsko zaključivanje tipova ili čak mogućnost automatske implementacije funkcije prema njenom tipu.

Korisno je pogledati jedan primjer te ga usporediti sa Javom. Usporedbu ćemo provesti na funkciji `misterij`.

```
1  misterij :: a -> a
```

Kod 2.1: Haskell misterij tipski potpis

```
1  <A> A misterij(A a)
```

Kod 2.2: Java misterij tipski potpis

Ovdje su dani samo tipski potpisi. Ako se zapitamo kako bi mogla izgledati implementacija primijetit ćemo da funkcija vraća isti tip podatka koji i prima. Kada shvatimo da ne znamo ništa tom tipu jedina moguća implementacija je da vratimo istu vrijednost koju smo i primili, dakle `misterij` je zapravo funkcija identiteta.

Na žalost, iako to logika nalaže, istu stvar ne možemo tvrditi i za moguću implementaciju u Javi. U Javi `misterij` može vratiti `null`, saznati klasu i napraviti novu instancu, usporediti pokazivač i sl. Drugim riječima, tipovi u Haskellu **ne lažu**.

3 Podatci

Kako bi aplikacija ispravno funkcionirala, potrebno je pribaviti neke osnovne podatke s kojima će baratati.

Bila bi idealna situacija kada bi postojao set podataka sa receptima, njihovim sastojcima, prosječnim cijenama sastojaka i dostupnim količinskim pakovanjima te korisnicima i njihovim najdražim receptima. Iz takvog bi se seta podataka moglo puno naučiti o korisničkim sklonostima prema receptima i sastojcima ali isto tako bi se podatci o cijenama i pakiranjima mogli iskoristiti za optimiziranje tjednog jelovnika.

Takav set podataka ipak ne postoji, te ga je potrebno napraviti. Ispada da postoje relativno praktični izvori podataka koji se mogu iskoristiti, a to su Coolinarka (servis za dijeljenje recepata) i Konzum Klik (konzumov servis za kupovinu preko Interneta).

Podatci o korisničkim sklonostima nisu nigdje javno dostupni, no i da jesu trebalo bi ih povezati sa podatcima o sastojcima i receptima koje smo prikupili iz drugih izvora što nije baš idealno te se sa relativno malim setom podataka zapravo svodi na čisto pogađanje.

Pošto je takva situacija i s obzirom da je fokus ovog rada na primjeni algoritama (ne nužno na realnom setu podataka) odabran pristup ovom problemu je da se naprave simulirani korisnici (botovi) koji će imati sklonost prema određenim atributima recepata i na osnovu toga će te recepte spremati u svoju privatnu kuharicu. Time je riješen problem nedostupnosti podataka o korisničkim sklonostima, samo što u ovom slučaju nećemo predviđati sklonosti ljudi već botova što nije veliki problem a ima i svojih prednosti, kao na primjer jednostavnost testiranja uspješnosti algoritma preporuka.

3.1 Coolinarka

Coolinarka sadrži strukturiranu stranicu sa popisom raznih namirnica koje se koriste u receptima. Svaka od namirnica na popisu ima svoju pod stranicu gdje se nalazi kratak opis i velika slika što su zapravo željene informacije.

Kako bi došli do tih podataka u prikladnom formatu potrebno je definirati strugač za coolinarkin hipertekstualni označni jezik (*HyperText Markup Language*, HTML). Ideja je prvo dohvatiti stranicu sa popisom sastojaka, sastrugati linkove koji vode na detalje o sastojcima te onda sastrugati detalje pojedinih sastojaka i spremiti ih na disk u praktičnom formatu kao što su zarezom odvojene vrijednosti (*Comma Separated Values*, CSV).

U tu svrhu korištena je programska zbirka `scalpel-core` koja pruža jednostavno monadsko sučelje za definiranje strugača.

```haskell
coolinarka :: String
coolinarka = "https://www.coolinarika.com"

type HTML = Text

ingredientURLsScraper :: Scraper HTML [URL]
ingredientURLsScraper = do
  links <- attrs "href" $
    "ul" // "li" // "h3" // "a" @: [match helper]
  maybe (fail "no links") pure $ parse links
  where
    parse = traverse (importURL . (coolinarka ++) . unpack)
    helper atr val = atr == "href" && "/namirnica/"
     ↪ `isPrefixOf` pack val
```

Kod 3.1: Coolinarka - Strugač liste sastojaka

Princip je vrlo sličan selektorima koje koriste kaskadni stilovi (*Cascading Style Sheet*, CSS). U ovom slučaju odabiremo *anchor* tagove koji se nalaze unutar `h3` taga koji je element liste te izvlačimo `href` atribut koji sadrži relativni jedinstveni lokator resursa (*Unique Resource Locator*, URL) sastojka.

Pošto su URL-ovi relativni, potrebno je dodati korijensku adresu web stranice kako bi mogli dohvatiti sadržaj tih podstranica što je učinjeno na način da je `coolinarka` dodana kao prefiks na URL koji je zatim provućen kroz `importURL` funkciju koja provjerava valjanost i vraća podatkovni tip `URL`.

Nakon što su sastrugani URL-ovi potrebno je sastrugati informacije o sastojcima. Za to je definiran novi strugač koji radi na istom principu.

```haskell
ingredientScraper :: Scraper HTML CoolIngredient
ingredientScraper = do
  n <- text $ "div" @: ["id" @= "content_header"] // "h1"
  i <- attr "src" $ "img" @: ["id" @= "content_image"]
  d <- (T.unwords . T.words . strip) <$> text ( "div" @: ["id"
   ↪ @= "content_header"] // "div" @: [hasClass "lead"] )
  pure $ CoolIngredient n ( U.slug n ) ( original i ) d
```

Kod 3.2: Coolinarka - Strugač informacija o sastojku

Ovdje možemo primijetiti da strugač vraća podatkovni tip `CoolIngredient` koji je struktura koja predstavlja sastojak uzet sa Coolinarke te ima slijedeću definiciju.

```haskell
data CoolIngredient = CoolIngredient
  { name :: Text
  , slug :: Text
  , image :: Text
  , description :: Text
  } deriving ( Eq, Show, Generic )

instance ToRecord CoolIngredient
instance FromRecord CoolIngredient
```

Kod 3.3: Coolinarka - `CoolIngredient`

Struktura, osim što definira razna polja za podatke vezane uz pojedini sastojak, derivira i nekoliko klasa. `Eq` klasa omogućuje vršenje provjere jednakosti, `Show` pruža funkciju za serijalizaciju u `String` dok `Generics` daje dodatne mehanizme za automatsku implementaciju instanci klasa koje to podržavaju.

U ovom slučaju iskorišten je `Generics` kako bi mogli automatski instancirali klase `ToRecord` i `FromRecord` koje dolaze iz programske zbirke `cassava` te omogućuju serijalizaciju podataka u CSV.

Nakon struganja svih tih informacija podatci su spremljeni na disk u CSV formatu, uz to slike su spremljene u zasebnu mapu za kasnije korištenje.

Popis kratica

CSS	*Cascading Style Sheet*	kaskadni stil
CSV	*Comma Separated Values*	zarezom odvojene vrijednosti
HTML	*HyperText Markup Language*	hipertekstualni označni jezik
URL	*Unique Resource Locator*	jedinstveni lokator resursa

Popis slika

Popis tablica

Popis kodova