

# Sadržaj

<b>1</b>	<b>Uvod</b>	<b>4</b>
<b>2</b>	<b>Haskell</b>	<b>5</b>
2.1	Čistoća . . . . .	5
2.2	Referencijalna transparentnost . . . . .	5
2.3	Lijenost . . . . .	6
2.4	Nepromjenjivost podataka . . . . .	6
2.5	Jak tipski sustav . . . . .	6
<b>3</b>	<b>Podatci</b>	<b>7</b>
	<b>Popis kratica</b>	<b>8</b>
	<b>Popis slika</b>	<b>9</b>
	<b>Popis tablica</b>	<b>10</b>
	<b>Popis kodova</b>	<b>11</b>

# 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.

Jedinstvani lokator resursa (*Unique Resource Locator*, URL) Hyper Text Markup Language (*Hyper Text Markup Language*, HTML)

## 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 žrtvovati 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 primijetiti 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**

U ovom poglavlju biti će obrađene tehnike prikupljanja i obrade podataka korištenih pri izradi aplikacije. Pošto je svrha aplikacije predviđati

## Popis kratica

**HTML**    *Hyper Text Markup Language*  
**URL**     *Unique Resource Locator*

Hyper Text Markup Language  
jedinstvni lokator resursa



## **Popis slika**

## **Popis tablica**

## Popis kodova

2.1	Haskell misterij tipski potpis . . . . .	6
2.2	Java misterij tipski potpis . . . . .	6