Web sučelje za označavanje bakterijskih kolonija

Tkalec, Filip

Master's thesis / Diplomski rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:861189

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-11-24**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

WEB SUČELJE ZA OZNAČAVANJE BAKTERIJSKIH KOLONIJA

Diplomski rad

Filip Tkalec

Osijek, 2017.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 22.09.2017.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Filip Tkalec
Studij, smjer:	Diplomski sveučilišni studij Računarstvo, smjer Procesno računarstvo
Mat. br. studenta, godina upisa:	D 707 R, 14.10.2014.
OIB studenta:	62627990960
Mentor:	Doc.dr.sc. Tomislav Matić
Sumentor:	Dr.sc. Ivan Vidović
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Ivan Aleksi
Član Povjerenstva:	Dr.sc. Ivan Vidović
Naslov diplomskog rada:	Web sučelje za označavanje bakterijskih kolonija
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U radu je potrebno izraditi višekorisničko web sučelje za manualno označavanje bakterijskih kolonija na slici. Sliku je potrebno učitati s web kamere ili s medija za pohranu podatak. Rezultate označavanja potrebno je spremiti lokalno ili na udaljenom poslužitelju s mogućnošću reprodukcije rezultata.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	22.09.2017.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 02.10.2017.

Ime i prezime studenta:	Filip Tkalec
Studij:	Diplomski sveučilišni studij Računarstvo, smjer Procesno računarstvo
Mat. br. studenta, godina upisa:	D 707 R, 14.10.2014.
Ephorus podudaranje [%]:	2%

Ovom izjavom izjavljujem da je rad pod nazivom**: Web sučelje za označavanje bakterijskih kolonija**

izrađen pod vodstvom mentora Doc.dr.sc. Tomislav Matić

i sumentora Dr.sc. Ivan Vidović

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1

1.1 Zadatak diplomskog rada 2

2. PREBROJAVANJE BAKTERIJSKIH KOLONIJA 3

3. WEB SUČELJE ZA PREBROJAVANJE BAKTERIJSKIH KOLONIJA 7

3.1 Ruby 8

3.2 Ruby on Rails 10

3.3 MVC (Model-View-Controller) 12

3.3.1 Model 13

3.3.2 View (pogled) 14

3.3.3 Kontroler 15

4. IMPLEMENTACIJA 16

4.1 Struktura web sučelja 16

4.2 Statične stranice 21

4.3 Korisnici 23

4.4 Bakterije 26

4.5 Slike i oznake 28

5. TESTIRANJE WEB SUČELJA 32

6. ZAKLJUČAK 35

LITERATURA 36

SAŽETAK 39

ABSTRACT 39

ŽIVOTOPIS 40

PRILOZI 41

1. UVOD

Brojanje bakterijskih kolonija iznimno je važan postupak na području mikrobiologije kojim se u, primjerice, prehrambenoj industriji određuje kvaliteta namirnica. Sličan postupak je primjenjiv i na brojanje gljivica, virusa, krvnih zrnaca, stanica itd. Manualno brojanje kolonija s hranjivih podloga (agar pločica) naporan je i repetitivan posao te je zbog samog umora i psiho-fizičkih faktora ljudi ovaj proces podložan je pogreškama. Nakon brojanja pločice se bacaju i uništavaju te nije moguće provesti analizu točnosti. Zbog problema skladištenja uzoraka nije moguće imati retrospektivni pregled brojanja te nije moguća reprodukcija rezultata na uništenom uzorku.

Razvojem tehnologije ovaj postupak moguće je obavljati uz pomoć računala. Korištenjem računala povećava se preciznost i skraćuje vrijeme brojanja te se na osnovu toga povećava produktivnost. Osim manualnog brojanja računala omogućuju automatsko i poluautomatsko brojanje. Problem čuvanja podataka i uzoraka u ovom slučaju nestaje zbog gotovo neograničene memorije računala te mogućnošću spremanja rezultata na udaljene baze podataka.

Zbog visoke cijene komercijalni sustavi za automatsko brojanje nisu dostupni obrazovnim ustanovama kao što su fakulteti. Cilj ovog diplomskog rada je izraditi web sučelje za manualno označavanje bakterijskih kolonija koje će biti lako dostupno. Korištenjem web aplikacije s jednostavnim i intuitivnim sučeljem moguće je olakšati sam posao brojanja i organiziranja podataka.

Tema ovog rada je razrađena u šest poglavlja. U drugom poglavlju je naglasak na način pripreme uzorka za brojanje. Objašnjen je proces razrjeđivanja, inkubiranja te na samom kraju tehnika manualnog brojanja kolonija. Kao primjer je ukratko pojašnjena najzastupljenija tehnika brojanja (APC). Opisuje se primjena metode u različitim djelatnostima i industrijama ali se stavlja naglasak na prehrambenu industriju. U trećem poglavlju je opisuju se mogućnosti web aplikacije. Uz opise mogućnosti predstavlja se i osnovna struktura modela i idejni način rada. Detaljno se opisuju alati koji su potrebni za izradu samog sučelja te pojašnjeni njihovi dijelovi radi lakšeg praćenja implementacije. U četvrtom poglavlju fokusira se na implementaciju zadatka. Opisuje se način rada svakog pojedinog djela te su prikazani, i objašnjeni, dijelovi koda. U petom poglavlju je prikazano testiranje web sučelja. Objašnjavaju se vrste testova koji su korišteni te komentari na rezultate testiranja. U zadnjem poglavlju je zaključak diplomskog rada, gdje su istaknuta najvažnije činjenice vezane za rad. Također su navedeni i prijedlozi za poboljšanja i nadogradnje web aplikacije.

1.1 Zadatak diplomskog rada

Zadatak ovog diplomskog rada je izraditi višekorisničko web sučelje za manualno označavanje bakterijskih kolonija na slici. Sliku je potrebno učitati s web kamere ili s medija za pohranu podatak. Rezultate označavanja potrebno je spremiti lokalno ili na udaljenom poslužitelju s mogućnošću reprodukcije rezultata.

2. PREBROJAVANJE BAKTERIJSKIH KOLONIJA

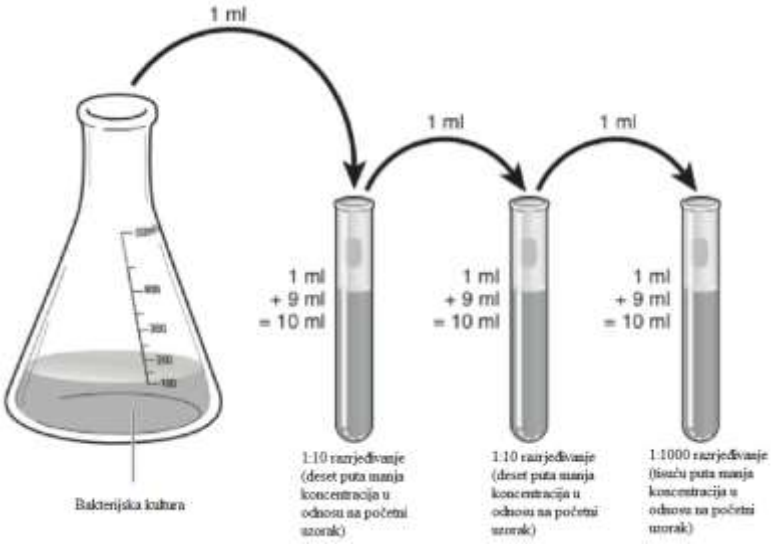
Brojanje kolonija jedna je od najosnovnijih mikrobioloških tehnika kojom se procjenjuje broj životnih, tj. živih, stanica u uzorku [1]. Postoji nekoliko tehnika brojanja koja se dijele u dvije skupine; posredno i neposredno brojanje. Primjeri tehnika su SPC (engl. *Standard Plate Count*), SPC (engl. *Spiral Plate Counter*), suhi petrifilmovi, MPN (engl. *Most Probable Number*), Turbidimetrijska metoda, itd. [2]

U praksi je uz samu vrstu bakterije potrebno poznavati i njihov broj. Brojanje bakterijskih kolonija koristi se u mnogim industrijama kao što su: prehrambena (prati se broj i tip bakterija u proizvodima), biotehnološka (pomno se prati rast bakterija te se manipuliraju bakterijama za izradu lijekova), vodoopskrba (testira se efikasnost filtera u sterilizaciji vode), itd. [3]

Proizvođači hrane obavezni su pratiti bakterijske kolonije u svojim proizvodima te osigurati sigurnost pri konzumiranju. U svakom proizvodu možemo pronaći određeni broj bakterija ali većina njih ne predstavlja opasnost pa je naglasak na održavanju, odnosno smanjivanju, štetnih bakterija na minimumu. Jedan od primjera praćenja bakterijskih kolonija u prehrambenoj industriji je nakon pasteriziranja proizvoda (uspoređuje se broj kolonija prije i poslije procesa pasterizacije). Kao što je već spomenuto princip brojanja nije strogo vezan za bakterije pa se pri proizvodnji piva i vina prati porast gljivica kvasca prilikom destilacije [3]. Također broj kolonija se može iskoristiti za predviđanje roka trajanja proizvoda. [4]

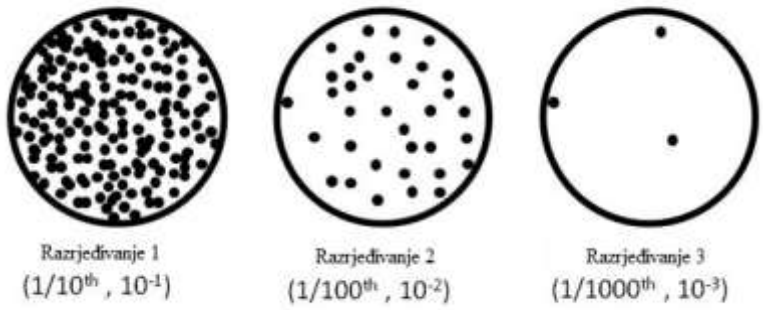
Prije samog brojanja bakterija potrebno je uzgojiti, odnosno inkubirati, bakteriju na hranjivoj podlozi. Pravilnim postupkom smanjuje se ljudska greška i rezultat brojanja je precizniji. Cijeli postupak od prikupljanja uzorka do brojanja kolonija se odvija u više koraka: prikupljanje uzorka, razrjeđivanje uzorka, priprema hranjive površine, nasađivanje bakterija, inkubiranje, brojanje i na samom kraju računanje broja bakterija u početnom uzorku. [1]

Prvi korak nakon prikupljanja uzorka je razrjeđivanje. Razrjeđivanje je proces u kojem početnu otopinu razrjeđujemo otapalom te se tako smanjuje broj bakterija u otopini. Ovisno o izvoru uzorka, početni broj bakterija se broji u tisućama ili čak milijunima. [1] Nasađivanje u ovoj fazi nije korisno jer je broj kolonija jako velik te je brojanje gotovo nemoguće. Potrebno je razrijediti početni uzorak kako bi se smanjio broj kolonija i tako postigao optimalan broj na hranjivoj podlozi.



Sl. 2.1. *Proces razrjeđivanja početnog uzorka. [5]*

Na Sl. 2.1. grafički je prikazan postupak razrjeđivanja. 1 ml početne otopine miješa se s 9 ml otapala. U prvom koraku se dobije koncentracija u omjeru 1:10, odnosno osiguravamo deset puta manju koncentraciju bakterija nakon prvog razrjeđivanja. Svakim sljedećim razrjeđivanjem koncentracija se umanjuje za 10 puta pa je tako nakon drugog razrjeđivanja koncentracija 10^{-2}, trećeg 10^{-3}, itd. Postupak je potrebno ponavljati sve dok se ne dobije broj bakterijskih kolonija pogodan za brojanje golim okom. Na Sl. 2.2. prikazano je smanjivanje kolonija na podlozi svakim narednim razrjeđivanjem. Optimalan broj kolonija na nasađenoj površini je od 25 do 250. [5]



Sl. 2.2. *Prikaz kolonija nakon svakog razrjeđivanja. [6]*

Nakon razrjeđivanja i određivanja optimalnog broja bakterija za brojanje potrebno je pripremiti hranjivu podlogu za uzgoj bakterija. Hranjiva podloga može biti u tekućem, krutom ili polutvrdom stanju i služi za uzgajanje mikroorganizama kao što su bakterije, virusi, gljivice itd. Najčešća podloga za uzgajanje je agar s dodanim nutrijentima za prehranu. Agar je mekani gel koji se dobiva

od algi. Ako je potreban selektivan uzgoj tada se koriste specijalne podloge. Primjer su virusi, koji su paraziti, čiji se uzgoj odvija na podlozi koja sadrži žive stanice. [7]

Hranjiva podloga se stavlja na petrijeve zdjelice. Nakon toga se, pažljivo, 0.1 ml otopine raspoređuje po površini uz pomoć steriliziranog staklenog štapića. Uobičajena je praksa nasađivanja svakog razrjeđivanja na više podloga kako bi se mogao izračunati prosjek te tako povećati preciznost konačnog rezultata. [1]

Podloge se nakon nanošenja otopine suše i stavljaju u inkubator. Temperatura i vrijeme inkubacije ovisi o podlozi i mikroorganizmu koji uzgajamo. Prilikom inkubacije individualne stanice koje nisu vidljive golim okom se razmnožavaju i stvaraju kolonije. Nakon određenog vremena kolonije postanu vidljive golim okom te ih je tada moguće prebrojiti. [1]

Najčešći oblik brojanja bakterija u prehrambenoj industriji je *aerobic plate count* (APC) koji služi kao indikator populacije bakterija na uzorku. Također se još naziva i *standard plate count* (SPC). [4] APC se vodi činjenicom da je broj kolonija nakon inkubacije jednak broju individualnih stanica prije inkubacije. [8] Brojanje kolonija se u većini slučajeva obavlja manualno na osvijetljenoj podlozi radi lakšeg raspoznavanja. Na Sl. 2.3. prikazan je način pravilnog označavanja bakterijskih kolonija. [4]

Nakon prebrojavanja potrebno je izračunati broj bakterija u početnom uzorku. [1] Ukupan broj bakterija možemo izračunati korištenjem formule:

$$N = n * 10^{x} * 10 \tag{2-1}$$

gdje je: N – ukupan broj bakterija u početnom uzorku, n – broj izbrojanih bakterijskih kolonija, x – broj razrjeđivanja početne otopine. Npr. ako je broj izbrojanih kolonija 50, a za brojanje je korištena 3 razina razrjeđivanja tada je ukupan broj bakterija u početnom uzorku 500 000. Nakon množenja broja bakterija s brojem razrjeđivanja potrebno je još jednom pomnožiti za 10 jer moramo uzeti u obzir da smo na hranjivu površinu stavili 1 ml od ukupnih 10 ml (na ovo gledamo kao na dodatno razrjeđivanje).

Nakon prebrojavanja petrijeve zdjelice se uništavaju. Na površinu uzorka najčešće se stavlja izbjeljivač koji ubija kulture, odnosno dezinficira površina. Petrijeve zdjelice se nakon ovog postupka ne mogu više koristiti te se bacaju. [9]

Sl. 2.3. *Primjer pravilnog označavanja bakterija. [10]*

Kao što možemo pretpostaviti, manualno brojanje je jako podložno ljudskim greškama te se nakon brojanja i uništavanja uzorka podatci ne mogu rekreirati. Ako je došlo do pogreške i ona nije uočena tijekom brojanja, tada se ona ne može ukloniti (nije moguće ponovno brojanje nakon uništavanja uzorka). Također je važno napomenuti kako se kod APC metode ne prepoznaju različite vrste bakterija. Kod prehrambene industrije broj kolonija nije u direktnoj povezanosti s brojem patogena i toksina (manji broj ne znači da opasnost nije prisutna) ali može biti koristan kao indikator kvalitete hrane. [8]

3. WEB SUČELJE ZA PREBROJAVANJE BAKTERIJSKIH KOLONIJA

U prethodnom poglavlju opisan je cijeli proces pripreme uzorka, njegovo inkubiranje te na kraju samo brojanje. Taj proces je dugotrajan i samo inkubiranje kolonija može potrajati više od 2 dana. Zbog visoke cijene aparata za automatsko brojanje najčešće se primjenjuje metoda manualnog brojanja.

Web aplikacija je jednostavan način zamjene manualnog brojanja i skladištenja rezultata. U gotovo neograničenu memoriju je moguće, u teoriji, spremiti neograničenu količinu podataka. Upravo ta činjenica omogućava čuvanje svakog pojedinog brojanja te u konačnici rekreiranje rezultata koji su stari i po nekoliko godina. Tako se vrlo lako mogu otkriti nepravilnosti i pogreške u prijašnjem označavanju.

Prije same implementacije potrebno je odrediti željene funkcije aplikacije. Cilj ovog rada je izraditi web aplikaciju za manualno brojanje bakterija koja će na jednostavan i intuitivan način povećati produktivnost te olakšati dokumentiranje, spremanje i rekreiranje rezultata. Temelj aplikacije je jednostavno sučelje koje nije zbunjujuće za korisnika te omogućuje korištenje svih mogućnosti uz minimalno vrijeme navikavanja.

Osnovna mogućnost aplikacije je spremanje fotografija u bazu podataka koje predstavljaju jedan uzorak (najčešće jednu petrijevu zdjelicu s nastanjenim bakterijama). Petrijevu zdjelicu je potrebno fotografirati te *uploadati* (učitati), a radi jednostavnijeg korištenja implementirana je mogućnost direktnog spremanja fotografije uz pomoć web kamere. Također je moguće iskoristiti i bilo koju kameru ili fotoaparat ako postoji mogućnost spajanja i dijeljenja slike s računalom.

Nakon spremanja fotografije moguće je direktno na sliku označiti bakterijske kolonije te aplikacija samostalno broji bakterije. U praksi je većinom potrebno brojati samo jednu vrstu bakterije po uzorku no aplikacija nudi mogućnost označavanja više vrsta bakterijskih kolonija. Vrste su kodirane bojama te je stoga razlikovanje vrlo jednostavno.

Aplikacija je osmišljena kao jeftinije i jednostavnije rješenje skupim aparatima za automatsko brojanje bakterijskih kolonija. Jedan od primjera primjene je u obrazovne svrhe. Aplikaciju je moguće koristiti na fakultetima i školama.

Uzevši u obzir primjer korištenja u edukacijske svrhe i korištenje aplikacije od strane više osoba potrebno je implementirati višekorisničko sučelje. Korisnik kroz registracijski proces dobije svoj

račun s repozitorijem te tako može pratiti svoju statistiku ali također mu se i nudi opcija pregleda drugih repozitorija.

Višekorisničko sučelje je potrebno implementirati s određenim mjerama zaštite kako bi se uvele restrikcije pri korištenju. Jedan od primjera je brisanje tuđeg korisničkog računa ili pak brisanje podataka unutar tuđih repozitorija od strane neovlaštenih korisnika.

Nakon objašnjavanja osnovnih funkcija aplikacije potrebno je pobliže objasniti alate potrebne za izradu web aplikacije. Ova aplikacija je napisana korištenjem programskog jezika Ruby i Ruby on Rails *frameworkom*. Također, opisuje se koncept rada MVC-a i prikazana je osnovna struktura aplikacije.

3.1 Ruby

Ruby je objektno orijentirani programski jezik koji je nastao u Japanu sredinom 90-ih godina prošlog stoljeća. Tvorac ovog jezika je Yukihiro Matsumoto poznat pod nadimkom Matz. Glavna ideja pri kreiranju ovog programskog jezika je kako programiranje treba biti zabavno za programere kako bi se povećala produktivnost. [11]

Ideja o kreiranju novog programskog jezika je začeta 1993. godine. Matz je smatrao kako objektno orijentirani jezici imaju svijetlu budućnost te je ga je zainteresirala ideja o skriptnom objektno orijentiranom jeziku. [12] Iako je Python tada postojao, nije ga smatrao istinskim objektno orijentiranim jezikom. Korištenjem svojih najdražih programskih jezika (Perl, Smaltalk, Eiffel, Ada i Lisp), odnosno uzimanjem najboljih elemenata svakog jezika, stvorio je novi i nazvao ga Ruby. [13]

Za nagli rast popularnosti ovog programskog jezika je zaslužan Ruby on Rails *framework* koji izlazi 2005. godine. Ruby se tada počinje koristiti u web programiranju. [13]Također Ruby je programski jezik otvorenog koda te bilo tko može sudjelovati u njegovom rastu i poboljšanju.

Osnovna Ruby sintaksa demonstrirana je kroz par primjera. Jezik je skriptni te ga zbog toga nije potrebno kompajlirati već je kod moguće direktno unositi u konzolu. Na Sl. 3.1. prikazan je primjer Hello world!

```
puts 'Hello world!'
```

Sl. 3.1. *Hello world!*

Prilikom kreiranja varijabli nije nužno koristiti deklaracije (char, int, string, itd.). Na Sl. 3.2. prikazana je razlika između lokalne, globalne i varijable instance.

```
Lokalna_varijabla
$globalna_varijabla
@varijabla_instance
@@varijabla_klase
```

Sl. 3.2. *Lokalna, globalna i varijabla instance.*

Jedna novost u Ruby jeziku su simboli. Simbol je objekt u Ruby jeziku kojim predstavljamo ime nečega i označava se dvotočkom ispred naziva (*:simbol*). Bitno je napomenuti kako simbol nije isto što i varijabla. Varijabla je promjenjiva, a simbol nije što znači da je bliži definiciji stringa.

Uzmimo u obzir Sl. 3.3. Na lijevoj strani kreirane su dvije varijable kojima su pridruženi isti simboli, a na desnoj strani dvije varijable kojima su pridružena dva ista stringa. U lijevom primjeru simbol će biti kreiran samo jedan put te će varijabla *a* i *b* pokazivati na isto mjesto u memoriji, dok će u desnom primjeru oba puta string biti kreiran te će varijable a i b pokazivati na različita mjesta u memoriji.

```
a = :simbol
b = :simbol
```

```
a = 'string'
b = 'string'
```

Sl. 3.3. *Razlika između simbola i varijable.*

Primjena simbola u Ruby jeziku i kasnije u Ruby on Rails frameworku jako je široka. Jedan od najčešćih primjera korištenja simbola je u *hashovima*. Koristi se kao vrijednost ključa u hash varijabli. *Hash* je objekt koji predstavlja imenik za spremanje vrijednosti u paru ključ-vrijednost, a njegov primjer možemo vidjeti na Sl. 3.4. Kreiran je hash kalendar koji u sebi ima dva para ključa-vrijednosti. Ključevi su 1 i 2, a vrijednosti tih ključeva su *Sijecanj* i *Veljaca*. Oznaka #{} označava ispis vrijednosti varijable umjesto teksta između navodnika.

```
kalendar = Hash.new
kalendar = {"1" => "Sijecanj", "2" => "Veljaca"}
puts „#{kalendar[1]}“ -ispisuje Sijecanj
```

Sl. 3.4. *Hash*

U mnogim programskim jezicima brojevi i ostali primitivni tipovi nisu objekti, što u Ruby jeziku nije slučaj. Moguće je pridijeliti metode apsolutno svemu. Na Sl. 3.5. možemo vidjeti poziv metode na broj. [13]

```
5.times { print "Ruby je zakon!" }
```

Sl. 3.5. *Poziv metode na broj*

Čak i osoba koja nema predznanja iz programiranja može pretpostaviti kako je rezultat poziva metode times na broj 5 ispisivanje rečenice: „Ruby je zakon!“ pet puta. Razlika između puts i print je ta što naredba puts dodaje novi red na kraju, a print ne.

Kao i kod svakog objektno orijentiranog programskog jezika moguće je kreirati klase i njihove metode. Na Sl. 3.6. možemo vidjeti deklariranje klase pas unutar koje je deklarirana metoda laje. Kreiran je objekt Oddie na kojem se poziva metoda laje. Poziv te metode rezultira lajanjem psa, odnosno ispisa Woof Woof!.

```
class Pas
 def laje
  puts "Woof woof!"
 end
end

Oddie = Pas.new
Oddie.laje
```

Sl. 3.6. *Primjer klase i metode*

U ovom potpoglavlju su pobliže objašnjene osnovne programskog jezika Ruby. Pojavom frameworka Ruby on Rails proces kreiranja stranica je drastično pojednostavljen.

3.2 Ruby on Rails

Ruby on Rails, ili skraćenije Rails, poslužiteljski je web *framework* napisan u Ruby programskom jeziku. Tvorac Railsa je danski programer David Heinemeier Hansson koji je 2004. godine po prvi puta predstavio Rails kao open source ali je privilegije za izmjenu dao tek 2005. godine. [14] Nakon toga Apple je objavio kako će Rails integrirati u OS X, što su i učinili 2007. godine, te tada Rails počinje dobivati na popularnosti. [15]

Rails je programska biblioteka koja nadopunjuje Ruby programski jezik, odnosno RubyGem koji se instalira korištenjem komandne linije ali je više od same biblioteke ili nekog API-ja. [16]

Možemo reći kako je Rails framework koji služi za izradu web stranica. Kao takav Rails uspostavlja konvencije za jednostavnu suradnju i održavanje. Ove konvencije su kodificirane kao Rails API (sučelje aplikacijskog programiranja ili direktive koja upravlja kodom) te su one dokumentirane na službenim stranicama. [17]

Rails kombinira programski jezik Ruby sa HTML-om, CSS-om i JavaScrtipt-om kako bi kreirao web aplikacije koje se pokreću na serveru. Upravo zbog toga ga smatramo back-end web aplikacijskom platformom.

Struktura web stranica je sastavljena od 3 glavna elementa: HTML, CSS i JS. Statične web stranice moguće je napisati korištenjem jedne datoteke. Ako stranica zahtjeva korištenje baze podataka ili generira dinamički sadržaj broj linija koda se povećava. Server korištenjem Ruby programskog jezika može dinamički slagati web stranicu iz više datoteka i tako stranica postaje dinamična što znači da prelazi iz web stranice u web aplikaciju. Rails framework korištenjem Ruby programskog jezika organizira razbacane datoteke i uz pomoć svojih konvencija slaže u jednu funkcionalnu web aplikaciju. Kreiran je upravo za olakšavanje pisanja web aplikacija. [16]

Rastavljanjem koda web aplikacije na više manjih datoteka postiže se čitljivost i modularnost. Više osoba može nesmetano raditi na implementaciji novog ili refaktoriranju postojećeg koda.

Datoteke Rails aplikacije su organizirane po specifičnoj strukturi. Struktura je identična za sve Rails projekte što je velika prednost jer omogućuje programeru rad na više projekata u isto vrijeme (struktura na svim projektima je ista). [16] Na Tab. 3.1. prikazana je struktura datoteka Rails projekta.

+-app	- glavni direktorij aplikacije koji organizira strukturu operacije i u sebi sadrži brojne druge direktorije (assets sadrži js, css fileove itd.)
| +-assets	
| |	
| |	
| +-controllers	- datoteke kontrolera
| +-helpers	- pomoćne datoteke (dodatne metode)
| +-mailers	- datoteke vezane uz slanje mailova
| +-models	- datoteke modela
| +-views	- datoteke prikaza (viewa)
+-config	- konfiguracijske datoteke (rute i slično)
+-db	- baza podataka
+-lib	
+-log	- biblioteke
+-public	- zapisi pogreški
+-script	- skripte

Tab. 3.1. *Prikaz strukture datoteka. [16]*

Rails aplikacije rade po MVC (engl. *Model-View-Controller*) uzorku. Ne samo da je MVC obrazac sadržan u strukturi datoteka aplikacije, već ona postoji i u obliku hijerarhije Rails klasa. Objašnjenje MVC-a i osnovna struktura aplikacije je objašnjena u narednim potpoglavljima.

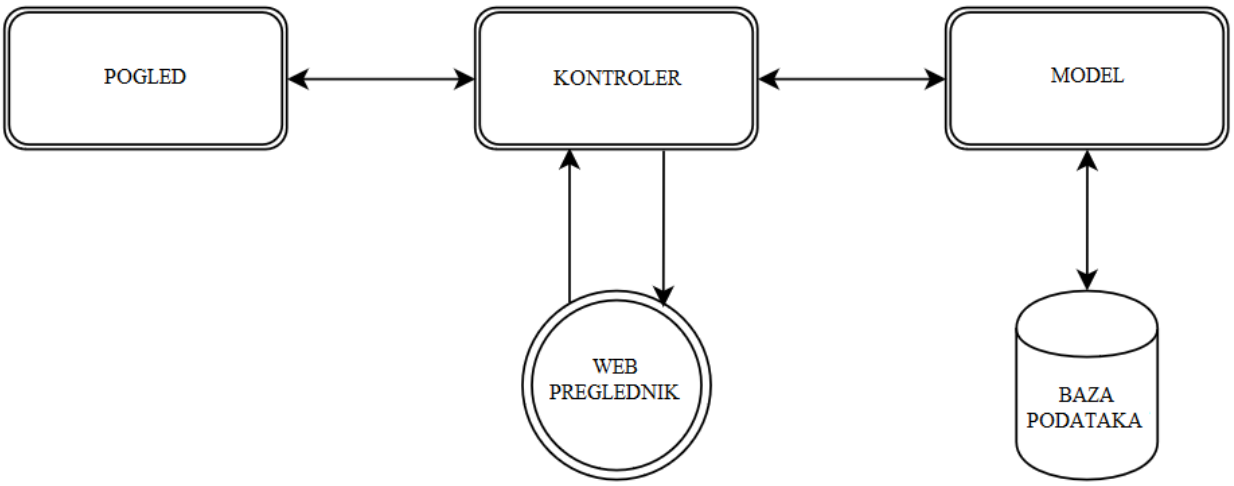
3.3 MVC (Model-View-Controller)

MVC je obrazac programske arhitekture. Koristi se u programskom inženjeringu za odvajanje pojedinih dijelova aplikacije u komponente ovisno o njihovoj namjeni. Razvijen je od strane Smalltalk programera, odnosno programera Trygve Reenskaug koji MVC po prvi puta opisuje u svom radu koji je objavljen 1979. godine. Tradicionalno se koristio za računalna grafička sučelja (GUI) ali je takva arhitektura postala popularna za dizajniranje web aplikacija pa čak i mobilnih aplikacija. [18]

Aplikacija je podijeljena u tri međusobno povezana dijela. To je učinjeno kako bi se odvojili interni prikazi informacija od načina na koji se podaci prikazuju i prihvaćaju od korisnika. MVC odvaja ove glavne komponente aplikacije kako bi se omogućila učinkovita upotreba koda i paralelni razvoj aplikacije. [19]

Kao što i sam naziv arhitekture govori aplikacija je podijeljena na modele, poglede i kontroler (upravljač). Svaka od tih komponenti izgrađena je za rukovanje pojedinim razvojnim aspektima aplikacije. Sama struktura Rails aplikacija nagovještava korištenje MVC arhitekture (unutar app direktorija se nalaze direktoriji controllers, models i views). Rails konvencija imenovanja modela je u jednini, a kontrolera u množini.

Na Sl. 3.7. prikazan je osnovni način rada MVC-a u Ruby on Rails okruženju. Web preglednik pri dolasku na web stranicu šalje zahtjev. Web server zaprima zahtjev i predaje ga upravitelju koji je zaslužan za odlučivanje o sljedećim koracima. U slučajevima statičnih stranica upravitelj odmah prikazuje pogled (view). U većini slučajeva web stranice su dinamične te ih je potrebno obogatiti podatcima iz baze. U tom slučaju upravitelj komunicira s modelom koji dohvaća potrebne podatke iz baze. Nakon dohvaćanja podataka počinje se slagati stranica korištenjem prikaza unutar kojih se prikazuju podatci te upravitelj vraća složenu stranicu web pregledniku u HTML formatu. [20]



Sl. 3.7. *Način rada MVC-a.*

U sljedećim potpoglavljima pobliže je objašnjen svaki od elemenata MVC-a te su prikazani primjeri korišteni za izradu ovog web sučelja.

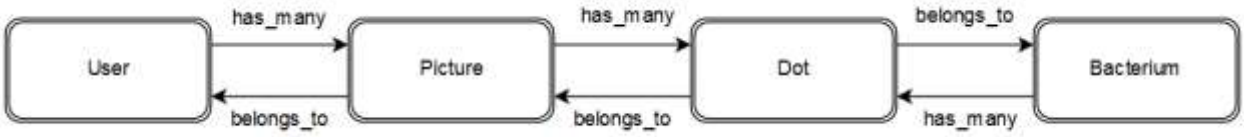
3.3.1 Model

Model je središnja komponenta uzorka koja predstavlja temeljnu logičku strukturu podataka u programskoj aplikaciji. Također održava odnose između objekata i baze podataka te obrađuje provjeru, povezivanje, transakcije, itd. Model obuhvaća podatke specifične za aplikaciju. Na primjer objekt modela može predstavljati korisnika, knjigu, igru, film, itd. [19]

Bitno je razlikovati kako model nije isto što i baza podataka. Model nam omogućuje transparentan i jednostavniji način prihvaćanja podatak iz baze, odnosno ActiveRecord nam omogućuje sučelje i vezanje između tablica u relacijskoj bazi podataka te tako možemo popunjavati i mijenjati bazu podataka korištenjem Ruby koda. U većini slučajeva je jedna tablica u bazi podataka predstavljena jednim modelom u MVC arhitekturi, ali to nije uvijek slučaj.

Na Sl. 3.8. prikazana je shema modela web sučelje za manualno brojanje bakterija se sastoji od četiri modela (korisnik, slika, točka i bakterija). Model korisnika je potreban za svako višekorisničko sučelje jer su u njemu sadržane informacije o svakom korisniku. Slika predstavlja uzorak na kojem obavljamo prebrojavanje. Naziv modela je proizvoljan te iako on predstavlja jednu petrijevu zdjelicu (uzorak) odabran je naziv slika radi lakšeg snalaženja u kodu. Točka predstavlja jednu bakterijsku koloniju na slici. Ona sadrži informacije o koordinatama točke,

veličini, itd. Zadnji model je bakterija koja predstavlja vrstu bakterije koja se označava te sadrži informacije o imenu i boji (kodirane su bojom).



Sl. 3.8. *Prikaz modela.*

Na Sl. 3.8 prikazani su modeli ali i veze između njih. Osim podataka unutar modela navode se i relacije između njih. Definiranjem veza Rails se, preko modela, automatski brine o povezivanju tablica unutar baze podataka pa nije potrebno koristiti komplekse SQL upite. Uzmimo za primjer model korisnika i slike. Jedan korisnik može postaviti mnogo slika, a jedna slika (uzorak) pripada samo jednom korisniku (onaj koji ju je postavio). Uz pomoć relacija has_many i belongs_to određujemo opisanu vezu između modela.

3.3.2 View (pogled)

View je vizualna reprezentacija, odnosno prikaz, svog modela. On je priključen na svoj model, ili njegov dio, i dobiva informacije za njegovu prezentaciju te zna kako odgovarati na inpute korisnika. Glavna zadaća mu je prikaz podataka modela i omogućiti korisniku sučelje za mijenjanje informacija kroz različite forme. Korisnici preko pogleda komuniciraju s modelom. [18]

Pogled može biti prikaz informacija poput grafikona, dijagrama, gumbova, formi itd. Nije potrebno prezentirati sve informacije koje model posjeduje pa zbog toga on djeluje i kao filter jer ograničava korisnikov pristup i potpunu izmjenu modela (samo ona količina koja mu je dopuštena kroz pogled).

Jedan model može imati više ili niti jedan view. Uzmimo za primjer model korisnika. Prvi dostupni view je stranica za registraciju. Tako, kroz kontrolirani pogled, dozvoljavamo korisniku izmjenu podataka (u ovom slučaju dodavanje novih informacija u bazu podataka). Drugi pogled istog modela je stranica za prijavljivanje. Različiti pogledi filtriraju različite informacije modela. Ako model nije potrebno prezentirati korisniku tada on nema niti jedan view.

View u Rails okruženju ima datoteku s ekstenzijom .erb (npr. user.html.erb) što označava *Embedded Ruby* datoteku. Kostur stranice je određen korištenjem HTML-a te je u ovom slučaju HTML prožet dinamičkim elementima koje popunjavamo korištenjem Ruby programskog jezika. Rails takvu datoteku popunjava dinamičkim elementima te ju prezentira web pregledniku kao HTML datoteku. Korištenjem opcije *inspect* u pregledniku možemo vidjeti kako su se dijelovi koda napisani u Ruby jeziku preveli u HTML.

3.3.3 Kontroler

Kontroler ili upravitelj služi kao posredničko sučelje između modela i pogleda te je njegova uloga odrađivanje sve poslovne logike i dolaznim zahtjevima. Također manipulira podatcima korištenjem modela i strukturira konačni prikaz za korisnika. [19]

Gledajući iz perspektive korisnika, upravitelj je mozak aplikacije. Služi kao veza između korisnika i samog sistema. Nakon interakcije korisnika s aplikacijom upravitelj obrađuje zahtjev te uz komunikaciju s modelom i pogledima sastavlja pravilan odgovor na korisnikovu akciju.

Kontroleru nije potrebno pridružiti model. Primjer su statične stranice koje upravitelj sastavlja iz više datoteka ali mu nisu potrebni podatci iz baze podataka. U web sučelju za brojanje bakterijskih kolonija postoji četiri modela i sedam upravitelja.

4. IMPLEMENTACIJA

U prethodnom poglavlju objašnjeni su osnovni alati i principi korišteni pri izradu web aplikacije. Potrebno je poznavati osnove radi lakšeg i boljeg razumijevanja same implementacije. Naredna poglavlja, uz prikaz isječaka kodova, objašnjavaju glavne funkcionalnosti web sučelja.

Rails sadrži jako velik broj pomoćnih naredbi pomoću kojih se, uz minimalno pisanje koda, u kratko vrijeme može napisati funkcionalna aplikacija. Naredbe se izvršavaju u terminalu (Linux) ili Comand Promptu (Windows). U Tab. 4.1. prikazan je popis osnovnih naredbi koje su korištene za izradu web sučelja.

`rails new`	`- kreiranje novog Rails projekta`
`rails generate`	`- naredba za kreiranje modela, kontrolera, pogleda, migracija i ostalog`
`rails server`	`- pokretanje servera`
`rails console`	`- pokretanje Rails konzole`
`rails test`	`- pokretanje testova`
`rails destroy`	`- naredba suprotna naredbi generate (briše sve pripadajuće datoteke)`

Tab. 4.1. *Osnovne Rails naredbe*

Cijela web aplikacija se sastoji od preko četiri tisuće datoteka i oko 1500 direktorija pa u ovom radu se neće prolaziti kroz svaki pojedini direktorij već će biti prikazani i objašnjeni samo ključni dijelovi aplikacije.

Implementacija koda podijeljena je u pet dijelova (struktura web sučelja, statične stranice, korisnici, bakterije, slike, dokumentacija i biblioteke) od kojih svaki pojedini dio predstavlja jednu tematsku cjelinu web aplikacije. Unutar svake cjeline prikazani su pogledi koji joj pripadaju s pripadajućim akcijama kontrolera i strukturom modela.

4.1 Struktura web sučelja

Zaglavlje s navigacijskim izbornikom i podnožje s informacijama o aplikaciji predstavljaju kostur aplikacije između kojeg se prikazuje sadržaj web stranice. Statične dijelove stranice nije potrebno definirati u svakoj view datoteci već se definiraju i layout datoteci koja se nalazi na putanji app/views/layout.

application.html.erb

```
<!DOCTYPE html>
<html>
<head>
  <title><%= full_title(yield(:title)) %></title>
  <%= csrf_meta_tags %>
  <%= stylesheet_link_tag    'application', media: 'all',
                             'data-turbolinks-track': 'reload' %>
  <%= javascript_include_tag 'application', 'data-turbolinks-track':
'reload' %>
  <%= render 'layouts/shim' %>
</head>
<body>
<%= render 'layouts/header' %>
<div class="container">
  <% flash.each do |message_type, message| %>
      <div class="alert alert-<%= message_type %>"><%= message %></div>
  <% end %>
  <%= yield %>
  <%= render 'layouts/footer' %>
  <%= debug(params) if Rails.env.development? %>
</div>
</body>
</html>
```

Sl. 4.1. *application.html.erb*

Kreiranjem projekta automatski je stvorena datoteka application.html.erb. Ona predstavlja temelj izgleda aplikacije te se svaki pogled generira se na osnovu ove nje. Sadržaj datoteke prikazan je na Sl. 4.1. i zbog svoje važnosti će biti u potpunosti objašnjen.

Struktura koda je identična HTML datotekama ali je u nju moguće, uz korištenje Ruby programskog jezika, dodavati dinamički generirane informacije. . <% %> i <%= %> služe za enkapsulaciju Ruby koda koji se prije prikaza u web pregledniku prevodi u HTML. Postoji razlika između navedenih oznaka. <% %> označava Ruby kod koji je potrebno izvršiti ali ga nije potrebno prikazati. Petlje i grananje su izvrstan primjer koda koji nije potrebno prikazivati. Ako se dio koda prikazuje korisniku i ima direktan utjecaj na pogled, tada se on piše unutar <%= %> oznaka.

Na početku dokumenta potrebno je objaviti web pregledniku koja verzija HTML-a je korištena pri pisanju. U ovom projektu korišten je HTML 5. Potrebno je naglasiti kako <!DOCTYPE> nije oznaka već instrukcija.

Nakon informacije o verziji, prepoznaje se klasična struktura HTML datoteke. Kod je omotan unutar <html> oznaka te je podijeljen na dva glavna dijela: glavu i tijelo. Unutar glave se

postavljaju informacije o metapodacima kao što su naslov stranice, skup znakova, stilove, veze, skripte, itd. Većina tih podataka se ne prikazuje korisniku.

U glavi su definirani podatci potrebni za rad stranice. csrf_meta_tags ,stylesheet_link_tag i javascript_include_tag su informacije potrebne za učitavanje CSS i JS datoteka. Unutar <title> oznaka po prvi puta se pojavljuje Ruby kod koji je omotan oznakama <%= %> što označava dio koda koji se prikazuje Title oznaka sadrži naslov web aplikacije koji će prikazan na kartici unutar web preglednika. Naslov stranice u ovom slučaju nije statičan već pozivamo metodu full_title kojoj predajemo yield(:title). Naredba yield označava sekciju unutar koje se prikazuje sadržaj iz drugog pogleda (view datoteke) ukoliko taj pogled predaje nešto sa simbolom :title. Ako pogled ne preda nikakav sadržaj onda ta sekcija ostaje prazna. U ovom slučaju metodi full_title predajemo naslov pogleda ili ništa.

Definiranje metoda unutar pogleda povećava broj linija koda i smanjuje njegovu čitljivost. Prateći konvenciju Rails aplikacije, metode korištene unutar pogleda spremaju se u helpere (pomoćnike) na putanji app/helpers i imenuju se po principu ime_helper.rb. Helperi su pomoćne datoteke koje služe za premještanje logike iz view datoteka kako bi pogledi bili što čišći i razumljiviji te se postiže modularnost aplikacije. Metode definirane u jednoj pomoćnoj datoteci se mogu koristiti u svim pogledima, što znači da helperi nisu nužno vezani za jedan pogled, model ili upravljač ali su često vezani za neku cjelinu. Helper datoteka sadrži samo Rails kod te zbog toga ima ekstenziju .rb.

Pozivom metode full_title u pogledu Rails automatski prepoznaje kako je ta metoda deklarirana u pomoćnoj datoteci (Sl. 4.2.). Ukoliko se metodi proslijedi naziv on se sprema u varijablu page_title, u suprotnom se ona ima vrijednost praznog stringa. Provjerava se vrijednost varijable page_title i na osnovu nje se određuje naslov cijele stranice.

application_helper.rb

```ruby
module ApplicationHelper

  # Returns the full title on a per-page basis.
  def full_title(page_title = '')
    base_title = "ColiCounter"
    if page_title.empty?
      base_title
    else
      page_title + " | " + base_title
    end
  end
end
```

Sl. 4.2. *Deklaracija full_title metode.*

Na kraju same glave HTML dokumenta (Sl. 4.1.) se nalazi još jedan dio koda napisan u Ruby jeziku <%= render 'layouts/shim' %>. Naredba render prikazuje partial datoteku, u ovom slučaju datoteku naziva shim koja se nalazi u layout direktoriju.

Partial datoteka sadrži djelomični dio view koda i označavaju se donjom povlakom (_parcijalna.html.erb). Služi kao ekstenzija pogleda koja se, kao i helper, koristi za organiziranje i čitljivost koda te nije namijenjen samostalnom korištenju već služi kao dio cjeline. Programer razbijanjem koda u više datoteka povećava produktivnost (jednostavnije se snaći u 3 datoteke s 50 linija koda, nego u jednoj datoteci sa 150 linija koda).

Naredbom render dio koda napisan u parcijalnoj datoteci se spaja s datotekom pogleda. Naredbom render 'layouts/shim' učitava se kod shim datoteke koji je potreban za prikaz stranice u pregledniku Internet Explorer (verzije manje od 9 ne podržavaju HTML 5). Kod je prebačen u parcijalnu datoteku jer je vezan za specifični slučaj te ga nije potrebno mijenjati.

U tijelu HTML dokumenta definiraju se svi ostali sadržaji stranice kao što su slike, tablice, tekst, liste, poveznice i ostalo. Zaglavlje i podnožje ostaje identično neovisno o tome gdje se nalazimo u aplikaciji i upravo iz tog razloga su definirani u kosturu aplikacije. Radi čitljivosti koda definirani su u parcijalnim datotekama. Na Sl. 4.3. je prikazan osnovni izgled sučelja.

Sl. 4.3. *Osnovni izgled sučelja.*

Sučelje je jako jednostavno i sastoji se od zaglavlja koji sadrži navigacijski izbornik i podnožja koji ima informaciju o autoru aplikacije te poveznicu na stranicu About. Zaglavlje i podnožje samo po sebi ne može biti kompletni prikaz ali funkcioniraju kao dio cjeline između kojeg se prikazuje dinamični sadržaj stranice.

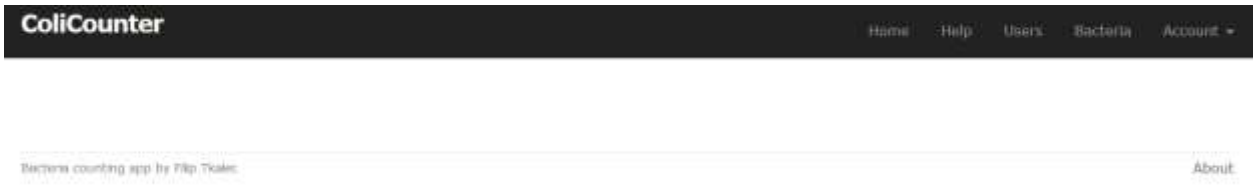
Navigacijski izbornik unutar zaglavlja nije u potpunosti statičan. U desnom uglu poveznica Log in (prijava) vodi na stranicu prijave korisnika. Nakon uspješne prijave, korisnika se usmjerava na početnu stranicu te tada izbornik u ovom stanju nije točan. Korisnik se uspješno prijavio na stranicu ali još uvijek mu se nudi opcija prijave. Na Sl. 4.4. prikazan je način kreiranja dinamičkih elemenata u Rails aplikacijama.

If grananje izvršava se između <% %> oznaka jer ništa nije potrebno predati pogledu. Metoda logged_in? je definirana u pomoćnoj datoteci te vraća true ako je korinik prijavljen i false ako nije. Ako je korisnik uspješno prijavljen tada mu se umjesto poveznice za prijavu prikazuju poveznice za koje samo prijavljeni korisnici imaju pristup i poveznica za odjavu.

header.html.erb

```
<% if logged_in? %>
    <li><%= link_to "Users", users_path %></li>
    <li><%= link_to "Bacteria", bacteria_path%></li>
                          .
                          .
                          .
          <%= link_to "Log out", logout_path, method: :delete %>
        </li>
      </ul>
    </li>
<% else %>
    <li><%= link_to "Log in", login_path %></li>
<% end %>
```

Sl. 4.4. *Isječak koda zaglavlja.*



Sl. 4.5. *Izgled sučelja nakon uspješne prijave.*

Na Sl. 4.5. prikazan je izgled sučelja nakon uspješnog prijavljivanja korisnika. Točkice između dijelova koda označavaju postojanje koda ali on nije potreban za samo razumijevanje pa zbog jednostavnosti nije niti prikazan.

Trenutno web sučelje je potpuno prazno jer generiran nikakav sadržaj. Između pozivanja zaglavlja i podnožja definirana je metoda za ispis poruka. Ako postoji flash poruka ona se ispisuje na tom dijelu. Flash poruke obavještavaju korisnika o radnjama kao što su uspješna registracija, bakterija je spremljena u bazu podataka i slično.

Najbitnija linija u ovoj datoteci je *<%= yield %>*. Kao što *yield(:title)* označava sekciju unutar koje će se prikazati sadržaj drugog viewa koji ima oznaku *:title*, u ovom slučaju yield prikazuje kompletan sadržaj druge view datoteke. Upravo zbog ovako strukturiranog koda govorimo kako je application.html.erb kostur web sučelja jer sve ostalo, osim sadržaja *yield*-a, ostaje identično u svakom prikazu.

Na samom kraju (Sl. 4.1. **Sl. *4.1*. Sl. *4.1*.**), ispod podnožja, ispisuju se informacije koje pomažu pri programiranju i pronalaženju pogreški (bugova). Ovaj blok informacija ispisuje se samo u developement načinu rada te prikazuje informacije o modelima i upravljačima koji se koriste na toj stranici. Rails ima 3 osnovna načina rada: produkcijsko, testno i razvojno. Produkcijsko okruženje je ono koje korisnik vidi. Developement služi za razvoj stranice i dodavanje novih mogućnosti, a test okruženje služi za provjeru i testiranje aplikacije.

4.2 Statične stranice

U ovoj aplikaciji postoje 4 statične stranice: home, webcam, help i about. Home predstavlja početnu stranicu web aplikacije i nije u potpunosti statična jer komunicira s modelom slika.

Rails projekt sadrži datoteku unutar koje su definirane sve rute aplikacije. Na Sl. 4.6. prikazane su rute statičkih stranica. Možemo vidjeti kako je root, odnosno početna stranica aplikacije, postavljena na 'static_pages#home. Na osnovu ove datoteke Rails prosljeđuje upite pravom upravljaču.

routes.rb

```
Rails.application.routes.draw do
  root 'static_pages#home'
  get  '/help',    to: 'static_pages#help'
  get  '/about',   to: 'static_pages#about'
  get  '/webcam',  to: 'static_pages#webcam'
end
```

Sl. 4.6. *Rute aplikacije.*

Svakom pogledu, odnosno akciji, pripada metoda unutar upravljača. Na Sl. 4.7. prikazana je metoda home koja je samo definirana ali stranica je pravilno prikazana. Iako je metoda home prazna, ona u sebi posjeduje veliku količinu ograđenih funkcija. Rješenje ove tajne se krije u samom Railsu i nasljeđivanju. Klasa StaticPagesController nasljeđuje klasu ApplicationController koja posjeduje potrebne metode te tako, bez dodatnog pisanja koda, metoda home koristi funckije deklarirane unutar ApplicationControllera.

Početna stranica (home) i webcam nisu u potpunosti statične jer nakon uspješne prijave korisnika postaju sučelje za spremanje slika koje se koriste u daljnjem procesu brojanja bakterija. Preostale stranice help i about su u potpunosti statične. Help stranica služio kao vodič pri korištenju sučelja, a about stranica sadrži kratki sažetak rada.

static_pages_controller.rb

```
class StaticPagesController < ApplicationController
  def home
  end
end
```

Sl. 4.7. *StaticPages#home.*

4.3 Korisnici

Korisnicima pri dolasku na web sučelje je dozvoljena samo osnovna interakcija i pregled statičnih stranica aplikacije (Help i About) Kreiranjem korisničkog računa korisnicima se dozvoljava pristup ostalim dijelovima aplikacije. Uspješnom prijavom, korisničko sučelje se mijenja te se korisniku prikazuju nove poveznice u navigacijskom izborniku do kojih nije moguće doći ako korisnik nije prijavljen.

Aplikaciju je potrebno zaštititi od neprikladnog pristupa definiranjem razina prava korisnicima te onemogućiti pristup određenim stranicama korisnicima koji nisu prijavljeni. Zaštita sustava je usko vezana s modelom korisnika pa se ona opsuje u ovom potpoglavlju.

Model korisnika implementiran je korištenjem CRUD principa. CRUD je skraćenica od *Create* (kreiranje), *Read* (čitanje), *Update* (izmjena) i *Delete* (brisanje) koja se često koristi u programiranju za opisivanje korisničkog sučelja. Model korisnika se idealno uklapa u CRUD način rada jer su mu omogućene sve četiri opcije. Rails u potpunosti podržava ovaj način rada te *rails generate scaffold* naredbom moguće je kreirati model, kontorler i prikaze za rukovanje modelom (generira kostur CRUD principa).

Dolaskom na početnu stranicu korisniku se nudi opcija registracije (ako nije prijavljen) te se klikom na gumb web preglednik preusmjerava /signup page koja sadrži formu prikazanu na Sl. 4.8.

Sl. 4.8. *Registracijska forma.*

Registracija korisnika zahtjeva tri informacije: ime kojim će se korisnik predstavljati na stranici, email adresa i lozinka. Forma je uređena korištenjem bootstrapa koji je jedna od najpopularnijih biblioteka za front end pomoću koje se na vrlo jednostavan način pišu responzivne stranice [21]. Svako polje forme sadrži primjer informacije koju je potrebno upisati te odabirom polja ispisuje se poruka s detaljnijim informacijama. Npr. postavljanjem miša na polje za unos lozinke ispisuje se instrukcija o njezinoj duljini (potrebno je minimalno 6 znakova).

Prilikom registracije korisnika potrebno je osigurati određene mjere zaštite i validacije unesenih podataka. Model komunicira s bazom podataka i daje sučelje za pregled i njihovo manipuliranje pa se zbog toga validacija podataka osigurava na razini modela. Na Sl. 4.9. prikazan je dio koda modela kojim se osigurava pravilnost unesenih podataka naredbom validates.

user.rb

```
validates :name, presence: true, length: { maximum: 50 }
VALID_EMAIL_REGEX = /\A[\w+\-.]+@[a-z\d\-]+(\.[a-z\d\-]+)*\.[a-z]+\z/i
validates :email, presence:   true, length: { maximum: 255 },
          format:     { with: VALID_EMAIL_REGEX },
          uniqueness: { case_sensitive: false }
has_secure_password
validates :password, presence: true, length: { minimum: 6 }, allow_nil: true
```

Sl. 4.9. *Validacija podataka korisnika.*

Ime korisnika ne smije biti prazno (presence: true). Također maksimalni broj dozvoljenih znakova je 50. Konstante u Ruby jeziku označavaju se s velikim slovima te s toga konstanta VALID_EMAIL_REGEX sadrži regularni izraz koji provjerava pravilnost formata upisane email adrese. Email adresa koristi se pri prijavljivanju korisnika pa je zbog toga, osim pravilnog formata, potrebno osigurati unikatnost svake mail adrese (u bazi ne smiju postojati dvije ili više identičnih mail adresa). Unikatnost osiguravamo korištenjem naredbe uniqueness.

Najveća razina sigurnosti je potrebna pri unošenju, validaciji i spremanju lozinke. Lozinka se mora sastojati od minimalno šest znakova koje je potrebno zamaskirati pri unošenju. Spremanje lozinke u bazu kao string je veliki sigurnosi propust.

U ovoj aplikaciji koristi se gem bycript koji predani string pretvara u hash. Hash je u ovom slučaju string koji se sastoji od znakova dobivenih algoritmom koji su napisali Niels Provos i David Mazières [22]. Korisnik unosi lozinku te nakon pritiska gumba za završetak, string se uz pomoć bycript gema pretvara u hash koji se onda sprema u bazu podataka. Čak niti administrator koji ima

ovlasti za pregledavanje baze podataka ne može vidjeti pravu vrijednost lozinke, samo hash. Ovaj proces nije reverzibilan te nije moguće iz hash izraza dobiti string koji je korisnik upisao. Ovo je bitno u slučaju da netko uspije doći do baze podataka, lozinke korisnika su sigurne. Prilikom prijave korisnika potrebno je provesti isti postupak. Od korisnika se zahtjeva unos lozinke koja se pretvara u hash te on uspoređuje sa hashem koji je spremljen u bazi podataka.

Nakon popunjavanja obrasca korisnik klikom na gumb završava proces registracije. Ako su sve informacije validne tada se korisnika preusmjerava na pregled njegovog korisničkog računa. Potrebno je napomenuti kako je korisnik automatski prijavljen nakon procesa registracije. Ako jedan od podataka nije unesen ili nije validan, tada se korisniku ispisuje upozorenje o nepravilnom unosu.

Proces izmjene korisnika gotovo je identičan kao i proces registracije. Kroz formu identičnoj za kreiranje računa unose novi podaci te model pronalazi u bazi korisnika na kojeg se izmjene odnose te mijenja podatke u bazi.

Uz pomoć sesije korisnik je prijavljen na stranicu sve dok ima otvoren preglednik. Zatvaranjem preglednika prekida se sesija te korisnik više nije prijavljen. Korisniku se tijekom prijave nudi opcija pamćenja prijave korištenjem kolačića. Korištenjem kolačića korisnik ostaje prijavljen na stranicu bez obzira na zatvaranje web preglednika i uništava se jedino korištenjem opcije odjave.

Pravilna implementacija kolačića je bitna zbog sigurnosnih razloga. U model korisnika je potrebno dodati novu informaciju, odnosno string, koji će označavati kolačić. Nakon uspješne prijave u model korisnika sprema se kriptirani kolačić vezan za korisnika. Ista informacija sprema se i na korisnikov web preglednik. Dolaskom na stranicu provjerava se postoji li kolačić na korisnikovom pregledniku i postoji li isti u bazi podataka aplikacije. Ako postoji, korisnik je automatski prijavljen.

Osim pregledavanja svog repozitorija i izmjene korisničkog računa moguće je dobiti popis svih korisnika i pregledavati njihove repozitorije. Indeks stranica se najčešće koristi u svrhu prikaza svih zapisa u bazi odabranog modela.

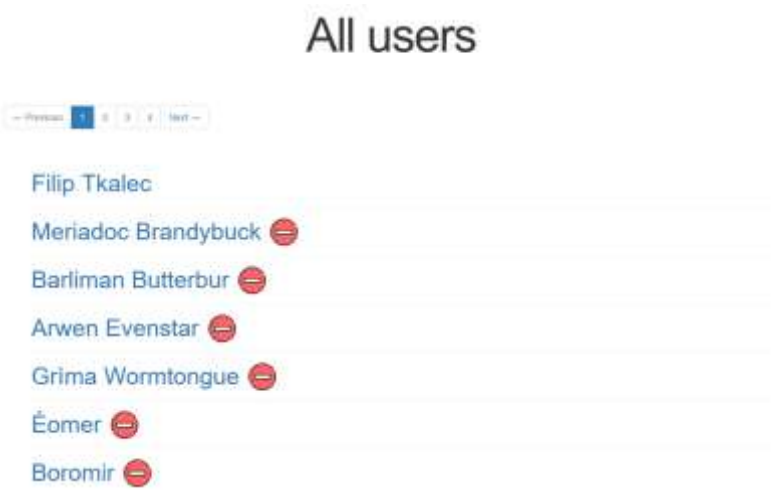
Lista korisnika ima dinamički prikaz koji ovisi o pravima. Obični korisnik je klikom preusmjeren na repozitorij tog korisnika, a administratoru je osim popisa korisnika dostupna opcija brisanja korisničkog računa. Na Sl. 4.10. prikazana su razina prava korisnika te index metoda.

users_controller.rb

```
before_action :logged_in_user, only: [:index, :edit, :update, :destroy]
before_action :correct_user,   only: [:edit, :update]
before_action :admin_user,     only: :destroy

def index
  @users = User.paginate(page: params[:page])
end
```

Sl. 4.10. *Indeks metoda i razina prava korisnika*

Index metoda u varijablu instance @user skuplja sve korisnike koji postoje u bazi podataka. Zbog velikog broja korisnika u bazi korišten je paginate gem koji omogućuje prikaz određenog broja korisnika u isto vrijeme (u ovom slučaju 30). Na Sl. 4.11. prikazana je lista korisnika s administracijskim pravima. Za demonstrativne svrhe korisnički računi su kreirani korištenjem faker gema (automatski popunjava bazu podataka).



Sl. 4.11. *Lista korisnika (administrator)*

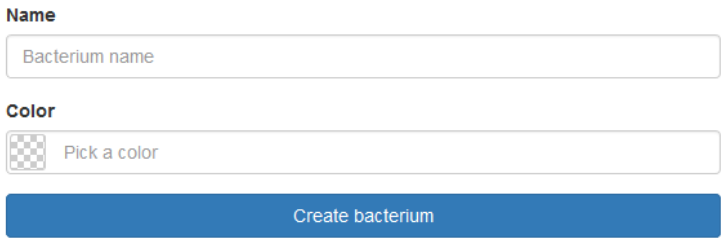
4.4 Bakterije

Model bakterija predstavlja stvarne bakterije koje se označavaju na slikama (uzorcima). Svaka vrsta bakterije kodirana je određenom bojom. Struktura je jednostavna i sastoji se od samo dvije informacije: ime bakterije i boje. Model bakterije direktno je vezan na model točke što znači da svaka oznaka na slici predstavlja jednu bakterijsku koloniju.

Kreiranje, brisanje i mijenjanje bakterija dozvoljeno je samo korisnicima s administratorskim pravima. Zbog globalnog korištenja ovog modela od strane svih korisnika potrebno je uvesti mjere zaštite. Brisanje bakterije u bazi podataka uzrokuje brisanje svih oznaka te bakterije na uzorcima.

Na Sl. 4.12. prikazan je obrazac za dodavanje nove bakterije. Potrebno je unijeti ime bakterije i odabrati boju. U polje je moguće upisati heksadekadni broj boje ili ju izabrati korištenjem birača boja koji je implementiran korištenjem minicolors gema.

Indeks stranica s listom bakterija prikazana je na Sl. 4.13. Svi prijavljeni korisnici imaju pristup ovoj stranici ali samo administratori imaju mogućnost izmjene i brisanja bakterija. Također zbog velikog broja bakterija koristi se pagination gem koji prikazuje samo određeni broj po stranici. Pritiskom na gumb + pokraj naslova korsika se (administratora) preusmjerava na obrazac za kreiranje nove bakterije.



Sl. 4.12. *Obrazac za dodavanje bakterije.*

Sl. 4.13. *Popis bakterija.*

4.5 Slike i oznake

U ovom potpoglavlju objašnjava se način implementacije glavne funkcionalnosti web sučelja. Model slike i oznake (točke) su usko vezani te ih nije moguće objasniti posebno. Slika označava uzorak, odnosno sliku koja je učitana s računala ili snimljena preko web kamere, a točka prikazuje bakterijsku koloniju na uzorku i sadrži informacije o koordinatama.

Prije korištenja funkcije označavanja potrebno je posjedovati korisnički račun. Uspješnim prijavljivanjem početna stranica otključava mogućnost postavljanja slike. Na Sl. 4.14. prikazana je forma za učitavanje slike s računala, a Sl. 4.15. prikazuje formu za web kameru. Oba načina zahtijevaju unos imena slike.

Sl. 4.14. *Obrazac za učitavanje slike sa računala.*

Sl. 4.15. *Obrazac za učitavanje slike web kamerom.*

Spremanje slike u bazu podataka implementirano je korištenjem carrierwave gema. Pri učitavanju s računala, slika se sprema u direktorij aplikacije i u bazu se pohranjuje putanja do slike. U slučaju web kamere slika se u bazu podataka sprema kao blob korištenjem base64 konverzije (postupak

pretvaranja slike u niz znakova). Proces je reverzibilan te se slika prije prikaza prevodi iz bloba u grafički oblik.

pictures_controller.rb

```
def create
  @picture = current_user.pictures.build(picture_params)
  if @picture.save
    flash[:success] = "Picture saved!"
    redirect_to edit_picture_path(@picture)
  else
    render 'static_pages/home'
  end
end

def edit
  @picture = Picture.find(params[:id])
end
```

Sl. 4.16. *PicturesController#create, edit, update*

Metode create i update prikazane na Sl. 4.16. služe za učitavanje slika i njihovu izmjenu. Uspješnim postavljanjem slike (if @picture.save) korisnik je preusmjeren na edit stranicu koja u ovom slučaju označava pogled za brojanje bakterija.

Na ***Sl. 4.17.*** prikazan je izgled sučelja za označavanje kolonija. Na samom vrhu nalazi se proizvoljno ime slike. Najveći dio zauzima slika petrijeve zdjelice koja predstavlja jedan utorak dok se na desnoj strani prozora nalaze alati za označavanje.

Sl. 4.17. *Sučelje za označavanje bakterijskih kolonija.*

U prvom prozoru na desnoj strani nalazi se padajući izbornik sa svim bakterijama u bazi podataka. Odabirom bakterije mijenja se boja oznake (boja je zapisana u modelu bakterije). Ime bakterije je prikazano u izborniku, dok je veličina i boje točke moguće provjeriti na demo točki koja se nalazi na dnu prozora. Gumbima + i – podešava se veličina oznake, odnosno mijenja se radijus točke.

Između prozora nalaze se dva gumba. Klikom na *show repository* korisnika se preusmjerava na njegov profil gdje je prikaz svih postavljenih slika. Pomoću drugog gumba možemo prikazati/sakriti oznake.

Drugi prozor sadrži statistiku označavanja. Na samom vrhu prikazuje se ukupan broj bakterijskih kolonija na uzorku koji se dinamički smanjuje/povećava. Omogućeno je označavanje više vrsta bakterijskih kolonija na istoj slici. Nakon označavanja jedne vrste moguće je u padajućem izborniku odabrati drugu i nastaviti sa označavanjem. Za detaljniju statistiku potrebno slati upit bazi podataka. Klikom na gumb osvježavanja postavlja se upit bazi koji vraća broj svake pojedine vrste bakterije. Potrebno je napomenuti kako se sam upit automatski pokreće pri otvaranju stranice pa ga nije potrebno manualno aktivirati svaki puta.

Kolonija se označava lijevim klikom na sliku koji aktivira ajax poziv prikazan na Sl. 4.18. i parametri oznake se automatski spremaju u bazu podataka. Ajax je asinkrona metoda koja omogućuje dinamičku promjenu strance tijekom koje nije potrebno osvježiti prozor web preglednika kako bi se rezultat akcije prikazao. Klikom se šalje POST HTTP zahtjev na rutu koja aktivira kreiranje nove oznake (dot). U pozivu je potrebno poslati sve podatke koji model točke sadrži u ovom slučaju *id* slike na koju se oznaka odnosi, id bakterije koju oznaka predstavlja te informacije o položaju i veličini oznake na slici.

script.html.erb

```
$.ajax({
    type: "POST",
    dataType: "script",
    url: "http://localhost:3000/dots",
    data: { dot: {picture_id: <%= @picture.id %>, x: x, y: y, r: r,
bacterium_id: dropdown_id} },
    success: function(resp){
    }
});
```

Sl. 4.18. *Ajax poziv za spremanje točke.*

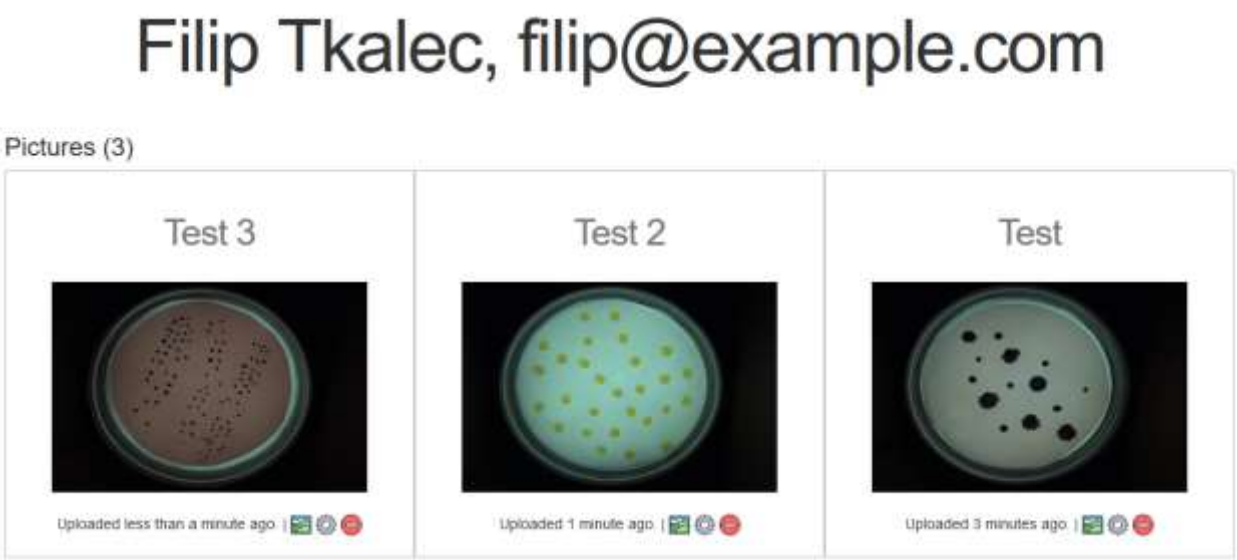
_script.html.erb

```
$.ajax({
    type: "POST",
    dataType: "script",
    url: "http://localhost:3000/dots/" + dot_id,
    data: { "_method":"delete"},
    success: function(resp){ }
});
```

Sl. 4.19. *Ajax piziv za brisanje točke.*

Oznaka je istog trenutka vidljiva na slici te ju je moguće obrisati klikom. Aktivira se drugi ajax poziv koji šalje DELETE zahtjev za brisanjem točke iz baze podataka (Sl. 4.19.). Rezultat ove akcije je brisanje točke sa slike i iz baze podataka.

Profil korisnika također služi kao repozitorij slika koji nudi retrospektivni prikaz svih označavanja. Samo korisnici koji su sliku postavili na web aplikaciju imaju mogućnost njezine izmjene i brisanja, dok su ostalim korisnicima te mogućnosti sakrivene. Na Sl. 4.20. prikazan je izgled repozitorija. Osim imena i email adrese korisnika prikazana je informacija s brojem slika koje je korisnik učitao. Glavni dio repozitorija je tablica koja sadrži smanjene prikaze slika s naslovima i informacijom o vremenu učitavanja slike. Slike su sortirane po vremenu postavljanja, s tim da su prvo prikazane novije slike. Klik na ikonicu slike vodi do većeg prikaza s detaljnim informacijama o označavanju.



Sl. 4.20. *Prikaz repozitorija.*

5. TESTIRANJE WEB SUČELJA

Pisanje testova osigurava pravilno funkcioniranje aplikacije ali i olakšava dodavanje novih mogućnosti. Svaki dio projekta bi trebao biti pokriven testom. Prilikom implementacije i dodavanja novog koda može se utjecati na stari dio koda koji nije u direktnoj povezanosti s izmjenama što može prouzročiti štetu ako se ne otkrije na vrijeme. Jednostavnim pokretanjem testova nakon svake izmjene osigurava se pravilna funkcionalnost cijele aplikacije.

Važnost testova pokazuje TDD *(test driven development)* pristup razvoja. Temelji se na principu pisanja testova prije implementiranja funkcionalnosti. Test će padati sve dok funkcionalnost nije ispravno implementirana. Ako je test dobro napisan tada prolazak označava pravilno implementiranu funkcionalnost.

Pisanje testova za Rails aplikacije je jednostavno jer on svojom strukturom datoteka potiče testiranje. Jedan od glavnih direktorija aplikacije je *test* koja sadrži test datoteke. Rails podržava četiri vrste testova: test modela, test kontrolera, test pogleda i test značajke. [23]

Kao što i ime govori model testovi služe za testiranje svakog pojedinog modela aplikacije i spremaju se na putanji test/models. Najčešće se testiraju validacije polja modela. Na Sl. 5.1. prikazan je samo dio testa modela korisnika. Na samom početku kreirana je varijabla @user koja predstavlja korisnika s pravilnim podatcima. Ova varijabla služi pri daljnjem testiranju. Svaki dio testa sadrži tekstualni opis provjere. Naredba assert očekuje uspješnu radnju, a assert_not neuspješnu. Prvo se naredbom assert @user.valid? provjerava ispravnost kreiranog korisnika. Nakon tog provjerava se pravilnost svakog polja modela: ime i email trebaju biti prisutni, ime ne smije biti duže od 50 znakova, itd. U testu se prolazi kroz sve moguće slučajeve i tako je osiguran pravilan rad modela korisnika.

Kontroler testovi osiguravaju pravilan rad kontolera. Služe za testiranje pravilnog postupka upravljača nakon određene akcije, npr. pravilno preusmjeravanja korisnika nakon registracije. Na Sl. 5.2. prikazan je dio testa upravljača. Kao i kod testa modela prvo je potrebno kreirati varijable korisnika. Prvi test provjerava preusmjeravanje korisnika na stranicu prijave ako je pokušao pristupiti stranici za izmjenu računa prije uspješne registracije. Također ovo je samo dio testova od kojih su pokriveni svi mogući događaji.

user_test.rb

```ruby
class UserTest < ActiveSupport::TestCase

  def setup
    @user = User.new(name: "Example User", email: "user@example.com",
                     password: "foobar", password_confirmation: "foobar")
  end

  test "should be valid" do
    assert @user.valid?
  end

  test "name should be present" do
    @user.name = ""
    assert_not @user.valid?
  end

  test "email should be present" do
    @user.email = "     "
    assert_not @user.valid?
  end

  test "name should not be too long" do
    @user.name = "a" * 51
    assert_not @user.valid?
  end
end
```

Sl. 5.1. *Test modela korisnika.*

user_controller_test.rb

```ruby
class UsersControllerTest < ActionDispatch::IntegrationTest

  def setup
    @user       = users(:filip)
    @other_user = users(:karlo)
  end

  test "should redirect update when not logged in" do
    patch user_path(@user), params: { user: { name: @user.name,
                                              email: @user.email } }
    assert_not flash.empty?
    assert_redirected_to login_url
  end

  test "should get new" do
    get signup_path
    assert_response :success
  end

  test "should redirect edit when logged in as wrong user" do
    log_in_as(@other_user)
    get edit_user_path(@user)
    assert flash.empty?
    assert_redirected_to root_url
  end
end
```

Sl. 5.2. *Test upravljača korisnika.*

User_signup_test.rb

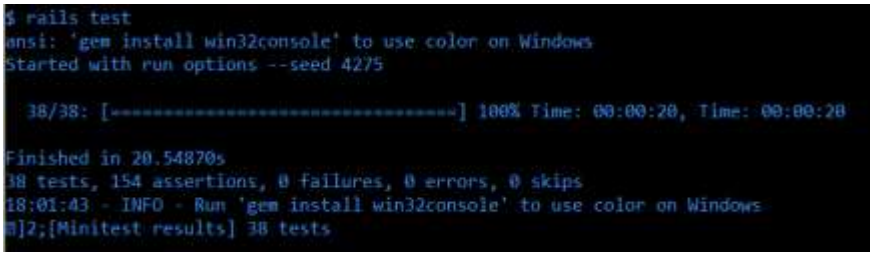
```ruby
class UsersSignupTest < ActionDispatch::IntegrationTest

  test "valid signup information" do
    get signup_path
    assert_difference 'User.count', 1 do
      post users_path, params: { user: { name:  "Example User",
                                         email: "user@example.com",
                                         password:              "password",
                                         password_confirmation: "password" }
}
    end
    follow_redirect!
    assert_template 'users/show'
    assert is_logged_in?
  end
end
```

Sl. 5.3. *Integracijski test kreiranja računa.*

Integracijski testovi su testovi visoke razine koji imitiraju korištenje stranice. Osiguravaju pravilno funkcioniranje svakog dijela aplikacije i nisu nužno vezani samo za model, kontroler ili pogled. Napisani su iz perspektive korisnika koji klika po stranici, ispunjava obrasce itd. Najpopularniji gemovi koji osiguravaju pisanje jednostavnih i kvalitetnih integracijskih testova su: RSpec i Capybara.

Na Sl. 5.3. prikazan je dio integracijskog testa za registraciju korisnika. U ovom slučaju pokriva se scenario uspješne registracije. Prvo se dohvaća stranica za registraciju i popunjava obrazac. Nakon slanja zahtjeva ispituje se preusmjeravanje na profil korisnika. Također se na samom kraju testa provjerava da li je korisnik automatski prijavljen nakon registracije.

Prije samog pokretanja testova potrebno je kreirati testnu bazu naredbom *rails db:test:prepare*. Koristi se testna baza zbog sigurnosnih razloga (produkcijska baza se ne bi trebala mijenjati). Ispis naredbe rails test možemo vidjeti na Sl. 5.4. Izvršavanje 38 testova je trajalo 20 sekundi i kao što možemo vidjeti svi su uspješno prošli što znači da svi dijelovi aplikacije pravilno funkcioniraju.



Sl. 5.4. *Ispis naredbe rails test.*

6. ZAKLJUČAK

Tema ovog diplomskog rada je implementacija web sučelja za manualno označavanje bakterijskih kolonija s mogućnošću reprodukcije rezultata. Kreirano je višekorisničko sučelje koje ima mogućnosti stvaranja, brisanja i izmjenu računa s naglaskom na sigurnost. Web aplikacija je implementirana korištenjem Ruby on Rails frameworka.

Aplikacija podržava učitavanja slika s računala ili korištenjem web kamere. Korisniku je nakon učitavanja slike omogućeno označavanje bakterijskih kolonija na slici korištenjem alata za izbor bakterija i veličine oznake. Osim označavanja implementirana je mogućnost reprodukcije rezultata preko profila korisnika koji služi i kao repozitorij. Detalji aplikacije su implementirani korištenjem gemova.

Web sučelja za manualno brojanje bakterija moguće je primijeniti u obrazovne svrhe. Sustavi za automatsko i poluatomatsko brojanje su nedostupni sveučilištima zbog svoje visoke cijene. Korištenje web aplikacije olakšava se proces brojanja i spremanja rezultata. Profesorima se također nudi mogućnost praćenja rada studenata.

Aplikacija služi kao kostur, odnosno dobra podloga za daljnje unapređivanje. Jedan od prijedloga unaprjeđivanja je implementacija filtera u repozitoriju s mogućnošću generiranja izvješća. Ova mogućnost je korisna ako je potrebno prikazati statistiku brojanja po određenim parametrima. Daljnje nadograđivanje može biti vođeno svrhom korištenja. Ukoliko će aplikacija uistinu biti korištena na fakultetima potrebno je implementirati razina prava profesorskog korisničkog računa, omogućiti predavanje dokumenata uz samo označavanje, mogućnost zadavanja i praćenja zadataka preko aplikacije itd. Kao što vidimo mogućnosti za daljnji napredak je velik.

Ovaj rad je rezultirao uspješnom web aplikacijom za brojanje bakterijskih kolonija koja može poslužiti kao osnova pri unapređivanju i daljnjem implementiranju u specifične svrhe.

LITERATURA

[1] D. H. Davey, »Microbial Growth Website,« Aberystwyth University, [Mrežno]. Available: http://users.aber.ac.uk/hlr/mpbb/index_files/Page299.html. [Pokušaj pristupa 6 9 2017].

[2] D. C. Oberg's, »MICROBIAL TESTING PROCEDURES,« Web State University, [Mrežno]. Available: http://faculty.weber.edu/coberg/class/3853/3853%20Microbial%20Testing%20Procedures.htm. [Pokušaj pristupa 6 9 2017].

[3] »Bacterial Counts - Quantitative Analysis of Microbes,« [Mrežno]. Available: https://www.researchgate.net/file.PostFileLoader.html?id=594e801ddc332dc50619443d&assetKey=AS%3A508791812784128%401498316829265. [Pokušaj pristupa 9 17 2017].

[4] I. V. E. S. F. T. Tomislav Matic´, »Semi-automatic Prototype System for Bacterial,« 2016.

[5] L. L. Hester, M. A. Sarvary i C. J. Ptak, »Mutation and Selection: An Exploration of Antibiotic,« 2014.

[6] D. H. Davey, »Microbial Growth Website,« Aberystwyth University, [Mrežno]. Available: http://users.aber.ac.uk/hlr/mpbb/index_files/Page1018.html. [Pokušaj pristupa 6 9 2017].

[7] L. U. Shijun Liu, »All About Agar,« [Mrežno]. Available: https://www.sciencebuddies.org/science-fair-projects/references/grow-microbes-agar. [Pokušaj pristupa 16 9 2017].

[8] M.-B. LABORATORIES, »Aerobic Plate Count,« [Mrežno]. Available: http://mb-labs.com/resources/aerobic-plate-count/. [Pokušaj pristupa 7 9 2017].

[9] »Bacteria Growing Experiments in Petri Plates,« SCIENCE Company, [Mrežno]. Available: https://www.sciencecompany.com/Bacteria-Growing-Experiments-in-Petri-Plates.aspx. [Pokušaj pristupa 16 9 2017].

[10] M. Rachel Watson, »THE VIRTUAL EDGE: Lab 5 Cultivation of Bacteria I,« [Mrežno]. Available: http://www.uwyo.edu/virtual_edge/results/enumeration_results.htm. [Pokušaj pristupa 21 9 2017].

[11] S. Turczyn, P. Ng i B. Griffth, »A Brief History of Ruby,« [Mrežno]. Available: https://launchschool.com/books/ruby/read/introduction. [Pokušaj pristupa 9 9 2017].

[12] D. Thomas i A. Hunt, »The Ruby Language FAQ,« [Mrežno]. Available: http://ruby-doc.org/docs/ruby-doc-bundle/FAQ/FAQ.html. [Pokušaj pristupa 16 9 2017].

[13] R. community, »About Ruby,« [Mrežno]. Available: https://www.ruby-lang.org/en/about/. [Pokušaj pristupa 9 7 2017].

[14] L. Grimmer, »Interview with David Heinemeier Hansson from Ruby on Rails,« 2006. [Mrežno]. Available: https://web.archive.org/web/20130225091835/http://dev.mysql.com/tech-resources/interviews/david-heinemeier-hansson-rails.html. [Pokušaj pristupa 16 9 2017].

[15] David, »Ruby on Rails will ship with OS X 10.5 (Leopard),« 2006. [Mrežno]. Available: http://weblog.rubyonrails.org/2006/8/7/ruby-on-rails-will-ship-with-os-x-10-5-leopard/. [Pokušaj pristupa 16 9 2017].

[16] D. Kehoe, Learn Ruby on Rails: Book One, 2016.

[17] »Rails Guides,« [Mrežno]. Available: http://guides.rubyonrails.org/. [Pokušaj pristupa 16 9 2017].

[18] J. Atwood, »Understanding Model-View-Controller,« Coding Horror, 5 5 2008. [Mrežno]. Available: https://blog.codinghorror.com/understanding-model-view-controller/. [Pokušaj pristupa 10 9 2017].

[19] »MVC Framework - Introduction,« tutorials point, [Mrežno]. Available: https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm. [Pokušaj pristupa 10 9 2017].

[20] M. Hartl, Ruby on Rails Tutorial: Learn Web Development with Rails (4th Edition), Addison-Wesley, 2017.

[21] »Bootstrap,« [Mrežno]. Available: http://getbootstrap.com/. [Pokušaj pristupa 20 9 2017].

[22] T. Schuck, »bcrypt-ruby,« [Mrežno]. Available: https://github.com/codahale/bcrypt-ruby. [Pokušaj pristupa 9 20 2017].

[23] »A Guide to Testing Rails Applications,« [Mrežno]. Available: http://guides.rubyonrails.org/testing.html. [Pokušaj pristupa 18 9 2017].

[24] »Getting Started with Rails,« [Mrežno]. Available: http://guides.rubyonrails.org/getting_started.html. [Pokušaj pristupa 16 9 2017].

SAŽETAK

U uvodnom dijelu ovog rada ističe se važnost brojanja bakterijskih kolonija na području mikrobiologije i primjena u industriji. U narednom poglavlju prikazan je proces pripreme uzorka od prikupljanja, preko inkubiranja do brojanja. Objašnjava se metoda manualnog brojanja i ističu njezine mane. Prije implementacije web sučelja objašnjeni su alati i principi korišteni pri izradi kao i osnovna struktura web sučelja. Implementacijski dio uz prikaze koda objašnjava način rada aplikacije za manualno brojanje bakterija. Zadnje poglavlje fokusira se na testiranje.

Ključne riječi: bakterije, brojanje, rails, web

ABSTRACT

WEB APLICATION FOR MARKING BACTERIAL COLONIES

The introductory part of this paper emphasizes the importance of counting bacterial colonies in the field of microbiology and its application in the industry. In the next chapter, a sample preparation process is presented, from collecting through incubation to counting. It explains the manual counting method and highlights its hive. Before implementation chapter, the tools and principles used in the design are explained and the basic structure of the web site is presented. The implementation section with code views explains the mode of operation. The last chapter focuses on testing.

Key words: bactera, counting, rails, web

ŽIVOTOPIS

Filip Tkalec rođen je 1. studenog 1992. u Novoj Gradišci. Odrastao je i živio u Novoj Gradišci gdje je završio osnovnu školu „Mato Lovrak“ 2007. godine. U istoj godini upisuje elektrotehničku školu u Novoj Gradišci ali se tijekom prvog razreda prebacuje u opću gimnaziju Nova Gradiška. Srednjoškolsko obrazovanje završava 2011. Iste godine upisuje preddiplomski studij računarstva na Elektrotehničkom fakultetu u Osijeku. Studij završava 2014.godine i postaje inženjer računarstva.

PRILOZI

Prilog 1. CD s radom u digitalnom formatu i kodom aplikacije