Razvoj web aplikacija u Go

Prekrat, Krunoslav

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Krunoslav Prekrat

Razvoj web aplikacija u GO

ZAVRŠNI RAD

Varaždin, 2019.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Krunoslav Prekrat

Matični broj: 42823/14-R

Studij: Informacijski sustavi

Razvoj web aplikacija u GO

ZAVRŠNI RAD

Mentor/Mentorica:

Prof. dr. sc. Dragutin Kermek

Varaždin, rujan 2019.

Krunoslav Prekrat

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Ovaj završni rad se bavi programskim jezikom Go – jezikom kojeg je 2009. godine kreirala grupa programera u Google-u. Ovaj rad se specifično bavi web komponentom programskog jezika Go, te je primarni fokus rada opis izrade same web aplikacije pomoću ranije spomenutog programskog jezika. Uz sami programski jezik, u radu je prikazana upotreba prikladnog alata za izradu aplikacije, komunikacija aplikacije sa drugim elementima (bazom podataka), te daljnji proces objavljivanja, tj. postavljanja aplikacije na javno dostupan servis.

Kroz ovaj rad spomenuti su koncepti koji omogućuju izradu finalnog proizvoda – same aplikacije, dok su uz one osnovne koncepte spomenuti i neki napredniji. Uz programski jezik Go još su spomenuta dva konkurentska jezika – PHP i C#, s kojima je dana usporedba u performansama i usporedba kompleksnosti izrade aplikacija pomoću tih tehnologija.

Kako bi se što bolje prikazao proces izrade aplikacije pomoću programskog jezika Go, jedno poglavlje ovog rada je posvećeno izradi cijele aplikacije od početka. Kroz ovakav primjer dan je prikaz različitih koncepata koji su objašnjeni kroz cijeli rad, a kroz primjer praktičnog rada svi ranije spomenuti koncepti su detaljno objašnjeni na stvarnom primjeru.

Ključne riječi: go; programiranje; web; vscode; sql; životopisi;

Sadržaj

1. Uvod

Sve od kasnog razdoblja 80-tih godina 20. stoljeća i ranih 90-tih, dolazi do laganog priljeva novih programskih jezika poput Perl-a, Python-a, Ruby-ja, PHP-a te JavaScript-a. U vrlo kratkom roku ovi jezici su popularizirani od strane velikog broja programera koji su prihvatili nove koncepte i krenuli u izradi aplikacija pomoću ranije spomenutih jezika. S takvim rastućim brojem korisnika koji redovito koriste ove jezike, ubrzo su se spomenuti jezici svrstali među najpopularnije programske jezike tog razdoblja. Višedretvenost, spremanje manjih podataka u memoriju i različiti API-ji omogućili su zajednički rad različitih sustava i aplikacija koje do tada nisu bile u mogućnosti raditi zajedno u isto vrijeme. [3]

Sav taj paralelni rad omogućio je brz i efikasan rast web-a, tj. aplikacija koje se nalaze na webu. No uz sav taj napredak, ti jezici nisu pokrili jedno ključno područje koje se danas opet pojavljuje kao atraktivan dio programerskog svijeta – moćni, kompilirani, višeplatformski jezik sa podrškom za višedretvenost (konkurentnost) usmjeren na sistemske programere. Kako ovo područje ne bi ostalo zakinuto, Google je 2009. objavio novi programski jezik kojim bi trebao riješiti upravo problem jezika koji je ranije spomenut. Taj jezik je Go – moćan, kompiliran, višeplatformski jezik sa podrškom za višezadaćnost, jednostavne sintakse kreiran od strane tima programera sa dugim i uspješnim iskustvom razvoja programskih jezika. Jezik je dan danas prihvaćen kao jako dobra alternativa za neke starije (puno kompliciranije) programske jezike, kao što su C, C++ i slični. [3]

Go je programski jezik opće namjene sa jednostavnom, tj. razumljivom sintaksom i naprednim mogućnostima. Popularnost je dobio upravo zbog nativne proširenosti na više različitih platformi i dostupnosti na tim platformama, te je uz pomoć svoje opširne dokumentacije postao poznat kao jedan od jezika koje je lako naučiti. Sve prethodno navedeno čini Go idealnim jezikom za početnika. [1]

Tim zadužen za razvoj Go-a poseban je naglasak stavio na nekoliko ključnih koncepata: [3]

- Statički tipovi podataka sa zaključivanjem podatkovnog tipa na temelju dane varijable
- Komplicirani koncepti iz jezika poput C++-a (pokazivači, paralelni rad i sakupljač smeća) su znatno pojednostavljeni
- Višezadaćnost i paralelni rad
- Brz kompilator
- Različite primjene programskog jezika (svestranost)

Važno je napomenuti da je Go kompilatorski jezik – za razliku od velikog broja modernih jezika za izradu Web aplikacija poput PHP-a, Pythona, Rubyja i sličnih. To omogućuje aplikacijama napisanim u Go-u neusporedivo bolje performanse kod izvedbe većine zadataka naspram prethodno spomenutih jezika. Tijekom procesa kompiliranja, kompilator provjerava dobiveni Go kod, pritom izvršavajući nekoliko ključnih procesa [2]:

- Vraća povratne informacije o greškama u kodu
- Optimizira dijelove aplikacije
- Brine se o čišćenju „smeća“ (eng. garbage collection)

Programski jezik Go je dostupan na većini modernih operacijskih sustava kao što su FreeBSD, Linux, Windows i macOS. To ga čini idealnim jezikom za izradu višeplatformskih aplikacija. [2]

Go kao jezik je spreman za web razvoj bez dodatnih ekstenzija i alata, ali ipak velik broj alata i razvojnih okvira za web razvoj nedostaju, dok su kod nekih drugih jezika odmah dostupni. Kako je tijekom vremena narastao broj korisnika ovog programskog jezika, tako je od njihove strane kreiran niz paketa koji donose niz funkcionalnosti koja je samom Go-u nedostajala. Zajedno sa svim tim alatima treće strane (eng. third-party), Go trenutno predstavlja cjelokupno rješenje za web razvoj. [3]

2. Osnove programiranja u Go-u

U ovom poglavlju opisani su koncepti koji se odnose na samu sintaksu programskog jezika Go, kao i način na koje se pojedine funkcije, tj. pojedini programski elementi mogu ostvariti u ovom programskom jeziku.

2.1. Osnovna sintaksa

Programski jezik Go je izrađen od strane iskusnih programera, te kao jedan od modernijih jezika predstavlja pokušaj rješenja problema izrade višezadaćnih umreženih aplikacija. Za programera to znači slijedeće – ukoliko sintaksa programskih jezika poput Jave ili C-a predstavlja problem za programera, sintaksa Go-a bi trebala programeru omogućiti bolje, tj. jednostavnije iskustvo programiranja. S druge strane, za programere koji su iskusni u programskim jezicima poput Ruby-ja, Python-a ili JavaScripta, Go predstavlja približno jednaku razinu kompleksnosti programiranja, dok uz tu jednostavnost dodaje benefite statičkih tipova podataka bez pretjeranog kompliciranja samog koda. [2]

Kao početni dio koda (prva linija), Go koristi koncept poznat pod nazivom „deklaracija paketa“ (eng. package declaration), što označava o kojem se paketu radi. Paketi su skupine programskog koda, tj. samostalni isječci koda koji se definiraju kako bi se sami kod mogao ponovo koristiti u drugim programima. U usporedbi s jezicima poput C-a i Jave, paketi predstavljaju biblioteke koje sadrže „višenamjenski“ kod. Ukoliko se radi o glavnom programu, tj. polaznoj datoteci koja će se kompilirati u binarni kod i zatim moći pokrenuti (kod windowsa će se kreirati .exe datoteka), paket koji treba koristiti naziva se **main**. [1] Pomoću ključne riječi **package** i naziva paketa u kodu će spomenuti dio izgledati ovako:

```
package main
```

Nakon definiranja paketa u čijem sklopu program radi, sljedeći ključni korak je uključivanje drugih paketa koji će se koristiti kroz program. Jedan od osnovnih paketa koji sadrži skup funkcija za ispis u konzolu naziva se **fmt** paket. [1] Uključivanje paketa se vrši ključnom riječi **import**:

```
import "fmt"
```

Kao konačni korak prije samog pokretanja osnovne aplikacije potrebno je definirati glavnu funkciju (main), koja će se izvršiti prilikom pokretanja izvršne datoteke. Funkcije u Go-u se definiraju pomoću ključne riječi **func**, te se glavna funkcija naziva **main()**. Unutar glavne funkcije potrebno je napisati kod koji će potvrditi da program radi. Najjednostavniji tip programa

za prvo testiranje programskog jezika je popularni "Hello world" program, koji na ekran (u konzolu) ispisuje riječi "Hello world". Ovakva funkcionalnost kod programskog jezika Go postiže se korištenjem ranije spomenutog "fmt" paketa, te pripadajuće funkcije **Println** iz spomenutog paketa. [1] Kombiniranjem glavne funkcije i Println funkcije, dobiveni dio koda će izgledati ovako:

```
func main() {
	fmt.Println("Hello World")
}
```

Izrađenu datoteku je potrebno spremiti pod nazivom **main.go**, kako bi pokretanjem go naredbe kompilator znao da se radi o glavnoj, tj. ulaznoj datoteci programa. Pokretanjem naredbe "**go run main.go**" u mapi sa kreiranom datotekom, kompilator programskog jezika Go kreirat će izvršnu datoteku (u slučaju windowsa .exe) te će ju nakon procesa kompiliranja automatski pokrenuti. Prethodno definirani dijelovi koda predstavljaju osnovnu sintaksu programskog jezika Go. [1]

2.1.1. Varijable

Kao što je spomenuto u prethodnom poglavlju, Go je programski jezik sa varijablama sa statičkim tipom podatka. Na taj način osigurano je da kreirana varijabla može poprimiti određeni oblik, te da taj oblik nije moguće u nekom trenutku promijenit za tu varijablu (npr. Varijabla tipa **integer** ne može u nekom trenutku postati **string**). Varijable se u Go-u definiraju na drugačiji način naspram jezika poput C++-a ili Jave. Definicija varijable se sastoji od tri ključna dijela:

- ključne riječi **var**
- **naziva** varijable
- riječi koja označuje **tip** varijable

Kao primjer, varijabla tipa **string** naziva **x** se definira na sljedeći način:

```
var x string
```

Kako bi se definiranoj varijabli pridružila vrijednost, koristi se operator jednakosti (=):

```
var x string = "Neki tekst"
```

Uz ručno definiranje tipova varijabli, Go kompilator ima ugrađenu mogućnost zaključivanja tipa varijable na temelju pridružene vrijednosti. Kod ovakvog načina definicije varijabli koristi se znak "**:**" (dvotočka) ispred jednakosti za pridruživanje vrijednosti. [1] Primjer definiranja nove varijable tipa „string“ bez ručnog pridruživanja tipa:

```
var x := „Neki tekst“
```

Uz varijable, u GO-u su dostupne i konstante. Poput varijabli konstante se definiraju na sličan način (koristi se ključna riječ **const** umjesto **var**), no za razliko od varijabli vrijednost konstanti se tijekom daljnjeg izvođenja programa ne može mijenjati. U konstante se spremaju vrijednosti koje će se kroz programski kod više puta koristiti, no njihova vrijednost će uvijek ostati ista. Primjer definiranja konstante:

```
const x := "Neki tekst"
```

Ukoliko je potrebno, konstante i varijable je moguće definirati u većem broju bez ponavljanja ključnih riječi (var ili const), a takvo definiranje vrši se na sljedeći način:

```
var (
      a = 5
      b = 10
)
```

Uz spomenute tipove „string“ i „int“, programski jezik Go ima ugrađene još neke tipove varijabli ključnih za izradu aplikacija. Od standardnih tipova još postoje **bool** (true or false, 0 ili 1), nekoliko tipova **integera** i **unsigned integera** (cjelobrojnih tipova podataka), **byte** (koji zapravo predstavlja **uint8** tip podatka), **rune** (koji zapravo predstavlja **int32** tip podatka), **float32, float64** tipovi (decimalni brojevi s pomičnim zarezom) i **complex64, complex128** tipovi podataka (matematički tip podatka za kompleksne brojeve s imaginarnim dijelom). [5]

Uz jednostavne tipove varijabli, Go još sadrži i kompleksne tipove podataka – takvi tipovi se sastoji od nekog drugog tipa podatka. U to spadaju tri tipa: **polje, dio** i **mapa**. [1] Polje (eng. array) predstavlja varijablu koja sadrži niz elemenata (varijabli) nekog jednostavnog tipa. Primjer definiranja polja s 5 elemenata tipa **integer**:

```
var x [5]int
```

Dio (eng. slice) za razliku od polja nema definiranu duljinu prilikom kreiranja varijable, već se duljina određuje dinamički kada se dijelu pridruži vrijednost. Dio pokazuje na određeni raspon nekog polja. [5] Primjer definicije dijela sa 3 elementa iz prethodno definiranog polja:

```
var s []int = x[1:4]
```

Mapa (eng. map), poznata i kao asocijativno polje, je niz elemenata s pridruženim ključem, tj. ključnom vrijednošću koja omogućuje lakše nalaženje određene vrijednosti u mapi. [5] Primjer definiranja mape s podacima tipa **integer**, a ključevima tipa **string**:

```
var x map[string]int
```

2.1.2. Selekcije i petlje

Kontrole strukture u programskim jezicima omogućuju implementaciju jednostavnih i kompleksnih algoritama, te predstavljaju ključan element programskog jezika. Kod Go-a, broj vrsta kontrolnih struktura je sveden na minimum, sve s ciljem kako bi jezik ostao što jednostavniji za naučiti i kako ne bi dolazilo do neodlučnosti koju strukturu koristiti. Od selekcija, dostupne su dvije kontrolne strukture – **if** i **switch**. [1]

If selekcija omogućuje donošenje programskih odluka na dijelovima koda. If selekcija se definira pomoću ključne riječi **if**, te nastavka koji sadrži operator usporedbe s pripadajućom frazom.

Tablica 1: Operatori usporedbe u Go-u

Operator	Naziv	Primjer
==	jednako	if x == 10
!=	nije jednako	if x != 10
<	manje	if x < 10
<=	manje ili jednako	if x <= 10
>	veće	if x > 10
>=	veće ili jednako	if x >= 10

Uz operatore usporedbe dostupni su i sljedeći logički operatori:

Tablica 2: Logički operatori u Go-u

Operator	Naziv	Primjer
&&	i	if x == 10 && y < 2
||	ili	if x == 10 || y < 2
!	ne	if !x

Uz ključne riječi **else if** i **else** nakon programskog bloka moguće je dodati druge slučajeve na odabranu if selekciju, kao što je to moguće i kod drugih programskih jezika (C, Java, Python...). [1]

Uz "if" selekciju dostupna je još jedna vrsta selekcije u programskom jeziku Go – **switch** selekcija. Switch selekcija je dostupna u istom formatu u drugim programskim jezicima poput C-a, Jave, Python-a i sličnih, te se ova selekcija koristi za slučajeve kada je potrebno

provjeriti više mogućih vrijednosti određene varijable. Primjer switch selekcije u Go-u koja na temelju proslijeđenog broja u rasponu 0 – 2 ispisuje naziv tog broja u konzolu:

```
switch i {
	case 0: fmt.Println("Nula")
	case 1: fmt.Println("Jedan")
	case 2: fmt.Println("Dva")
	default: fmt.Println("Nepoznati broj")
}
```

Uz selekcije druga vrsta kontrolne strukture u programiranju je petlja. Petlje se koriste kako bi isti programski blok bilo moguće izvršiti nekoliko puta sa različitim vrijednostima. Kod klasične **for** petlje, taj programski blok se izvršava sve dok je uvjet petlje zadovoljen. U Go-u, petlje koje su u drugim programskim jezicima poznate pod nazivima "while" i "foreach" su integrirane u petlju "for", te se za takve tipove petlji koristi ista ključna riječ. [2]

Najosnovniji oblik **for** petlje je petlja koja zamjenjuje petlju poznatu pod nazivom **while**. Takva petlja u Go-u ima samo jedan segment u zaglavlju petlje, te je za potrebe izlaza iz petlje potrebno implementirati zasebnu logiku. Primjer for petlje koja iterira sve dok je vrijednost varijable **i** manja od 10 i ispisuje vrijednost:

```
i := 0
for i < 10 {
	i++
	fmt.Println("Vrijednost je: ", i)
}
```

Kada bi programski blok u tijelu petlje iz prethodnog primjera bio uklonjen, tada bi se petlja beskonačno izvršavala. Cijeli prethodni primjer je moguće napisati kao klasičnu **for** petlju sa segmentima inicijalizacije i koraka, kao što je prikazano u sljedećem primjeru:

```
for i := 0; i < 10; i++ {
	fmt.Println("Vrijednost je: ", i)
}
```

Uz navedene petlje u programskom jeziku Go dostupan je još jedan oblik petlje, poznat kao for petlja s klauzulom **raspona**. Takva petlja će iterirati kroz vrijednosti polja sve dok ne dođe do zadnjeg elementa u polju.

Petlja se zadaje pomoću ključne riječi for i segmenta s definicijom indeksne varijable i vrijednosne varijable, kao i definicijom samog polja. [2] Primjer petlje s rasponom:

```
brojevi := []int{1, 2, 3, 4, 5}
for i, n := range brojevi {
      fmt.Println("Vrijednost je: ", n, " na indeksu ", i)
}
```

2.1.3. Funkcije

Kao još jedan od ključnih segmenata programskog jezika Go dostupne su funkcije. Funkcije predstavljaju programske blokove koji su odvojeni u cjeline, kako bi ih bilo moguće koristiti na više mjesta u kodu. Funkcije se definiraju pomoću ključne riječi **func**, te još tri dijela: naziva funkcije, parametara funkcije (ukoliko su potrebni) i tipa vrijednosti kojeg vraća funkcija (ukoliko funkcija vraća neku vrijednost). Jedna od osnovnih funkcija u Go-u je ranije spomenuta funkcija pod nazivom **main**, te se sadržaj te funkcije izvršava prilikom pokretanja programa. Ukoliko je potrebno funkciji proslijediti neki parametar, taj parametar mora biti definiran prilikom definiranja funkcije, a tip parametra mora biti određen. [2] Primjer definiranja funkcije koja ispisuje sadržaj u konzolu ovisno o ulaznom parametru:

```
func ispisiBroj(broj int){
      fmt.Println("Uneseni broj je: ", broj)
}
```

Prethodno definirana funkcija koja ispisuje poruku s brojem 2 se poziva na sljedeći način:

```
ispisiBroj(2)
```

Ukoliko funkcija treba vratiti neku vrijednost, to se definira unutar funkcije putem ključne riječi **return**, te varijable ili vrijednosti koju funkcija vraća. Prilikom definiranja funkcije, potrebno je definirati tip vrijednosti koju funkcija vraća (ukoliko funkcija vraća neku vrijednost). Primjer funkcije koja vraća umnožak dvaju ulaznih parametara:

```
 func pomnoziBrojeve(a int, b int) int{
      return a * b
}
```

Uz vraćanje jedne vrijednosti, funkcije u Go-u mogu vratiti više izlaznih parametara, tj. izlaznih vrijednosti. Na taj način funkcije u Go-u su optimizirane da prilikom zvanja funkcije, ta

funkcija može kao jednu od izlaznih vrijednosti vratiti grešku, tj. podatke o greški koja se dogodila prilikom izvedbe funkcije. Takva funkcionalnost se učestalo koristi kod http paketa, koji je detaljnije objašnjen u četvrtom poglavlju. Primjer funkcije koja vraća više od jedne vrijednosti:

```
func podijeliBrojeve(a int, b int) (float32, string) {
	if b == 0 {
		return 0, "Drugi broj ne može biti 0. "
	}
	return float32(a/b), ""
}
```

2.1.4. Strukture

Određene podatke u Go-u potrebno je definirati kao skup više drugih tipova podataka iz kojih je moguće saznati više različitih informacija o nekom zapisu. Kod većine modernih programskih jezika (Java, C#, C++, Python...) to se postiže putem klasa i njihovih instanci – objekata. Go nije objektno orijentirani programski jezik i kao takav ne koristi klase za definiranje kompleksnijih tipova, tj. skupova podataka, već se kod Go-a koriste strukture (eng. struct).

Struktura, tj. **struct** je skup podataka sa definiranim tipovima. Strukture omogućuju način referenciranja više različitih podataka kroz zajednički naziv varijable. Na taj način različite vrijednosti vezane uz isti ključni podatak mogu biti dohvaćene kroz isti naziv varijable, a različiti naziv atributa. [2] Primjer korištenja strukture za definiranje osnovnih podataka o nekoj osobi:

```
type Osoba struct {
	Ime		string
	Prezime		string
	Dob		int
}
```

Te kada bi se kreirala varijabla tipa "Osoba", u kodu bi to bilo definirano na sljedeći način:

```
o := Osoba { Ime: "Ivan", Prezime: "Horvat", Dob: 24, }
```

Važno je napomenuti da struktura može kao podatak imati vrijednost tipa neke druge strukture, te da je na taj način moguće ugnježđivati više struktura jedna u drugu. [2]

Prilikom korištenja struktura važno je napomenuti da programski jezik Go koristi koncept pokazivača za referenciranje strukture. Pokazivači su često korišteni programski elementi iz starijih programskih jezika poput C-a i C++-a, a Go ih implementira na sličan, ali i pojednostavljen način. Pokazivači su varijable koje pokazuju na memorijsku adresu na kojoj se neki podataka nalazi. Oni omogućuju da se varijabla nekog tipa referencira kroz neku drugu varijablu, a sve promjene vrijednosti definiranih putem pokazivača izvršit će se na prethodno definiranom podatku. Na taj način moguće je funkcijama kao parametar proslijediti varijablu nekog kompleksnijeg tipa (tj. strukture) bez da mu se prosljeđuje cijeli podatak, već samo memorijska lokacija gdje se taj podatak nalazi. Na taj način smanjena je upotreba memorije i optimizirano je izvršavanje programa. [2] Na sljedećem primjeru vidljivo je kako se koriste pokazivači u programskom jeziku Go (za strukturu uzeta je struktura iz prethodnog primjera pod nazivom Osoba):

```
o := Osoba { Ime: "Ivan", Prezime: "Horvat", Dob: 24, }

pok := &o
```

Varijabla **pok** je pokazivač koji pokazuje na memorijsku lokaciju varijable **o**, koja je tipa strukture **Osoba**. U nastavku je prikazana promjena vrijednosti imena varijable **o** putem pokazivača **pok**, i ispis te vrijednosti putem varijable **o**:

```
pok.Ime = "Marko"

fmt.Println(o.Ime)
```

Vrijednost koja će se ispisati na ekran biti će "Marko", jer se putem pokazivača mijenja vrijednost varijable "o" na koju taj pokazivač pokazuje. U slučaju da se prilikom definiranja varijable **pok** nije koristio znak **&**, vrijednost varijable "o" bi se kopirala u varijablu "pok" te se promjenom vrijednosti varijable "pok" ne bi mijenjale vrijednosti varijable "o". [2]

2.2. Rad s konzolom

Kroz ranija poglavlja spomenut je paket naziva „fmt“, koji sadrži funkcije za komunikaciju s konzolom, tj. s terminalom u Go-u. Konzola je ključan oblik komunikacije između programera i programa tijekom testiranja aplikacije, te služi kao dobar izvor informacija koje krajnji korisnik ne treba vidjeti, što vrijedi posebno kod web aplikacija. Jedna od ključnih funkcija koje se koriste iz **fmt** paketa je "Println()" funkcija. Ta funkcija služi za ispis podataka u konzolni prozor, te se koristi na sljedeći način:

```
fmt.Println("Ispis teksta")
```

Ukoliko je potrebno, moguće je unijeti određene vrijednosti putem konzolnog unosa. Takve radnje se izvršavaju putem "Scan" funkcija, te omogućuju jednostavan unos podataka putem konzolnog prozora. Primjer unosa podataka putem Sscan funkcije:

```
var x string

fmt.Sscan(x)
```

Uz fmt paket za rad sa konzolom, moguće je koristiti i paket "os" za unos i čitanje veće količine podataka. Uz funkcije za čitanje podataka iz konzole, "os" paket sadrži i niz drugih funkcija za čitanje datoteka i sličnih elemenata koji pripadaju operacijskom sustavu. Pomoću takvih jednostavnih funkcija za komunikaciju s konzolom, moguće je jednostavno implementirati cijelu dijagnostički cjelinu aplikacije.

Slika 1: Primjer ispisa primljenih zahtjeva web aplikacije u konzolni prozor

2.3. Paketi

Kod većine modernih programskih jezika koristi se pojam biblioteka – skupova koda povezanih u cijelinu kako bi se taj kod mogao koristiti kroz nekoliko različitih aplikacija. Kod programskog jezika Go, takvu funkcionalnost poprimaju paketi (eng. package). Paketi su skupovi programskog koda sa nizom funkcija i struktura kreiranih kako bi se modularizirala izrada aplikacije. Korištenjem paketa određeni programski kod je moguće koristiti kroz više aplikacija, a isto tako je moguće izdvojiti dio koda iz glavne programske datoteke (main.go) kako bi se kod bolje strukturirao. Ranije spomenuti paket **main** predstavlja naziv glavnog paketa u programu koji sadrži funkciju **main**, a ona predstavlja glavni kod programa koji će se prvi pokrenuti prilikom izvršavanja aplikacije. Za početak, pakete je moguće podijeliti na tri vrste:

- Ugrađeni go paketi
- Lokalni paketi
- Vanjski paketi dohvaćeni s različitih izvora

Ugrađeni **go** paketi (npr. spomenuti paket **fmt**) su paketi koji se prilikom instalacije Go razvojnog okruženja i kompilatora automatski dodaju u glavnu Go biblioteku s dostupnim paketima. Takvi paketi sadrže funkcije i strukture napisane od strane tima za razvoj programskog jezika Go, te su dostupni za korištenje već nakon instalacije Go razvojnog okruženja na računalo. Za uključivanje, tj. korištenje ovakvog paketa u nekoj Go datoteci, potrebno je na vrhu datoteke kod **import** naredbe definirati ime paketa pod navodnicima:

```
import "fmt"
```

Lokalni paketi su paketi koji se koriste u nekoj Go datoteci, a kreirani su lokalno negdje u strukturi trenutnog projekta. Takve pakete će Go-ov kompilator automatski pronaći, uz napomenu da se oni uključuju u neku drugu datoteku s prefiksom "**./**":

```
import "./funkcije"
```

Prema službenoj Go dokumentaciji, iako podržani, lokalni paketi referencirani prefiksima "**./**" ili "../" ne predstavljaju preporučen način za referenciranje paketa. Prema preporuci iz Go službene dokumentacije, lokalni paketi unutar programskog jezika Go referenciraju se svojom relativnom putanjom naspram mape zadane putem varijable okoline programskog sustava (eng. Environment variable) zajedničke za sve projekte programskog jezika Go. U slučaju da se glavna Go programska datoteka (paket "main") nalazi u mapi s imenom "goprojekt“, tada bi referenca lokalnog paketa "alati" bila definirana na način:

```
import "goprojekt/alati"
```

Ukoliko se radi o vanjskog paketu s izvora treće strane (eng. third party package), tada se on referencira kao i ugrađeni paket, no prvo ga je potrebno instalirati na razvojno računalo. To se obavlja pozivom naredbe **go get** iz terminala, ukoliko je paket javno dostupan na internetskom poslužitelju:

```
go get github.com/golang/example/stringutil
```

Nakon što se vanjski paket uspješno dohvati i instalira na razvojno računalo, moguće ga je koristiti kao i ugrađeni paket, pozivom njegovog naziva putem **import** naredbe. Neki od najčešće korištenih Go paketa prikazani su narednoj tablici:

Tablica 3: Najčešće korišteni paketi programskog jezika Go

Naziv paketa	Kratki opis
fmt	Paket s funkcijama za ispis podataka u konzolni prozor te obradu unesenih skupova znakova
net/http	Paket s funkcijama za izradu web poslužitelja, dohvaćanje http zahtjeva i rad s tim zahtjevima
html/template	Paket s funkcijama za obradu html predložaka
errors	Paket s funkcijama za obradu nastalih grešaka
database/sql	Paket s zajedničkim funkcijama za spajanje aplikacije s bazom podataka i daljnji rad s bazom podataka
time	Paket s funkcijama za manipulaciju s vremenskim tipovima podataka

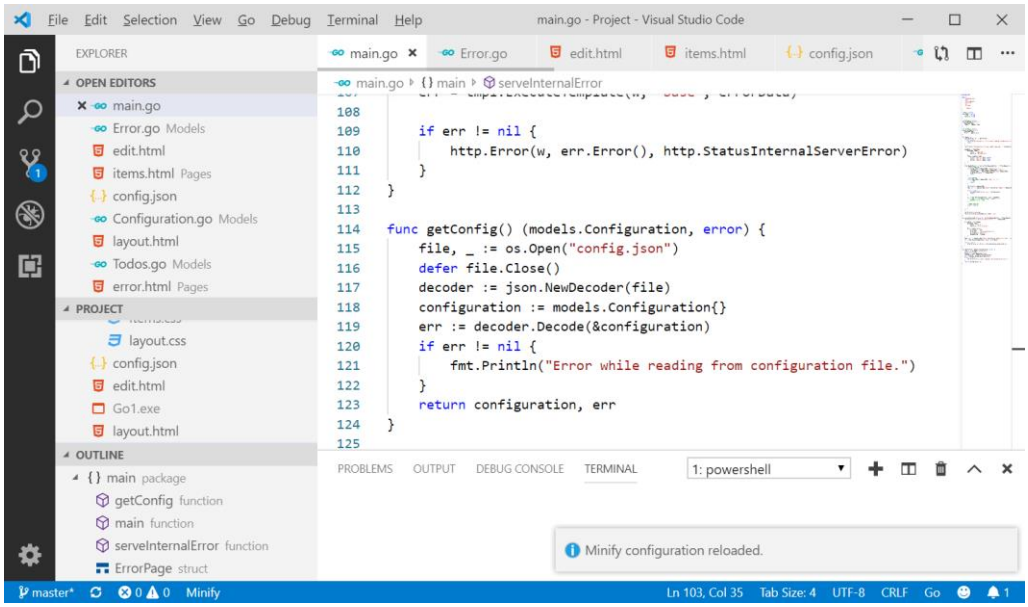
3. Opis radne okoline i kreiranja novog projekta

Kod izrade aplikacije putem programskog jezika Go, potrebno je odabrati odgovarajuće razvojno okruženje, tj. program s kojim će izrada programa biti jednostavna i dobro organizirana. Za ovaj projekt odabran je program pod nazivom „Visual Studio Code“.

3.1. Visual Studio Code

Visual Studio Code je besplatni programski alat tvrtke Microsoft, te omogućuje rad s različitim programskim jezicima i alatima. Upravo to je razlog njegove rastuće popularnosti, te uz programske jezika poput C#-a, PHP-a, C-a i sličnih čini dobru podlogu za izradu Go aplikacija. Također, Visual Studio Code je „lagan“ a istovremeno vrlo moćan uređivački program izvornog koda za Windows, macOS i Linux operacijske sustave. Prilikom početne instalacije spomenutog programa, na računalo se instalira podrška za neke jezike koji se koriste uz Go za web razvoj, tako da je dio početnih postavki razvojnog okruženja odmah gotov. [6]

Visual Studio Code je također poznat zbog ugrađenog „Intellisense“ sustava koji automatski sugerira sljedeće dio koda kojeg korisnik želi upisati. Na taj način radikalno se ubrzava sami razvoj aplikacija uz minimalnu količinu upisivanja istog dijela koda na više mjesta u datotekama.



Slika 2: Izgled razvojnog sučelja Visual Studio Code

3.2. Instalacija plugin datoteka za rad s Go-om

Kako bi na razvojnom računalu bilo moguće izrađivati Go aplikacije, potrebno je instalirati povezane programe koji to omogućuju. Uz samo razvojno sučelje, potrebno je instalirati kompilator na razvojno računalo kako bi .go datoteke bilo moguće pretvoriti u izvorni kod – format kojeg računalo može pročitati. Prilikom pokretanja instalacijskog paketa dostupnog na službenoj Go stranici (izvor [5]) na računalo se instalira kompilatorski program s povezanim datotekama i pred-instaliranim Go paketima. Uz instalaciju, automatski se u niz konzolnih naredbi dodaje niz naredbe s prefiksom „go“, koje će detaljnije biti objašnjene u idućem poglavlju.

Prilikom kreiranja prve .go datoteke i pokušaja rada s njom u Visual Studio Code-u, program će automatski kreirati obavijest o potrebnom dohvatu dodatnih datoteka koje su potrebne za rad s Go-om putem Visual Studio Code programskog alata. Instalacijom programskog alata za verzioniranje **git** i njegovim pokretanjem pomoću naredbe unutar razvojnog alata Code dohvaćaju se datoteke koje optimiziraju razvojno okruženje za razvoj Go aplikacija. Uz takve alate potrebno je dohvatiti i ekstenziju za Visual Studio Code koja spaja instalirane alate sa samim Code programom, te u potpunosti integrira Go funkcionalnost.

Nakon uspješne instalacije svih potrebnih ekstenzija i alata za rad u definiranom programskom sučelju, potrebno je kreirati glavnu, tj. polaznu datoteku koja će sadržavati glavnu funkciju (main) te će predstavljati polaznu točku novog programa. Unutar glavne mape od projekta, kreirat će se datoteka pod nazivom **main.go**, koja predstavlja upravo tu polaznu datoteku koja će sadržavati glavni (polazni) kod za pokretanje aplikacije. Osnovna struktura takve aplikacije opisana je u drugom poglavlju, te je nakon osnovnog strukturiranja moguće kompilirati datoteku u program kojeg je moguće pokrenuti.

3.3. Kompiliranje i pokretanje programa

Kompiliranje i pokretanje Go programa se vrši pomoću ugrađenih funkcija go-ovog kompilatora i funkcija operacijskog sustava. Naredba za kompiliranje Go izvornog koda u izvršnu programsku datoteku je „**go build**“, te se pokreće unutar samog foldera u kojem se nalazi „main.go“ datoteka. Primjer pokretanja naredbe za kompiliranje:

```
C:Go/projects/Primjer> go build
```

Pokretanjem ovakve naredbe na windows operacijskom sustavu unutar mape projekta kreirat će se **.exe** izvršna datoteka sa nazivom jednakim nazivu projektne mape. Ukoliko je potrebno prilikom kreiranja izvršne datoteke tu datoteku nazvati nekim drugim imenom, moguće je dodati klauzulu s prefiksom "-o" i željenim nazivom. [5] Npr. ukoliko je izvršnu datoteku prethodno kompiliranog projekta potrebno nazvati "Testiranje.exe" tada će se za kompiliranje koristiti sljedeća naredba:

```
C:Go/projects/Primjer> go build -o Testiranje.exe
```

Ukoliko je nakon procesa kompiliranja izvršnu datoteku potrebno pokrenuti, to je moguće napraviti na standardni način pokretanja izvršne datoteke, kako je to dostupno u operacijskom sustavu. No tijekom testiranja je poželjno pokretati datoteke odmah nakon procesa kompiliranja, pa je za takve potrebe dostupna naredba "**go run**", koja će nakon build naredbe odmah pokrenuti izvršnu datoteku. [5] Primjer korištenja:

```
C:Go/projects/Primjer> go run
```

Ukoliko je potrebno ukloniti spremljene i postojeće datoteke, osim ručnim putem, to je moguće napraviti pomoću naredbe "**go clean**". Ovakva naredba će automatski očistiti mapu trenutnog projekta od svih datoteka nastalih nakon kompiliranja i pokretanja aplikacije.

Još jedna od ključnih naredbi je naredba "**go get**". Preko spomenute naredbe moguće je dohvatiti paket sa neke lokacije izvan projekta. Na taj način optimizirano je dijeljenje i dohvaćanje paketa izrađenih od strane drugih korisnika. Primjer dohvaćanja paketa:

```
C:Go/projects/Primjer> go get go get -u github.com/gorilla/mux
```

Izvođenjem prethodno definirane naredbe u glavnu mapu sa paketima će se dodati novi paket, te će se moći uključivati u bilo koji Go projekt na računalu na isti način kao i ugrađeni Go paket (npr. "fmt").

4. Web programiranje u Go-u

U ovom poglavlju opisani su napredniji dijelovi programskog jezika Go – specifično paketi i sintaksa potrebna za izradu web aplikacija.

4.1. Obrada HTTP zahtjeva u Go-u

Kako bi se zahtjev određenog korisnika prema poslužitelju mogao izvršiti, web aplikacija mora imati ugrađen sustav za osluškivanje HTTP zahtjeva i vraćanje odgovora na te zahtjeve. Kod programskih jezika poput PHP-a i C#-a, za takve potrebe obrade zahtjeva dostupni su poslužiteljski programi. Takvi programi predstavljaju prvu točku prilikom obrade HTTP zahtjeva, te zahtjev dalje prosljeđuju samoj aplikaciji. Kod programskog jezika Go, zahtjev direktno dolazi do web aplikacije. To omogućuje brzi dohvat zahtjeva od strane aplikacije i brzo vraćanje odgovora prema korisniku, no zahtjeva nešto dodatnog koda. Da bi se pisanje takvog koda skratilo, dostupan je **„net/http“** paket.

4.1.1. Net/HTTP paket

Net/HTTP paket omogućuje programiranje logike jednostavnog zaprimanja **http** zahtjeva i vraćanja odgovora na zaprimljene zahtjeve kroz nekoliko linija programskog koda.

Tijekom starih dana weba, cijeli internet je funkcionirao na principu direktnog vraćanja statičkih datoteka na dobivene zahtjeve. Na ovaj način cijela web lokacija je bila skup statičkih datoteka čiji bi se sadržaj direktno vratio kao odgovor na zahtjev. Taj princip je moguće jednostavno implementirati pomoću „net/http“ paketa. [3]

```
http.ListenAndServe(":8080", http.FileServer(http.Dir("var/www")))
```

Prethodni kod će osluškivati sve http zahtjeve na portu **8080**, te će kao odgovore vraćati datoteke koje se nalaze na disku na lokaciji „**var/www“** (relativno od lokacije izvršne datoteke). Na taj način moguće je realizirati aplikaciju kroz malu količinu programskog koda.

Ukoliko je u aplikaciji potrebno koristiti modernije principe web aplikacija (sesije, spajanje na bazu podataka, POST i GET zahtjeve i slično), potrebno je koristiti **http.HandleFunc** funkciju. Ta funkcija će na temelju zadanog parametra (koji definira url kojeg je korisnik tražio) odraditi zadani dio koda. Spomenuta **HandleFunc** funkcija prima dva parametra – prvi predstavlja oblik URL-a dobivenog zahtjeva. Ukoliko zahtjev odgovara uzorku postavljenom za prvi parametar funkcije, tada će se izvršiti funkcija definirana drugim parametrom. Ta funkcija prima dva ulazna parametra – **http.ResponseWritter** varijable koja

se koristi kao podloga za izradu odgovora na zahtjev i ***http.Request** varijable pomoću koje je moguće doći do ostalih podataka poslanih putem dobivenog zahtjeva.

```
http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {...
```

Prethodno definirani kod će na bilo koji zahtjev izvršiti funkciju definiranu kao drugi parametar **handleFunc** funkcije. Različite zahtjeve je moguće „pratiti“ pomoću više handleFunc funkcija. Primjer:

```
http.HandleFunc("/pocetna", pocetnaAkcija)

http.HandleFunc("/postavke", pocetnaAkcija)

http.HandleFunc("/autorizacija/prijava", pocetnaAkcija)
```

Tako strukturiran kod je vrlo jednostavan, te omogućuje brzu izradu jednostavnih web aplikacija kroz nekoliko linija programskog koda. Nažalost takav pristup nije prigodan za veće aplikacije i kompleksnije web servise, stoga je za takve slučajeve potrebno koristiti neki razvojni okvir za web aplikacije ili proširiti funkcionalnost osnovnih funkcija za upravljanje **http** zahtjevima. Od razvojnih okvira za Go web razvoj, dostupan je **mux** paket iz **gorilla** skupa alata. Mux proširuje funkcionalnost upravljanja zahtjevima dodajući funkcionalnosti poput regularnih izraza (eng. Regular expression), naprednog preusmjeravanja i ostalih funkcionalnosti. [3]

Ukoliko se u aplikaciji ne koristi pristup pomoću mux preusmjerivača (eng. router-a), do putanje iz zahtjeva moguće je doći pomoću druge varijable u funkciji koja odgovara na zahtjev, tj. pokazivač *http.Request sadrži atribut **RequestURI** koji označuje putanju koja je poslana prema poslužitelju. Na temelju vrijednosti tog atributa može se izvršiti specifična akcija, te je iz samog zahtjeva moguće dohvatiti još neke vrijednosti, kao što su kolačići, GET i POST parametri (ovisno o metodi), vrijednosti forme i slično.

4.1.2. Statičke datoteke

Uz .go datoteke koje sadrže programski kod koji se izvršava na poslužitelju (nakon što se datoteka kompilira u izvršnu datoteku), web aplikacija mora sadržavati i neke datoteke koje će se direktno poslati prema klijentskom računalu, tzv. **statičke datoteke**. Statičke datoteke su skup datoteka koji se nalazi na određenoj lokaciji na poslužitelju, te se njihov sadržaj bez prethodne modifikacije direktno šalje prema klijentu kao odgovor na http zahtjev. Takve datoteke mogu biti slike, video sadržaj, zvukovni sadržaj, a isto tako mogu biti i programski kod koji se izvršava na klijentskom pregledniku, kao što su **JavaScript** ili **css** datoteke.

Statičke datoteke će se u Go web aplikaciji nalaziti u specifičnoj mapi, npr. mapa pod imenom **static**, sa podmapama nazvanih ovisno o sadržaju koji se nalazi u njima (images, css,

js itd.). Prilikom dohvaćanja zahtjeva za statičkom datotekom, potrebno je obratiti pažnju na dolaznu putanju po kojoj je datoteka pristigla do poslužitelja. Ukoliko se klijent nalazi na stranici s putanjom "/kategorije/sport", te na toj stranici šalje do poslužitelja zahtjev za statičkom datotekom "/static/images/slika1.jpg", tada će do poslužitelja doći zahtjev sa putanjom "kategorije/sport/static/images/slika1.jpg". Zato je ključna stvar kod vraćanja statičkih datoteka da se njihova putanja obradi kao apsolutna na poslužitelju, te da se uvijek datoteke iz **static** mape dohvaćaju tako da je mapa static uvijek početna mapa putanje. Kada se prethodno definirana radnja prebaci u kod, primjer će izgledati ovako:

```
http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
      if strings.Index(r.RequestURI, "static") != -1 {
            ...
      }
}
```

U prethodno definiranom dijelu koda pomoću **http.HandleFunc** funkcije prvo se osluškuje bilo koji dolazeći zahtjev (kako je objašnjeno u poglavlju 4.1.1.). Ukoliko **RequestURI**, tj. putanja zahtjeva sadrži naziv "static" u sebi, tada se radi o zahtjevu za statičkom datotekom te se izvršava sljedeći kod:

```
lastIndex := strings.Index(r.RequestURI, "static")
r.RequestURI = r.RequestURI[lastIndex:len(r.RequestURI)]
http.ServeFile(w, r, r.RequestURI)
return
```

U prethodnom kodu, prvo se dohvaća indeks posljednjeg pojavljivanja riječi "static" u putanji zahtjeva. Tada se **RequestURI** varijabla, koja predstavlja putanju zahtjeva, mijenja tako da bude jednaka samo onom dijelu nakon "static" riječi, ukoliko ona već nije bila početna. Na kraju se kao odgovor na zahtjev vraća statička datoteka iz "static" mape, te se završava "http.HandleFunc" funkcija.

4.2. Sustav predložaka (templates)

Prilikom odgovaranja na korisničke zahtjeve za određenim web stranicama, klijentu će se vratiti sadržaj **html** datoteka koje se nalaze na poslužiteljskom računalu. Sadržaj html datoteka će u tom slučaju uvijek biti isti, te kako bi se na stranicama nešto promijenilo potrebno je koristi JavaScript za izmjene kod klijenta. No moguće je, prilikom odgovaranja na zahtjev, modificirati sadržaj html datoteke na temelju Go-ovog ugrađenog sustava predložaka. [3]

Go predlošci omogućuju da se dio logike sa go datoteka prebaci na html datoteke, tj. da se na temelju zadanih parametara definiraju petlje, selekcije, varijable i neke funkcije u html datoteci. Kako bi se html datoteka mogla obraditi prema sustavu predložaka, potrebno je datoteku parsirati (obraditi) prilikom dogovaranja na http zahtjev:

```
tmpl,err :=
template.New("base").ParseFiles("Pages/"+r.RequestURI+".html",
"layout.html")

err = tmpl.ExecuteTemplate(w, "base", pageData)
```

Prilikom obrade predloška, html stranici je moguće pridružiti određeni model podataka. Tada se prema tom modelu određuju vrijednosti varijabli u predlošku, te je moguće izraditi jednostavniju programsku logiku unutar predloška ovim putem. Unutar html datoteke, varijable koje se obrađuju u sklopu predloška definiraju se na sljedeći način:

```
<head>

     <title>{{.BasePage.Title}}</title>

</head>
```

U prethodnom primjeru prikazano je kako će predložak poprimiti vrijednost naslova stranice jednaku vrijednosti atributa **Title** kod objekta naziva **BasePage** koji je atribut glavnog modela stranice (PageModel). Sve varijable koje se koriste tijekom obrade predloška moraju biti definirane negdje u go kodu prije početka obrade predloška.

Web aplikacija će sadržavati nekoliko dostupnih stranica kroz koje se obavljaju sve aktivnosti na web aplikaciji. Zasebne stranice aplikacije imaju drugačiji sadržaj, no njihov osnovni izgled i neke funkcionalnosti su zajedničke. Ukoliko se sustav predložaka ne koristi prilikom obrađivanja i vraćanja sadržaja stranica kao odgovora na zahtjeve, u svakoj html datoteci će se određeni dio koda ponavljati. Kako bi se takvo nepotrebno ponavljanje koda izbjeglo, potrebno je definirati jednu glavnu stranicu, tj. layout page – html datoteku koja će imati uključene zajedničke statičke datoteke i sadržaj zaglavlja i podnožja, te će imati

definirane sekcije u koje će se uključivati sadržaj sa pojedinih stranica koje se učitavaju kao dio glavne stranice.

Sustav predložaka u Go-u omogućuje definiranje određenih sekcija unutar html datoteka koje označuju mjesta gdje će se nalaziti sadržaj sa ostalih html stranica / datoteka. Sekcija mora imati definiran naziv, te se unutar datoteka definira pomoću ključne riječi "**define**", dok se u glavnoj stranica dohvaća pomoću ključne riječi „**template**“. Primjer korištenja sekcija:

```
<head>
      {{template "stilovi" .}}
</head>
```

Prethodna linija koda će unutar zaglavlja glavne stranice definirati područje u koje će se prebaciti sadržaj definiran u drugoj stranici pod nazivom sekcije „**stilovi**“.

```
{{define "stilovi"}}
      <link href="static/css/layout.css" rel="stylesheet" />
{{end}}
```

Prethodna linija koda će se nalaziti u html datoteci neke stranice (ne glavne), te će se taj kod prebaciti u sekciju definiranu pod nazivom „**stilovi**“. Na taj način, kroz korištenje sekcija unutar predloška, lako je moguće više različitih dijelova stranice modificirati ovisno o stranici koja se obrađuje. Prilikom obrade zahtjeva za određenom stranicom, kroz Go kod potrebno je parsirati glavnu stranicu (layout) sa trenutnom stranicom koja se obrađuje.

```
tmpl,err := template.New("predlozak").ParseFiles("stranica.html",
"glavna_stranica.html")
```

Ukoliko prilikom izvršavanja prethodnog isječka koda ne dođe do greške, parsiranje, tj. spajanje sadržaja dviju stranica je uspješno prošlo. Zatim, pomoću funkcije "ExecuteTemplate", kreirana stranica se vraća klijentu kao odgovor na zahtjev. Kroz spomenutu funkciju moguće je još i stranici proslijediti skup vrijednosti kao model stranice, no to nije obvezna funkcionalnost.

4.3. Rad s bazom podataka

Kao moderan pristup podacima kroz web aplikacije, koriste se baze podataka. Go kao moderan jezik ima dostupan niz biblioteka koje omogućuju povezivanje s različitim bazama podataka kao što su postgresql, mysql, oracle, sqlLite i slični. Za rad sa SQL rezultatima upita koristi se zajednički Go paket.

4.3.1. Spajanje na SQL bazu podataka

SQL kao jezik za rad s relacijskim bazama podataka će kao rezultat upita vratiti sličan tip podatka (tablicu ili više njih) kod različitih baza podataka, npr. MySQL-a i PostgreSQL-a. Rad s dobivenim podacima se obavlja pomoću ugrađenog Go paketa "**database/sql**", te se funkcije tog paketa mogu koristiti neovisno o tipu baze sa koje aplikacija dohvaća podatke. Za spajanje na određenu bazu podataka potrebno je koristiti specifični upravljački paket za tu bazu. Za neke jezike potrebni paketi su definirani u nastavku:

- PostgreSQL: `import _ "github.com/lib/pq"`
- MySQL: `import _ "github.com/go-sql-driver/mysql/"`
- Oracle: `import _ "gopkg.in/goracle.v2"`

Nakon definicije potrebnog paketa za komunikaciju s bazom, potrebno je definirati potrebne parametre za spajanje s bazom: poslužitelj, korisničko ime, lozinku i slične podatke, ovisno o bazi na koju se aplikacija spaja. To se definira kroz niz teksta spremljen u **string** varijablu (eng. Connection string), te se veza s bazom otvara na sljedeći način:

```
dsn := fmt.Sprintf("host=localhost port=3306 dbname=postgres
user=postgres password=lozinka sslmode=disable")

db, err := sql.Open("postgres", dsn)
```

U prethodnom primjeru prikazano je spajanje na postgreSQL bazu podataka. Kroz "fmt.Sprintf" funkciju definirani su potrebni parametri za spajanje, te je pomoću te funkcije složen pravilni oblik **string** varijable kakav se koristi za spajanje na bazu podataka. Kroz drugu liniju koda otvorena je veza prema bazi, te se potrebni upravljački paket definira pomoću prvog parametra funkcije "**sql.Open**", a drugi parametar se koristi za pristup do navedene baze. Ukoliko ta funkcija ne vrati grešku, aplikacija je uspješno spojena sa bazom, te je moguće izvršiti upit nad spomenutom bazom do zatvaranja veze. Sql paket ima ugrađeni sustav za zatvaranje veze, tako da nije potrebno ručno zatvarati postojeću vezu s bazom. [3]

4.3.2. Izvršavanje upita nad bazom

Kako bi se iz relacijske baze mogli dohvatiti ili izmjeniti podaci, potrebno je koristiti upite nad bazom. Kako je ranije spomenuto, za izvršavanje operacija nad bazom iz Go aplikacije, dostupan je zajednički paket koji se koristi za sve relacijske baze – "database/sql" paket. Paket sadrži niz funkcija koji omogućuju rad s podacima dobivenim iz baze podataka te komunikaciju sa spomenutom bazom. Nakon što se pomoću upravljačkog paketa uspješno izvrši spajanje na bazu podataka, moguće je izvršiti upit pomoću funkcije "**db.Query()**", gdje varijabla **db** predstavlja otvorenu vezu s bazom podataka.

```
upit := "SELECT * FROM korisnik;"

redovi, greska := db.Query(upit)
```

Rezultat „query“ funkcije će se u prethodnom primjeru spremiti u varijablu naziva "redovi", a ta varijabla će sadržavati kolekciju redova podataka dobivenih pozivom SQL upita. Nakon što se upit uspješno izvrši bez dobivene greške, potrebno je dohvatiti sadržaj redova. Nakon čitanja podataka iz redova, potrebno je spomenute redove „zatvoriti“, tj. označiti programu kraj čitanja redova kako bi kompilator automatski znao kada je potrebno zatvoriti vezu s bazom. To će se izvesti pomoću ključne riječi "**defer**", a ona označuje da se objekt ne „zatvara“ dok se vanjska funkcija (unutar koje se objekt nalazi) ne vrati (eng. return). [7]

```
defer redovi.Close()
```

Nakon što se odgodi zatvaranje redova, te redove je potrebno pročitati i spremiti njihov sadržaj u varijable programskog jezika Go. Za takvu funkcionalnost koristit će se petlja **for**, te će se ona ponavljati sve dok ne dođe do zadnjeg reda, pomoću funkcije "**redovi.Next()**":

```
for redovi.Next() {

      user := User{}

      redovi.Scan(&user.ime,&user.prezime,&user.email, &user.korime)

}
```

U prethodnom primjeru, varijabla user predstavlja strukturu za spremanje podataka o korisniku, a funkcija **redovi.Scan** pojedini red pretražuje na način da odabrane varijable (po redu kako su dohvaćene putem upita) sprema kao vrijednosti atributa varijable **user**. Kroz takvih nekoliko linija programskog koda, programski jezik Go omogućuje brzo i jednostavno izvršavanje upita nad bazom i spremanje podataka. Ključna prednost naspram drugih programskih jezika je u tome da Go koristi jedan paket za rad sa svim relacijskim bazama, dok kod drugih jezika postoje razlike prilikom rada sa različitim bazama podataka. [3]

4.4. Paralelni rad (goroutines)

Kako je spomenuto u uvodnom poglavlju, programski jezik Go ima ugrađene koncepte za jednostavno postizanje paralelnog rada u aplikacijama. Glavni mehanizam Go-a za postizanje višezadaćnosti nosi naziv „goroutines“. Pomoću ključne riječi "go" moguće je kreirati novi goroutine, tj. novu programsku instancu koja će se izvršavati paralelno s ostatkom programa. Kada se neka funkcija pokrene pomoću ključne riječi "go" ispred poziva funkcije, program će se odmah vratiti iz te funkcije, no funkcije će se uz glavni dio koda nastaviti izvršavati neovisno u pozadini. Ukoliko se kreira više različitih goroutines objekata istovremeno, nije moguće znati kojim će se redom oni izvršiti – to ovisi o raspoređivaču procesa operacijskog sustava i opterećenosti sustava. [7]

Postoje dva glavna načina za kreiranje goroutine objekta – pozivom regularne, definirane funkcije i pozivom anonimne funkcije. Ukoliko goroutine mora izvršiti neki manji dio koda koji se ne ponavlja na više mjesta u kodu, tada je moguće koristiti anonimne funkcije:

```
go func() {
	for i := 10; i < 20; i++ {
		fmt.Print(i, " ")
	}
}...
```

Prethodna anonimna funkcija će ispisati brojeve u rasponu od 10 do 19 u sklopu zasebnog goroutine objekta, tj. u sklopu zasebne dretve. Ukoliko je funkcija kompleksnija ili se ponavlja na više mjesta u kodu, pomoću "go" poziva moguće je pokrenuti neku definiranu funkciju, kao što je prikazano u sljedećem primjeru:

```
func IspisiNesto() {
	fmt.Print("Nesto.")
}
go IspisiNesto()
```

U prethodnom primjeru će se funkcija naziva "IspisiNesto" pokrenuti u novoj dretvi pozivom "go IspisiNesto", te će u zasebnoj dretvi ispisati u konzoli definiranu riječ.

5. Praktični rad - aplikacija

U ovom poglavlju opisani su pojedini dijelovi aplikacije izrađene sa programskim jezikom Go, te je dan pregled dijelova samog Go projekta.

5.1. O aplikaciji

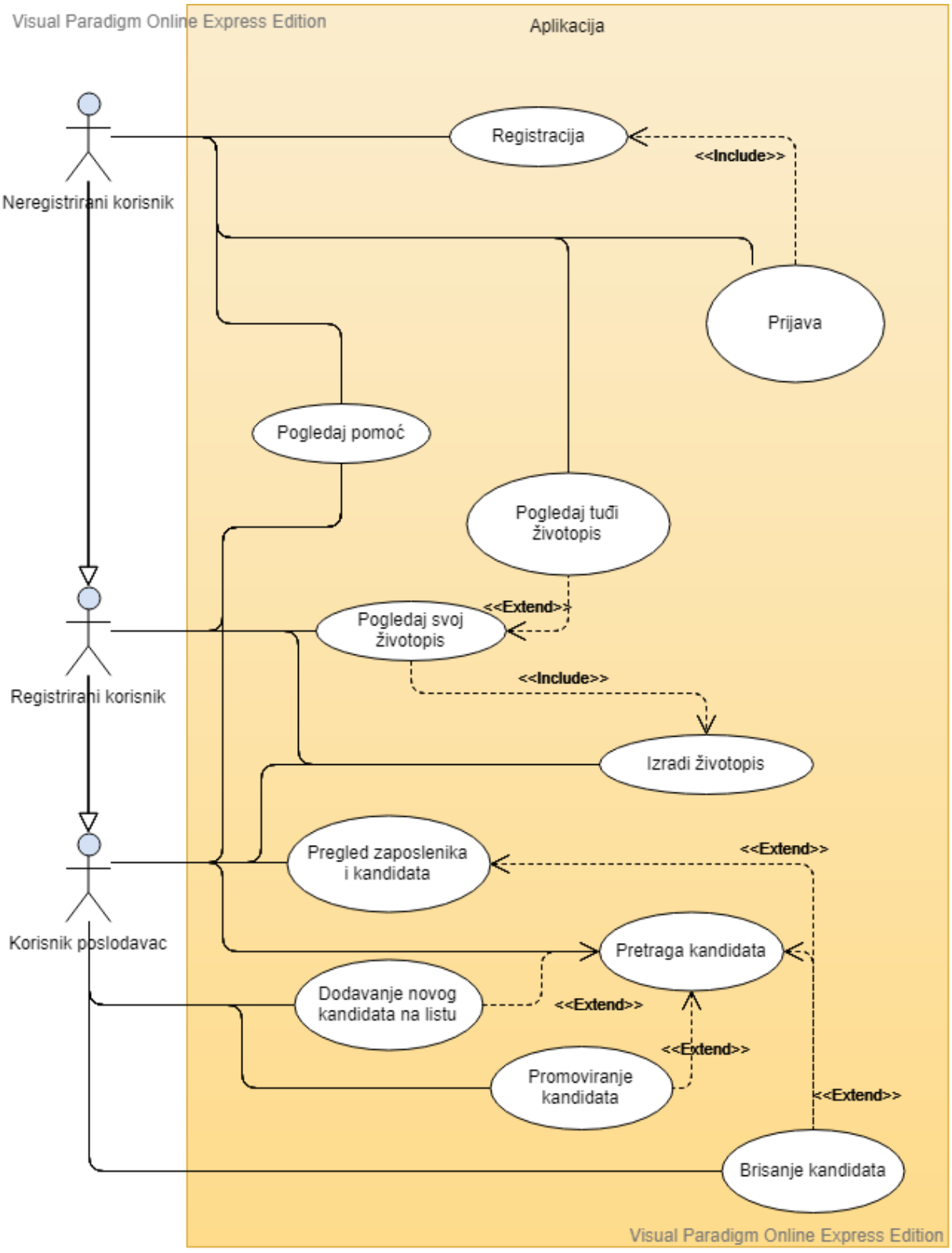
Aplikacija obrađena u ovom poglavlju je web aplikacija za izradu digitalnih verzija životopisa i njihovo međusobno dijeljenje između korisnika i kompanija koje koriste spomenutu aplikaciju. Aplikacija koristi testni naziv „openCV“, te bi trebala omogućiti bilo kojem korisniku jednostavnu izradu životopisa pomoću nekoliko polja za unos podataka.

Iako su kroz nekoliko internetskih portala dostupni slični alati za izradu životopisa, svi oni su namijenjeni kako bi se životopis nakon izrade ispisao putem pisača, te se predao u papirnatom obliku. openCV aplikacija bi trebala omogućiti izradu životopisa i njegov digitalni prikaz u prezentacijskom obliku, no ispis životopisa nije primarni zadatak ovog alata. Svakom gotovom životopisu se dodijeljuje jedinstveni kod kako bi jednostavno bilo kasnije dohvatiti određeni životopis. Životopis se sprema u bazu podataka kroz nekoliko relacija definiranih u nadolazećim poglavljima.

Uz izradu životopisa pomoću jednostavnog alata, aplikacija također treba omogućiti korisniku prijenos osobne slike (koja se pojavljuje na životopisu) i još nekoliko manjih postavki vezanih uz korisničko iskustvo. Kao konačnu funkcionalnost aplikacije, korisnik bi trebao moći kreirati račun poduzeća pomoću kojeg može uspoređivati životopise nekoliko različitih kandidata, te bi kao dodatnu funkcionalnost mogao dodavati određene životopise u razmatranje za zaposlenje unutar poduzeća.

Cijeli serverski kod aplikacije će biti implementiran putem programskog jezika Go, dok će SQL baza aplikacije biti PostgreSQL relacijska baza. Za potrebe klijentskih funkcija, koristit će se samo posebni fontovi za tekst i ikone, dok od JavaScript biblioteka neće biti korištena niti jedna.

Za lakši prikaz funkcionalnosti aplikacije dostupan je dijagram slučajeva korištenja, prikazan na slici u nastavku:



Slika 3: Dijagram slučajeva korištenja

5.2. Struktura projekta

Kako bi se aplikacija mogla s vremenom održavati potrebno je prilikom definiranja projekta dobro strukturirati aplikaciju. Glavna mapa s projektom (nazvana Project) podijeljena je na nekoliko mapa i dvije datoteke:

- config.json

- main.go

config.json datoteka sadrži konfiguracijske parametre koji se učitavaju prilikom pokretanja aplikacije. **main.go** datoteka je glavna (početna) datoteka s Go izvornim kodom, te ona prikazuje ulaznu točku prilikom kompiliranja aplikacije u izvršnu datoteku. Uz te dvije datoteke dostupno je i nekoliko mapa:

- Api
- Framework
- Handlers
- Models
- Pages
- static

Mapa **Api** sadrži .**go** datoteke sa potrebnim kodom za pristup bazi podataka. Kroz funkcije koje se nalaze u datotekama iz ove mape moguće je poslati upit do baze i dobiti odgovor.

Mapa **Framework** sadrži **.go** datoteke sa ugrađenim funkcijama koje se koriste na više mjesta kroz aplikaciju. Primjer takvih funkcija su funkcije za dohvaćanje konfiguracijskih parametara iz konfiguracijske datoteke, funkcije za rad s greškama i slične funkcije.

Mapa **Handlers** sadrži **.go** datoteke sa programskom logikom odgovora na http zahtjeve. Funkcije iz ovih datoteka se direktno pozivaju u main.go datoteci, a sva ostala programska logika iza određenog zahtjeva se nalazi upravo u tim funkcijama.

Mapa **Models** sadrži **.go** datoteke s tipovima podataka i strukturama koje se koriste kroz aplikaciju.

Mapa **Pages** sadrži **.html** datoteke sa strukturama pojedinih stranica aplikacije. Također sadrži i dijeljene stranice kao što je layout.html, stranica koje se parsira zajedno s drugim stranicama prilikom odgovora na http zahtjev.

Mapa **static** sadrži statičke datoteke koje se serviraju putem http zahtjeva. Mapa je podijeljena na još četiri mape: **images, lib, scripts** i **styles**. U svakoj mapi se nalazi određena vrsta datoteka, te je u mapu **images** moguće spremiti slike koje je korisnik aplikacije prenio na poslužitelj.

5.3. Izrada stranica aplikacije

Ključni korak izrade web aplikacije predstavlja izrada html stranica koje će se slati do krajnjeg korisnika. Jedna od ključnih stranica koja se dijeli između svih ostalih stranica je **layout** stranica, tj. zajednička stranica koja sadrži zaglavlje i podnožje, te stilske datoteke sa zajedničkim elementima. Ta stranica ima definirana četiri glavna dijela:

- „Glava“ web stranice
- Zaglavlje
- Glavni dio
- Podnožje

U „glavi“ web stranice se nalaze skripte i reference na statičke datoteke (te sam naslov trenutne stranice) koje se koriste na trenutnoj stranici. Ovo je jedan od glavnih dijelova html web stranica stoga je prisutan i u ovom projektu.

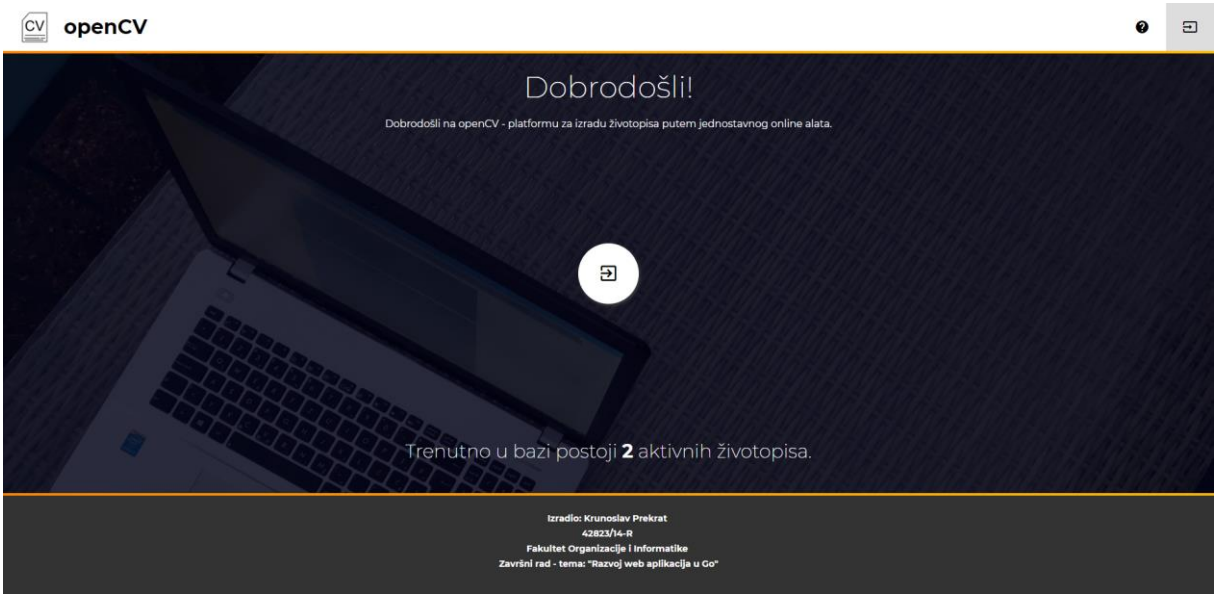
Zaglavlje web stranice sadrži zajednički dio koji je jednak na svim stranicama aplikacije, te on sadrži naziv aplikacije sa pripadajućom ikonom, kao i dugmad za izvršavanje određenih akcija kao što su otvaranje glavnog izbornika, odlazak na stranicu pomoći i prijavu, tj. odjavu iz aplikacije.

Glavni dio predstavlja sadržaj stranice koja se učitava kao dio layout stranice. Kao što mu i ime kaže, glavni dio sadrži sve glavne elemente određene web stranice.

Podnožje sadrži neke informacije o aplikacije dostupne na svakoj stranici, te se ovisno o učitanoj stranici sadržaj podnožja može mijenjati (dodavati).

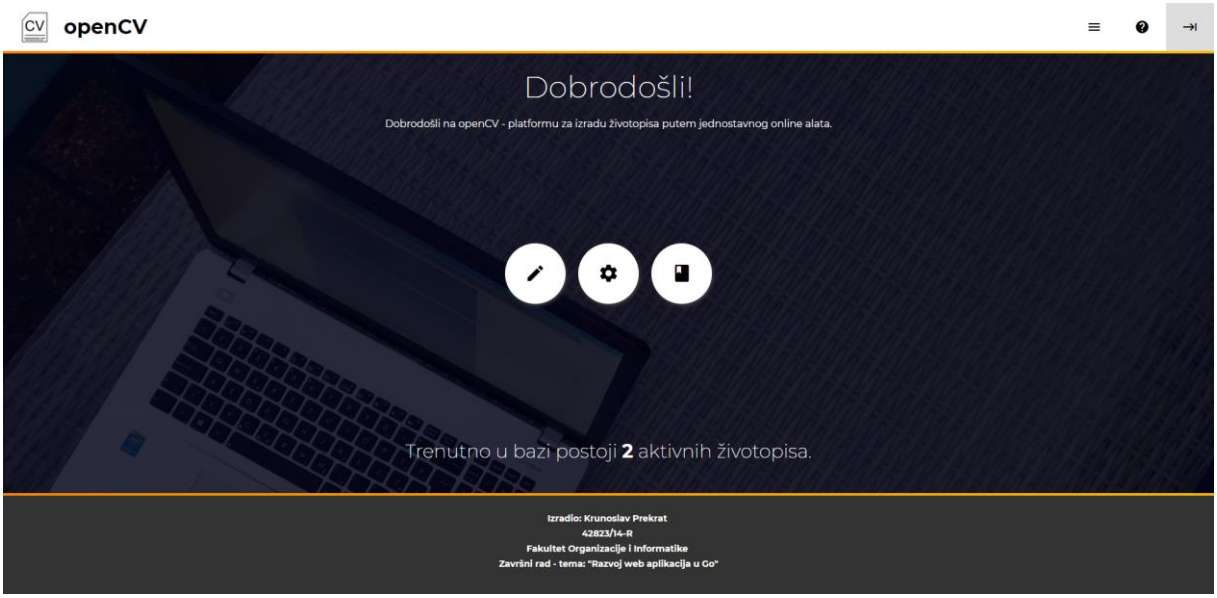
Važno je napomenuti da je aplikacija izrađena sa idejom da se kao model stranice uvijek kao jedan od parametara proslijeđuje parametar pod nazivom **BasePage** (osnovna stranica), čiji se dijelovi mogu pročitati na zajedničkom dijelu stranica, tj. layout-u. Taj parametar predstavlja strukturu sa nekoliko vrijednosti, a među vrijednostima se nalaze naziv stranice i podaci o korisniku koji je trenutno prijavljen u aplikaciju. Na takav način, glavna stranica uvijek može doći do tih podataka neovisno o specifičnoj stranici koja je učitana, te će obradu predložaka stranica uvijek biti moguće izvršiti.

Uz model stranice i njegove vezane atribute, u sklopu predložaka na glavnoj stranici se nalaze definirana mjesta na kojima će se ubaciti dijelovi sa pojedinačnih stranica, kao što su reference na **css** i **js** datoteke.



Slika 4: Izgled početne stranice web aplikacije

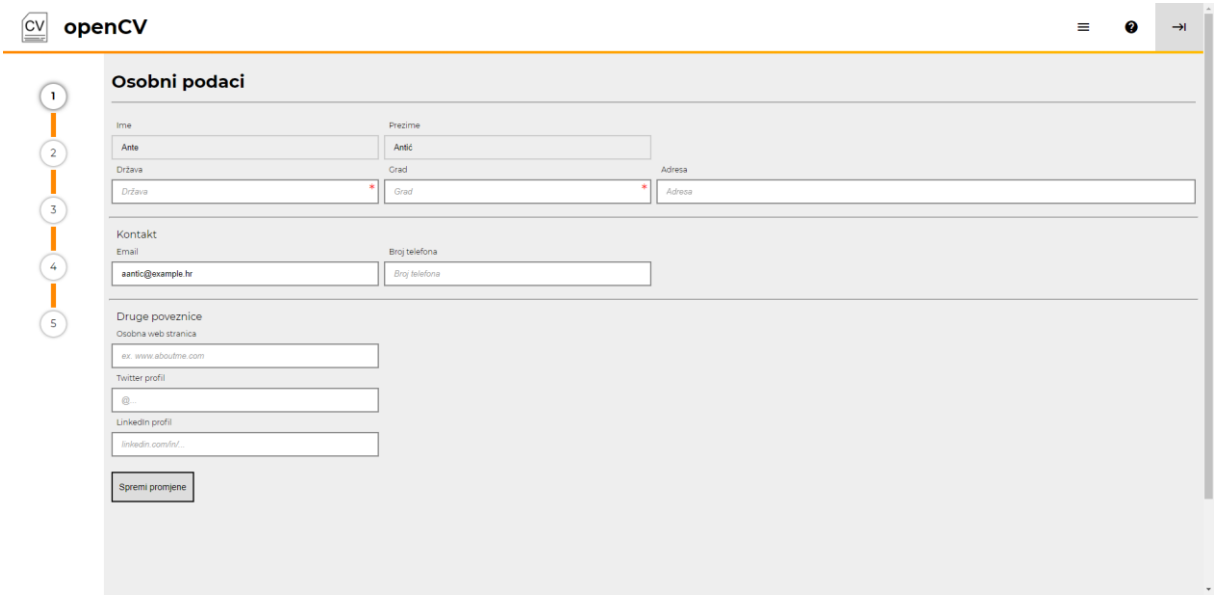
Na slici 3. vidljivi su glavni elementi stranice web aplikacije za izradu životopisa. U zaglavlju se nalaze spomenute akcije za pomoć i prijavu u aplikaciju, dok se u podnožju nalaze informacije o projektu. Prikazana stranica je početna stranica aplikacije, te sadrži poruku dobrodošlice, prečac na stranicu s registracijom i informacija o trenutnom broju životopisa u bazi podataka. Ukoliko se korisnik prijavi u aplikaciju s ispravnim korisničkim podacima, prethodno spomenuti ekran će se promijeniti, kako je prikazano na slici 4.



Slika 5: Izgled početne stranice web aplikacije nakon uspješne prijave u aplikaciju

Kako je vidljivo na slici 4., sadržaj zaglavlja i glavnog dijela stranice se nakon korisničke prijave promijenio. Zaglavlje sada sadrži i akciju za otvaranje glavnog izbornika, dok se akcija za prijavu promijenila u akciju za odjavu iz aplikacije. U glavnom dijelu stranice izbor dostupnih prečaca također se promijenio.

Ključna stranica cijele aplikacije je stranica pod nazivom **editor**, te predstavlja alat za izradu i uređivanje životopisa. Kada je korisnik uspješno prijavljen u aplikaciju otvara mu se pristup do spomenute stranice. Stranica ima pet sekcija podijeljenih prema strukturi životopisa, te je stranici moguće pristupiti putem jednog od dostupnih prečaca.



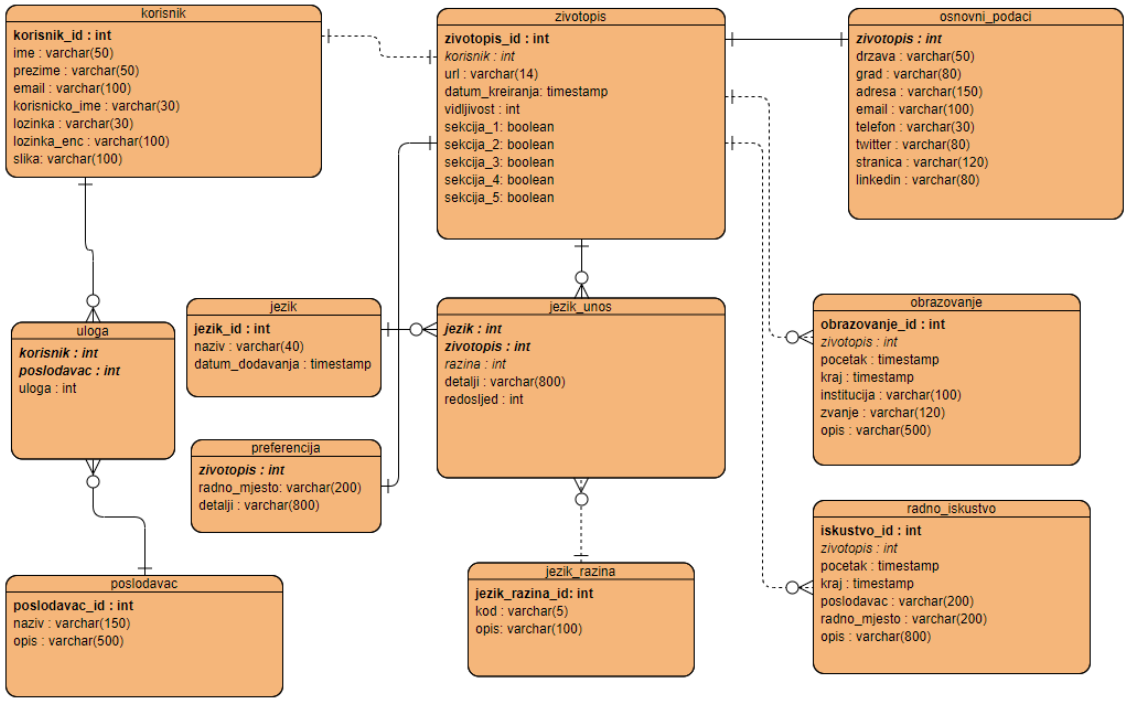
Slika 6: Stranica s alatom za uređivanje životopisa

Unutar glavne sekcije stranice, nalaze se dva panela – panel sa navigacijom po sekcijama životopisa i glavni panel sa poljima za uređivanje podataka unutar sekcije. Ovisno o odabranoj sekciji, na stranici će se prikazati određeni skup podataka. Na dnu sekcije se nalazi gumb za spremanje podataka, te ovisno o odabranoj sekciji, taj gumb može voditi na slanje ispunjene forme putem POST metode, ili može putem asinkronog poziva pozvati metodu na poslužitelju koja će zatim spremiti dobivene podatke. Takav pristup se koristi kada sekcija sadrži listu grupa podataka, kao što su npr. sekcija s obrazovanjem i jezicima.

Ostatak stranica kroz aplikaciju koristi dizajn sličan stranici sa slike 5, uz izmjene i prilagodbe vezane uz specifičnu stranicu. Izgled zaglavlja i podnožja je na svim stranicama isti, sa svrhom da se korisnika ne zbunjuje s posebnim izgledom dijelova na specifičnim stranicama.

5.4. Struktura baze podataka i spajanje kroz aplikaciju

Baza podataka je ključan dio web aplikacije, te se u aplikaciji za izradu životopisa koristi za spremanje podataka o korisnicima, dijelovima životopisa i poslodavcima. Za izradu baze podataka odabran je sustav PostgreSQL. Baza podataka je strukturirana sa nekoliko tablica prikazanih putem ERA dijagrama:



Slika 7: ERA dijagram baze podataka

Kroz ERA dijagram na slici 6., vidljiva je struktura baze podataka, te su prikazane sve relacije koje se koriste u web aplikaciji za izradu životopisa.

Osnovna relacija koja se koristi za prijavu / odjavu iz aplikacije je relacija **korisnik**. Ova relacija sadrži osnovne podatke o korisnicima koji imaju registriran korisnički račun u aplikaciji, a podaci koje relacija sadrži su: id korisnika, ime, prezime, email, korisničko ime, lozinka i naziv slikovne datoteke od korisnika. Važno je napomenuti da je za potrebe testiranja aplikacije lozinka dostupna u dva oblika – standardnom i kriptiranom. Prilikom prijave u aplikaciju za provjeru točnosti podataka koristi se kriptirani oblik lozinke, dok je standardni oblik dostupan zbog jednostavnije provjere podataka od testnih korisnika.

Uz korisnika, druga ključna relacija je **zivotopis**. Ta relacija sadrži nekoliko ključnih podataka vezanih uz životopise, te sadrži referencu na korisnika kojem taj životopis pripada –

trenutno je to uvijek veza 1 na 1 (kako je prikazano na ERA dijagramu), no kroz daljnji razvoj aplikacije može doći do potrebe da ta veza bude 1 na više. Relacija zivotopis sadrži sljedeći skup podataka: id životopisa, id korisnika, url adresu životopisa, datum kreiranja životopisa, vidljivost i nekoliko atributa koji predstavljaju kompletnost određenih sekcija životopisa. Ova relacija se referencira kroz većinu ostalih relacija, tako da predstavlja ključan dio cijelog sustava.

Relacija **osnovni_podaci** predstavlja podatke prikazane na prvoj sekciji životopisa. Povezana je vezom 1 na 1 s relacijom zivotopis, sa svrhom da se odvoje slojevi podataka po sekcijama kroz zasebne relacije. Relacija sadrži sljedeće podatke: id životopisa, država, grad, adresa, email (ukoliko se razlikuje od email-a za prijavu u aplikaciju), telefon, poveznica na twitter profil, poveznica na vlastitu web stranicu, poveznica na linkedin profil.

Dostupno je još nekoliko relacija koje prikazuju podatke o određenim dijelovima životopisa, a to su sljedeće relacije:

- **obrazovanje** – jedan ili više zapisa o prethodnom obrazovanju korisnika, sadrži osnovne podatke vezane uz obrazovanje korisnika kao što su početak i kraj obrazovanja, obrazovna ustanova, zvanje i opis obrazovanja.
- **jezik_unos** – jedan ili više zapisa o govornoj sposobnosti određenog jezika, te podaci o detaljima vezanim uz poznavanje jezika, kao detalji obrazovanja. Relacija se veže na još dvije relacije: **jezik** i **jezik_razina**, a one definiraju o kojem se točno jeziku radi i o kojoj jezičnoj razini se radi.
- **radno_iskustvo** – jedan ili više zapisa o prethodnom radnom iskustvu. Struktura relacije je jako slična relaciji obrazovanje, te prikazuje podatke o prethodnom radnom iskustvu korisnika.
- **preferencija** – relacija sadrži naziv i opis preferiranog radnog mjesta za trenutno prijavljenog korisnika. Predstavlja točno jedan zapis po korisniku.

Uz spomenute relacije dostupne su još dvije relacije koje sadrže podatke o poslodavcu kojeg određeni korisnik može registrirati u aplikaciji. Ukoliko korisnik ima administratorska prava nad određenim poslodavcem (upisanim poduzećem), tada taj korisnik može koristiti alat za uspoređivanje životopisa u aplikaciji. Na taj način, kroz aplikaciju je omogućeno jednostavno regrutiranje kandidata i usporedba njihovih podataka.

Za potrebe buduće funkcionalnosti aplikacije, relacija **uloga** se koristi za dodjeljivanje uloge korisniku unutar poduzeća (relacija **poslodavac**). Ukoliko se određeni korisnik zaposli unutar poduzeća, kroz aplikaciju je to moguće evidentirati, te se ta informacija prikazuje kod drugih korisnika ukoliko uspoređuju životopis zaposlenog korisnika.

Nakon izrade baze podataka prema prethodno definiranoj strukturi, aplikaciji je potrebno mogućiti spajanje na bazu podataka. Programski jezik Go, za potrebe spajanja na različite relacijske baze podataka, ima dostupan niz upravljačkih paketa koji su detaljnije opisani u poglavlju 4.3. Za potrebe spajanja aplikacije za izradu životopisa, koristi se postgreSQL paket za spajanje na bazu podataka.

Kao prvi korak povezivanja aplikacije s bazom potrebno je dohvatiti odgovarajuće podatke za spajanje, a oni se nalaze u konfiguracijskoj datoteci od aplikacije. Čitanjem konfiguracijske datoteke moguće je pročitati liniju pod nazivom "ConnString" (skraćeno od eng. connection string), koja predstavlja jednolinijski tekst sa svim potrebnim informacijama za spajanje na bazu podataka. Putem naredbe **sql.Open** otvara se veza prema aplikaciji, te se objekt s otvorenom aktivnom vezom na bazu podataka vraća do drugih funkcija. Ukoliko je veza na bazu uspješno uspostavljena, moguće je izvršiti upit iz aplikacije:

```
query = "SELECT zivotopis_id, url, vidljivost, sekcija_1::text as sekcija1, sekcija_2, sekcija_3, sekcija_4, sekcija_5 FROM zivotopis WHERE korisnik=" + userid + ";"

rows, err := db.Query(query)
```

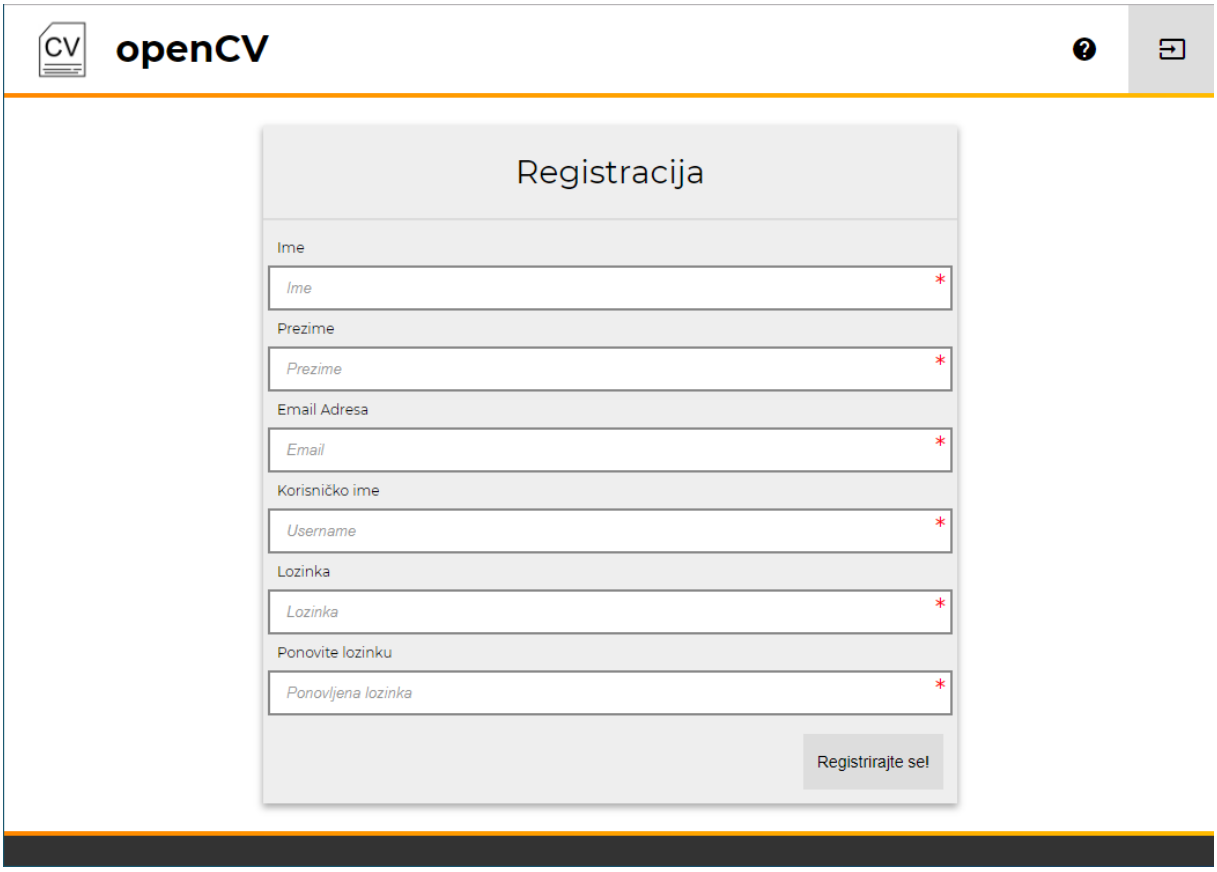
Prethodni kod će izvršiti upit nad bazom podataka i ukoliko se upit uspješno izvrši – vratiti skup redova kao rezultat izvršenog upita. Dobiveno rezultat je potrebno obraditi pomoću funkcije **rows.Scan**, koja će podatke iz zasebnog reda spremiti u varijable unutar programskog jezika Go.

```
for rows.Next() {
	rows.Scan(&id, &url, &visibility, &sec1, &sec2, &sec3, &sec4, &sec5)
	sec1bool, _ = ParseDBBool(sec1)
	sec2bool, _ = ParseDBBool(sec2)
	...
}
```

Na prethodnom kodu vidljiv je način na koji se podaci spremaju u zasebne varijable. Isto tako prikazan je i jedan od nedostataka programskog jezika Go – bool tip podatka iz PostgreSQL baze podataka nije moguće automatski spremiti u varijablu, već je potrebno prvo varijablu spremiti u **string** tip podatka, a zatim pomoću funkcije izvršiti konverziju. Na tako definiranim principima moguće je izvršiti svaki upit na bazu podataka iz aplikacije.

5.5. Sustav za prijavu / registraciju

Bez sustava za prijavu i registraciju korisnika, web aplikaciju poput ove nije moguće realizirati. Kako bi se omogućio jednostavan pristup korisniku, obrazac za registraciju mora biti jednostavna i mora biti jednostavno doći do nje.



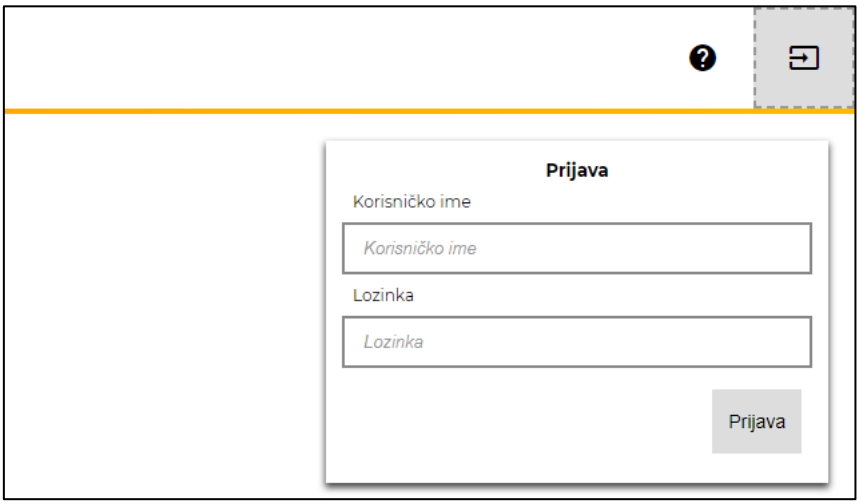
Slika 8: Obrazac za registraciju unutar aplikacije

Registracija se vrši pomoću obrasca prikazanog na slici 7., te je za uspješnu registraciju potrebno unijeti neke osobne podatke (Ime i Prezime) te nekoliko podataka pomoću kojih će se nakon registracije vršiti prijava u aplikaciju. Prilikom slanja podataka iz obrasca prema poslužitelju, izvršit će se provjera poklapanja lozinke. Ukoliko je provjera uspješna, na poslužitelju će se obraditi poslani podaci, te će se prije spremanja u bazu podataka lozinka kriptirati. Kriptiranje će se izvršiti pomoću SHA256 algoritma, kako je prikazano u sljedećem isječku koda:

```
h := sha256.New()
h.Write([]byte(password))
passwordEnc := base64.URLEncoding.EncodeToString(h.Sum(nil))
```

Nakon kriptiranja lozinke, svi podaci će se pomoću upita spremiti na bazu podataka, ukoliko korisnik nije unio već postojeću (u bazi) e-mail adresu ili korisničko ime koje već postoji u sustavu. Ukoliko dođe do greške prilikom procesa registracije, korisniku će se prikazati greška iznad obrasca za registraciju.

Ukoliko je registracija protekla uspješno, korisniku će se prikazati poruka o uspješnoj registraciji i informaciji o prijavi u aplikaciju. Prijava se može izvršiti sa bilo kojeg ekrana (jer je dio layout-a), a do obrasca za prijavu dolazi se klikom u gornji desni ugao aplikacije.



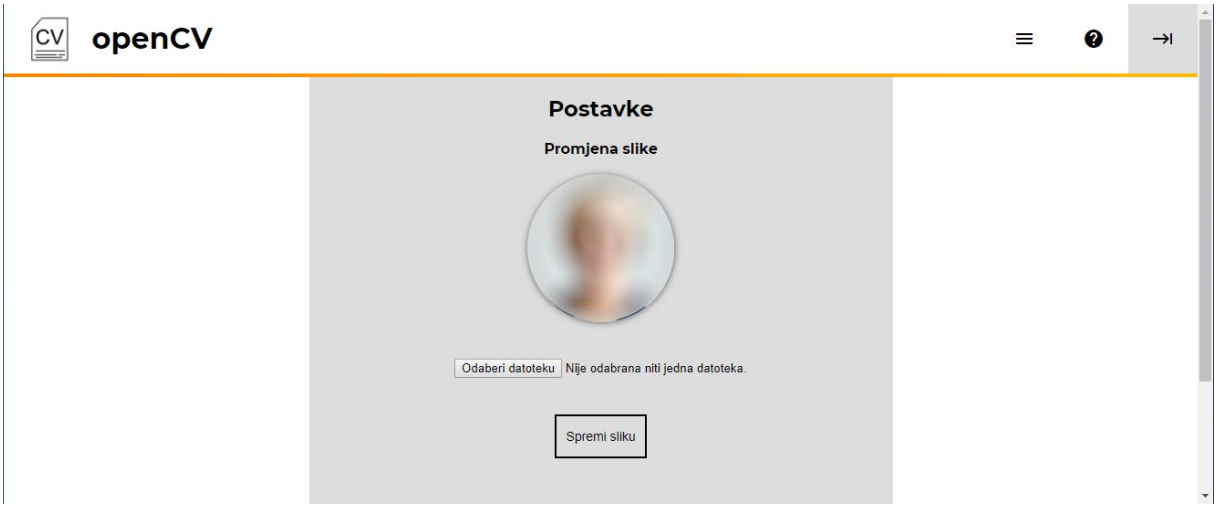
Slika 9: Obrazac za prijavu u aplikaciju

Prilikom prijave potrebno je, kroz obrazac za prijavu (slika 8) unijeti korisničko ime i lozinku. Provjerom unesenih podataka aplikacija će omogućiti korisniku pristup ostalim mogućnostima aplikacije ukoliko su uneseni ispravni podaci točni. Prilikom provjere potrebno je ponoviti proces kriptiranja lozinke i slanja takve lozinke do baze podataka. Ukoliko prijava prođe uspješno, podaci od korisnika se dohvaćaju i spremaju u aktivnu sesiju. Taj proces se obavlja s razlogom da se prilikom odlaska na bilo koju stranicu aplikacija ne mora uvijek spajati s bazom kako bi došla do korisničkih podataka, već su ti podaci dostupni u lokalnoj sesiji, spremljeni kroz kolačić na korisničkom računalu.

U programskom jeziku Go sesije nisu standardni dio net/http paketa, već je za korištenje takvih funkcionalnosti potrebno koristiti poseban Go paket. Za upravljanje sesijom korišten je Go paket "github.com/gorilla/sessions", a paket sadrži niz funkcija koje omogućuju jednostavno korištenje sesija. Ukoliko se korisnik u nekom trenutku želi odjaviti iz aplikacije, trenutna sesija za tog korisnika se izbriše.

5.6. Prijenos datoteke na poslužitelj

Jedna od mogućnosti aplikacije za izradu životopisa je dodavanje vlastite slike za svakog korisnika kako bi se omogućio lakši pregled više životopisa istovremeno. Svaki prijavljeni korisnik ima pristup stranici s postavkama, te kroz tu stranice može odabrati vlastitu profilnu sliku i prebaciti je na poslužiteljsko računalo. Slika će se, nakon spomenutog procesa, prikazivati u glavnom izborniku aplikacije kada je korisnik prijavljen.



Slika 10: Stranica s postavkama

Na slici 9. prikazana je stranica s postavkama na kojoj je moguće promijeniti sliku korisnika. Korisnik odabire datoteku sa vlastitog računala, te klikom na gumb „Spremi sliku“ prenosi odabranu slikovnu datoteku na poslužitelj. Izvršavanjem prethodne akcije, do aplikacije će doći http zahtjev koji u sebi sadrži podatke s obrasca, te se putem funkcije **ParseMultipartForm** obrađuje dobiveni sadržaj obrasca. Iz obrađenog zahtjeva aplikacije će pročitati sadržaj datoteke kao i njen naziv, te će pročitani sadržaj biti potrebno spremiti.

Prilikom obrade sadržaja obrasca, početna provjera unutar aplikacije je provjera veličine datoteke. Datoteka ne smije prijeći određenu veličinu prilikom prijenosa, a ona standardno iznosi 10 megabajta. Nakon provjere veličine, aplikacija provjerava da li je datoteku moguće dohvatiti, tj. da li ona uopće postoji. Ukoliko druga provjera prođe uspješno, aplikacija izvršava zadnju provjeru – provjeru tipa datoteke. Aplikacija će dopustiti prijenos nekoliko slikovnih datoteka (jpg, jpeg i png formati), te nakon završne provjere prelazi na spremanje datoteke.

Datoteka se kreira putem **ioutil.TempFile** funkcije, a naziv datoteke se formira na temelju nasumičnog skupa znakova i ID-ja trenutno prijavljene osobe. Na taj način, stara

datoteka sa sadržajem slike korisnika (ukoliko ona postoji) se neće odmah izbrisati, tako da u se korisniku može omogućiti vraćanje stare slike ukoliko nije zadovoljan novom.

```
tempFile,err := ioutil.TempFile("static/images/profiles", newFilename)

defer tempFile.Close()

fileBytes, err := ioutil.ReadAll(file)

tempFile.Write(fileBytes)
```

Prethodno prikazani isječak koda izvršava spremanje slikovne datoteke na poslužiteljsko računalo. Kako bi se datoteka kasnije mogla evidentirati kao aktivna slika za korisnika, naziv datoteke je potrebno spremiti u bazu podataka.

U relaciji **korisnik** nalazi se polje pod nazivom "slika", a ono predstavlja dodijeljeni naziv slike koju je korisnik prenio na poslužitelj. U slučaju da korisnik nije prenio sliku, polje će biti prazno, a prilikom prikaza slike na aplikaciji koristit će se osnovna slika za sve korisnike.

5.7. Slanje elektroničke pošte

Prilikom izvršavanja određenih akcija kroz aplikaciju, poželjno je korisniku javiti status o izvršavanju nekih akcija putem email poruke. Taj proces se odrađuje kontaktiranjem SMTP poslužitelja koji će poslati email poruku određenog sadržaja na korisničku email adresu. Na aplikaciji za izradu životopisa, primjer takvog slanja email poruke nalazi se već prilikom registracije u aplikaciju. Kada se novi korisnik uspješno registrira u aplikaciju, na njegovu se email adresu šalje poruka sa tekstom dobrodošlice, kao što je prikazano na slici 10:

Slika 11: Pozdravna email poruka openCV aplikacije

Proces slanja email poruke u programskom jeziku Go sastoji se od nekoliko koraka.

Iz konfiguracijske datoteke aplikacija za početak mora pročitati „svoju“ email adresu, tj. email adresu s koje će se poslati email poruka do korisnika. Čitanje konfiguracijske datoteke je već spomenuto u ranijem poglavlju, a iz datoteke je potrebno dohvatiti dvije vrijednosti:

```
config, err := framework.GetConfig()

from := config.SiteMail

pass := config.SiteMailPass
```

Konfiguracijski objekt će vratiti email adresu s koje se mail šalje, kao i lozinku za autentifikaciju na email servisu, kako bi bilo moguće autentificirati aplikaciju kroz servis. Nakon uspješnog učitavanja konfiguracijskih vrijednosti, sadržaj poruke će se kreirati pomoću nekoliko vrijednosti, kao što su **predmet** poruke, **primatelj** i **tijelo** poruke, tj. glavni sadržaj poruke. Nakon spremanja kreiranog teksta u string varijablu, email će se poslati putem sljedeće naredbe:

```
err = smtp.SendMail("smtp.gmail.com:587",
      smtp.PlainAuth("", from, pass, "smtp.gmail.com"),
      from, []string{to}, []byte(msg))
```

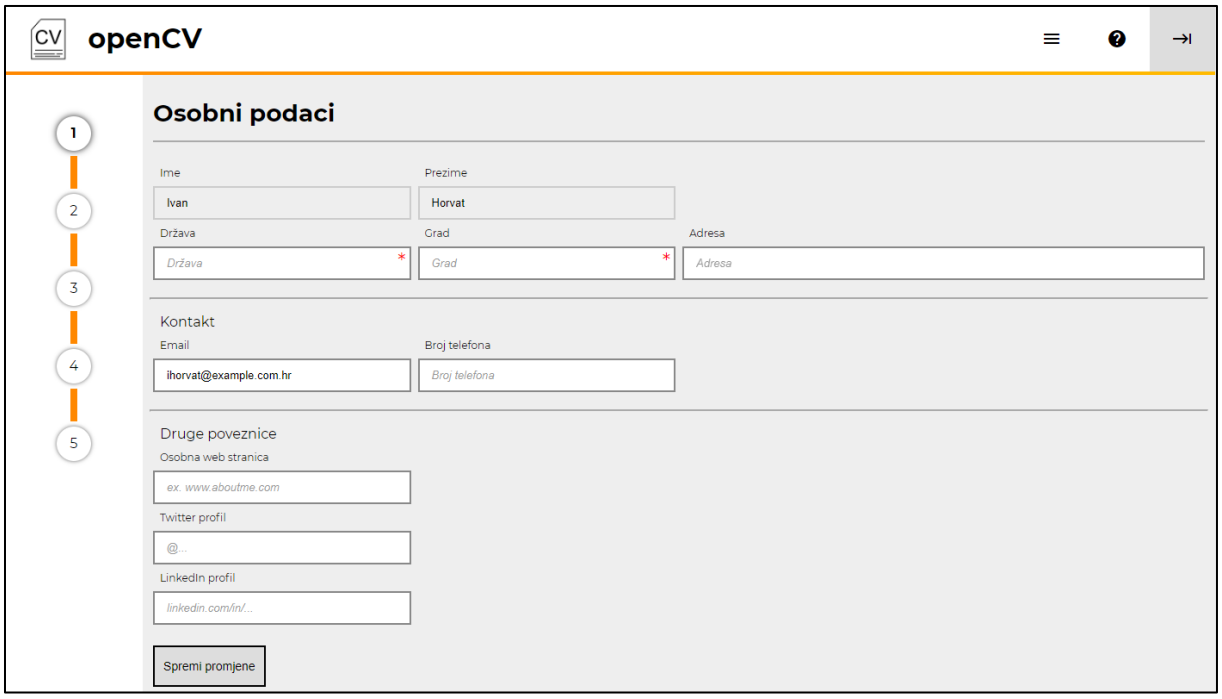
Prethodno definirani isječak koda odraditi će slanje email poruke kroz nekoliko koraka:

1. Autentifikacija na email poslužitelju
2. Obrada podataka vezanih uz slanje maila
3. Slanje emaila na korisničku email adresu

Ukoliko dođe do greške prilikom slanja email poruke, aplikacija će kroz konzolni prozor javiti grešku. Email funkcionalnosti je moguće koristiti i kroz ostale dijelove aplikacije, npr. prilikom usporedbe kandidata kroz neko poduzeće – kandidatu će biti poslana email poruka da je njegov / njezin životopis u procesu razmatranja kod određenog poslodavca. Također, na određenim stranicama postoji mogućnost slanja email poruke na više korisničkih email adresa istovremena, što će se najjednostavnije ostvariti korištenjem **goroutine** funkcionalnosti.

5.8. Izrada i pregled životopisa

Ključna funkcionalnost openCV aplikacije, kako je to ranije spomenuto, je jednostavna izrada životopisa putem čarobnjaka za izradu, te prikaz životopisa u web formatu (kao web stranica). Za potrebe kreiranja životopisa potrebno je otići na stranicu od „uređivača“, tj. čarobnjaka za izradu životopisa, a stranica je dostupna kroz poveznice na glavnom izborniku ili na početnoj stranici. Prvim odlaskom na stranicu, u bazi podataka kreirat će se unos u relaciji "zivotopis", koji će sadržavati prazne vrijednosti za trenutno prijavljenog korisnika. Odlaskom na spomenutu stranicu, korisniku će se prikazati sljedeći ekran:



Slika 12: Obrazac za unos osnovnih podataka

Prvi ekran predstavlja unos osobnih podataka za trenutno prijavljenog korisnika. Putem ovog ekrana, korisnik unosi podatke vezane uz mjesto prebivališta, kontakt podatke i podatke o postojećim korisničkim računima na drugim mrežama. Taj ekran predstavlja osnovne podatke za unos, te bez osnovnih podataka ispunjenih na tom ekranu životopis ne može biti kreiran. Podatke poput imena i prezimena nije potrebno ispunjavati jer je njih potrebno ranije unijeti prilikom registracije. Email se također unosi putem registracijskog obrasca, no ukoliko korisnik želi koristiti drugačiji email u životopisu od onog koji se koristi kroz aplikaciju, taj podatak je moguće izmijeniti. Segment s drugim poveznicama je opcionalan, te ga nije potrebno ispuniti kako bi životopis bilo moguće dalje uređivati. Kroz programski kod, dohvat osnovnih podataka vrši se putem implementirane funkcije, sljedećim pozivom:

```
baseInfo, err := api.GetBaseInfo(cvdata.CVID)

...

func GetBaseInfo(cvid int) (*models.CVBaseInfoRecord, error) {

      db, err := GetDBInst()

      query = "SELECT drzava, grad, adresa, email, telefon, twitter,
      stranica, linkedin FROM osnovni_podaci WHERE zivotopis=" +
      strconv.Itoa(cvid) + ";"

      rows, err := db.Query(query)

      obj := new(models.CVBaseInfoRecord)

      defer rows.Close()

      for rows.Next() {

            rows.Scan(&obj.Drzava, &obj.Grad, &obj.Adresa, &obj.Email,
            &obj.Telefon, &obj.Twitter, &obj.Stranica, &obj.LinkedIn)

      }

      return obj, nil

}
```

Kroz prethodni dio koda prikazano je kako se dohvaćaju podaci iz baze podataka i spremaju u privremenu memoriju u programskom jeziku Go. Specifično, osnovni podaci sa životopisa će se dohvatiti iz baze podataka na temelju poslanog identifikatora životopisa. Nakon slanja upita do baze podataka, dobiveni objekt s podacima se obrađuje putem funkcije "rows.Scan", a dohvaćeni skup podataka se zatim sprema u predviđene varijable. Funkcija vraća strukturirani objekt sa podacima dohvaćenim iz prethodno izvršenog upita i podatke o greški ukoliko je došlo do nje prilikom izvršavanja funkcije.

```
data = models.CVBaseInfoPageModel{

      BasePage:   *pageData,

      CV:         *cvdata,

      ActivePage: activePage,

      Info:       *baseInfo,

}
```

Prethodni dio koda predstavlja kreiranje modela podataka koji se zatim obrađuje u sklopu html predložaka programskog jezika Go, te se pomoću podataka iz zadanog modela na stranici prikazuju tražene vrijednosti. U prethodnom slučaju, parametar "Info" predstavlja

dohvaćene podatke iz baze podataka, spremljene u objekt koji se zatim referencira unutar objekta koji se obrađuje u predlošku.

Klikom na gumb spremi, uneseni podaci se putem POST metode prijenosa podataka obrađuju na poslužitelju, te se šalju do baze podataka kako bi se trajno pohranili. Aplikacija prepoznaje da li se radi o zahtjevu s predajom podataka prema dohvaćenoj vrijednosti gumba za spremanje, tj. ukoliko vrijednost "submitBtn" kontrole obrasca nije prazna, dobiveni zahtjev je potrebno dodatno obraditi kako bi se ostali podaci s njega pohranili u bazi. Provjera se u programskom kodu izvršava na sljedeći način:

```
if r.PostFormValue("submitBtn") != "" {
      err = api.SaveBaseInfoData(r, cvdata.CVID)
      ...
}
```

Nakon preusmjeravanja na funkciju za spremanje podataka, potrebne vrijednosti iz dobivenog podatkovnog obrasca je potrebno dohvatiti. Podaci s obrasca su poslani POST metodom, tako da se dohvaćanje vrijednosti vrši na način:

```
country := r.PostFormValue("country")
city := r.PostFormValue("city")
address := r.PostFormValue("address")...
```

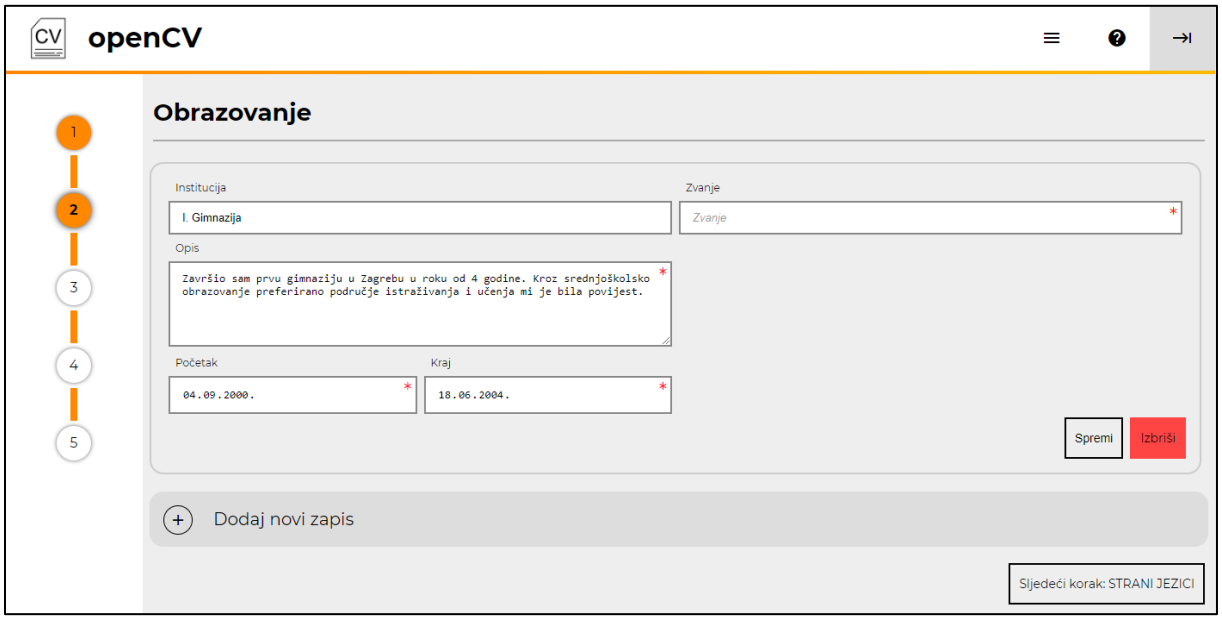
Gdje je parametar "**r**" http zahtjev koji dolazi od strane klijenta. Nakon obrade svih vrijednosti iz obrasca, određene vrijednosti se kombiniraju u SQL upit, koji se putem funkcije "fmt.Sprintf()" obrađuje u završni skup znakova koji se šalje prema bazi. Ukoliko već u bazi postoji unos u relaciji "osnovni_podaci" za trenutno uređivani životopis, podaci će biti ažurirani putem upita, a u suprotnom slučaju kreirat će se novi unos u prethodno navedenoj relaciji.

```
if foundCount == 0 {
      query = fmt.Sprintf("INSERT INTO osnovni_podaci (zivotopis,
      drzava, grad, adresa, email, telefon, twitter, stranica,
      linkedin) VALUES (%s,'%s','%s','%s','%s','%s','%s','%s','%s');",
      strconv.Itoa(cvid), country, city, address, email, phoneNumber,
      personalWebsite, twitterProfile, linkedinProfile)
      } else if foundCount == 1 {
            query = fmt.Sprintf("UPDATE osnovni_podaci SET drzava='%s',
            grad='%s',    adresa='%s',    email='%s',    telefon='%s',
            twitter='%s',    stranica='%s',    linkedin='%s'    WHERE
```

```
                zivotopis=%s;", country, city, address, email, phoneNumber,
                personalWebsite,      twitterProfile,      linkedinProfile,
                strconv.Itoa(cvid))
        } else {
return errors.New("pronađeni broj stranica nije u rasponu (0, 1)")
        }
```

Ukoliko je operacija spremanja podataka u bazu uspješno izvršena, trenutna stranica će se osvježiti te će korisnik vidjeti koji su podaci uneseni.

Sljedeći korak, nakon uspješnog ažuriranja osnovnih podataka, predstavlja unos prethodnog obrazovanja koje je korisnik obavio, te pripadajućih titula koje je korisnik stekao putem prethodnog obrazovanja. Prilikom odlaska na poveznicu „Sljedeći korak“ sa ekrana sa unosom osnovnih podataka, aplikacija će korisnika preusmjeriti na obrazac za ispunjavanje podataka o prethodnom obrazovanju. Obrazac je po svom izgledu sličan kao i prethodni obrazac, no sadrži nekoliko ključnih izmjena. Kako bi bilo moguće dodati više zapisa o prethodnom obrazovanju, ekran je podijeljen na odjeljke koji označuju različite zapise o obrazovanju, od početnih zapisa poput podataka o osnovnoj školi, pa sve do kasnijih zapisa o višem obrazovanju. Obrazac je prikazan na sljedećoj slici:



Slika 13: Obrazac za unos podataka o prethodnom obrazovanju

Kada korisnik prvi puta otvori ekran s unosom podataka o obrazovanju, korisnik će dobiti stranicu sa opcijom za dodavanje novog zapisa o obrazovanju. Pritiskom miša na

poveznicu za dodavanje novog zapisa aplikacija će kroz novi upit dodati prazan zapis u bazu podataka, te će se korisnička stranica osvježiti.

Nakon osvježavanja stranice, korisniku će biti vidljiv jedan prazan zapis te opcija za dodavanje još zapisa. U praznom zapisu moguće je urediti podatke o obrazovanju: obrazovna institucija, zvanje nakon položenog obrazovanja, opis obrazovanja kao i datumi početka i kraja obrazovnog procesa u odabranoj ustanovi. Nakon ispune obrasca, korisnik navedeni obrazac može spremiti.

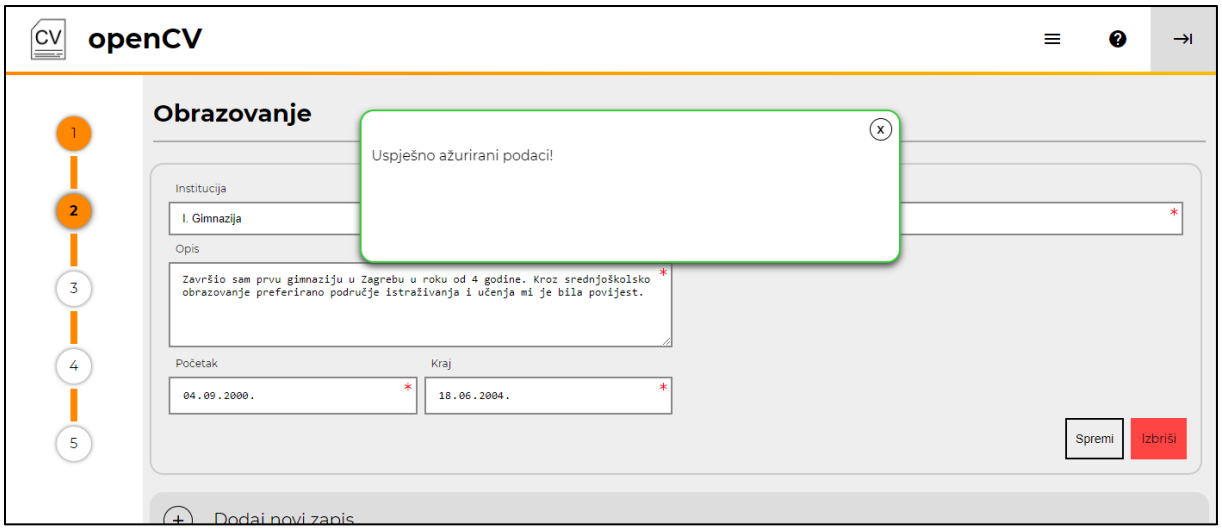
Pritiskom na gumb „Spremi“, korisnikov preglednik će putem asinkronog poziva kontaktirati aplikaciju na poslužitelju. Za kontaktiranje preglednik će koristiti navedenu web adresu prema aplikaciji ("/editor/edu/save"), te će se za prijenos sadržaja koristiti parametar "**data**". Navedeni parametar predstavlja JSON podatkovnu strukturu sa sadržajem svih polja unesenih kroz obrazac određenog zapisa. Zahtjev će biti poslan putem POST metode slanja podataka, kako bi se poslani podaci bolje zaštitili. Prilikom obrade zahtjeva na aplikaciji, prvi dio obrade je sličan kao i kod slanja osnovnih podataka:

```
r.ParseForm()

data := r.Form.Get("data")
```

U varijablu "**data**" spremit će se sadržaj iz obrasca, te će se dobiveni sadržaj deserijalizirati (pretvoriti iz skupa znakova u objekt) putem dostupne funkcije programskog jezika Go i paketa "encoding/json":

```
var eduRecord models.CVEduRecord

err := json.Unmarshal([]byte(data), &eduRecord)
```

Vrijednosti dobivenog objekta (parametri) će se zatim obraditi u aplikaciji tako da stvore niz znakova koji predstavlja upit na bazu podataka, kao što je bio i slučaj kod spremanja osnovnih podataka. Dobiveni upit će se zatim izvršiti na bazi podataka, te ovisno o dobivenoj grešci prilikom izvršavanja skupa operacija, do klijentskog preglednika doći će informacija o uspješnosti spremanja promjena. Kod obrasca za unos prethodnog obrazovanja, sva polja mogu biti prazna osim unesenih vrijednosti datuma koje moraju biti ispunjene, u protivnom, prilikom spremanja podataka u bazu podataka javit će se greška. Ukoliko su podaci uspješno pohranjeni u bazu podataka, tada će se u korisničkom pregledniku prikazati poruka kako je operacija protekla uspješno i kako su podaci ažurirani. Izgled takve poruke je prikazan na sljedećoj slici:



Slika 14: Poruka o uspješnosti spremanja podataka

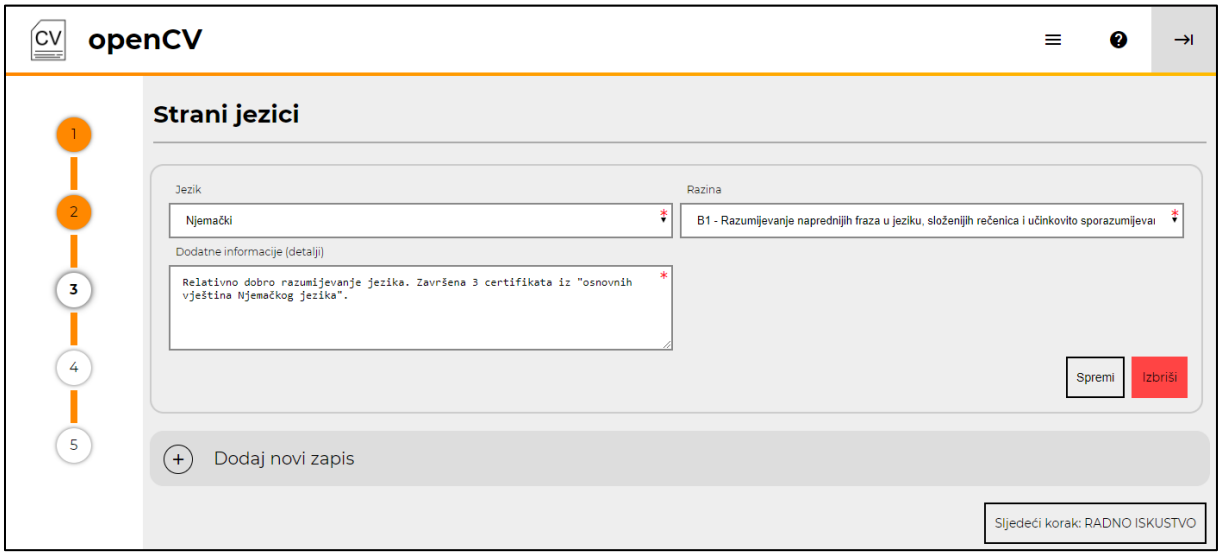
Kada je zapis sa uređenim podacima spremljen, na stranici će nakon osvježavanja biti vidljivi podaci iz tog zapisa, sortirani prema datumu kraja i datumu početka. Prilikom prikaza podataka o datumima, standardni Go template sustav datume prikazuje u html kodu na način na koji kontrole za odabir datuma ne mogu prepoznati zadane datume (krivo formatiranje). Zato, kako bi prikaz radio na korisničkom pregledniku, datumi se prilikom dohvata podataka iz baze spremaju u zasebno polje u prilagođenom formatu, što se obavlja na sljedeći način:

```
obj.PocetakFormat = fmt.Sprintf("%d-%02d-%02d", obj.Pocetak.Year(), obj.Pocetak.Month(), obj.Pocetak.Day())

obj.KrajFormat = fmt.Sprintf("%d-%02d-%02d", obj.Kraj.Year(), obj.Kraj.Month(), obj.Kraj.Day())
```

Pritiskom na gumb „Izbriši“, trenutno odabrani zapis o obrazovanju će se izbrisati iz baze podataka. Ova opcija može biti korisna ukoliko korisnik zabunom doda prevelik broj zapisa na stranicu, te se realizira na sličan način kao i dodavanje novog zapisa. Putem akcije na poslužitelju "/editor/edu/delete" šalje se identifikacijska oznaka odabranog zapisa prema poslužitelju. Aplikacija obrađuje zahtjev, te obradom dobivenog obrasca dohvaća se oznaka pomoću koje se zatim slaže upit prema bazi podataka koji će izbrisati zapis iz relacije "obrazovanje". Nakon uspješnog brisanja trenutno stranica na klijentskom pregledniku će se osvježiti, te ukoliko je brisanje prošlo uspješno, izbrisani zapis više neće biti prikazan. Ukoliko je došlo do greške prilikom brisanja podataka (npr. dobivena identifikacijska oznaka nije pronađena u bazi podataka) korisniku će se prikazati stranica s informacijama o grešci.

Nakon uspješnog dodavanja željenog broja zapisa o obrazovanju i spremanja izmjena nad njima, korisnika će aplikacija usmjeriti na sljedeće korak izrade životopisa – dodavanje

podataka o poznavanju stranih jezika. Procedura kod dodavanja, izmjene i brisanja stranih jezika poprilično je slična procedurama na prethodnoj stranici (obrazovanju). Proces dodavanja novog zapisa u stranom jeziku je praktički identičan onome kod dodavanja novog zapisa o obrazovanju – korisnik klikom na gumb „Dodaj novi zapis“ dodaje novi zapis o stranom jeziku u bazu podataka. Kod stranih jezika, za razliku od obrazovanja, polja „Jezik“ i „Razina“ nisu polja s ručnim unosom tekstualne vrijednosti, već su polja s odabirom jedne od ponuđenih opcija. Kako bi odabir opcija pravilno radio, prilikom dohvata podataka, uz upit za dohvaćanje zapisa o stranim jezicima, dohvaćaju se i podaci o dostupnim jezicima za odabir i podaci o dostupnim jezičnim razinama.

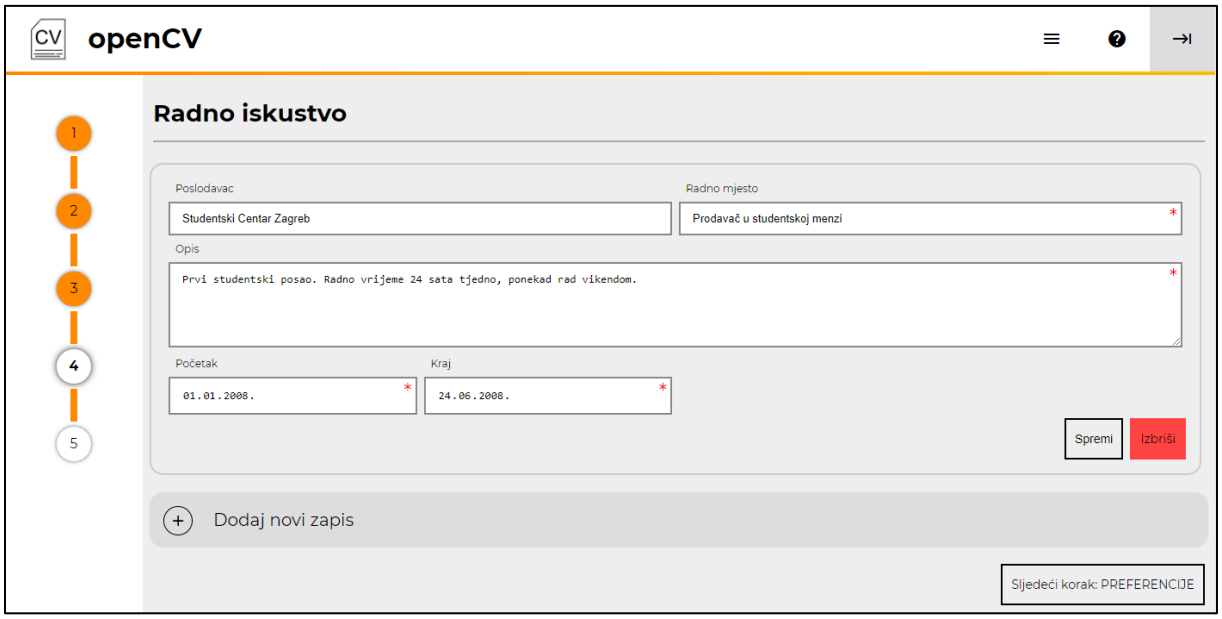


Slika 15: Obrazac za unos zapisa o stranom jeziku

Prilikom spremanja zapisa u bazu podataka, prema aplikaciji se šalje samo identifikacijska vrijednost jezika i razine, koja se zatim šalje prema bazi podataka, gdje ona predstavlja primarni ključ u relacijama "jezik" i "jezik_razina". Spremanje se vrši na sličan način kao i kod spremanja podataka o zapisu obrazovanja – putem asinkronog zahtjeva podatak se, spremljen u jedan JSON objekt pod nazivom "data", šalje od preglednika do poslužitelja, na kojem ga aplikacija obrađuje. Obradom JSON objekta dobivaju se tražene vrijednosti koje se zatim spremaju u bazi podataka. Brisanje zapisa se vrši na isti način kao i kod brisanja podataka o obrazovanju.

Važno je napomenuti, da relacija "jezik_unos" koja sadrži zapise o poznavanju stranih jezika, predstavlja slabi entitet u bazi podataka, između relacija "životopis" i "jezik". Stoga, prilikom dodavanja novog zapisa o poznavanju stranog jezika, nije moguće dodati više zapisa od broja stranih jezika koji se nalaze u bazi podataka.

Kao predzadnji korak izrade životopisa, dostupno je dodavanje radnog iskustva. Obrazac je sličan obrascu za dodavanje zapisa o obrazovanju, te je njegov izgled prikazan na sljedećoj slici:

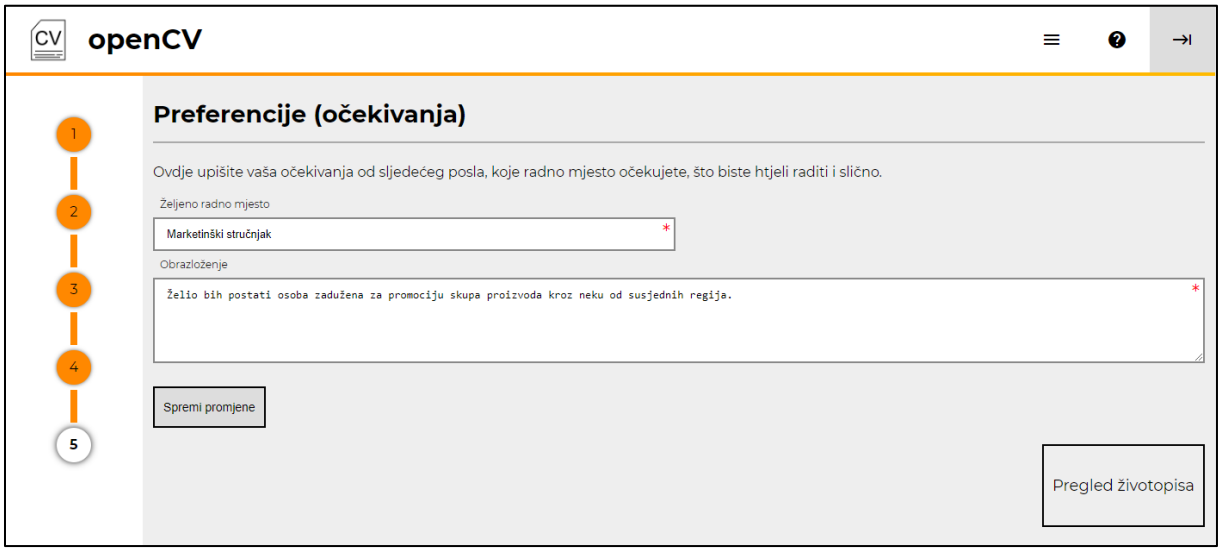


Slika 16: Obrazac za unos podataka o radnom iskustvu

Za potrebe dodavanja zapisa o radnom iskustvu potrebno je, kao i kod dodavanja zapisa o obrazovanju, unijeti datume početka i kraja određenog zaposlenja. Prilikom dohvata liste svih zapisa o radnom iskustvu, zapisi će se poredati prema unesenim datumima na svakom zapisu. Postupak spremanja i brisanja zapisa je isti kao i kod zapisa o obrazovanju, uz razliku u relaciji u koju se podaci upisuju na bazi podataka – za spremanja podataka o radnom iskustvu koristi se relacija "radno_iskustvo". Kao i kod obrazovanja, prilikom dohvata zapisa o radnom iskustvu, datume je, zbog prikaza na korisničkom pregledniku, potrebno formatirati u poseban format kako bi ih bilo moguće pravilno prikazati. Za asinkroni poziv prilikom spremanja podataka o radnom iskustvu koristi se sljedeći isječak koda (JavaScript):

```
var xhttp = new XMLHttpRequest();

    var json_data = "data=" + JSON.stringify(returnObj);

    xhttp.onreadystatechange = function () {...};

    xhttp.open("POST", "/editor/exp/save", true);

    xhttp.setRequestHeader("Content-Type",  "application/x-www-form-
    urlencoded");

    xhttp.send(json_data);
```

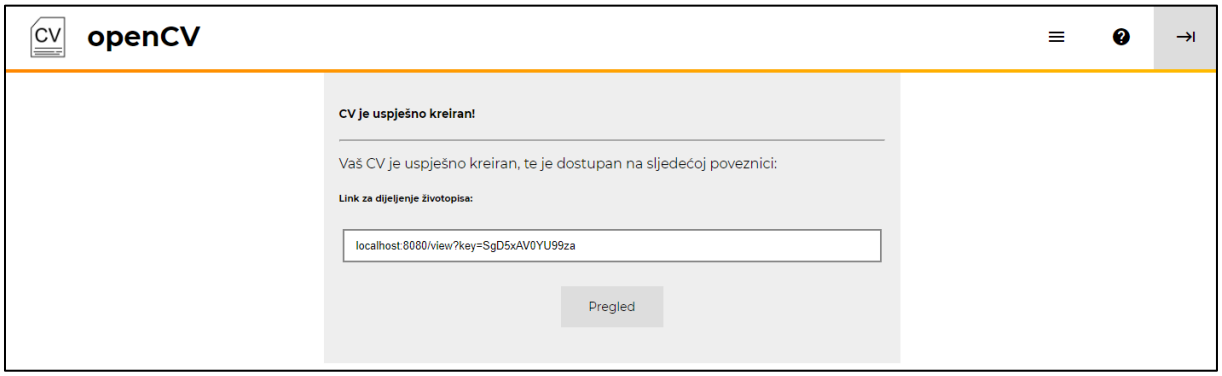
Zadnji korak prilikom izrade životopisa je upis osobne preferencije za željenu poziciju i mjesto rada. Kroz ovaj završni korak, korisnik će prikazati svoja osobna očekivanja od budućeg poslovnog okruženja, te će na taj način budućem poslodavcu dati do znanja da li je on pravi zaposlenik za tog poslodavca. Preferencija predstavlja jedan zapis u bazi podataka po jednom životopisu, te se automatski dodaje u bazu podataka kada korisnik spremi uneseni sadržaj kroz prikazanu stranicu:



Slika 17: Obrazac za unos korisnikovih očekivanja

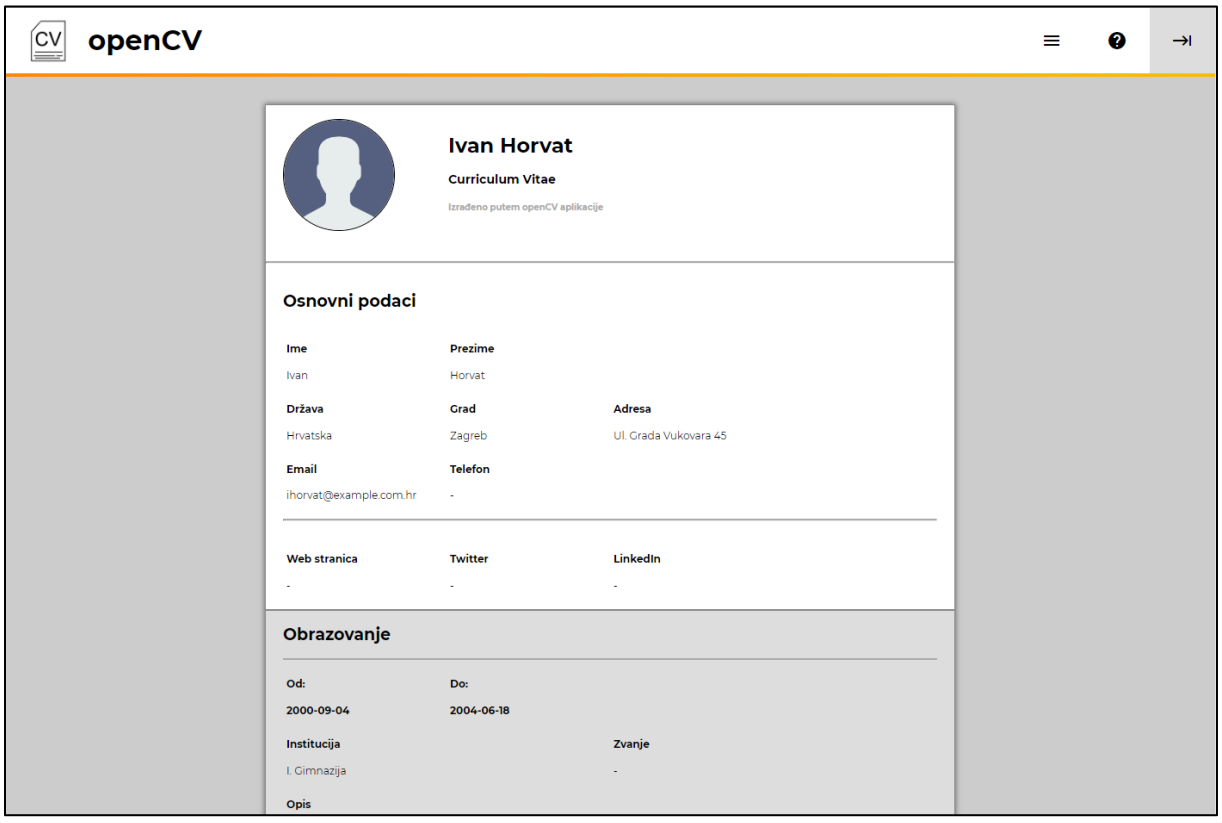
Uneseni podaci o korisničkim preferencijama unose se u bazu podataka predajom obrasca putem POST metode – isto kao i kod spremanja osnovnih podataka, prvog obrasca čarobnjaka za izradu životopisa. U bazu podataka se spremaju dva podatka – željeno radno mjesto i obrazloženje preferencije. Prema identifikacijskom broju trenutno aktivnog životopisa u bazi se ažurira određeni zapis o preferencijama. Nakon spremanja podataka, tj. obrade istih kroz aplikaciju na poslužitelju, životopis je u potpunosti ispunjen (ukoliko neki od prethodnih koraka nije preskočen), te se korisniku omogućuje da pregleda svoj životopis.

Klikom na gumb "Pregled životopisa", korisnik će biti preusmjeren na stranicu s porukom potvrde o uspješnoj izradi životopisa. Ukoliko korisnik nije ispunio prvu sekciju životopisa s osnovnim podacima (ostale sekcije mogu biti prazne), korisniku će se prikazati poruka o nedovršenosti životopisa, te će životopis morati dovršiti. U suprotnom, korisniku se prikazuje poruka „CV je uspješno kreiran“, te se korisniku prikazuje poveznica koju može podijeliti s drugim korisnicima, a poveznica otvara prikaz životopisa s podacima od korisnika koji ju je podijelio. Uz poveznicu za dijeljenje životopisa, na dnu stranice nalazi se prečac na pregled vlastitog životopisa.



Slika 18: Stranica s potvrdom o završenoj izradi životopisa

Odlaskom na pregled životopisa, korisniku će se prikazati sljedeća stranica:

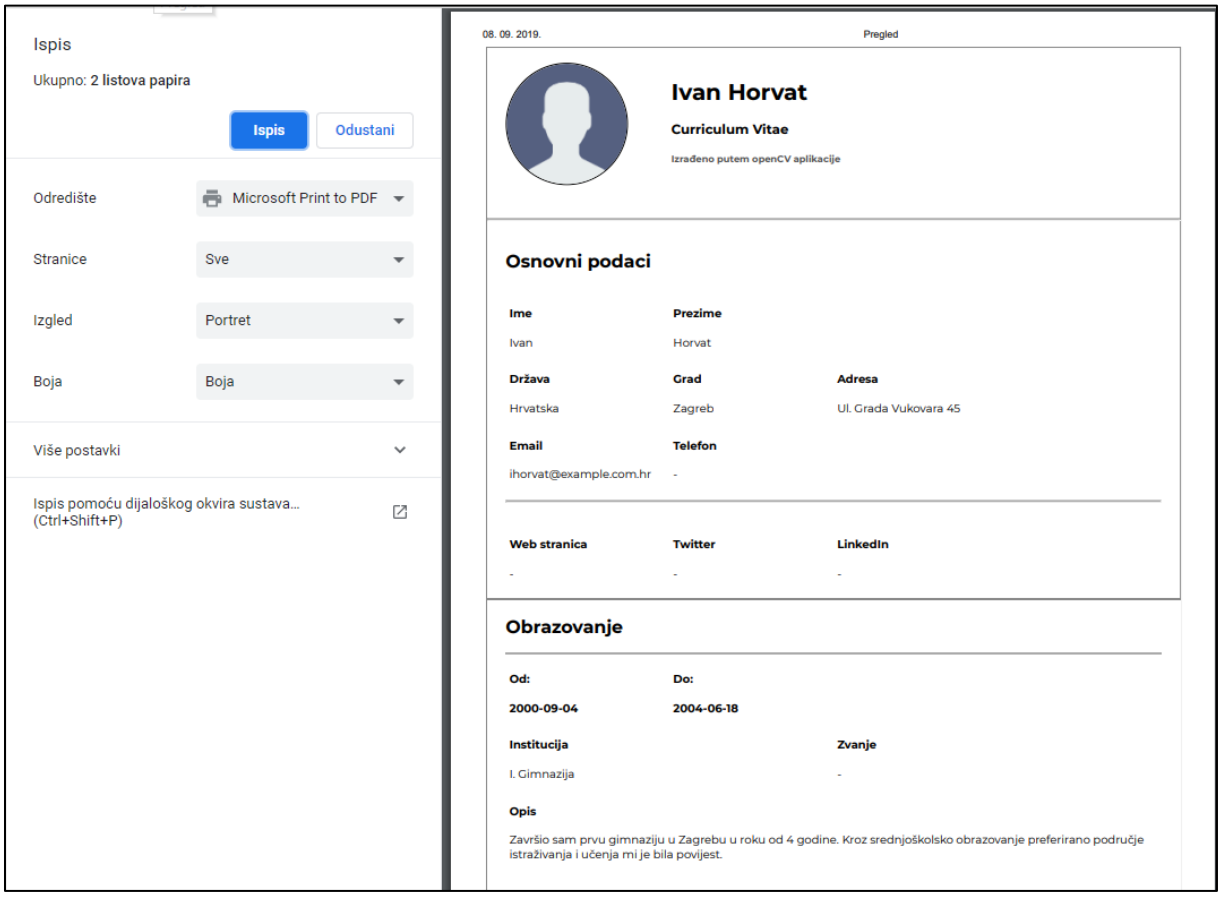


Slika 19: Stranica s pregledom životopisa

Pregled životopisa na strukturiran način prikazuje izrađeni životopis za određenog korisnika. Sekcije životopisa su posebno odvojene kako ne bi došlo do zabune prilikom pregleda podataka, te su zasebni zapisi na određenim sekcija također posebno odvojeni. Na

taj način, korisniku se prikazuje gotovi izgled životopisa nakon što ga korisnik izradi putem obrazaca dostupnih u alatu za izradu životopisa. Ukoliko se stranica otvori bez dodatnog parametra pod nazivom "key" (ključ), stranica će učitati podatke za trenutno prijavljenog korisnika. Ukoliko korisnik, prilikom odlaska na web lokaciju za pregled životopisa, unese parametar koji predstavlja identifikacijski ključ životopisa (niz od 14 znakova), tada će se korisniku prikazati životopis identificiran pomoću priloženog ključa. Ukoliko ključ nije pronađen u bazi podataka, korisniku će se prikazati greška. U slučaju da neprijavljeni korisnik pokuša pristupiti stranici za pregled životopisa bez ključa, aplikacija će ga preusmjeriti na početnu stranicu.

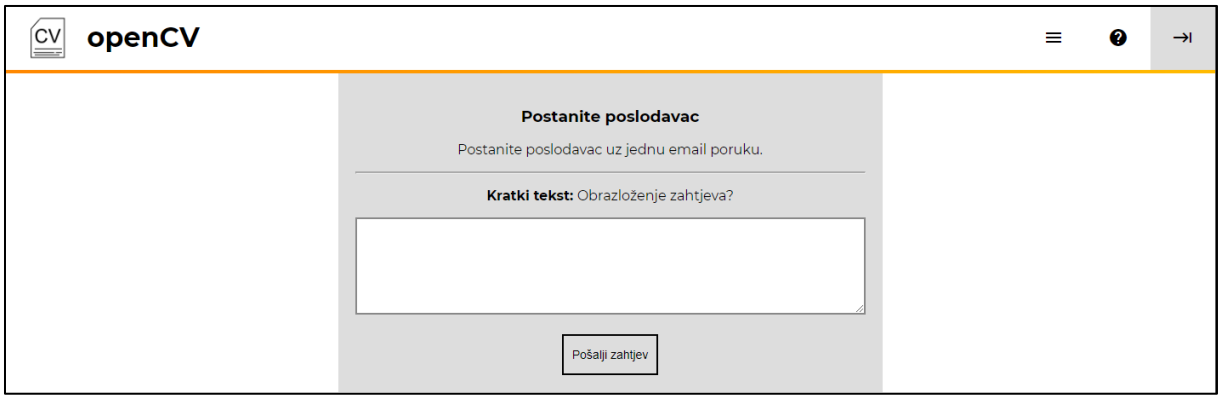
Na dnu stranice s pregledom gotovog životopisa, nalazi se gumb, čijim se klikom otvara životopis u prikazu za ispis. Prilikom takvog prikaza, određeni stilski dijelovi se uklanjaju sa stranice, te ostaje pregled životopisa s bijelom pozadinom, bez dodatnih dijelova iz zaglavlja ili podnožja aplikacije. Korisnik, ukoliko želi, dobiveni prikaz životopisa može ispisati putem pisača ili spremiti u nekom od formata koje omogućuje korišteni preglednik.



Slika 20: Prikaz životopisa za ispis

5.9. Funkcije poslodavca

Uz regularne korisnike koji kroz aplikaciju izrađuju životopise i dijele ih sa drugim korisnicima, kroz aplikaciju su dostupne funkcije koje koriste poslodavci. Kako bi određeni korisnik postao poslodavac, korisnik mora posjetiti stranicu "company/apply", na kojoj će predajom kratkog obrazloženja korisnik poslati email poruku do putem aplikacijske email adrese. Na email adresi od aplikacije pojavit će se poruka s unesenim zahtjevom i ID-jem korisnika kojeg je potrebno promovirati, tj. pretvoriti njegov korisnički račun u račun poslodavca.



Slika 21: Obrazac za slanje zahtjeva za korisnički račun poslodavca

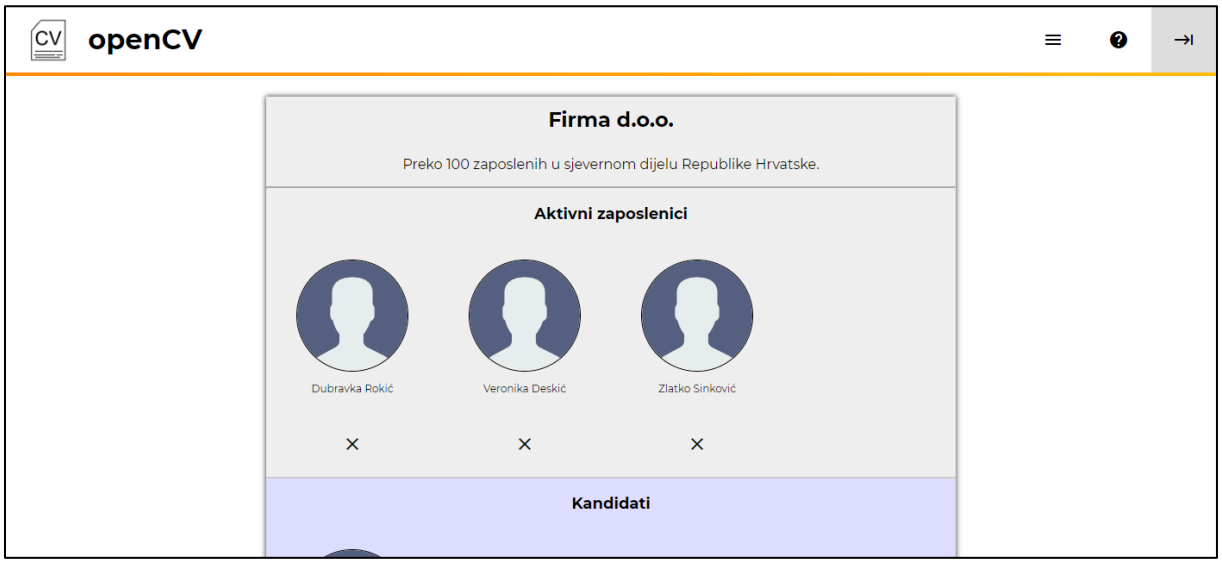
Putem dobivene email poruke, administrator aplikacije koji ima pristup email adresi može odobriti korisnikov zahtjev slanjem direktnog upita prema bazi podataka. Email poruka je složena na slijedeći način:

```
mime := "MIME-version:1.0;\nContent-Type:text/html;charset=\"UTF-8\";
subject := "Subject: Novi poslodavac za korisnika #" + strconv.Itoa(userID) + "!\n"
link := fmt.Sprintf("%s/company/activate?userid=%d", r.Host, userID)
body := fmt.Sprintf("%s%s<html><body>Aplikacijski mail za novog poslodavca - ID: %d. Sadržaj: <br><br>\"%s\"<br><br> Aktivacijski ID: <b'%d'</b>.</body></html>", subject, mime, userID, desc, userID)
```

Kada je korisniku odobren status poslodavca, u bazi podataka će se kreirati novi zapis o poslodavcu, te će na ekranu s postavkama korisnik moći urediti naziv i opis svojeg poduzeća ili kompanije. Ukoliko je korisnik zadovoljan s kreiranim podacima, korisnik može krenuti upravljati sa podacima vezanim uz status poslodavca, tj. korisnik može dodavati nove kandidate u svoje virtualno poduzeće.

Odlaskom na stranicu "/company", čiji se prečac prikazuje u glavnom izborniku ukoliko korisnik ima status poslodavca, korisniku se otvara prikaz poduzeća sa svim članovima koje je korisnik odabrao kao aktivne članove svojeg poduzeća (zaposlenike) ili koje je proglasio potencijalnim kandidatima za zaposlenje. Spomenute uloge definirane su kroz relaciju "uloga" u bazi podataka, te na temelju vrijednosti polja uloga unutar te relacije za određenog korisnika i poslodavca definiran je odnos korisnika sa poslodavcem. Ukoliko je uloga za određenog korisnika i poslodavca 1, tada je taj korisnik vlasnik poduzeća koje se promatra. Ukoliko je uloga 2, tada je korisnik zaposlenik u poduzeću koje se promatra. Ukoliko je uloga 3, tada je korisnik potencijalni kandidat za zaposlenje u poduzeću koje se promatra. Kandidata je moguće dodati u poduzeće na stranici s životopisom za određenog zaposlenika, klikom na prečac na vrhu stranice pod nazivom "Dodajte kandidata".

Kada je kandidat dodan u poduzeće, on će se prikazati odlaskom na ranije spomenutu stranicu u sekciji "Kandidati". Odlaskom na spomenutu stranicu, kandidate je moguće izbaciti iz poduzeća (klikom na gumb ispod imena kandidata "Ukloni s popisa", ili ih je moguće promovirati u aktivne zaposlenike.

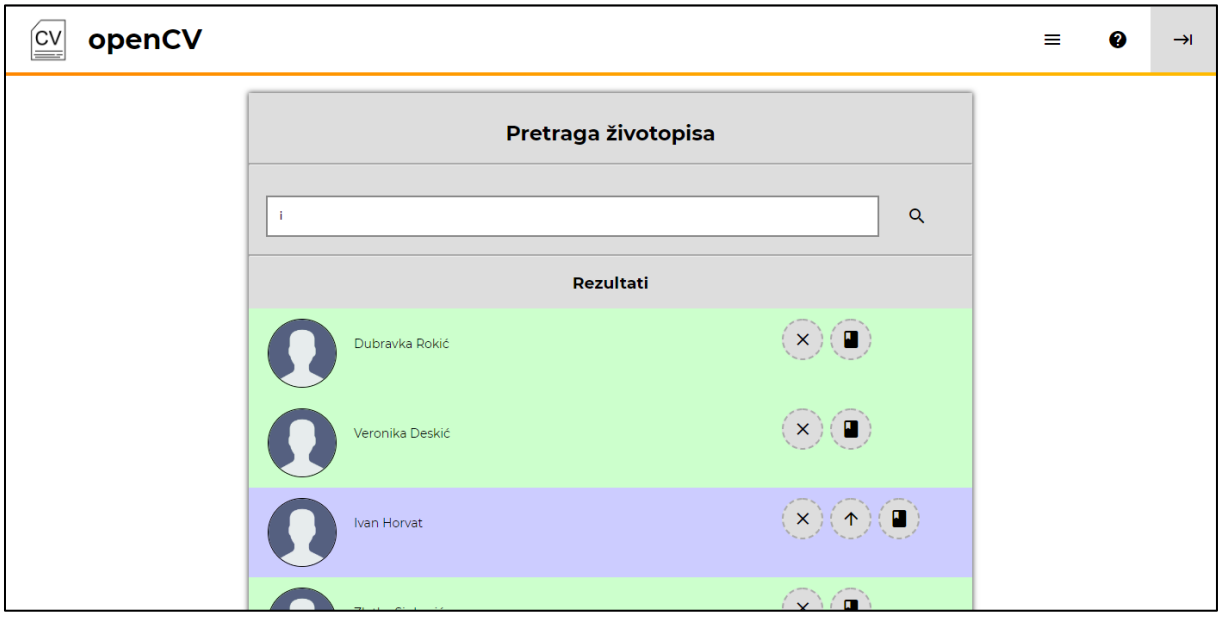


Slika 22: Stranica s pregledom zaposlenika i kandidata

Kako bi stranicu s popisom zaposlenika i kandidata bilo lakše podijeliti na dvije sekcije, model podataka koji se obrađuje prilikom dohvata stranice sadrži dvije liste s korisničkim podacima, jedna lista sadrži aktivne zaposlenike, dok druga sadrži kandidate. Prilikom čitanja redova dobivenih kao rezultat upita na bazu podataka, prilikom obrade svakog reda uloga korisnika u poduzeću se provjerava kako bi se podaci pohranili u pravu listu, a to se provodi putem programskog koda definiranog u nastavku:

```
if obj.Uloga == 2 {
      pageModel.Employees = append(pageModel.Employees, *obj)
} else if obj.Uloga == 3 {
      pageModel.Candidates = append(pageModel.Candidates, *obj)
}
```

Kada je korisnik promoviran u status poslodavca, za tog korisnika otvara se još jedna mogućnost, tj. stranica – pretraga zaposlenika. Pomoću navedene stranice moguće je brzo pretražiti bazu podatak prema imenu i prezimenu korisnika. Traženje korisnika obavlja se putem asinkronog zahtjeva sa korisničkog preglednika, šaljući GET zahtjev prema poslužitelju. Rezultat zahtjeva je JSON objekt koji predstavlja listu objekata, gdje je svaki objekt skup podataka vezanih uz pojedinog zaposlenika.



Slika 23: Stranica s pretragom korisnika

Korišteni programski kod za pretvorbu podataka iz dijela (eng. slice) u JSON objekt u programskom jeziku Go definiran je u nastavku:

```
returnObj, err := json.Marshal(*results)
```

Kroz programski jezik JavaScript, kod za slanje asinkronog zahtjeva je sljedeći:

```
var inputVal = document.getElementById("mainInput").value;
var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function () {
```

```
            ...
        };
        xhttp.open("GET", "/company/search?phrase=" + inputVal, true);
        xhttp.setRequestHeader("Content-Type",    "application/x-www-form-
urlencoded");
        xhttp.send();
```

Kada su podaci dohvaćeni, ovisno o dobivenom rezultatu i kodnom broju statusa izvršenja zahtjeva (ukoliko je 200 zahtjev je uspješno izvršen), obrađuje se dobiveni zahtjev. Ukoliko je dobivena vrijednost jednaka "null", tada nije pronađen niti jedan rezultat. Ukoliko je dobivenu vrijednost moguće pretvoriti u objekt (putem funkcije "JSON.parse"), tada se dobiveno polje objekata obrađuje. Za svaki objekt iz spomenutog polja kreira se jedna kartica s podacima o korisniku, te se ovisno o ulozi u poduzeću (ako postoji) za korisnika kartica na drugačiji način označuje, te se dobivene akcije po korisniku razlikuju.

Pronađeni korisnici koji su ujedno zaposlenici u poduzeću od trenutno prijavljenog korisnika označeni su zelenom bojom pozadine, te je za njih dostupna akcija uklanjanja iz poduzeća. Pronađeni korisnici koji su kandidati u poduzeću od trenutno prijavljenog korisnika označeni su plavom bojom pozadine, te je uz njihovo uklanjanje iz poduzeća dostupna i akcija promoviranja u aktivne zaposlenike. Za ostale zaposlenike koji nemaju ulogu kod promatranog poslodavca dostupna je akcija za dodavanje u poduzeće. Uz tako definirane akcije, za svakog pronađenog korisnika koji ima definiran "url" parametar životopisa dostupna je akcija za pregled pripadajućeg životopisa.

Sve akcije predstavljaju prečace na akcije koje obavljaju određenu radnju, te su zadani svojim "url" parametrom na aplikaciji i pripadajućim parametrom na temelju kojeg se akcija može izvršiti za određenog zaposlenika. Akcije su sljedeće:

- Brisanje: "/company/remove?id=[ID Zaposlenika]"
- Dodavanje: "company/add?id=[ID Zaposlenika]"
- Promocija: "company/promote?id=[ID Zaposlenika]"
- Pregled životopisa: "view?key=[URL ključ životopisa]"

Izuzevši akciju za pregled životopisa, sve ostale akcije predstavljaju direktne pozive prema aplikaciji koje aplikacija obrađuje prilikom izrade upita na bazu podataka i šalje kreirani upit prema bazi podataka.

5.10. Hosting – prebacivanje gotove aplikacije na poslužitelj

Nakon provedene faze testiranja aplikacije, web aplikaciju je potrebno postaviti na javno dostupan poslužitelj kako bi ona postala svima dostupna. Programski jezik Go je kompilatorski programski jezik, što ga ne čini idealnim za web sustave kod kojih razvojni programer nema potpunu kontrolu nad računalom na koje se postavlja aplikacija. Srećom, tvorac programskog jezika Go, Google, nudi niz servisa koje olakšavaju postavljanje gotovih aplikacija na javno dostupne poslužitelje, te međusobno povezivanje više servisa kao što je i SQL baza podataka. Za potrebe prebacivanja gotove Go aplikacije na Google Cloud platformu Google je razvio skup alata koji olakšavaju taj proces.

Cloud SDK je Google-ov programski alat koji pomoću nekoliko konfiguracijskih parametara ostvaruje vezu na Cloud platformu, te omogućuje daljnji rad bez dodatne autentifikacije. Prilikom pokretanja alata, kroz nekoliko upita alat će postaviti sigurnu vezu do Cloud Platform poslužitelja, te će omogućiti izvršavanje daljnjih akcija nad poslužiteljem. Kada se programer odluči prebaciti aplikaciju na poslužitelj, aplikaciju je potrebno sa svim pripadajućim datotekama prebaciti na lokaciju na računalu definiranu unutar **gopath** varijable. Sa te lokacije, Google-ov alat će dohvatiti sve potrebne pakete i dodatne reference koje se koriste aplikaciju, te će spomenute reference prebaciti zajedno s glavnom aplikacijom na poslužitelj. Početak prebacivanja pokreće se iz mape sa sadržajem aplikacije naredbom:

```
gcloud app deploy
```

Servis tada odrađuje ostatak radnji, a one su vidljive kroz konzolni prozor. Prva radnja je provjera svih programskih datoteka programskog jezika Go, te provjera svih referenci. Ukoliko prva provjera prođe bez grešaka, sljedeća radnja definira osnovne informacije o aplikaciji i traži provjeru od strane korisnika:

```
Services to deploy:

descriptor:      [C:\Users\User\go\src\open-cv\app.yaml]

source:          [C:\Users\User\go\src\open-cv]

target project:  [opencv-248810]

...

Do you want to continue (Y/n)?  y
```

Ukoliko se korisnik koji prebacuje aplikaciju slaže s dobivenim vrijednostima, servis će krenuti na sljedeću akciju – kopiranje datoteka sa lokalnog računala na poslužitelj. Sve datoteke (osim .go datoteka) biti će prebačene na poslužitelj u ovom koraku, a nakon uspješnog prebacivanja u konzolnom prozoru se javlja poruka:

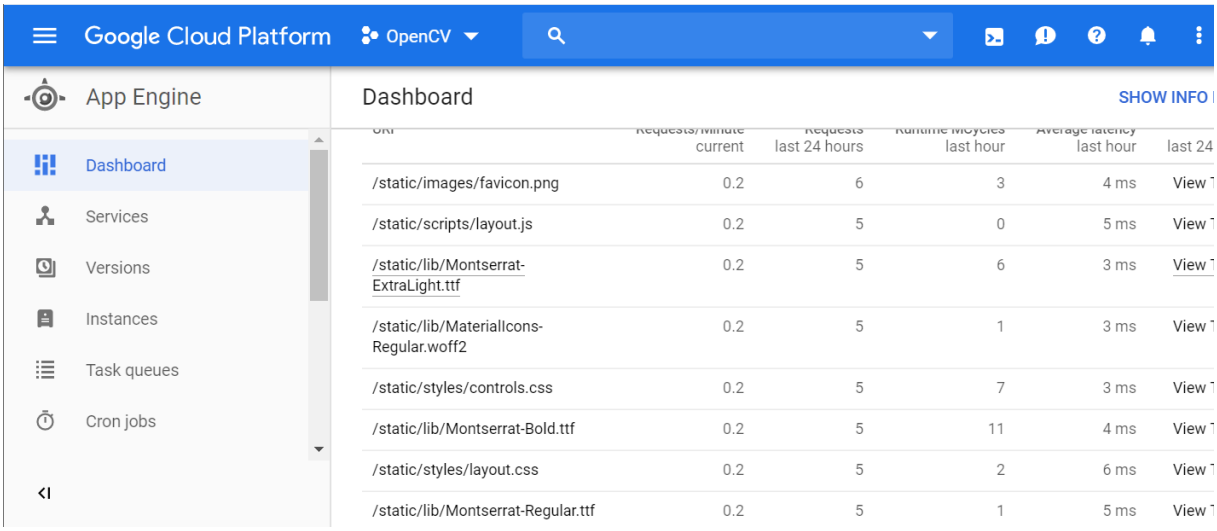
```
╔═══════════════════════════════════════════════════════════╗
╠═ Uploading 52 files to Google Cloud Storage              ═╣
╚═══════════════════════════════════════════════════════════╝

File upload done.
```

Nakon prebacivanja svih datoteka na poslužitelj, servis mora odraditi određena podešavanja kako bi aplikacija radila bez obzira na količinu prometa prema stranici. Završetak ove faze podešavanja servis javlja kroz niz poruka:

```
Updating service [default]...done.

Setting traffic split for service [default]...done.

Deployed service [default] to [https://aplikacija.appspot.com]
```

Ukoliko je završni proces protekao bez grešaka, aplikacija će javiti korisniku da je prebacivanje i aktivacija aplikacije uspješna, te će ponuditi opcije kako pristupiti aplikaciji. Putem dobivene web adrese moguće je pristupiti aplikaciji, a za upravljanje aplikacije biti će zadužen Google-ov servis na Google Cloud platformi.

Pregled zaprimljenih zahtjeva kroz određeni vremenski period i punu statistiku web aplikacije moguće je dohvatiti odlaskom na web stranice Google Cloud platforme, kao što je prikazano na slici 11. S tim korakom, kompletan proces izrade aplikacije je gotov, te je kroz platformu moguće ažurirati aplikaciju ukoliko je potrebno izvršiti neke promjene nad njom. [8]



Slika 24: Pregled statistike web aplikacije na Google Cloud platformi

6. Usporedba s drugim jezicima za izradu web aplikacija

Kako je ranije spomenuto, velik dio karakteristika programskog jezika Go odnosi se na brzinu rada aplikacija napravljenih pomoću tog programskog jezika. Kao dokaz performansi promatranog programskog jezika kroz trenutno poglavlje, dostupno je nekoliko analiza putem kojih su direktno uspoređene performanse još dva programska jezika.

6.1. Usporedba s PHP-om

Programski jezik PHP predstavlja jednu od starijih tehnologija za izradu web aplikacija, no još uvijek predstavlja validnu opciju za izradu jednostavnih i kompleksnih web aplikacija. Ključna prednost programskog jezika PHP nad jezicima poput Go-a je jednostavnost izrade web aplikacija. Nakon kreiranja skupa stranica sa PHP programskim kodom, kod nije potrebno kompilirati, već je aplikaciju odmah moguće testirati kroz web preglednik. Takav pristup omogućuje programeru koji razvija aplikaciju vrlo jednostavan i brz postupak testiranja aplikacije, bez potrebe izvršavanja dodatnih akcija prije pregleda aplikacije.

Uz to što je PHP interpreterski jezik, PHP je po svojoj načinu izrade web aplikacija također jedan od jednostavnijih jezika. PHP kod koji se izvršava na poslužitelju nalazi se u .php datotekama koje uz poslužiteljski kod, sadrže i html kod koji definira strukturu web stranica aplikacije. Uz niz dostupnih proširenja i razvojnih okvira za strukturiranje PHP programskog koda, broj korisnika tog programskog jezika je daleko veći od broja Go programera.

Iako programski jezik PHP ima niz prednosti koje ga na prvi pogled čine idealnim izborom za izradu web aplikacija, taj jezik ima nekoliko ključnih mana koje je potrebno uzeti u obzir. Za razliku od programskih jezika Go i C#, interpreterski jezik poput PHP-a je potrebno prilikom svakog dohvata stranice ponovo interpretirati – na taj način performanse aplikacije znatno brže padaju prilikom većeg broja zahtjeva u usporedbi s kompiliranim programskim jezicima. Kod manjih web aplikacija koje nemaju velik broj funkcionalnosti takav pad performansi nije tako lako uočljiv – no kod kompleksnijih aplikacija taj pad performansi može predstavljati značajan problem.

Uz performanse, ključan nedostatak PHP u usporedbi s programskim jezikom Go vidljiv je prilikom izrade većih web stranica aplikacije – html sadržaj stranice je teško odvojiti od PHP koda, ne postoje koncepti poput Go-ovih predložaka, tako da je određene dijelove koda na svakoj stranici potrebno ponavljati (ukoliko se ne koristi neki razvojni okvir).

6.2. Usporedba s .NET C#-om

C# je programski jezik kojeg je američka kompanija Microsoft lansirala 2000. godine, kao konkurenciju programskom jeziku Visual Basic (kojeg je kasnije C# i zamijenio). Programski jezik C# dio je Microsoftove .NET inicijative za izradu modernog programskog razvojnog okvira za izradu različitih oblika aplikacija. Kroz naredno razdoblje tijekom kojeg je razvijeno niz poboljšanja za .NET razvojni okvir, programski jezik C# pretvorio se u jedan od najčešćih izbora programera prilikom izrade web i desktop aplikacija, a relativan uspjeh je postigao i u mobilnom segmentu. [9]

Poput Go-a, C# je kompiliran programski jezik – prije pokretanja aplikacije potrebno je programski kod putem kompilatorskog programa prebaciti u strojni kod. Za razliku od programskog jezika Go, gdje je cijeli kod potrebno prebaciti u strojni prije pokretanja aplikacije, .NET programski okvir sadrži kompilator koji može kompilirati dijelove koda u stvarnom vremenu. Ta značajka razvojnog okvira omogućuje C#-u fleksibilnost prilikom izrade aplikacija, posebice web aplikacija kod kojih dio koda može biti pomiješan sa html strukturnim kodom za prikaz stranice. Uz sve navedeno, C#-ov kompilatorski program je tijekom niza godina bio podvrgnut optimizacijama i razvoju novih značajki, stoga je kompilator vrlo temeljit, te omogućuje jednostavno nalaženje grešaka u kodu i ispravljanje istih.

U usporedbi s programskim jezikom Go, glavne prednosti C#-a dolaze iz niza funkcionalnosti koje su implementirane putem .NET platforme. Mnoge funkcionalnosti koje su inicijalno podržane od strane C#-a dostupne su u Go-u putem paketa izrađenih od strane Go razvojne zajednice, te nisu tako detaljno dokumentirane kao što je slučaj kod cijele .NET platforme. Uz niz funkcionalnosti, za razvoj C# aplikacija dostupan je službeni Microsoftov razvojni alat – Visual Studio. Visual Studio sadrži ugrađeni kompilatorski alat za kompiliranje C# programskih datoteka, kao i jedan od najrazvijenijih sustava za predviđanje upisanog koda – Intellisense sustav.

Glavna prednost programskog jezika Go nad .NET platformom je brzina izvođenja aplikacije. .NET platforma, zbog svoje kompleksnosti i velikog broja dostupnih razvojnih biblioteka, ne može se mjeriti sa brzinom programskih jezika poput Go-a. Uz to, .NET platforma je prvenstveno napravljena s naumom da radi samo na Windows operacijskom sustavu, no zahvaljujući novoj inačici .NET platforme to nije više slučaj. .NET Core je programski okvir nastao na temelju .NET okvira, te se odlikuje podrškom za različite operacijske sustave i povećanim performansama u odnosu na staru platformu. Za detaljnu analizu određenih karakteristika svakog od tri navedena programska jezika dostupna je tablica usporedbe ključnih karakteristika.

Tablica 4. Podaci o programskim jezicima Go, PHP i C#

Jezik/Karakteristika	**Go**	**PHP**	**C#**
Godina prvog izdanja	2009	1995	2000
Trenutna stabilna inačica	1.13	7.3.9	7.3
Razvojna kompanija	Google	PHP razvojni tim	Microsoft
Naziv ekstenzije programske datoteke	.go	.php	.cs
Glavni dizajner/i	Rob Pike, Ken Thompson	Rasmus Lerdorf	Anders Hejlsberg
Vrste aplikacija	Web, konzolne desktop aplikacije	Web	Web, konzolne desktop aplikacije, desktop aplikacije s grafičkim sučeljem, mobilne aplikacije
Način prevođenja u strojni kod	Kompilator	Interpreter	Kompilator
Programski princip	Proceduralan	Proceduralan	Objektno orijentiran
Tipovi podataka	Statički	Dinamički	Statički

Podaci u prethodnoj tablici dohvaćeni su sa službenih stranica za spomenute programske jezike, te su ažurni na datum 12.09.2019.

6.3. Usporedba performansi tablično i grafički

Kao pravi pokazatelj razlika u performansama između tri programska jezika spomenuta u prethodnom poglavlju, u svakom programskom jeziku izrađena je aplikacija s tri dostupne stranice – svaka stranica izvršava određenu vrstu performansnog testiranja. U sva tri testa za potrebe mjerenja vremena izvedbe korišteni su dostupni koncepti u programskom jeziku, te je uvijek mjereno vrijeme za izvedbu programskog segmenta s traženim testom. Svi rezultati prilikom testiranja su prikazani u milisekundama, dok je za svaki parametar testa testiranje provedeno pet puta, te je kao konačni rezultat izračunata aritmetička sredina. Svi testovi su izvedeni na istom računalu u istim uvjetima.

6.3.1. Prvi test – generiranje rezultata

Prvi test performansi se sastoji od generiranja niza podataka koji se zatim spremaju u određenu podatkovnu strukturu – ovisno o programskom jeziku ta struktura može biti **polje** (eng. Array), **lista** (eng. List) ili **dio** (eng. Slice). Ovisno o ulaznom GET parametru stranice koji predstavlja broj elemenata u generiranoj listi, generirati će se niz objekata koji se sastoji do dva parametra – indeksa elementa i hash generirane vrijednosti. Svakom objektu pridružit će se indeks koji predstavlja redni broj tog elementa za kasniji prikaz u tablici s rezultatima, te će se pridružiti vrijednost koja se generira putem zadanog algoritma. Spomenuta vrijednost se generira na sljedeći način:

1. Aplikacija generira nasumični skup od 10 znakova
2. Nad generiranim skupom znakova provodi se SHA256 hash algoritam
3. Generirano polje bajtova se sprema u znakovnom obliku (base64)

Test bi za rezultat trebao pokazati koliko pojedini jezik brzo može provesti odabrani algoritam, te kakve mogućnosti jezik ima po pitanju upravljanja sa skupom podataka. Za potrebe testiranja za parametar koji određuje količinu generiranih objekata uzet je sljedeći skup: 1000, 10000, 100000 i 1000000.

U nastavku se nalaze programski kodovi za svaki pojedini jezik pomoću kojeg je testiranje realizirano. U programskom jeziku Go, dio funkcije na temelju kojeg se mjeri proteklo vrijeme testiranja je sljedeći:

```
h := sha256.New()

for i := 0; i < count; i++ {

      var comp testOneComponent

      comp.Index = i + 1
```

```
        shaVal := randStringBytes(10)
        h.Write([]byte(shaVal))
        comp.Code = base64.URLEncoding.EncodeToString(h.Sum(nil))
        *coll = append(*coll, comp)
}
```

Prije procesa iteriranja kroz N iteracija, kod programskog jezika Go kreira se nova instanca objekta tipa "sha256" (iz "crypto/sha256" paketa). Iteriranjem N puta, kreirat će se N nasumičnih nizova teksta (putem funkcije za nasumično generiranje teksta), te će se za svaki niz kreirati "hash" vrijednost. Dobivena vrijednost će se zatim pretvoriti u znakovni niz (iz skupa bajtova) putem "base64.URLEncoding.EncodeToString" funkcije za obradu skupova podataka, te će se dobiveni podataka zajedno sa rednim brojem dodati kao objekt na kraj početno definiranog polja objekata. U programskom jeziku PHP mjereni dio koda je sljedeći:

```
for($i = 0; $i < $count; $i++) {
        $randStr = generateRandomString(10);
        $tempHash = hash('sha256', $randStr);
        array_push($collection, (object) [
            'index' => $i,
            'hash' => $tempHash
        ]);
 }
```

Kod programskog jezika PHP, mjerena funkcija je kraća, no u pravilu je strukturirana isto kao i funkcija kod Go-a. Valja napomenuti da je kod PHP-a, pretvorba skupa bajtova u znakovni niz automatska, te da se cijeli „hashing“ proces obavlja kroz jednu liniju koda. U programskom jeziku C# mjereni dio koda prikazan je u nastavku:

```
List<FirstComponentModel> modList = new List<FirstComponentModel>();
Random rand = new Random(DateTime.Now.Millisecond);
using (SHA256 mySHA256 = SHA256.Create())
  {
  for (int i = start; i < end; i++)
    {
```

```
            StringBuilder hashValue = new StringBuilder();
            byte[] byteVal = mySHA256.ComputeHash(getRandStringBytes(rand));
            for(int j=0; j < byteVal.Length; j++)
            {
                  hashValue.Append(byteVal[j].ToString("x2"));
            }
            FirstComponentModel tempModel = new FirstComponentModel();
            tempModel.Index = i;
            tempModel.Sha = hashValue.ToString();
            list.Add(tempModel);
         }
      }
```

U programskom jeziku C#, mjereni dio koda je nešto veći u usporedbi s druga dva jezika, no finalni rezultat je isti. Iteriranjem N puta, kreira se N znakovnih nizova, te se zajedno s rednim brojevima spremaju u obliku objekta u listu kreiranu na početku programskog bloka. Ukoliko se u testu koriste polja umjesto lista kao skupovi podataka, performanse su gotovo identične, stoga ta postavka ne predstavlja utjecaj na testu performansi. Konačni skup rezultata prikazan je u narednoj tablici.

Tablica 5. Rezultati prvog testa performansi

N	Go	.NET Core C#	PHP
1000	1,8	5,4	2,8
10000	15	46	28,4
100000	156,2	512,6	277,8
1000000	1557,2	5290,4	2903,4

Prilikom prvog testiranja gdje je generirano 1000 rezultata u svakom programskom jeziku, uočljivo je kako programski jezik Go ostaje dosljedan navodima koji ističu brzinu tog jezika, dok već na relativno malom broju rezultata C# izvršava test tri puta sporije od Go-a. PHP je svojom brzinom pozicioniran između druga dva jezika, te je oko 64% sporiji od programskog jezika Go.

Prilikom testiranja sa 10000 rezultata, programski jezik Go ponovo najbrže izvršava zadani test, te C# ostaje u istom odnosu s Go-om kao i prilikom testiranja s 1000 rezultata –

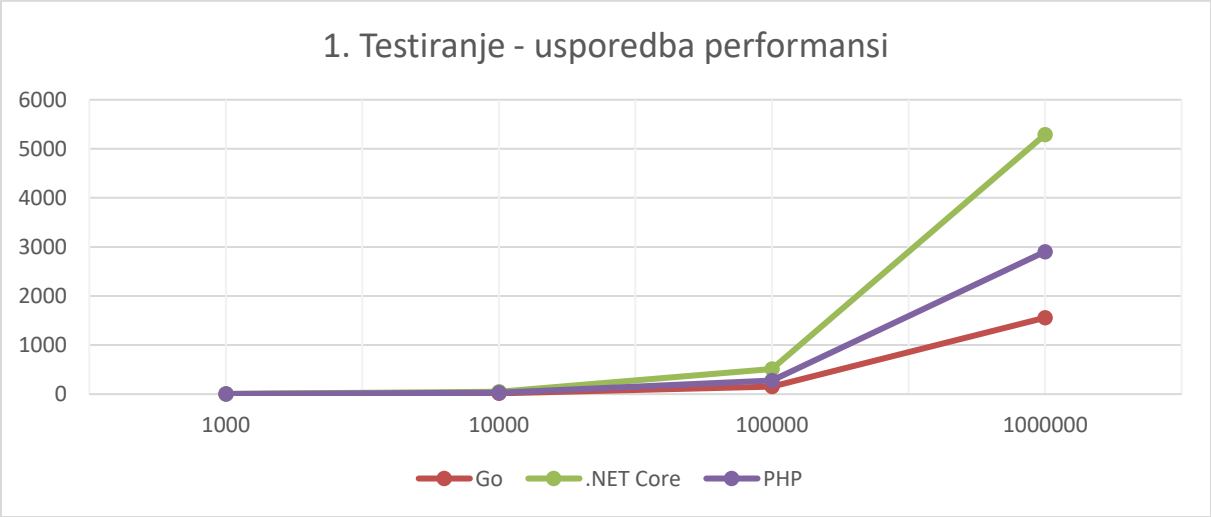
tri puta sporije izvršava zadani kod. Brzina PHP-a u odnosu na Go je nešto veća nego u prvom testiranju, te je u ovom testu Go gotovo 50% brži od PHP-a.

Na 100000 rezultata, Go i dalje predstavlja najbrži programski jezik kod ovog testa, te se u odnosu na C# razlika performansi između dva jezika dodatno povećava. Razlika između programskih jezika Go i PHP ostaje slična onoj iz prethodnog testiranja sa 10000 rezultata.

Prilikom testiranja sa milijun rezultata, Go ostaje najbrži programski jezik kod izvršavanja ovog testa, te je razlika između Go-a i C#-a još dodatno povećana u odnosu na test sa 100000 rezultata. Test uvjerljivo prikazuje kako je za potrebe hash algoritama programski jezik Go najbolja opcija, no isto tako pokazuje i jedan od ključnih nedostataka programskog jezika PHP. Prilikom izvođenja testa sa milijun rezultata na PHP aplikaciji, interpreterski program aplikacije pokazuje neočekivanu grešku:

```
Fatal error: Allowed memory size of 134217728 bytes exhausted...
```

Prethodno prikazana poruka daje informaciju o nemogućnosti alokacije dovoljno velikog polja za spremanje podataka. Grešku je moguće izbjeći povećanjem maksimalne količine memorije koju PHP poslužitelj može alocirati za pojedinu skriptu. Povećanjem količine memorije sa 128 MB na 512 MB, skripta će se uspješno izvršiti. Rezultati testiranja prikazani su i grafički:



Slika 25: Grafički prikaz rezultata prvog testa performansi

6.3.2. Drugi test – paralelno generiranje rezultata

Nakon provedbe prvog testa, vidljivo je da je programski jezik C#, uparen sa platformom .NET Core, najsporije rješenje za izvedbu prethodnog testa. Tijekom cijelog testiranja programski jezik Go je svojim performansama uvijek bio tri ili više puta brži od programskog jezika C#, dok je PHP uvijek bio minimalno dvostruko brži od C#-a. Jedan od ključnih aspekata koji se nameće kao idejno rješenje takvih problema na .NET Core platformi je korištenje „Zadataka“ (eng. Tasks) – funkcionalnosti za postizanje višedretvenog rada aplikacije.

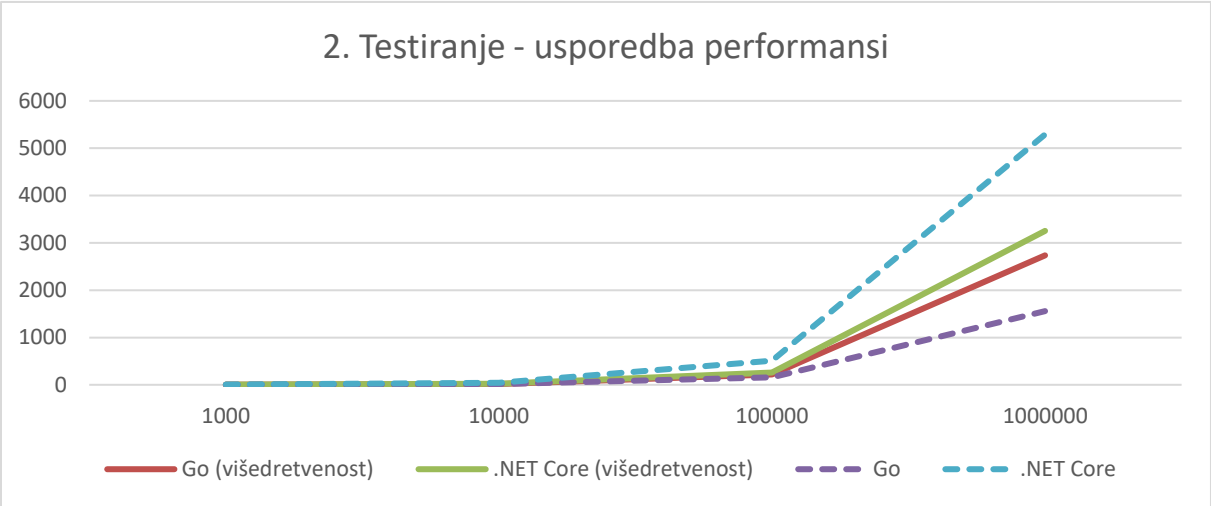
Drugi test je isti kao i prvi test, no jedna razlika koja mijenja svrhu testiranja je uključivanje višedretvenog rada u test. Prilikom generiranja svakih 1000 rezultata, kreirat će se zasebna dretva, tj. zadatak, koji će kreirati 1000 rezultata i spremiti ih u predviđenu listu, tj. dio. Za ovaj test, kod programskog jezika Go korištena je programska struktura „goroutine“, dok je kod programskog jezika C# korišten koncept asinkronih zadataka. Programski jezik PHP nije uključen u ovo testiranje zbog nemogućnosti kreiranja zasebnih dretava kod web aplikacije. Parametri testiranja su isti kao i u prvom testu. Programski kod nije naveden zbog identičnosti s kodom iz prvog testiranja, uz izuzetak dodanih funkcija za postizanje višedretvenog rada.

Tablica 6. Rezultati drugog testa performansi

Jezik/Rez	Go (višedretvenost)	.NET Core (višedretvenost)	Go	.NET Core
1000	2,2	4,8	1,8	5,4
10000	17,6	23,4	15	46
100000	220,2	257,6	156,2	512,6
1000000	2735,2	3253	1557,2	5290,4

U prethodnoj tablici prikazani su rezultati drugog testa. U tablicu su dodani i rezultati za programske jezike Go i C# iz prvog testa kako bi se lakše usporedili sa višedretvenim testom. Rezultati testiranja programskog jezika Go najbolje prikazuju koliko je jezik optimiziran za višedretveni rad – bez dodavanja višedretvenih funkcija, vrijeme izvršavanja testa je bilo manje nego kada su one dodane. Razlog tome je jednostavnost testa – algoritam koji se izvršava u testu je previše jednostavan za dijeljenje na više zasebnih procesa, te samo kreiranje novog pod-procesa (dretve) i dodavanje novog pod-procesa u red zahtjeva više procesorskog vremena nego generiranje rezultata.

Programski jezik C# kroz platformu .NET Core prikazuje suprotnu sliku prilikom izvođenja testiranja. Kroz dodavanje zadataka, izvedba odabranih dijelova koda je znatno brža u usporedbi s radom sa samo jednom dretvom. Usporedivši rezultate s prvim testom, brzina izvedbe C# aplikacije se gotovo udvostručila, čineći rad sa zadacima obveznim dijelom razvoja .NET Core aplikacije. U usporedbi s Go-om, rezultati drugog testa prikazuju mnogo manje razlike u brzini izvođenja aplikacija. Rezultati su prikazani i grafički, te su uz rezultate drugog testa prikazani i rezultati prvog testa za usporedbu:



Slika 26: Grafički prikaz rezultata drugog testa performansi

6.3.3. Treći test – učitavanje podataka iz baze

Treći test provjerava mogućnosti, tj. brzinu čitanja podataka iz baze podataka, te je za potrebe ovog testa odabrana mySQL relacijska baza podataka. U zadanoj bazi podataka nalazi se određena količina podataka unutar jedne tablice. Tablica sadrži podatke o različitim osobama – id, ime, prezime, grad, dob, te sadrži dovoljan broj redova za testiranje. Svaka aplikacija mora ostvariti vezu s bazom, te izvršiti upit nad bazom podataka koji će vratiti određeni broj redova iz tablice s podacima, limitiranih brojem dobivenim iz ulaznog parametra web stranice. Nakon dohvaćanja podataka, iste podatke je potrebno spremiti kroz određenu programsku strukturu podataka u listu. Za ulazne parametre odabran je sljedeći skup: 100, 1000, 10000, 100000. Mjereni kod za treće testiranje u programskom jeziku Go prikazan je u nastavku:

```
coll := new([]testThreeComponent)

db, err := sql.Open("mysql", "***:***@/speedtest")
```

```
defer db.Close()
results, err := db.Query("SELECT * FROM main LIMIT " +
strconv.Itoa(count) + ";")
for results.Next() {
      var comp testThreeComponent
      results.Scan(&comp.ID, &comp.Ime, &comp.Prezime, &comp.Grad,
&comp.Dob)
      *coll = append(*coll, comp)
}
```

Kod prethodno prikazanog koda varijabla "coll" predstavlja pokazivač na skup (dio) elemenata dobivenih obradom rezultata upita, dok varijabla "count" predstavlja broj redova dohvaćenih upitom. U programskom jeziku PHP, promatrani programski kod prikazan je u nastavku:

```
$conn = new mysqli($servername, $username, $password, $dbname);
$sql = "SELECT * FROM main LIMIT $count;";
$result = $conn->query($sql);
$collection = [];
if ($result->num_rows > 0) {
   while($row = $result->fetch_assoc()) {
   array_push($collection, (object) [
       'id' => $row["id"],
       'ime' => $row["ime"],
       'prezime' => $row["prezime"],
       'grad' => $row["grad"],
       'dob' => $row["dob"]
     ]);
   }
}
$conn->close();
```

Skup operacija u programskom jeziku PHP sličan je onome u programskom jeziku GO za treće testiranje, te se nakon uspostavljanja veze s bazom podataka i izvršenja upita kreiraju

objekti čiji se sadržaj sprema u polje objekata pod nazivom "collection". Svakom objektu se, koristeći operator pridruživanja "=>", pridružuje skup atributa čije su vrijednosti dobivene iz rezultata izvršenog upita. Zadnji testni isječak koda pripada programskom jeziku C#, te je definiran na sljedeći način:

```
List<ThirdComponentModel> modList = new List<ThirdComponentModel>();
MySqlConnection connection = new MySqlConnection("Server=localhost;
database=speedtest; UID=***; password=***");
connection.Open();
MySqlCommand command = connection.CreateCommand();
command.CommandText = $"SELECT * FROM main LIMIT {n};";
MySqlDataReader reader = command.ExecuteReader();
while (reader.Read())
{
   ThirdComponentModel comp = new ThirdComponentModel();
   comp.ID = reader.GetInt32(0);
   comp.Ime = reader.GetString(1);
   comp.Prezime = reader.GetString(2);
   comp.Grad = reader.GetString(3);
   comp.Dob = reader.GetInt32(4);
   modList.Add(comp);
}
connection.Close();
```

U programskom jeziku C# i razvojnom okviru .NET Core, objekt "modList" predstavlja listu svih dobivenih objekata kroz obradu izvršenog upita nad bazom podataka. Varijabla "n" sadrži broj dobivenih redova, te se iteriranjem po redovima tablice dobivene izvršavanjem upita dohvaćaju zasebni objekti sa sadržajem definiranim kroz bazu podataka.

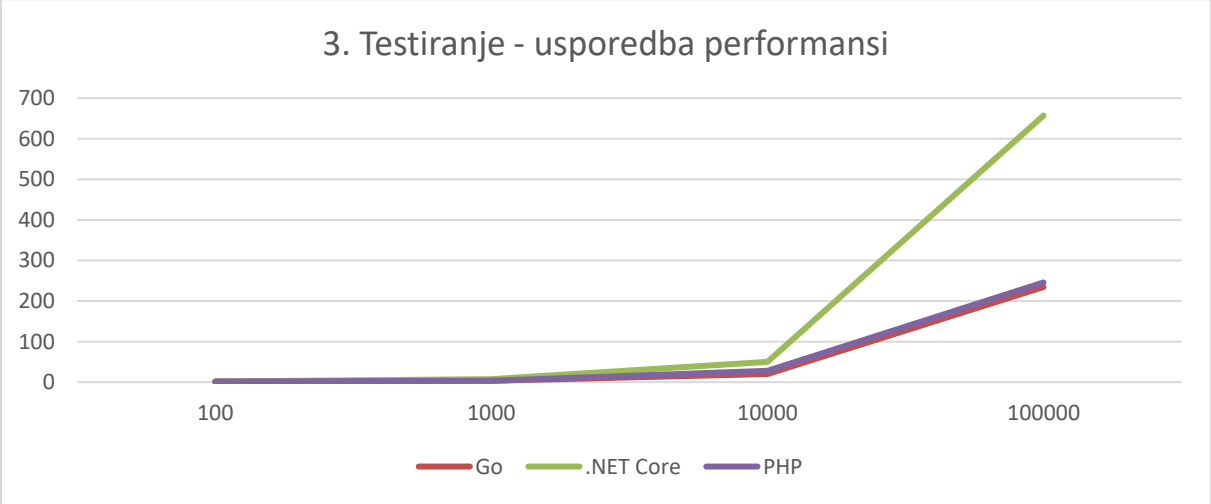
Potrebno je napomenuti kako je prvo pokretanje prilikom testiranja aplikacija sporije od budućih pokretanja. Razlog tome je inicijalna provedba autentifikacije od strane mySQL servisa. Prilikom uzimanja vrijednosti testiranja prva vrijednost je ignorirana, te su uzete ostale vrijednosti za izračun aritmetičke sredine.

Tablica 7. Rezultati trećeg testa performansi

Jezik/Rez	Go	.NET Core	PHP
100	1,4	1,6	1,6
1000	3,2	7,2	3,8
10000	19,8	49,8	28,2
100000	234,6	657,4	245,8

Prilikom izvršavanja trećeg testa, programski jezik Go ponovo prikazuje veću brzinu izvršavanja kada se usporedi sa druga dva jezika prisutna u testu. Treba ipak uzeti u obzir da jedan dio trećeg testa ovisi i o brzini mySQL servisa, te je shodno tome vidljivo da programski jezici Go i PHP ostvaruju slične performanse, gdje je Go brži za svega nekoliko milisekundi na testovima s većom količinom podataka. .NET Core i u posljednjem testu pokazuje najgore performanse, te prikazuje sličan odnos performansi sa Go-om kao i u prethodnim testovima.

U nastavku se nalazi i grafički prikaz usporedbe između tri programska jezika u posljednjem testu performansi.



Slika 27: Grafički prikaz rezultata trećeg testa performansi

7. Zaključak

Programski jezik Go cilja na nekoliko ključnih stvari koje ga čine pogodnim za izradu web aplikacije, što se istovremeno reflektira i na pozitivne strane razvoja u tom programskom jeziku. Go se kao jezik može pohvaliti jednostavnom sintaksom koja ga čini dobrim izborom za programere početnike, a istovremeno predstavlja i dobar odabir za iskusnije programere zbog sličnosti sa nekim drugim jezicima (python, scala). Go kao takav može programeru omogućiti brz razvoj u usporedbi s jezicima poput C-a ili C++-a, koju mu zbog svoje brzine predstavljaju glavnu konkurenciju.

Upravo zbog spomenute brzine izvođenja programa, Go predstavlja puno bolju alternativu za izradu aplikacija kod kojih se javljaju kritični dijelovi koji moraju biti obavljeni u stvarnom vremenu – za razliku od jezika poput PHP-a ili Python-a, koji zbog svoje jednostavnosti gubi brzinu izvođenja programa. Za brz rad programa također je zaslužena i konkurentnost (višezadaćnost) kod programskog jezika Go, funkcionalnost koju je također jednostavno implementirati u Go-u, dok kod nekih drugih jezika (C, Java) to nije slučaj.

Brz kompilatorski program za pretvaranje programskog koda u izvorni kod također predstavlja prednost, posebice kod programa s velikom količinom programskog koda. Uz svoju brzinu, kompilator nudi i mnoge koncepte koji olakšavaju održavanje koda, kao što je i optimizacija određenih dijelova programa, kako bi program radio još brže.

Kao i svi programski jezici, tako i programski jezik Go nije jezik bez mana. Sve prednosti programskog jezika Go dolaze s određenim nedostacima, a glavni razlog iza većine nedostataka je starost jezika – Go kao jezik je novitet, što ga čini donekle problematičnim za razvoj aplikacija.

Kao nov jezik, glavni problem programskog jezika Go je manjak podrške od strane Go zajednice. Velik broj grešaka koji se događaju kroz korištenje standardnih paketa nije dokumentiran, te je potrebna određena količina vremena da se takve greške otklone. Velik broj javnih paketa dostupnih kroz različite poslužitelje ima problema upravo sa nedostatkom korisničke podrške za otklanjanje grešaka koji nastaju kroz korištenje tih paketa.

Sintaksa samog jezika, iako jednostavna i dobra za početnike, može se pokazati kao problem za programere koji dolaze sa drugih programskih jezika, kao što su C, Java, C#.

No jedan od ključnih, možda i najvećih problema programskog jezika Go predstavlja prebacivanje jeziak na poslužitelj (hosting). Go, primarno jezik za izradu web aplikacija i servisa, je kompilirani jezik – njegovo stavljanje na poslužitelj nije toliko jednostavno kao kod interpretiranih jezika poput Python-a, PHP-a, Ruby-a itd. Nedostatak korisničke podrške i

poprilično sažeta dokumentacije od strane tvoraca programskog jezika Go čine korištenje ovog programskog jezika u produkcijskim scenarijima vrlo kompliciranim i ponekad nemogućim.

Programski jezik Go je nov, zanimljiv i predstavlja dobru opciju za programere početnike. Njegova jednostavna sintaksa, brz kompilator i skup naprednih koncepata na prvi pogled čine ga idealnim za izradu modernih web aplikacija. Go zaista ostvaruje velik broj mogućnosti obećanih od strane njegovih tvoraca, te velik broj ljudi odabire Go kao prvi izbor za programiranje novih aplikacija.

Nažalost, Go još uvijek ima mnogo nedostataka koji ga čine nepogodnim za izradu velikih produkcijskih aplikacija. Na sreću, većina nedostataka nije vezana uz sami programski jezik, već uz korisničku podršku oko jezika, tako da on i dalje predstavlja dobru opciju za početak programiranja. Problemi oko migracije gotove aplikacije na javno dostupan poslužitelj predstavljaju značajan problem ovog programskog jezika, te dok se taj problem ne riješi, Go ne može konkurirati vodećim jezicima za izradu web aplikacija – PHP, C#, Python, Java.

Uzevši u obzir prethodno spomenute činjenice, Go trenutno nije isplativ jezik za korištenje u produkcijskim aplikacijama i web servisima. Ipak, treba napomenuti da je Google-ova web platforma „moćan“ i razvijen alat, te da kroz niz mogućnosti koje ona nudi, daje korisniku mnoštvo opcija za povezivanje Go aplikacije sa drugim servisima, te zajedno u kombinaciji, mogu poslužiti kao dobra podloga za izradu manjih web aplikacija i web servisa.

Popis literature

[1] C. Doxsey, *An Introduction To Programming In Go*, 1. izd., 2012., [Na internetu]. Dostupno: https://www.golang-book.com/public/pdf/gobook.3186517259.pdf [pristupano 7.7.2019].

[2] G. Ornbo, *GO in 24 Hours Sams Teach Yourself*, *Next Generation Systems Programming with Golang*, 1. izd., Indiana, Indianapolis, USA: Pearson Education, Inc., 2018

[3] N. Kozyra, M. Ryer, *Go: Building Web Applications,* 1. izd., Livery Place, Birmingham, UK: Packt Publishing, 2016.

[4] M. Zamski, D. Singer, *Go for Javascript Developers* (web knjiga), [Na internetu]. Dostupno: http://www.pazams.com/Go-for-Javascript-Developers/ [pristupano 7.7.2019].

[5] „Documentation“ (bez dat.) u „The Go Programming Language“, službena web dokumentacija za Go razvoj. Dostupno: https://golang.org/ [pristupano 7.7.2019].

[6] „Go in Visual Studio Code“ (bez dat.) u „Visual Studio Code“, službena web dokumentacija za Visual Studio Code. Dostupno: https://code.visualstudio.com /docs/languages/go [pristupano 7.7.2019].

[7] M. Tsoukalos, *Mastering Go,* 1. izd., Livery Place, Birmingham, UK: Packt Publishing, 2018.

[8] „Cloud Platform Docs“ (bez dat.) u „Google Cloud Platform“, službena web dokumentacija za servise dostupne na Google Cloud platformi. Dostupno: https://cloud.google.com/go/docs/ [pristupano 15.8.2019].

[9] „What is .NET?“ (bez dat.), službena web stranica za uvod u .NET razvoj. Dostupno: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet [pristupano 03.09.2019.]

Popis slika

Popis tablica