Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Boštjan Skok

Spletna aplikacija microCOMB za določanje komponent genske ekspresije

diplomsko delo na
visokošolskem strokovnem študiju

Mentor: doc. dr. Tomaž Curk

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Preučite najnovejše tehnologije za razvoj odzivnih spletnih aplikacij. V izbrani tehnologiji izdelajte spletni vmesnik za program microCOMB. Program omogoča uporabnikom-biologom enostavno primerjavo eksperimentalno izmerjenih genskih ekspresij kvasovke S. cerevisiae z ekspresijskimi podatki kvasovke v obsežni, javno objavljeni zbirki na www.yeastgenome.org. Glavni izhod programa microCOMB je določitev glavnih komponent oz. modulov genske ekspresije, ki so podobni ekspresijam izmerjenim v drugačnih eksperimentalnih pogojih. V okviru diplomskega dela razvite spletno aplikacijo, ki od uporabnika sprejme podatke o eksperimentalno izmerjenih genskih ekspresijah in jih nato primerja z ekspresijskimi podatki v lokalni zbirki, pri čemer naj uporabi program microCOMB. Aplikacija naj hrani zgodovino rezultatov analiz, skrbi za posodabljanje lokalne zbirke javno dostopnih ekspresijskih podatkov ter uporabnikom omogoča določanje pravic za dostop do podatkov v skupni rabi ali skupini uporabnikov.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Boštjan Skok z vpisno številko 63050412 sem avtor diplomskega dela z naslovom:

Spletna aplikacija microCOMB za določanje komponent genske ekspresije (angl. *microCOMB web application for the identification of gene expression components*)

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Tomaža Curka,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 31. avgusta 2016 Podpis avtorja:

Zahvaljujem se družini in vsem svojim bližnjim, ki so me spodbujali in podpirali tekom študija. Zahvalil bi se rad tudi mentorju, doc. dr. Tomažu Curku za vse nasvete in pomoč pri izdelavi diplomskega dela.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
CSS	Cascading Style Sheets	kaskadne slogovne podloge
URL	Uniform Resource Locator	enolični krajevnik vira
JWT	JavaScript Web Token	spletni žeton JavaScript
JSON	JavaScript Object Notation	objektni zapis JavaScript
NPM	Node Package Manage	upravljavec paketov Node
IDE	Integrated development environment	integrirano razvojno okolje
SoC	Separation of Concerns	oblikovno vodilo za razdelitev računalniškega programa v več delov, ki opravljajo vsak svojo nalogo
API	Application Programming Interface	vmesnik za programiranje aplikacij
SPA	Single-Page Application	spletna aplikacija, sestavljena iz ene strani HTML, ki se dinamično spreminja ob interakciji z uporabnikom
GPU	Graphics processing unit	grafični procesor

kratica	angleško	slovensko
ORM	Object-relational mapping	programerska metoda za pretvorbo podatkov med nekompatibilnimi podatkovnimi sistemi v objektno orientiranih programskih jezikih
REST	Representational state transfer	arhitekturni slog za porazdeljene hipermedijske sisteme
HTML	Hyper Text Markup Language	označevalni jezik za nadbesedila
RDBMS	Relational database management system	sistem za upravljanje relacijskih podatkovnih baz

Povzetek

Naslov: Spletna aplikacija microCOMB za določanje komponent genske ekspresije

Cilj diplomskega dela je bil izdelati spletno aplikacijo, ki deluje kot grafični vmesnik za uporabnike microCOMB-a in vzdržuje bazo genskih ekspresij. Glavne funkcije aplikacije so omogočiti uporabnikom posredovanje ekspresijskih podatkov v analizo in prikazati njene rezultate, vodenje zgodovine analiz in skrbeti za ažurnost javne baze ekspresijskih podatkov. V delu so opisane uporabljene tehnologije, arhitektura sistema, razvojni proces ter končna funkcionalnost aplikacije. Ob razvoju smo strmeli k čim bolj modularni arhitekturi ter preprostosti nadaljnjega nadgrajevanja in vzdrževanja. Uporaba zabojnikov Docker nam je omogočila visoko stopnjo modularnosti in neodvisnosti od programske in strojne opreme. Aplikacija je sestavljena iz strežniškega dela in uporabniškega vmesnika. Strežniški del je razvit z uporabo programskega jezika Python z ogrodjema Pyramid in Cornice, uporabniški vmesnik pa z ECMAScript6 in ogrodjema React in Redux. Za hranjenje podatkov na strežniku smo se odločili za kombinacijo relacijske baze PostgreSQL in tekstovnih datotek. Za dodatno stopnjo modularnosti smo na strežniku uporabili SQLAlchemy, kar aplikaciji omogoča, da je agnostična do uporabljene tehnologije RDBMS.

Ključne besede: microCOMB, genska ekspresija, spletna aplikacija, EcmaScript6, Python, Pyramid, React, Redux, Docker.

Abstract

Title: microCOMB web application for the identification of gene expression components

The goal of this thesis is to develop a web application that functions as user interface for microCOMB and manages it's gene expression database. The main functions of the application are to enable the user to upload expression profiles to be analyzed and show it's result, store user history of completed analyses and keep the public database up to date. In the thesis we describe the technologies used, architecture, development process and application functionality. During the development and design process we focused on modularity, maintainability and extendability. Using Docker containers we achieved a high degree of modularity and decoupling from the underlying hardware and software. The application is split into server and client side. The server side is developed using Python and two frameworks Pyramid and Cornice. Client side uses ECMAScript6 as the main language and React in Redux frameworks. To store data on the server side we use a combination of PostgreSQL and text files. To add another degree of modularity we used SQLAlchemy as the ORM on the server side. Using an ORM we made the application RDBMS agnostic.

Keywords: microCOMB, gene expression, web application, EcmaScript6, Python, Pyramid, React, Redux, Docker.

Poglavje 1

Uvod

Analiza profilov ekspresije genov v organizmih je pomembna na raziskovalnih področjih, kot so medicina, biologija in farmacija. Nameni analize profilov se razlikujejo med področji. Na primer, v medicini raziskovalci želijo izvedeti, kako se spremeni genska ekspresija v celicah izpostavljenim določenim patogenom kot so virusi in baterije. V farmaciji pa raziskovalci želijo izkoristiti mikroorganizme za proizvodnjo določenih beljakovin. To dosežejo tako, da z eksperimentiranjem določijo pogoje, ki vplivajo na ekspresijo genov, ki nadzorujejo proizvodnjo tarčnega proteina. Razlogov in načinov uporabe analize profilov genskih ekspresij je veliko, vsi si pa delijo skupno lastnost, da je ekspresija rezultat nekega zunanjega vpliva na organizem ali notranje spremembe v organizmu.

Aplikacija microCOMB [1] je razvita z namenom določitve glavnih komponent genske ekspresije, ki so podobne ekspresijam iz drugih eksperimentov. Tako raziskovalci izvejo za druge eksperimente, kjer se isti geni spremenijo na podoben način, kot so se spremenili geni v danem eksperimentu. S primerjavo pogojev takih eksperimentov, v katerih so se geni podobo odzvali, lažje določijo faktorje, ki uravnavajo ekspresijo skupine genov. Določitev faktorjev in njihovih medsebojnih vplivov jim omogoči izbiro pogojev za naknadne eksperimente, ki bodo z večjo verjetnostjo proizvedli pričakovane rezultate. MicroCOMB je trenutno osredotočen na primerjave genskih eks-

presij kvasovke *Saccharomyces cerevisiae* pridobljenih z uporabo tehnologije mikromreže DNA (DNA microarray).

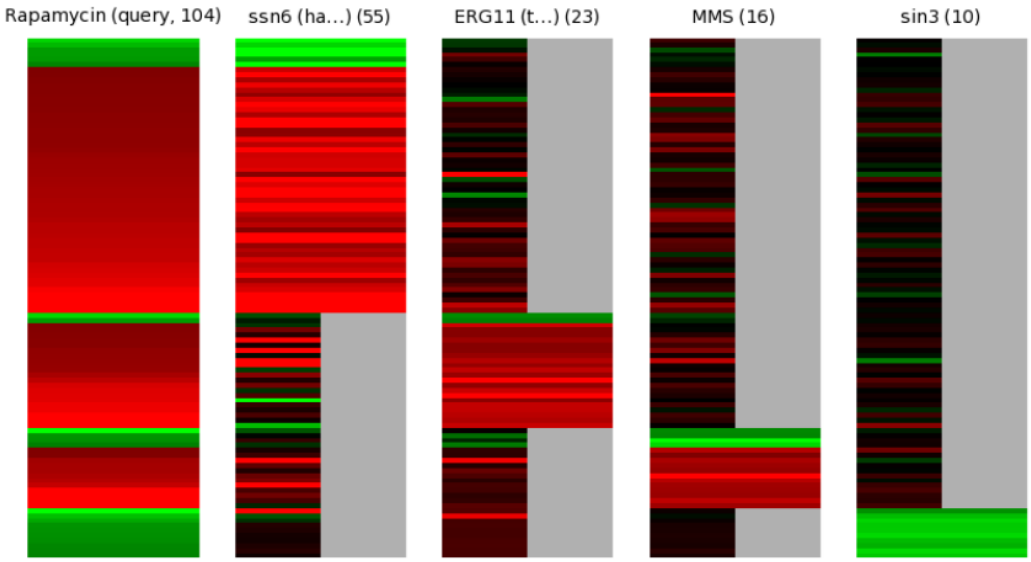
Tehnologija mikromreže DNA se je skozi leta izkazala za zanesljivo metodo hkratnega merjenja nivojev ekspresij velikega števila v naprej določenih genov. Meritve z mikromrežami DNA predstavljajo večino objavljenih profilov genskih ekspresij. V našem primeru so to cDNA mikromreže celotnega genoma kvasovke *S. cerevisiae*. Takšna mikromreža(čip) je trdna podlaga na katero so v mikroskopskih gručah pritrjene v naprej določene niti DNA. Nanjo se v prvi fazi nanese izolirane niti DNA kontrolne skupine, ki so obarvane s fluorescentnim barvilom. V večini primerov je to zeleno rumeno barvilo Cy3. Počaka se, da steče proces hibridizacije, kjer se združijo komplementarne niti DNA. Potem sledi proces izpiranja tako, da na mreži ostanejo samo trdno vezane niti. Postopek se ponovi z nitmi DNA eksperimentalne skupine obarvane z rdečim barvilom Cy5. Sledi ponovno izpiranje. Tako obdelano mikromrežo se optično prebere v namenski napravi, ki glede na odziv obarvanih niti DNA na svetlobo lahko določi na katere skupine se je vezala kontrola in na katere eksperiment, ter do kakšne mere. Pridobljene podatke se glede na znano strukturo mikromreže analizira in pretvori v pare gen in njegova stopnja ekspresije tipično izračunana po formuli:

$$log_2(FC_i) = log_2(\frac{R_i}{G_i}), \tag{1.1}$$

kjer je R_i stopnja ekspresije i-tega gena eksperimenta G_i pa ekspresija kontrole. Rezultat je dvojiški logaritem razmerja ekspresij.

Na sliki 1.1 je primer grafičnega dela rezultata analize microCOMB za najbolj perspektivno kombinacijo treh profilov ekspresij najdenih v bazi. Graf prikazuje ekspresije 104 genov v skupaj štirih eksperimentih. Vrstice obarvane zeleno pomenijo gen, ki se izrazi v kontroli ne pa v eksperimentu. Rdeče vrstice predstavljajo izraz v eksperimentu. Svetlost barve prikazuje stopnjo ekspresije. Črna barva predstavlja gene, ki se sploh niso izrazili. Z zoženjem stolpca najdenih komponent(siva barva), označimo gene v najdenih eksperimentih, ki niso del komponente, ker se ne ujemajo. Ob pogledu na graf

je uporabniku razvidno, da je prva komponenta precej podobna. Če ga ekspresija teh genov zanima lahko sledi povezavi do študije povezane s tem eksperimentom in poišče opis pogojev tega eksperimenta. Razvidno je tudi, da se z zadnjo komponento ujemata samo z ekspresijo kontrole. Ter da se večina ostalih 104 genov te dekompozicije v tem eksperimentu sploh ni izrazila (črna barva). Pogoji takega eksperimenta ga verjetno sploh ne zanimajo.



Slika 1.1: Grafični prikaz najbolj perspektivne kombinacije komponent za poizvedbo Rapamycin

V prikazanem primeru je uporabnik posredoval microCOMB-u profil genetske ekspresije za svoj eksperiment rapamycin. MicroCOMB je kot rezultat vrnil spisek kombinacij treh eksperimentov prisotnih v bazi, razvrščen padajoče po oceni ustreznosti. Način ocenjevanja ustreznosti uporabnik sam določi ob zagonu analize. S prikazano analizo je bila uporabljena formula 1.2, ki je med testiranjem izstopala po kvaliteti rezultatov.

$$\frac{k * p}{r} \tag{1.2}$$

Kjer je k stopnja korelacije ekspresij genov v komponentah, p število genov, ki jih komponente pokrivajo in r povprečna razdalja med njimi.

Aplikacija trenutno sprejema in objavlja rezultate preko preprostih statičnih strani HTML in zahteva ročno posodabljanje baze javnih poizkusov, kar predstavlja težavo za uporabnike. Zato je cilj tega diplomskega dela razvoj uporabniku prijazne spletne aplikacije, ki bo olajšala delo s sistemom microCOMB ter avtomatizirala posodabljanje baze javnih poizkusov. Aplikacija mora uporabniku omogočati zasebno posredovanje ekspresijskih vektorjev, ogledovanje rezultatov analiz, dostop do zgodovine njegovih posredovanih vektorjev ter rezultatov analiz, ter omogočiti dodajanje vektorja v javno dostopno bazo. V začetnem delu diplomskega dela opišemo tehnologije uporabljene pri razvoju aplikacije, nato podrobno opišemo arhitekturo sistema ter njegovo načrtovanje. Sledi opis razvoja in funkcionalnosti strežniškega dela aplikacije ter razvoj uporabniškega vmesnika. V končnem delu predstavimo razvito aplikacijo in opišemo njene funkcije ter v sklepu naštejemo nekaj možnih izboljšav ter potencialnih nadgradenj sistema.

Poglavje 2

Razvojno okolje

Aplikacijo sestavljajo dve ločeni celoti razviti z različnima tehnologijama. Zato potrebujemo dve ločeni razvojni okolji.

Minimalne potrebe za razvoj strežniškega dela so tolmač za Python verizija 3.5 in orodje pip [2] za namestitev paketov. Za lažji in hitrejši razvoj smo uporabili IDE PyCharm [3], katerega prednosti in uporabo opišemo v podpoglavju 2.4.

Osnova razvojnega okolja odjemalnega dela so orodja:

- program Node.js [4] verzija 4 ali novejša omogoča izvajanje ostalih orodij;
- program NPM [5] dela paketa Node.js, ki skrbi za nameščanja knjižnic JavaScript potrebnih za delovanje razvojnega okolja in za izvajanje same aplikacije;
- program WebPack [6], ki iz več med seboj odvisnih modulov JavaScript ustvari statičen paket kode JavaScript, optimizirane za prenos v brskalnik.
- prevajalnik Babel, ki pretvori kodo iz standarda ES6 v EcmaScript 5, katerega podpira večina brskalnikov.

Za razvoj kode smo izbrali IDE WebStorm [7], ki je podrobnejše opisan v podpoglavju 2.5, ker je specifično namenjen izdelavi aplikacij JavaScript. V

nasprotju z drugimi popularnimi splošni IDE-ji, kot sta atom in SublimeText, uporabniku ni potrebno iskati in nameščati vtičnikov, ki jih prilagodijo razvoju aplikacij v JavaScript-u. Za učinkovit razvoj s knjižnicami React in Redux so se izkazale kot nepogrešljiva še dodatka za brskalnik react-devtools [8] in redux-devtools-extension [9].

Dodatek React Developer Tool omogoča ogled hierarhije komponent React na trenutni strani in vpogled ter spreminjanje njihovih lastnosti in stanja. Ker React vhodne podatke spremeni v statičen HTML, je razhroščevanje tega postopka brez dodatka zelo oteženo. Saj je potrebno slediti kodi v knjižnici React, kar pa je zamudno in razen v redkih primerih nepotrebno.

Dodatek Redux DevTools pa omogoča ogled trenutnega stanja Redux shrambe in beleži zgodovino sprememb. Potrebujemo ga, ker je vpogled v shrambo preko samega razhroščevalnika otežen zaradi nivoja gnezdenja objektov v spominu.

2.1 Uporabljene tehnologije

Za razvoj aplikacije smo želeli izbrati moderne tehnologije, ki sledijo predvidenim smernicam razvoja spletnih aplikacij. Na področju, kot so spletne tehnologije, ki se stalno spreminja, je to precej oteženo. Ob času izdelave te naloge sta po našem mnenju izstopala Facebook-ov React.js ter Google-ov Angular2. Angular2 je naslednja iteracija popularnega ogrodja Angular za razvoj spletnih aplikacij. Angular2 je celovito ogrodje za razvoj aplikacij, ki striktno definira strukturo aplikacije v stilu MVC. Glavna slabost Angular2 je, da je še v beta fazi razvoja. V ostalih faktorjih, kot sta podpora in perspektivnost, sta si precej podobna. Velika prednost React-a pa je hitrost s katero posodablja uporabniški vmesnik, ker se izogne ponovnemu izdelovanju DOM-a.

ECMAScript 2015 je specifikacija JavaScript programskega jezika, ki ga standardizira neprofitna organizacija Ecma International [10].V juniju mesecu, 2015 leta so izdali standard, znan tudi pod kratico ES6, ki je vzbudil

veliko zanimanja med razvijalci spletih aplikacij. Prinesel je veliko potrebnih izboljšav in predvsem možnost definiranja razredov ter modulov, kar je omogočilo razvoj kompleksnih aplikacij. Podpora novih standardov s strani brskalnikov je ponavadi dolgotrajni proces. Tako, da po enem letu od standardizacije ES6 je ta dobro podprt samo v brskalnikih Firefox, Edge in Chrome. Razvijalcem se je v čakanju na podporo brskalnikov uspelo izogniti z razvojem prevajalnikov. Ti kodo ES6 prevedejo v starejše verzije JavaScript-a, ki so bolje podprte.

Prevajalnik Babel [11] je trenutno najbolj popularen prevajalnik iz ECMAScript 6 v ECMAScript 5. Zaradi rasti v kompleksnosti kode JavaScript v spletnih aplikacijah je uporaba standarda ES6 postala v praksi skoraj obvezna. K popularnosti tudi prispeva njegova podpora drugih različic JavaScript-a, kot sta TypeScript in React-ov JSX, bogat ekosistem vtičnikov.

Docker [12] je odprtokodni program, ki avtomatizira postavitev Linux aplikacij znotraj zabojnikov. Zabojniki Docker ovijejo aplikacijo v virtualni datotečni sistem, ki vsebuje vse kar aplikacija potrebuje za delovanje. Na ta način zabojniki zagotovijo, da se bo aplikacija izvajala enako ne glede na okolje v katerem se nahaja. Docker za delovanje izkorišča funkciji Linux jedra, ki omogočata izolacijo virov sistema. Tako se zabojniki Docker izvajajo v sistemu kot procesi in ne kot ločeni virtualni stroji. To mu omogoča manjšo porabo virov sistema. Zabojniki so predvsem hitri ob zagonu, saj ne postavljajo operacijskega sistema ampak samo aplikacijo. Za vodenje polj kompleksnih kombinacij zabojnikov, ki so za delovanje odvisni med seboj, so pri Docker-ju razvili aplikacijo docker-compose [13]. Ta na podlagi konfiguracijske datoteke ob zagonu koordinira vzpostavitev zabojnikov in komunikacij med njimi. To omogoča ponovljive postavitve kombinacij aplikacij.

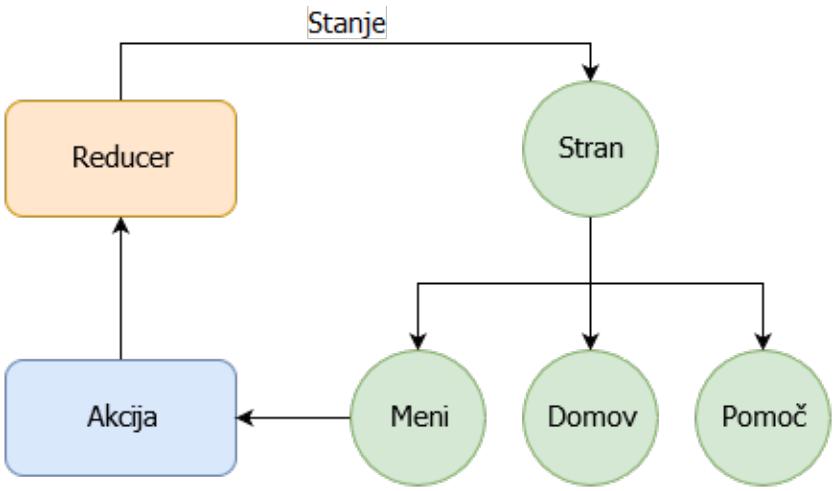
React.Js [14] je odprtokodna knjižnica razvita v Facebook-u in objavljena kot prost odprtokodni projekt marca meseca leta 2015. Od ostalih okolij za razvoj SPA aplikacij se razlikuje po tem, da ne zahteva obnove DOM-a s strani brskalnika, ampak poišče razlike med svojim internim DOM-om in DOM-om brskalnika in ga spremeni, kjer je to potrebno. Tak način

delovanja se izkaže za zelo učinkovitega in omogoča React-u hitro osveževanje prikaza podatkov v primerjavi s konkurenco. Spreminjanje DOM-a je tako učinkovito, da ga je večina konkurenčnih knjižnic, implementirala direktno ali preko vtičnikov.

Glavne prednosti knjižnice React:

- **Enosmerni podatkovni tok**, podatki so posredovani komponentam kot lastnosti značke HTML in so nespremenljivi. Prikazane podatke se lahko spremeni samo s povratnim klicem metode JavaScript.
- **Virtualni DOM** je interna kopija DOM-a brskalnika, ki jo React hrani v spominu. Koda JavaScript razvijalca spreminja le virtualni DOM in ne pravega. Ko React zazna spremembo izvede primerjavo virtualnega in realnega DOM-a strani. Najdene razlike prenese v DOM brskalnika. Na ta način brskalnik ustvari celoten DOM samo enkrat, kar omogoča React-u učinkovito osveževanje uporabniškega vmesnika
- **JSX** je posebna različica JavaScript jezika, ki razvijalcu omogoča učinkovito mešanje elementov HTML in kode JavaScript. Ker se JSX prevede v JavaScript, je mogoče zaznati napake v kodi že ob prevodu in ne šele ob izvajanju. Ker se koda JSX prevaja, ima tudi to prednost, da v primerjavi z drugimi knjižnicami, lahko razvijalcu sporoči vrstico v kodi, kje se nahaja napaka.

Redux [15] je knjižnica, katere cilj je čim bolj preprosto upravljanje s stanjem aplikacije JavaScript. Razvit je bil kot alternativa knjižnici Flux, katera je bila razvita na Facebooku s poudarkom na preprostosti. Redux dovoli definirati eno drevo stanja za celotno aplikacijo. Njegovo stanje lahko spremenimo samo z oddajo akcije, ki je objekt z opisom spremembe stanja. Na sliki 2.1 je prikazan postopek spreminjanja stanja. V tem primeru komponenta Meni sproži povratno funkcijo, ta pridobi ali spremeni določene podatke in jih v obliki objekta akcija odda naprej. Akcijo obravnava reduktor (ang. Reducer), ki prepozna tip akcije. Reduktor spremeni stanje aplikacije in Redux poskrbi, da se vse komponente odvisne od teh podatkov osvežijo.



Slika 2.1: Diagram spremembe stanja v Redux

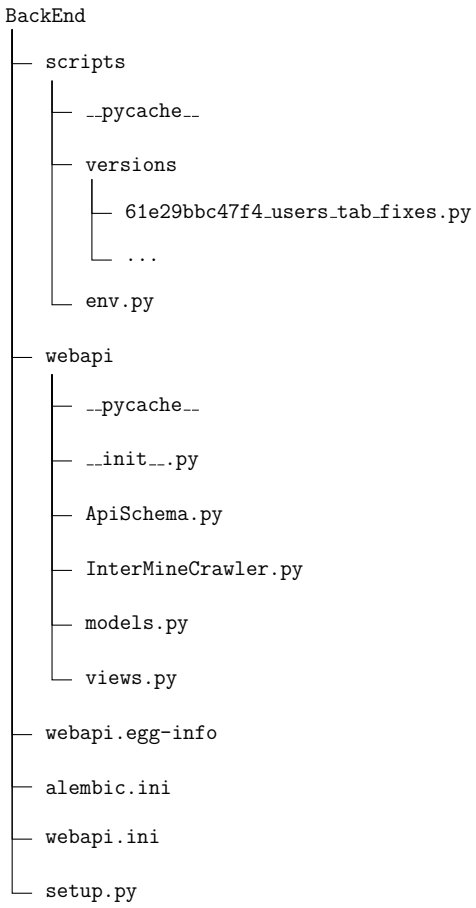
Tako Redux doseže enosmerni tok podatkov enako kot React in lastnost, da se stanje spremeni samo znotraj reduktorja in omogoča lažje razhroščevanje. Kombinacija Redux-a in React-a je zaradi preprostosti uporabe postala zelo popularna.

2.2 Zgradba projekta

Projekt je razdeljen na dva dela, kar je razvidno tudi iz zgradbe korenske mape. V njej se nahajajo mapa FrontEnd, v kateri je koda za React Redux SPA aplikacijo, ki teče v brskalniku. V mapi BackEnd hranimo kodo REST strežnika.

Opis zgradbe strežniškega dela prikazanega na sliki 2.2:

- Naloga **__init__.py** je vzpostavitev aplikacije. Izvajanje se začne v metodi main.
- **webapi.ini**, konfiguracijska datoteka strežnika definira predvsem ključ za šifriranje žetonov JWT ter podatke za dostop do baze podatkov.
- **alembic.ini**, nastavitve za modul alembic, predvsem nastavitve do-



Slika 2.2: Zgradba mape BackEnd

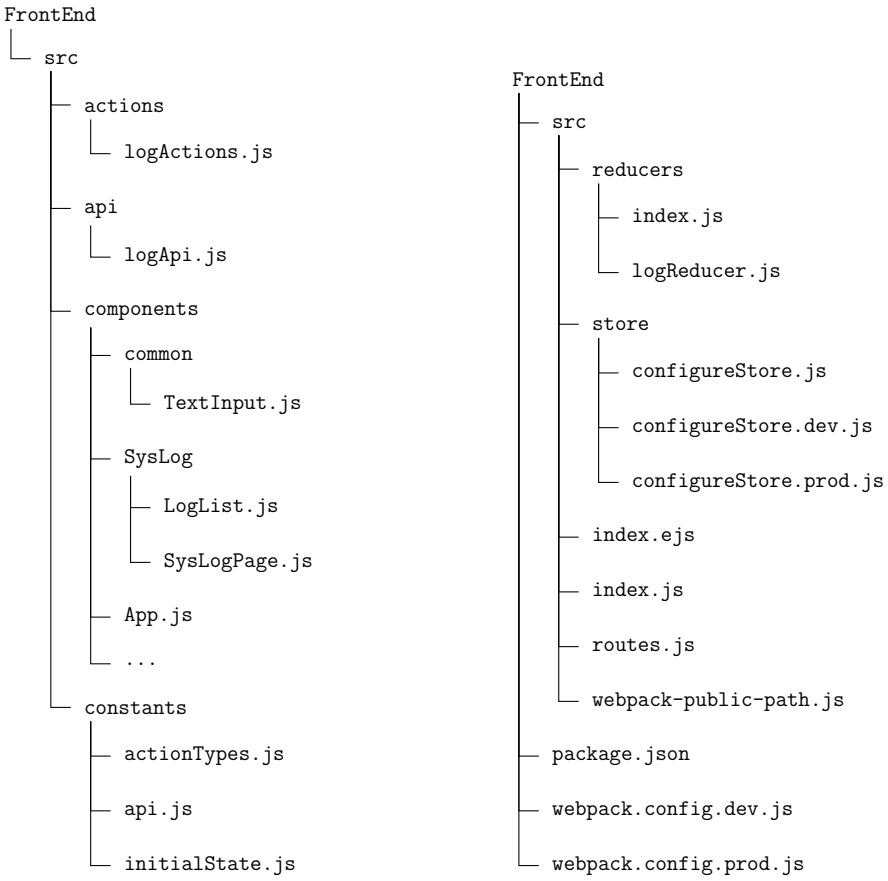
stopa do baze podatkov.

- **setup.py**, standardna Python skripta, ki namesti aplikacijo in pakete od katerih je odvisna.
- **ApiSchema.py**, v nje definiramo obliko veljavnih podatkov za naše spletne storitve.
- **InterMineCrawler.py**, skripta, ki skrbi za posodobitev baze microCOMB.
- **models.py**, vsebuje definicije objektov SQLAlchemy, ki uporabljajo pri tvorbi sheme baze podatkov.
- **views.py**, vsebuje definicije spletnih storitev ogrodja Cornice.
- Mapo **scripts** uporablja modul alembic za svoje skripte.
- **env.py**, v skripti alembic modulu definiramo, kako dostopa do meta podatkov našega modela SQLAlchemy.
- Datoteke v mapi version kot na primer **61e29bbc47f4_users_tab_fixes.py** vsebujejo migracijske skripte modula alembic. Format imena je unikatna šifra, kateri sledi ime migracije.
- **__pycache__**, so mape, kjer Python prevajalnik hrani zložno kodo.
- **webapi.egg-info**, mapo uporablja setuptools orodje za nameščanje Python modulov.

Odjemalni del aplikacije temelji na odprtokodnem projektu react-slingshot [16]. Ta je popularna osnova za razvoj React Redux aplikacij in definira osnovno zgradbo projekta ter razvojno okolje.

Opis zgradbe odjemalnega dela prikazanega na sliki 2.3:

- **package.json**, glavna konfiguracijska datoteka razvojnega okolja. Definira potrebne knjižnice node.js. Vsebuje skripte za zagon testnega strežnika, zagon testov ter ustvarjanje produkcijske verzije aplikacije.



Slika 2.3: Zgradba mape FrontEnd

- **webpack.config.*.js**, konfiguracija webpack aplikacije za testno in produkcijsko okolje.
- Mapa **src** hrani izvorno kodo aplikacije. Ostale mape FrontEnd-a, ki niso prikazane na sliki 2.3, pripadajo programu npm ali webpack-u.
- V mapi **actions** so definirane vse Redux akcije v programu ločene v module glede na entiteto.
- **api** mapa vsebuje kodo, sestavi in izvede poizvedbe HTTP.
- **components** hrani definicije komponent React ločene v pod mape glede na njihovo funkcijo.
- **common** mapa vsebuje definicije osnovni komponent React. Na primer, **TextInput.js** ovije input element HTML-ja in mu doda lastnosti potrebne za delo s poljem v sistemu React.
- **SysLog** mapa vsebuje vse komponente React za prikaz strani Log.
- **App.js** je glavna komponenta React. Definira osnovno zgradbo aplikacije. Vse ostale komponente se prikazujejo znotraj nje.
- **constants** definira vrednosti stalnih spremenljivk v aplikaciji.
- **initialState.js** vsebuje začetno stanje shrambe Redux. Datoteka **api.js** hrani URL naslove potrebne za komunikacijo s spletnimi storitvami. Datoteka **actionTypes.js** vsebuje definicije tipov akcij Redux.
- V mapi **reducer** definiramo reduktorje. Datoteka **index.js** vsebuje definicijo korenskega reduktorja. Ostale datoteke vsebujejo reduktorje za posamične entitete. Reduktor poimenujemo tako, da nazivu entitete dodamo besedo Reducer.
- **store** mapa vsebuje module, ki skrbijo za inicializacijo shrambe Redux. Ločeni so glede na konfiguracijo okolja **configureStore.dev.js** za razvojno okolje **configureStore.prod.js** pa za produkcijsko.

- **index.ejs** definira osnovno index.html stran aplikacije.
- **index.js** zažene React in Redux.
- **routs.js** definira navigacijske poti usmerjevalnika React.

2.3 Vagrant

Aplikacija Vagrant služi postavljanju razvojnih okolji, na preprost in ponovljiv način. Služi kot visoko nivojski ovoj virtualizacijskih tehnologi kot so VirtualBox, VMware ali KVM ter orodij za upravljanje konfiguracij, kot so Chef, Puppet in Salt. Uporabili smo ga za postavitev testnega strežnika na način, ki je ponovljiv vsem razvijalcem. Za postavitev enakega testnega strežnika, kot ga imajo ostali razvijalci, mora razvijalec namestiti VirtualBox in Vagrant in v mapi projekta zagnati ukaz *Vagrant up*. Vagrant bo uporabil konfiguracijsko datoteko Vagrantfile in avtomatsko prenesel potrebne slike virtualnih strojev, jih zagnal ter vzpostavil komunikacijo za aplikacijo v virtualnem okolju preko preslikav vrat strežnika. Vagrant tudi deli vsebino mape projekta z virtualnim okoljem.

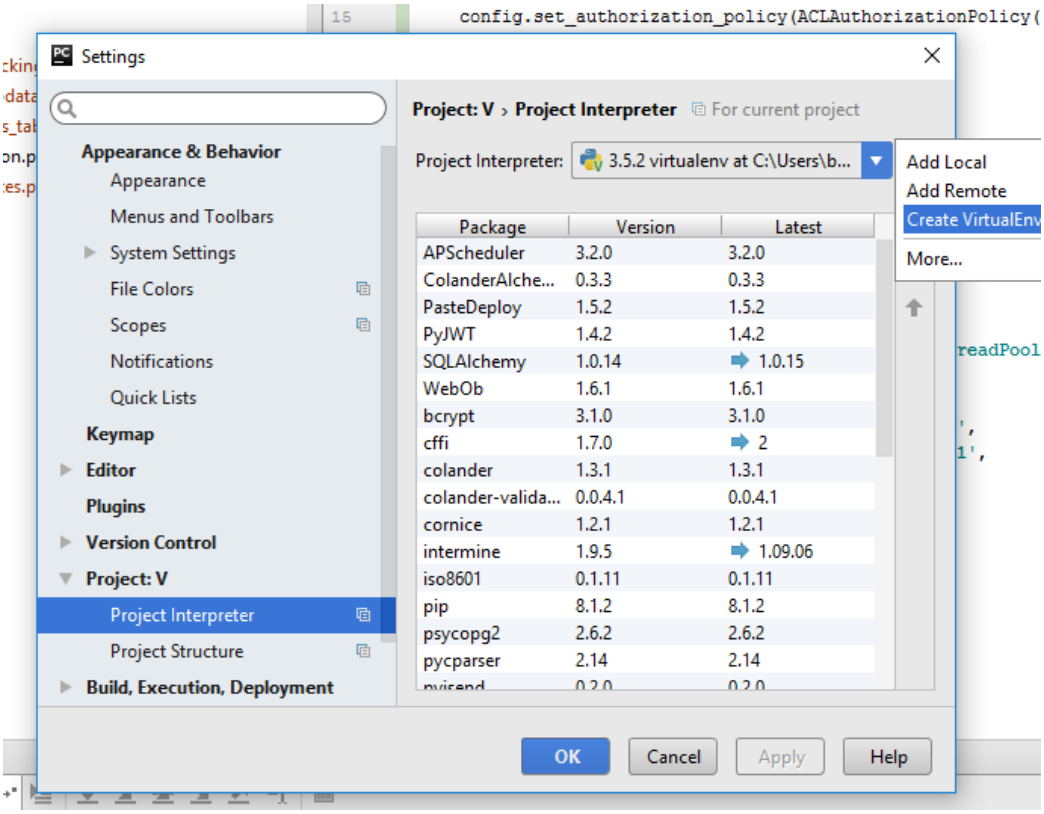
2.4 Uporaba IDE-ja PyCharm

IDE PyCharm smo izbrali, ker je preprost za uporabo in namestitev ter zastonj za uporabo na odprtokodnih aplikacijah. Postavitev razvojnega okolja je preprosta in dokumentirana v dokumentacij [34] Pyramid projekta. Potrebno je nastavit virtualno okolje Python-a ter ga uporabiti pri nastavitvi opcije Run\Debug.

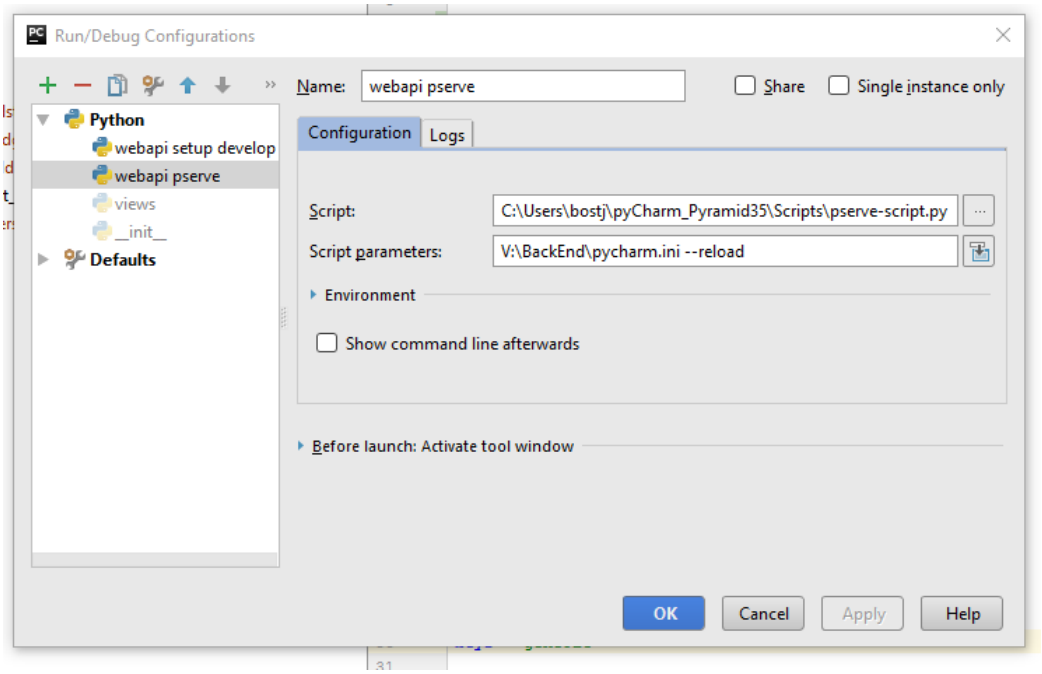
Python virtualno okolje v PyCharm-u nastavimo v meniju File→ Settings → Project:webapi→ Project Interpreter s pritiskom na gumb Create VirtualEnv kot je razvidno na sliki 2.5. Vpišemo ime novega okolja in potem novemu okolju dodamo potrebne Python module naštete v datoteki setup.py v install_requires.

```
ENV['VAGRANT_DEFAULT_PROVIDER'] = 'virtualbox'
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/trusty64"

  config.vm.provider "virtualbox" do |v|
      v.customize ["setextradata", :id, "VBoxInternal2/
    SharedFoldersEnableSymlinksCreate/v-root", "1"]
  end
  #DEBUG postgres direct connection
  config.vm.network "forwarded_port", guest: 5432, host: 5432
  #webapi
  config.vm.network "forwarded_port", guest: 8000, host: 8000
  #web
  config.vm.network "forwarded_port", guest: 80, host: 80

  config.vm.provision :docker
  config.vm.provision "shell", inline: <<-EOC
    sudo apt-get -y install python-pip; \\
    sudo pip install docker-compose
  EOC
  #config.ssh.pty = true
#    screen -L -S VagrantCmd;\\

  config.vm.provision "shell",run: "always", inline:<<-EOC
    sudo docker stop $(docker ps -a -q);\\
    cd /vagrant; \\
    docker-compose pull;docker-compose up
  EOC
```

Koda 2.1: Konfiguracijska datoteka Vagrantfile



Slika 2.4: Nastavitve PyCharm

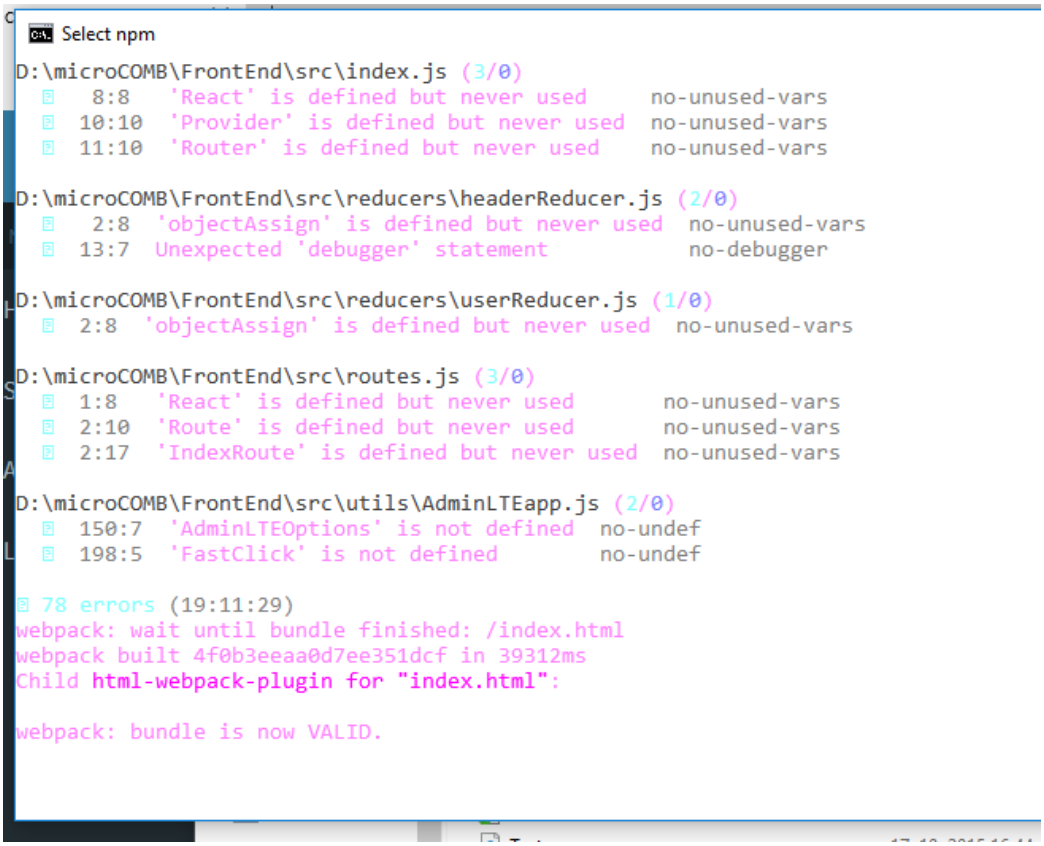


Slika 2.5: Nastavitve PyCharm Run\Debug

Nastavitve Run→ Debug so dosegljive prek menija Run→ Edit Configurations. Dodamo dve novi konfiguraciji Python. Prva konfiguracija namesti aplikacijo s klicem setup.py z argumentom develop. Druga konfiguracija požene skripto pserve-script.py z argumentom *webapi.ini –reload* v okolju, ki smo ga prej definirali. Uporabnik mora zamenjati še sqlalchemy.url parameter v webapi.ini, ki aplikaciji pove kako se povezati na bazo podatkov.

2.5 Razvoj odjemalnega dela z IDE-jem WebStorm

IDE WebStorm smo izbrali, ker je specifično namenjen razvoju JavaScript aplikacij in prinaša precej olajšav razvijalcem, ravno tako kot PyCharm je zastonj za razvoj odprtokodnih aplikacij. Čeprav lahko uporabimo interni razvojni strežnik WebStorm-a za lažje delo, smo se odločili zato, da bi ostali

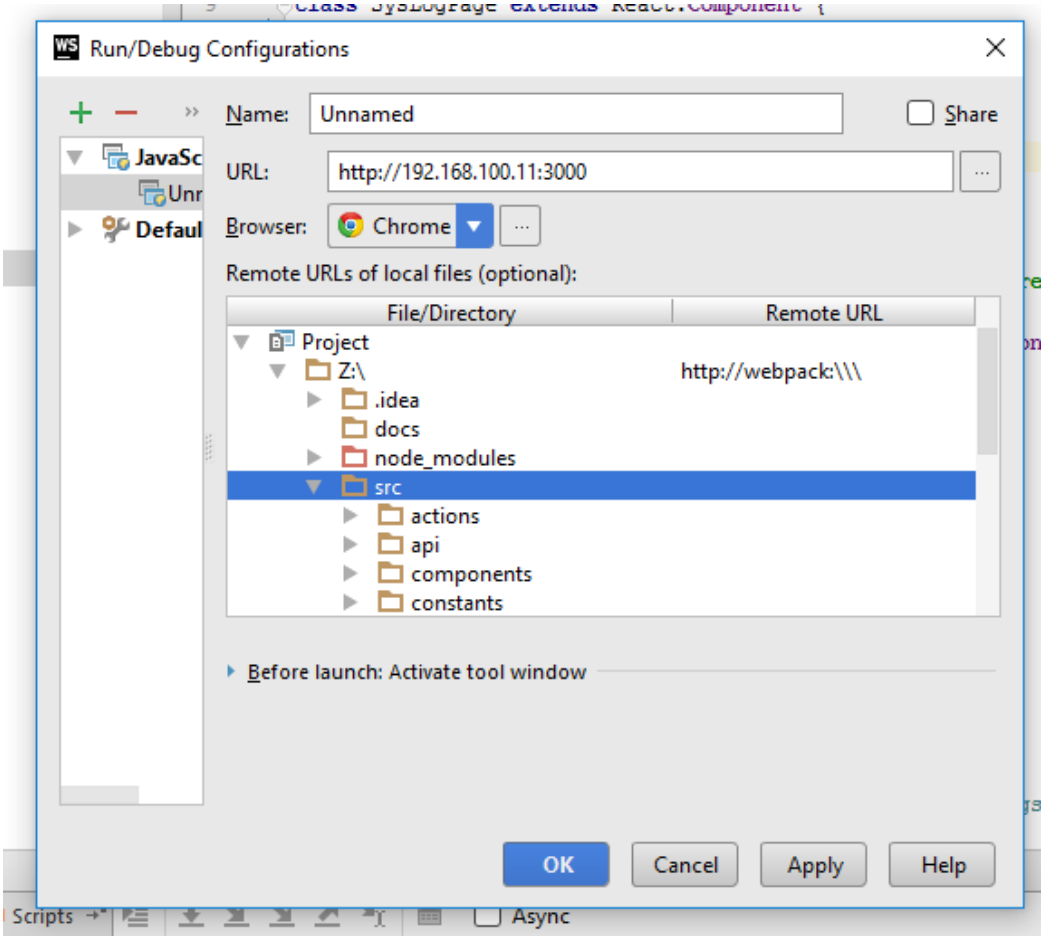


Slika 2.6: Konzola WebPack dev server

bližje okolju React-SlingShot. Zato za postavitev strežnika v ukazni vrstici IDE-ja zaženemo ukaz *npm install*, ki bo namestil vse manjkajoče knjižnice v datoteki package.json. Ko se postopek zaključi, namestimo še razvojni strežnik webpack z ukazom *npm install webpack-dev-server*. Zdaj lahko z *npm start -s* zaženemo start skripto iz datoteke package.json in postavi se naš lokalni strežnik. Med delovanjem se bodo v oknu ukazne vrstice izpisovale napake v kodi in status osveževanja strani, ko spremenimo kodo.

Razvojni strežnik opazuje stanje kode programa in ko se ta spremeni, sproži osvežitev strani. Da bo aplikaciji omogočen dostop do podatkov na strežniku, mora uporabnik spremenit vrednost BASE_URL spremenljivke v \src\constants\api.js datoteki na instanco delujočega strežnika za spletne storitve.

Za razhroščevanje aplikacije se uporablja source-map funkcija aplikacije webpack. Webpack ustvari source-map datoteko, ki jo razhroščevalniki uporabijo za preslikavo kode, ki se izvaja v brskalniku v izvorno kodo v IDE-ju. V našem primeru je potrebno preveriti, ali je v datoteki webpack.config.dev.js atribut devtool nastavljen na vrednost *eval-source-map*. Za prikaz strani uporabimo brskalnik Google Chrome, ker zanj obstaja vtičnik JetBrains IDE Support [35], ki ga poveže z IDE-jem. Potrebno je še povedati IDE-ju, kako naj uporabi source-map. Odpremo meni Run→ Edit Configurations→ dodamo novo JavaScript Debug konfiguracijo kot na sliki 2.7. Vpišemo URL naslov strani na našemu razvojnemu strežniku in v polju browser izberemo brskalnik Chrome. Pomembno je še, da v oknu Remote URL of local files izberemo korensko mapo odjemalnega dela ter v njeno polje Remote Url vpišemo *http://webpack:*\\\.



Slika 2.7: Nastavitve WebStorm JavaScript

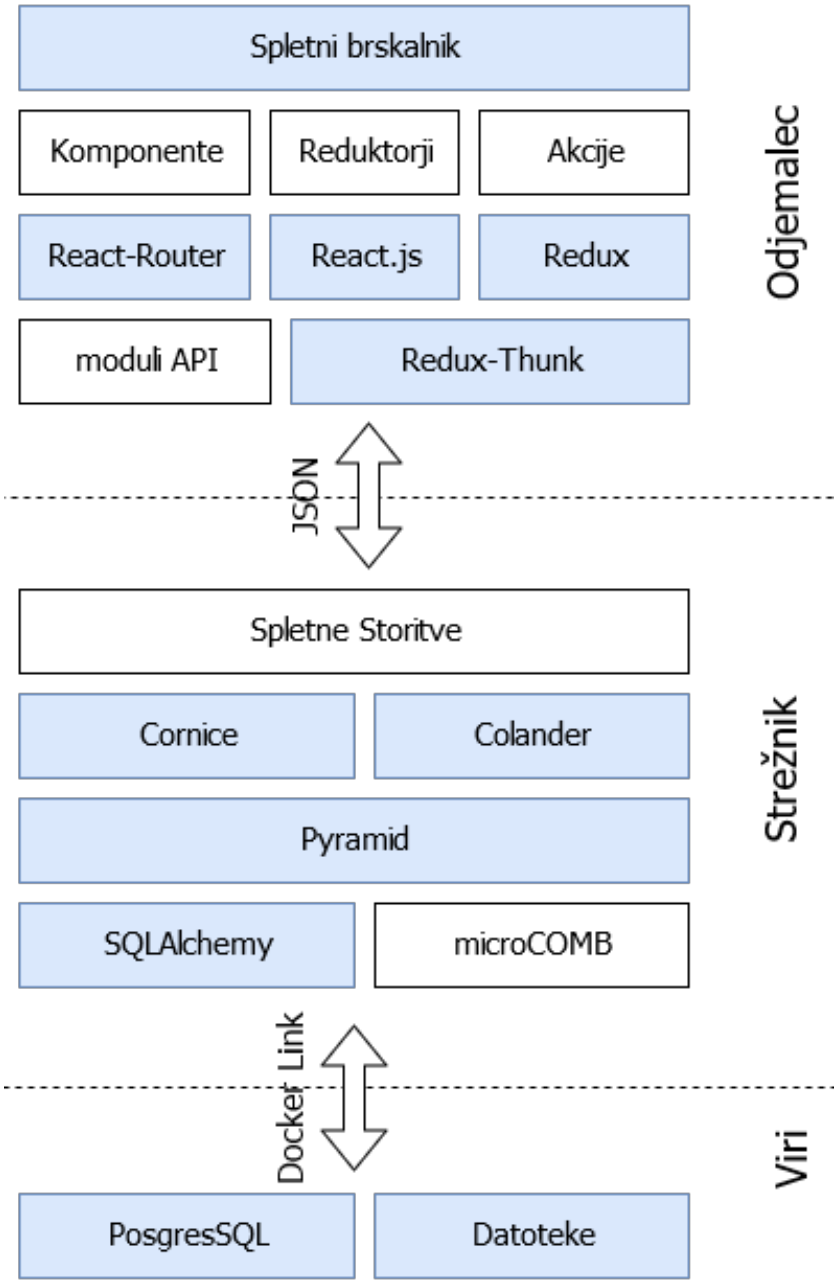
Poglavje 3

Arhitektura sistema

Aplikacija je sestavljena iz dveh ločenih celot: aplikacijski (back-end) in uporabniški vmesnik (front-end). Taka ločitev izhaja iz oblikovnega vodila SoC in prinaša določene prednosti – neodvisnost delov aplikacije omogoča lažje testiranje ter neodvisen razvoj in nadgradnje. Ker je strežnik neodvisen, je za uporabniški vmesnik, ki teče na drugi platformi, potrebna samo podpora protokola HTTP.

Strežnik izvaja spletno storitev brez pomnjenja stanja, kar pomeni, da je vsaka poizvedba zaključena celota in neodvisna od trenutnega stanja sistema. Medsebojna neodvisnost poizvedb omogoča preprosto testiranje in razhroščevanje kode. Za razvoj strežnika smo, zaradi njegove vsestranskosti in bogatega ekosistema programskih knjižnic, uporabili programski jezik Python [33]. Ker sam Python ne podpira streženja po protokolu HTTP in je razvoj takega ogrodja zamuden, smo uporabili Waitress aplikacijo kot vmesnik med protokolom HTTP in Python-om. Za definicijo spletnih storitev smo uporabili odprtokodno ogrodje Cornice [21].

Uporabniški vmesnik je zgrajen kot enostranska aplikacija (SPA), ki se izvaja v uporabnikovem spletnem brskalniku. Vmesnik je razvit v ECMAScript-u 2015. Ker le-tega starejši brskalniki ne podpirajo, se ga pred objavo prevede s prevajalnikom Babel. Za definiranje grafične podobe vmesnika smo uporabili ogrodja Bootstrap [22] in React, za obdelavo in hrambo podatkov v



Slika 3.1: Logična arhitektura aplikacije

vmesniku pa skrbi knjižnica Redux.

3.1 Implementacija z zabojniki Docker

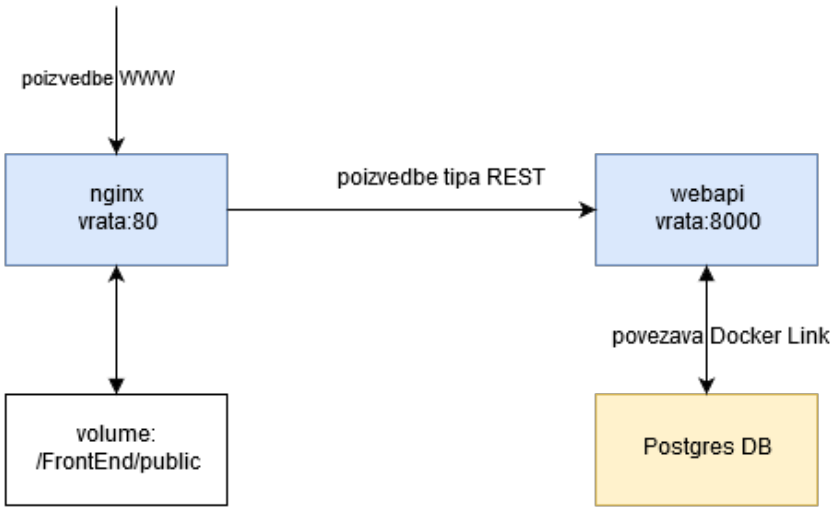
Strežniški del aplikacije smo virtualizirali z uporabo aplikacije Docker. Z virtualizacijo strežnika se izognemo problemom, ki izhajajo iz razlik programske in strojne opreme na fizičnih strežnikih. Strežnik je sestavljen iz več zabojnikov Docker, katerih konfiguracija je definirana v tekstovnih datotekah in njihove spremembe so izsledljive ter vodene kot različice v sistemu git [32]. To omogoča učinkovito izsledljivost hroščev, ki se pojavijo zaradi sprememb v programski opremi. Zato, ker se aplikacija med razvojem izvaja in testira z zabojniki, ki so enaki tistim na produkcijskemu strežniku, je verjetnost napak zmanjšana. Organiziranje in vzpostavljanje zabojnikov ter komunikacije med njimi je zapleten proces, ki je podvržen napakam. V izogib zapletom smo uporabili orodje Docker-compose, ki definira arhitekturo sistema s konfiguracijsko datoteko in preprosto vzpostavi komunikacijo med zabojniki.

Strežnik sestavljajo trije zabojniki:

- PostgreSQL [23] strežnik baze podatkov;
- Cornice strežnik izvaja Spletne storitve;
- Nginx [24] strežnik streže uporabniški vmesnik;

```yaml
version: '2'
services:
  webapi:
    image: bostjanskok/corniceserver
    command: /bin/ash -c "cd /BackEnd;python setup.py;
                          pserve webapi.ini --reload"
    ports:
     - "8000:8000"
    volumes:
     - ./BackEnd:/BackEnd/
    depends_on:
     - db

  db:
    image: postgres
    ports:
    - "5432:5432"
    environment:
                  PGUSER: postgres
                  POSTGRES_PASSWORD: 123456
                  POSTGRES_DB: microCombDb
  website:
    image: nginx
    volumes:
     - ./FrontEnd/public:/FrontEnd/
    ports:
    - "80:80"
     environment:
       - NGINX_PORT=80
```

Slika 3.2: Datoteka docker-compose.yml za aplikacijo



Slika 3.3: Diagram arhitekture strežnika

Poglavje 4

Strežniški del

4.1 Avtentikacija uporabnikov

Dostop do zasebnih podatkov uporabnikov omejujemo s preverjanjem žetona tipa JWT, ki je dodeljen uporabniku ob uspešni avtentikaciji. Json spletni žeton (JWT) je odprti standard RFC 7519 [40] za ustvarjanje žetonov, ki definira obliko objekta in podprte načine šifriranja. Strežnik ob prijavi v žeton zapiše poljubne izjave, v našem primeru unikatno številko uporabnika in čas veljavnosti, in celoto digitalno podpiše.

Prijava uporabnika poteka tako, da uporabniški vmesnik preko metode POST posreduje s strani uporabnika vnešen e-poštni naslov in geslo. Prenos podatkov poteka v golem besedilu, zato je potrebno na strežniku urediti TLS šifriranje povezave. V primeru, da povezava ni šifrirana, obstaja možnost napada človeka v sredini. Ko strežnik prejme zahtevek za prijavo, posredovano geslo povzame z uporabo zgoščevalne funkcije bcrypt. Rezultat primerja z zapisom v podatkovni bazi, in če se ujemata, se uporabniku dodeli JWT. Z uporabo zgoščevalne funkcije bcrypt se zaščiti uporabniška gesla v primeru kraje baze podatkov. Za to funkcijo smo se odločili, ker je dobro podprta in je zaradi pogostega dostopa do spomina ni mogoče učinkovito pohitriti z uporabo GPU-jev. Pri iskanju zgoščevalnega algoritma smo premišljevali o uporabi Argon2 zmagovalcu PHC [17], a se zanj nismo odločili, saj trenutno

```
@login.post(schema=ApiSchema.LoginSchema)
def login_post(request):
    email = request.json_body.get('email').strip().lower()

    user = DBSession.query(User).filter(User.email == email).first()
    if user is None:
        request.errors.add(name='Login', description='user with email:' +
    email + ' not found', location='/user')
        return

    # Calculating a hash
    password = request.json_body.get('password').encode('utf-8')
    # hashed = bcrypt.hashpw(password, bcrypt.gensalt())
    # Validating a hash (don't use ==)
    if bcrypt.checkpw(password, user.hash.encode('utf-8')):
        token = request.create_jwt_token(user.user_id)
        return jsend.success({'JWT': token})
    request.errors.add(name='Login', description='login failed', location='/
    user')
    return
```

Koda 4.1: Metoda za prijavo

ne obstaja stabilni modul za Python. Natančneje, strežnik uporablja Python modul bcrypt [20] s privzetimi parametri.

Za ustvarjenje JWT smo uporabili knjižnico PyJWT [19], ki uporablja metodo HMAC RFC 2104 [36] z zgoščevalno funkcijo SHA256 [37] za podpis žetona. Zasebni ključ, uporabljen pri šifriranju, je definiran s poljem *private_key* v konfiguracijski datoteki.

Veljavnost žetona je prav tako nastavljiva z definicijo polja "expiration". Vsebina žetona je čas veljavnosti in interni identifikator uporabnika, kateremu le-ta pripada. Ogrodje Pyramid [25] avtomatično prebere žeton iz glave poizvedbe in nastavi polje *authenticated_userid* v objektu *request*. Zahteve po zasebnih podatkih, ki nimajo veljavnega žetona, so zavrnjene.

```
1 user = DBSession.query(User).filter(User.user_id == request.
     authenticated_userid).first()
2     if user is None:
3         request.errors.add(name='GetUser',
4          description='user with id not found',location='/user')
5         return
```

Koda 4.2: Primer iskanja uporabnika s SQLAlchemy

4.2 Upravljanje z bazo podatkov

Aplikacija ne dostopa neposredno do baze podatkov, ampak uporabi ORM SQLAlchemy [26] kot vmesnik. Ker je koda aplikacije vezana samo na SQLAlchemy, lahko na preprost način zamenjamo RDBMS z zamenjavo gonilnika DBAPI. V našem primeru smo se odločili za bazo PostgreSQL in uporabili DBAPI gonilnik psycopg2 [27]. Prednost uporabe ORM-ja je avtomatska preslikava podatkov v bazi v objekte Python, s katerimi lažje delamo, in zaščita pred vrivanjem SQL-a. ORM skrije podrobnosti pisanja in izvajanja poizvedb SQL tako, da dosežemo enako funkcionalnost kot pri uporabi v naprej definiranih metod. Spreminjanje podatkov je preprosto, saj se spremembe atributov objekta prebranega iz baze avtomatično shranijo v bazo ob zaključku transakcije.

Zato, da lahko deluje SQLAlchemy, moramo definirati objekte, ki se hranijo v bazi. To lahko naredimo na dva načina: preslikamo obstoječo bazo v definicije objektov SQLAlchemy s pomočjo *automap* metode, ali definiramo objekte SQLAlchemy in nato uporabimo metodo *MetaData.create_all()*, ki ustvari potrebne tabele.

Ker se med razvojem aplikacij definicije objektov pogosto spreminjajo, smo si delo olajšali z uporabo knjižnice alembic [18]. Knjižnica izvede primerjavo novega modela v Python-u in trenutne sheme podatkovne baze. Na podlagi primerjave izdela migracijsko skripto SQL, ki uskladi shemo baze z modelom. Takšno postopno usklajevanje je še posebej pomembno pri nadgrajevanju baz, ki že vsebujejo podatke, saj v večini primerov ni potrebno

```
class Publication(Base):
    __tablename__ = 'Publication'
    id = Column(Integer, primary_key=True)
    pubMedId = Column(Integer, nullable=False, unique=True)
    title = Column(Text)
    summary = Column(Text)
    organismSN = Column(String, nullable=False)
    addedToMatrix = Column(Boolean)
    dataSetsDownloaded = Column(Boolean)
    added = Column(DateTime)
    dataSets = relationship("DataSet")
    status = Column(String)
    statusLog = Column(Text)

    @classmethod
    def from_json(cls, data):
        return cls(**data)

    def to_json(self):
        to_serialize = ['id', 'pubMedId',
                'Title','Summary', 'OrganismSN',
                'AddedToMatrix', 'DataSetsDownloaded','Added']
        d = {}
        for attr_name in to_serialize:
            d[attr_name] = getattr(self, attr_name)
        return d
```

Koda 4.3: Primer definicije objekta SQLAlchemy v models.py

```
1
2 $ alembic revision --autogenerate -m "Added account table"
3 INFO [alembic.context] Detected added table 'account'
4 Generating /path/to/foo/alembic/versions/27c6a30d7c24.py...done
5
6 $ alembic upgrade head
7 INFO  [alembic.context] Context class PostgresqlContext.
8 INFO  [alembic.context] Will assume transactional DDL.
9 INFO  [alembic.context] Running upgrade 1975ea83b712 -> ae1027a6acf
```

Koda 4.4: Alembic primer kreiranja nove migracije in nadgradnja baze

njihovo obnavljanje. Ustvarjanje migracijskih skript ima tudi to prednost, da se spremembe sheme baze nadzirajo kot ostala izvorna koda. Tako pridobimo prednosti, kot je sledljivost hroščev, možnost vrnitve na prejšnje verzije in preprostejše sodelovanje med več razvijalci.

4.3 Podatkovna baza

Za podatkovno bazo PostgreSQL smo se odločili iz več razlogov. Ker je odprtokodna in usklajena s standardom ANSI-SQL, je podprta v večini razvojnih okoljih. Podpira večino operacijskih sistemov in velik nabor podatkovnih tipov. Z več kot petnajst let razvoja za seboj, PostgreSQL podpira nabor funkcionalnosti primerljiv s komercialnimi rešitvami. Predvidevamo, da bo PostgreSQL zadostila potencialnim potrebam v nadaljnjem razvoju aplikacije. Same baze ne namestimo ampak uporabimo uraden zabojnik Docker za PostgreSQL dosegljiv na https://hub.docker.com/_/postgres/. To storimo s sekcijo konfiguracijske datoteke docker-compose.yml, ki je prikazana na sliki 4.1. Ključ image določi zabojnik iz hub.docker.com, ki se uporabi. Ključ *PGUSER* definira uporabniško ime privzetega uporabnika. *POSTGRES_PASSWORD* določi geslo privzetega uporabnika. *POSTGRES_DB* pa ime naše podatkovne baze. To so podatki na podlagi katerih tvorimo *sqlalchemy.url* v konfiguracijiski datoteki aplikacije webapi.ini.

Opis entitet, ki so razvidne v Entitetnemu diagramu 4.2:

```
db:
  image: postgres
  ports:
  - "5432:5432"
  environment:
        PGUSER: postgres
        POSTGRES_PASSWORD: 123456
        POSTGRES_DB: microCombDb
```

Slika 4.1: Izsek datoteke docker-compose.yml, ki vzpostavi bazo PostgreSQL

- **User**, entiteta hrani podatke uporabnika:
  - user_id, je unikatna številka uporabnika, ki jo določi baza
  - email, uporabljamo kot uporabniško ime in zahtevamo, da je unikaten
  - hash, hrani rezultat bcrypt funkcije, potreben za avtentikacijo uporabnika
- **log**, entiteta je dnevnik napak aplikacije:
  - log_id, zaporedna številka vnosa določi sama baza
  - msg, tekst sporočila napake
  - stack, tekstovni zapis sledi sklada napak
- **alembic_version**, entiteto ustvari aplikacija alembic in v njej vodi evidenco migracij izvedenih nad bazo
- **DbStatus**, entiteta hrani stanje baze javnih poskusov:
  - id, avtomatsko določena zaporedna številka
  - name, ime baze, ki se ujema z imenom datoteke, kjer je ta shranjena v datotečnem sistemu
  - lastUpdate, čas zadnje posodobitve baze

  - completed, zastavica je postavljena na resnično, ko je datoteka z bazo uspešno posodobljena
  - dataSetNum, število meritev v trenutni bazi
  - publicationsNum, število publikacij vnešenih v bazo

- **Publication**, entiteta vsebuje podatke o publikaciji, kateri pripadajo meritve:
  - id, avtomatsko določena zaporedna številka
  - pubMedId, unikatna identifikacijska številka članka v zbirki PubMed [41]
  - title, naslov članka
  - summary, povzetek članka
  - organismSN, kratko standardno ime organizma
  - addedToMatrix, vrednost je resnična, ko so vse meritve genskih ekspresij v publikacija dodane v bazo microCOMB
  - dataSetsDownloaded, vrednost je resnična, ko so meritve prenesene na strežnik
  - added, čas zapisa entitete publikacije v bazo
  - status, trenutno stanje publikacije. Možne so tri vrednosti: čaka na prenos meritev, čaka na integracijo v bazo microCOMB ali napaka
  - statusLog, zapišejo se podatki o morebitni napaki pri prenosu ali dodajanju v bazo

- **DataSet**, entiteta meritev v določeni publikaciji:
  - id, avtomatsko določena zaporedna številka
  - publication_id, tuj ključ unikatna številka publikacije
  - downloaded, označeno kot resnično, če je meritev prenesena na strežnik

  - conditionName, oznaka pogojev eksperimenta vira meritve
  - path, pot v datotečnem sistemu do datoteke, ki vsebuje meritve

- **UserQuery**, entiteta hrani uporabnikovo poizvedbo:
  - id, avtomatsko določena zaporedna številka
  - user_id, je unikatna številka uporabnika
  - query_file, pot v datotečnem sistemu do datoteke, ki vsebuje uporabnikove meritve
  - query_name, oznaka poizvedbe določena s strani uporabnika
  - queryExpTh, razmerje v katerem so zapisane ekspresijske vrednosti v posredovanih podatkih
  - minCompSize, minimalna velikost komponent
  - hitsN, maksimalno število dekompozicij komponent v rezultatu
  - scoreFunction, funkcija uporabljena pri ocenjevanju rezultatov
  - part1 do part4, določi število komponent v rezultatu
  - useDataSets, spisek eksperimentov v formatu JSON, katere upoštevamo pri izračunu rezultata
  - description, opis poizvedbe
  - status, trenutno stanje poizvedbe. Možne so tri vrednosti: čaka (and. Queued), končano (ang. Done) ali napaka (ang. Error)
  - decomposition, najdene dekompozicije genskih ekspresij

- **Decomposition**, hrani podatke o dekompoziciji, ki je del analize:
  - id, avtomatsko določena zaporedna številka
  - userQ_id, tuj ključ unikatna številka poizvedbe uporabnika
  - components, komponente, ki so del te dekompozicije
  - gene_coverage, število pokritih genov

  - correlation, stopnja korelacije pokritih genov
  - average_distance, povprečna razdalja med geni
  - score, ocena dekompozicije
  - query_genes, pokriti geni v uporabnikovem vektorju in njihove ekspresijske vrednosti

- **Component**, komponente rezultatov analiz:
  - id, avtomatsko določena zaporedna številka
  - name, oznaka komponente določena s strani microCOMB-a
  - decomposition_id, tuj ključ unikatna številka dekompozicije, kateri pripada komponenta
  - genes, geni, ki pripadajo komponenti in njihove ekspresijske vrednosti

- **GeneToDataSetCacheMap**, predstavlja gen v komponenti:
  - id, avtomatsko določena zaporedna številka
  - component_id, tuj ključ unikatna številka komponente, kateri pripada gen
  - dataset_cache_id, tuj ključ unikatna številka podatkov gena v določenem vektorju
  - dataset_cache, podatki o genu in njegovo kratko ime, ekspresijska vrednost in javna meritev kateri pripada

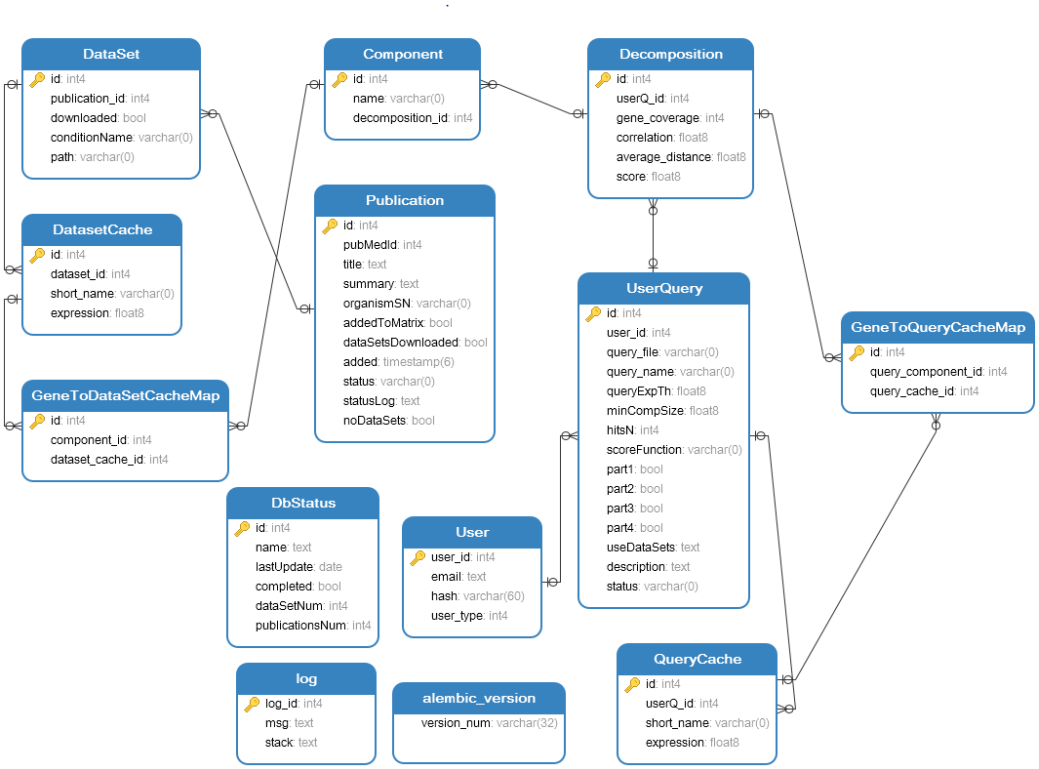
- **DatasetCache**, entiteta hrani podatke o genu v določeni javni meritvi:
  - id, avtomatsko določena zaporedna številka
  - dataset_id, tuj ključ unikatna številka javne meritve
  - short_name, kratek naziv gena
  - expression, ekspresijska vrednost

- **GeneToQueryCacheMap**, predstavlja gen v uporabnikovi meritvi:
  - id, avtomatsko določena zaporedna številka
  - query_decomposition_id, tuj ključ unikatna številka dekompozicije, kateri pripada gen
  - query_cache_id, tuj ključ unikatna številka podatkov gena v uporabnikovi meritvi
  - query_cache, podatki o genu in njegovo kratko ime, ekspresijska vrednost in uporabnikova meritev kateri pripada
- **QueryCache**, entiteta hrani podatke o genu v uporabnikovi meritvi:
  - id, avtomatsko določena zaporedna številka
  - userQ_id, tuj ključ unikatna številka uporabnikove poizvedbe
  - short_name, kratek naziv gena
  - expression, ekspresijska vrednost

4.4 Ogrodje za spletne storitve Cornice

Za definiranje spletnih storitev smo uporabili ogrodje Pyramid in njegovo nadgradnjo Cornice. Pyramid je eno od večjih Python ogrodij za spletne aplikacije, katerega cilj je preprostost in razširljivost. Cornice je nadgradnja Pyramid-a, katere cilj je poenostaviti razvoj spletnih storitev tipa REST. Definicija spletne storitve tako postane Python metoda označena z atributom v datoteki views.py. S pomočjo atributov iz te datoteke lahko definiramo tudi postopek validacije prejetih JSON struktur. V tej nalogi smo za validacijo uporabili modul colander [28], priporočan s strani razvijalcev Cornic-a.

Na izseku kode 4.5 je razviden postopek definicije spletne storitve. Prva vrstica definira, na kateremu URL-ju je storitev dosegljiva, in njeno ime. V tretji vrstici je določen atribut za metodo, ki pove ogrodju, da se do nje dostopa z metodo HTTP GET. Določena je tudi funkcija za validacijo vhodnih podatkov.



Slika 4.2: Model podatkovne baze za spletno aplikacijo microCOMB

```
1  userService = Service(name='user', path='/user', description="User info ",
       cors_origins=('*',))
2
3  @userService.get(validators=has_valid_userid)
4  def get_user(request):
5      user = DBSession.query(User).
6      filter(User.user_id == request.authenticated_userid)
7      .first()
8      if user is None:
9          request.errors.add(name='GetUser',
10                            description='user with id not found',location='/user')
11         return
12     return jsend.success(user.to_json())
```

Koda 4.5: Primer definicije spletne storitve

4.5 Uvoz javno objavljenih ekspresijskih podatkov

Za iskanje podobnosti v genskih ekspresijah med eksperimenti potrebuje microCOMB ažurno bazo javnih ekspresijskih vektorjev. Kot del aplikacije smo zato razvili modul, ki se poveže na javno bazo Intermine [29] ter izvozi ekspresijske vektorje. Ker je potrebno občasno preveriti, ali se je javna baza spremenila, smo uporabili knjižnico apscheduler [30], ki skrbi, da se izvede izvoz podatkov ob določenih intervalih. Postopek posodobitve baze poteka v naslednjih fazah:

1. Bazo Intermine vprašamo za identifikacijske številke vseh objavljenih člankov, ki vsebujejo ekspresijske vektorje;
2. Pridobljen spisek primerjamo s članki, prisotnimi v lokalni bazi, in manjkajoče dodamo;
3. Iz baze se prebere spisek publikacij, ki nimajo pripadajoče datoteke z ekspresijski vektorji;
4. Za vsak manjkajoč ekspresijski vektor preberemo podatke iz Intermine baze, jih preslikamo v format microCOMB-a, ter shranimo v datoteko;
5. V lokalno bazo publikacij zapišemo pot do datoteke z vektorji;
6. Posodobimo entries.txt v microCOMB-u;
7. Sprožimo obnovo microCOMB baze z izvedbo skripte buildDB.py.

Datoteka ekspresijskih vektorjev microCOMB je sestavljena iz poravnanih stolpcev. Prva dva stolpca sta vedno prisotna. Prvi stolpec definira enolično oznako gena (Systematic Name v bazi Intermine) drugi pa možna imena gena (Gene Name v bazi Intermine). Vsi stolpci, ki sledijo, predstavljajo vsak svoj vektor genskih ekspresij pri določenih pogojih.

```
    scheduler = BackgroundScheduler({
        'apscheduler.executors.default': {
            'class': 'apscheduler.executors.pool:ThreadPoolExecutor',
            'max_workers': '1'
        },
        'apscheduler.job_defaults.coalesce': 'false',
        'apscheduler.job_defaults.max_instances': '1',
        'apscheduler.timezone': 'UTC',
    })
    scheduler.add_job(crawle, 'interval', days=1)
    scheduler.start()
```

Koda 4.6: Inicializacija storitve apscheduler

Poglavje 5

Odjemalni del

Arhitektura uporabniškega vmesnika je tipa SPA in temelji na treh JavaScript knjižnicah:

- React.js skrbi za prikazovanje podatkov v obliki HTML. Omogoča dinamično posodabljanje DOM-a glede na spremembe podatkov;
- Redux.js skrbi za hrambo in pridobivanje podatkov ter obveščanje potrebnih modulov o spremembah podatkov. Definira v naprej predpisan postopek, kako se podatki vnesejo v aplikacijo in spreminjajo znotraj nje s ciljem, da so spremembe podatkov čim bolj predvidljive;
- React-Router uporabi React za virtualizacijo navigacije znotraj aplikacije. Ob navigaciji na druge poglede strani spremeni samo DOM.

5.1 Implementacija navigacije

Osnovna stran aplikacije sestoji iz štirih komponent: glave, stranskega menija in noge strani, ki so fiksne komponente in ne podpirajo navigacije, ter glavnega dela strani, ki podpira React-Router navigacijo in služi kot ovoj za druge komponente.

Navigacijo lahko sproži vsaka komponenta React, ki uvozi usmerjevalnik in tako dobi dostop do njegove instance. Navigacijo lahko sprožimo na

```
import React from 'react';
import { Route, IndexRoute } from 'react-router';

import App from './components/App';
import HomePage from './components/HomePage';
// eslint-disable-line import/no-named-as-default
import NotFoundPage  from './components/NotFoundPage';
import SysLogPage  from './components/SysLogPage';
import DbStatusPage  from './components/DbStatusPage';
import ComposeDbPage  from './components/ComposeDbPage';
import ResultPage  from './components/ResultPage';s
import UserPage  from './components/UserPage';

export default (
  <Route path="/" component={App}>
    <IndexRoute component={HomePage}/>
    <Route path="Logs" component={SysLogPage}/>
    <Route path="Status" component={DbStatusPage}/>
    <Route path="ComposeDb" component={ComposeDbPage}/>
    <Route path="Result/:id" component={ResultPage}/>
    <Route path="User/:id" component={UserPage}/>
    <Route path="*" component={NotFoundPage}/>
  </Route>
);
```

Koda 5.1: Konfiguracija usmerjevalnika React

dva načina: neposredno v kodi, tako da želeno lokacijo porinemo na sklad browser-History z ukazom *push*, ali z uporabo komponente Link, kateri v atribut *to* vpišemo želeno ciljno lokacijo. Komponenta Link ob kliku avtomatsko sproži navigacijo. V naši aplikaciji le-to upravlja levi navigacijski meni, sestavljen iz Link komponent. Občasno pa je potrebno ob zaključku operacij z drugimi komponentami sprožiti navigacijo tudi ročno.

5.2 Uporaba React.js in Redux

Za delovanje Redux-a je potrebno ob zagonu aplikacije inicializirati njegovo shrambo podatkov. Za inicializacijo potrebujemo tri objekte:

- Korenski reduktor, ki deluje kot usmerjevalnik med ostalimi reduktorji in se inicializira s funkcijo combineReducers katere parameter je spisek vseh reduktorjev;
- Začetno stanje shrambe podatkov v obliki JSON;
- Vmesno programsko opremo, ki je v našem primeru samo instanca Redux-Thunk-a [31], ki nam omogoča asinhrono branje podatkov iz strežnika.

Inicializiranje React-a poteka tako, da se ob zagonu aplikacije sproži *render* metoda, ki izvede *document.getElementById('app')*. React si od tega trenutka dalje lasti vsebino HTML elementa z id-jem 'app' in dinamično posodablja DOM tega elementa. V našem primeru je začetna stran HTML aplikacije index.ejs in vsebuje samo en div element z id-jem "app". React dinamično definira ostali HTML naše aplikacije.

Implementacijo in uporabo React-a in Redux-a znotraj aplikacije je najbolje razložiti s podrobnim opisom preprostega primera uporabe aplikacije. Izbrali smo primer prikaza zadnjih vrstic dnevnika napak strežnika. Maska je preprosta in sestoji iz gumba za osvežitev ter izpisa v obliki tabele zadnjih vrstic dnevnika napak. Postopek se začne z navigacijo na komponento

SysLogPage. Končnica Page v imenu komponente je popularna konvencija v aplikacijah React in označuje komponente, ki upravljajo s podatki. Njihova naloga je podobna kontrolerju v arhitekturi MVC.

Kot je razvidno na izseku 5.2 v svoji metodi *render* SysLogPage definira samo Naslov strani in kot otroka prikaže komponento LogList, katere naloga je prikaz podatkov in gumba za osveževanje. Komponenta LogList je podobna pogledu v arhitekturi MVC. Ker LogList ni vezan na shrambo Redux, in ne more dostopati do podatkov, mu jih posreduje SysLogPage. V svoji metodi *render* SysLogPage poda LogList-u referenco na svojo metodo za osveževanje podatkov kot lastnost "refresh" in objekt s trenutnimi podatki kot lastnost sysLogs. Kako SysLogPage pridobi podatke, je razvidno iz zadnjih vrstic izseka kode 5.2. Metoda *connect* poveže komponento SysLogPage s shrambo Redux. Metoda *mapStateToProps* preslika podatke pridobljene iz shrambe v lokalne lastnosti komponente. Naloga metode *mapDispatchToProps* je predvsem poenostaviti kodo za izvedbo akcije. Ob pritisku na gumb se izvede akcija *getLogs*, ker pa je le-ta definirana kot *thunk*, se izvede asinhrono v ozadju. Glavna nit programa medtem nadaljuje z izvajanjem. Šele, ko se akcija *getLogs* zaključi, se izvedejo ukazi v metodah *then()* in *catch()*, ki sta vezani nanjo.

V izseku kode 5.3 si lahko ogledamo definicijo akcije *getLogs*. Akcija izvede metodo *logApi.getLogs*, ki iz strežnika prebere želene podatke in jih vrne kot rezultat v obliki JSON. Akcija preveri, ali je poizvedba uspela tako, da prebere polje status na vrnjenem objektu. Če poizvedba ni uspela, je status enak "fail"ali "error". V tem primeru akcija sproži izjemo. V primeru, da poizvedba uspe, akcija izvede drugo akcijo *loadLogsSuccess*, in jih kot parameter poda rezultat poizvedbe. Metoda *loadLogsSuccess* vrne objekt z definiranim poljem "type". Če obstaja reduktor, ki obravnava akcijo tega tipa, se njegova koda izvede. V našem primeru je to logReducer. Naloga reduktorjev je spreminjanje stanja podatkovne shrambe Redux. Da bi se izognili potratnim primerjavam stanj, se morajo reduktorji držati enega pravila: objektov v shrambi se ne spreminja, ampak se jih v celoti zamenja. Objekt, ki ga vrne

```
reloadLogs(event) {
    event.preventDefault();
    this.props.actions.getLogs().then( toastr.success('refresh success')).
    catch(error => {
      debugger;
      error.hasOwnProperty('status') ?
        error.errors.forEach(x=> toastr.error(x.description))
        : toastr.error(error);
      this.setState({executing: false});
    });
  }

  render() {
    const {sysLogs} = this.props;
    return (
      <div>
        <h1>
          BackEnd Log last 40 lines
        </h1>
        <LogList refresh={this.reloadLogs} sysLogs={sysLogs}/>
      </div>
    );
  }
}
function mapStateToProps(state, ownProps) {
  return {
    sysLogs: state.sysLogs
  };
}
function mapDispatchToProps(dispatch) {
  return {
    actions: bindActionCreators(logActions, dispatch)
  };
}
export default connect(mapStateToProps, mapDispatchToProps)(SysLogPage);
```

Koda 5.2: Izsek skripte SysLogPage.js

```
export function loadLogsSuccess(logs) {
  return {type: types.LOAD_LOGS_SUCCESS, logs};
}

export function getLogs() {
  return function (dispatch, getState) {
    dispatch(beginAjaxCall());
    let state = getState();
    return logApi.getLogs(state.header).then(response => {
      if (response.status === 'success') {
        dispatch(loadLogsSuccess(response.data));
      } else {
        throw(response);
      }
    }).catch(error => {
      dispatch(ajaxCallError(error));
      throw(error);
    });
  };
}
```

Koda 5.3: Definicija akcij za dnevnik napak

```
export default function logReducer(state = initialState.sysLogs, action) {
  switch (action.type) {
    case types.LOAD_LOGS_SUCCESS:
      return action.logs;
    default:
      return state;
```

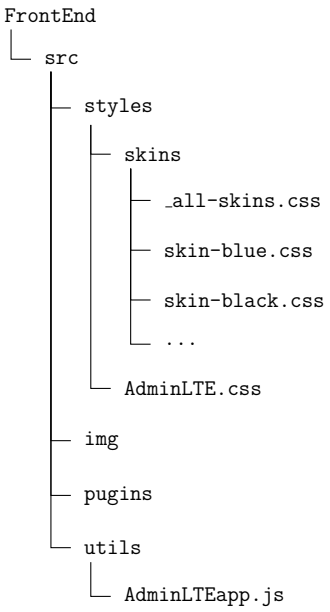
Koda 5.4: Izsek skripte logReducer.js

reduktor, postane novo stanje shrambe. Ker nas prejšnje stanje ne zanima, ga v našem izseku 5.4 kar zamenjamo z novim. Ko se stanje shrambe spremeni, Redux obvesti vse nanj vezane komponente React. V tem primeru je to SysLogPage. Ko le-ta prejme nove podatke, jih preda komponenti LogList, ki se ponovno izriše in novi podatki postanejo vidni uporabniku.

Reduktorji operirajo samo nad delom celotne shrambe. Po konvenciji se jim dodeli polje shrambe, ki nosi isto ime kot reduktor ob inicializaciji v korenskemu reduktorju.

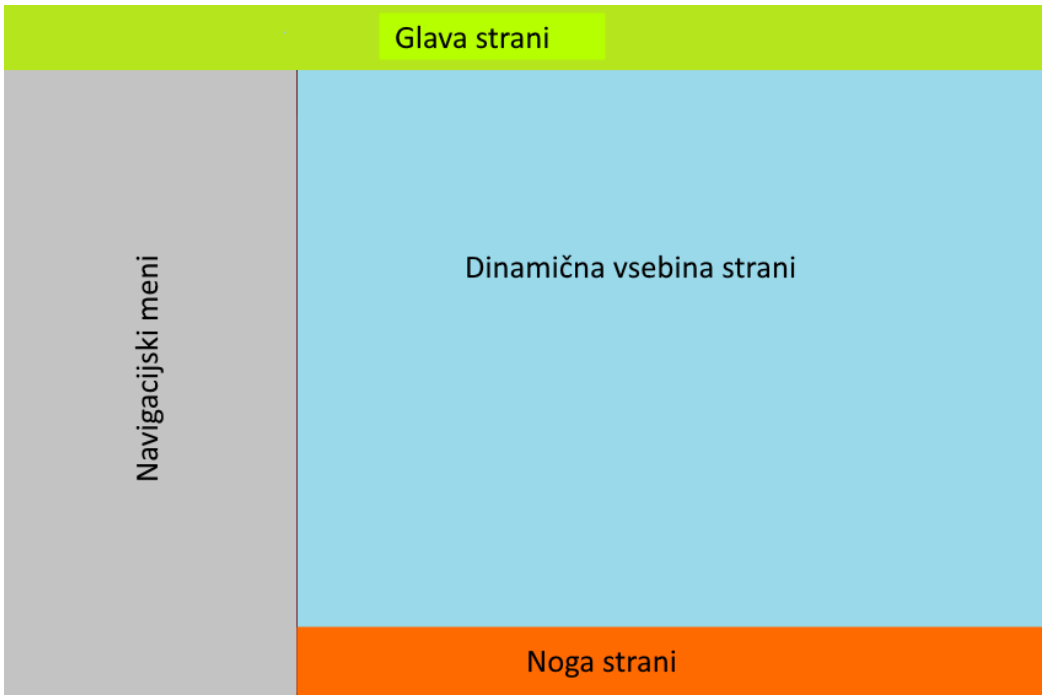
5.3 Oblikovanje uporabniškega vmesnika

Za oblikovanje uporabniškega vmesnika smo uporabili ogrodje bootstrap, ki olajša razvoj odzivnih uporabniških vmesnikov. Bootstrap definira stil osnovnih elementov, kot so gumbi za tekstovna polja in podobno, in da se le ti prilagajajo velikosti zaslona uporabnika ter tipu naprave na kateri se aplikacija izvaja. Za okolje bootstrap je mogoče ustvariti predlogo, ki spremeni barvno shemo in obliko elementov aplikacije. Veliko takih predlog je objavljenih v obliki odprtokodnih projektov in so proste za uporabo. Za našo aplikacijo smo izbrali AdminLTE [38], ki jo je pod MIT licenco objavil Almsaeed Studio [39]. Predlog vsebuje veliko nam neuporabnih vtičnikov, ki omogočajo kompleksne prikaze diagramov in animacije elementov. Zato smo iz predloge, kot je razvidno na sliki 5.1, uporabili samo njen osnovni css stil v datoteki AdminLTE.css, barvne sheme v mapi skins ter AdminLTEapp.js skripto JavaScript, ki skrbi za prilagajanje predloge na velikost zaslona.



Slika 5.1: Datoteke, ki definirajo obliko aplikacije

Iz predloge smo obdržali tudi strukturo osnovne strani, ki jo sestavljajo: levi navigacijski meni, glava strani, ki prikazuje podatke o uporabniku, ter omogoča prijavo in sredinski del strani, ki smo ga uporabili za prikaz glavnih pogledov v aplikaciji.



Slika 5.2: Struktura uporabniškega vmesnika

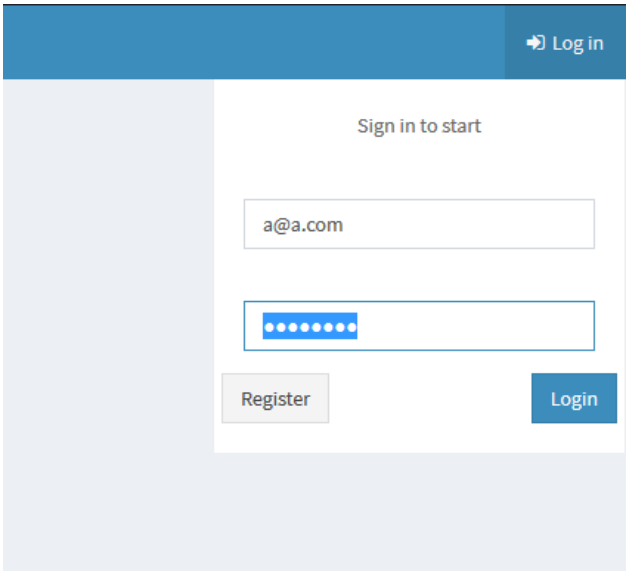
Poglavje 6

Predstavitev aplikacije

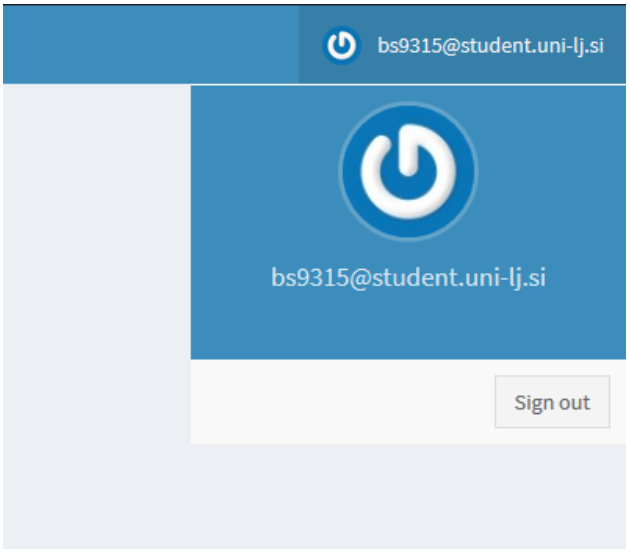
V tem poglavju podrobneje opišemo glavne funkcionalnosti razvite aplikacije. Osnovna zahteva aplikacije je, da omogoči uporabnikom prijavo in zasebno hrambo rezultatov analiz ter posredovanih ekspresijskih podatkov. Za izvedbo samih analiz smo izdelali uporabniški vmesnik preko katerega uporabnik določi parametre analize in, katere javne ekspresijske vektorje v bazi naj analiza upošteva. Za prevedbo stanja baze microCOMB smo izdelali še vmesnik stanja uvoza javnih ekspresijskih vektorjev v bazo. Uporabnik ima svojo domačo stran, kjer vidi zgodovino svojih izvedenih analiz ter dostopa do njihovih rezultatov. Lahko se tudi odloči za javno objavo svojih ekspresijskih profilov, ki se nato uvozijo v bazo microCOMB. Objavljeni ekspresijski profili so potem na voljo drugim uporabnikom pri izbiri vektorjev za analize.

6.1 Prijava uporabnika

Prijavo v aplikacijo omogoča meni Login v zgornjem desnem kotu uporabniškega vmesnika. Ko uporabnik ni prijavljen, klik na meni prikaže vnosno masko za prijavo. V masko uporabnik vnese naslov svoje elektronske pošte in geslo. Če je prijava uspešna, se v taistem meniju vnosna maska skrije. Namesto nje se prikažejo uporabnikovi podatki. V primeru, da uporabnik še nima računa in se želi registrirati, izpolni masko in pritisne gumb Register.



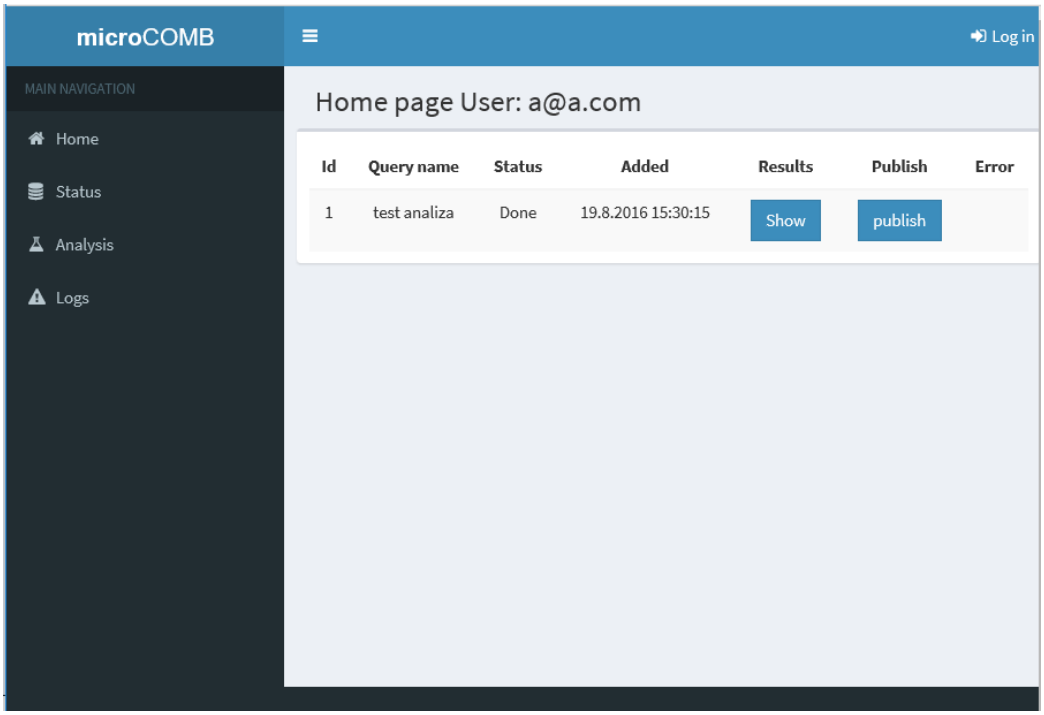
Slika 6.1: Maska za prijavo



Slika 6.2: Meni po uspešni prijavi uporabnika

6.2 Domača stran uporabnika

Glavna naloga uporabniške strani je, da uporabniku prikaže zgodovino njegovih analiz ter mu ob kliku nanjo ponovno prikaže rezultate analize. Sekundarna naloga uporabnikove strani pa je javna objava posredovanega ekspresijskega vektorja. Stran sestavlja tabela uporabnikovih analiz, ki prikazuje unikatno številko analize, čas posredovanja, njeno stanje, gumb za prikaz rezultatov ter gumb za javno objavo. Analize imajo lahko štiri stanja: na čakanju, v obdelavi, zaključena ali napaka. Pri kliku na gumb za javno objavo se zaradi preprečevanja napak uporabnika vpraša, ali je prepričan, ali želi objaviti svoj ekspresijski vektor.



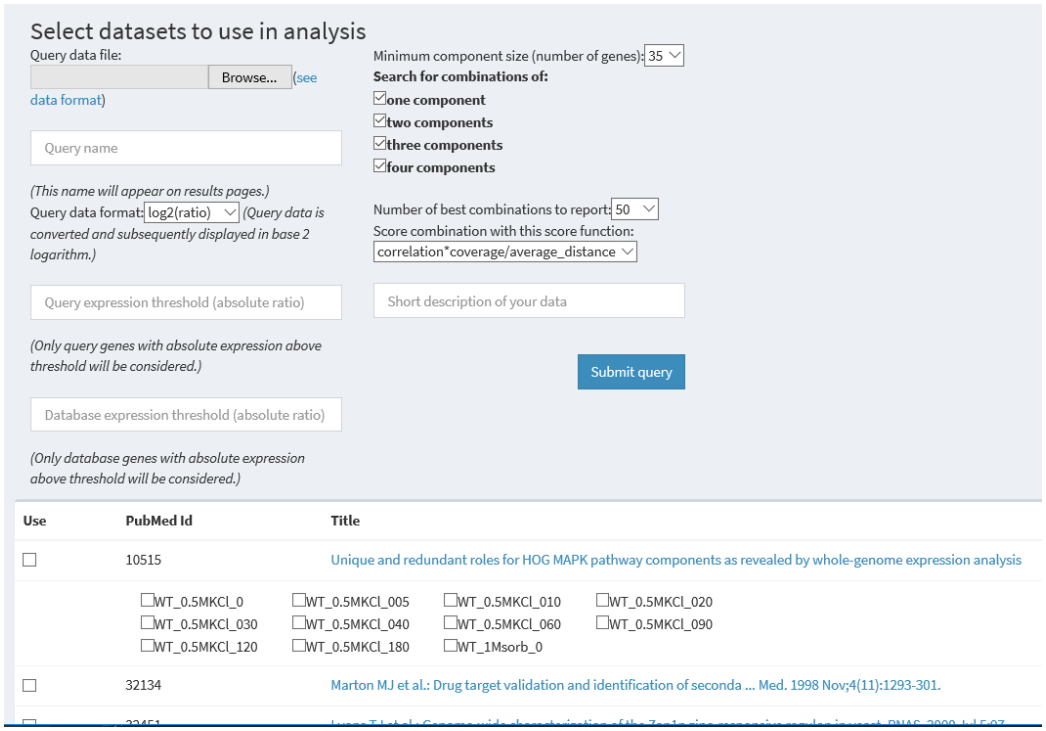
Slika 6.3: Primer domače strani uporabnika

6.3 Podajanje parametrov analize in izvedba analize

Na vrhu strani uporabnik vnese svojo datoteko ekspresijskih vektorjev in določi obvezne parametre analize. Nato uporabnik določi s katerimi ekspresijskimi vektorji v bazi želi primerjati svojega. Za izbiro publikacij aplikacija, prikaže spisek vseh vnosov v bazi v obliki tabele. V prvem stolpcu tabele so potrditvena polja, in če so le-ta potrjena, se v analizi upoštevajo vsi ekspresijski vektorji publikacij. V primeru, da uporabnik želi izbrati samo določene vektorje v publikaciji, lahko to stori s klikom na vrstico publikacije. Nato se polje razširi in prikažejo se potrditvena polja za vektorje v publikaciji. Ob pritisku na gumb poizvedi, se s strežnika prenesejo informacije o izbranih poljih. Strežnik te informacije shrani v uporabnikovo tabelo analiz in njeno stanje postane Na čakanju (ang. Queued). Uporabnik je nato preusmerjen na svojo domačo stran, kjer čaka, da se stanje analize spremeni v Zaključeno (ang. Done). Strežnik v ozadju po vrsti prebere čakajoče analize iz baze in podane parametre ter požene analizo microCOMB. Ko se analiza zaključi, strežnik osveži stanje analize.

6.4 Prikaz rezultata analize

Namen strani je prikazovanje rezultatov analize uporabniku na čim bolj pregleden način. Odločili smo se, da stran razdelimo na dva dela. Levo imamo spisek najdenih kombinacij v obliki tabele, razvrščen padajoče po njihovih ocenah, ker uporabnika zanimajo predvsem visoko ocenjene kombinacije. Ob kliku na kombinacijo pa se podrobnosti prikažejo na desni polovici strani. Podrobnosti so sestavljene iz pregleda ter opisov komponent, ki so prisotne v njej. Pregled vsebuje: grafični prikaz rezultatov; splošne podatke o izbrani kombinaciji; oceno kombinacije; prisotne komponente in korelacijo. Vrednosti ekspresije posameznega gena se izpišejo v podrobnostih komponent. Uporabniku je omogočen spustni meni, v katerem lahko izbira pregledovanje ka-



Slika 6.4: Izvedba analize

terekoli prisotne komponente.

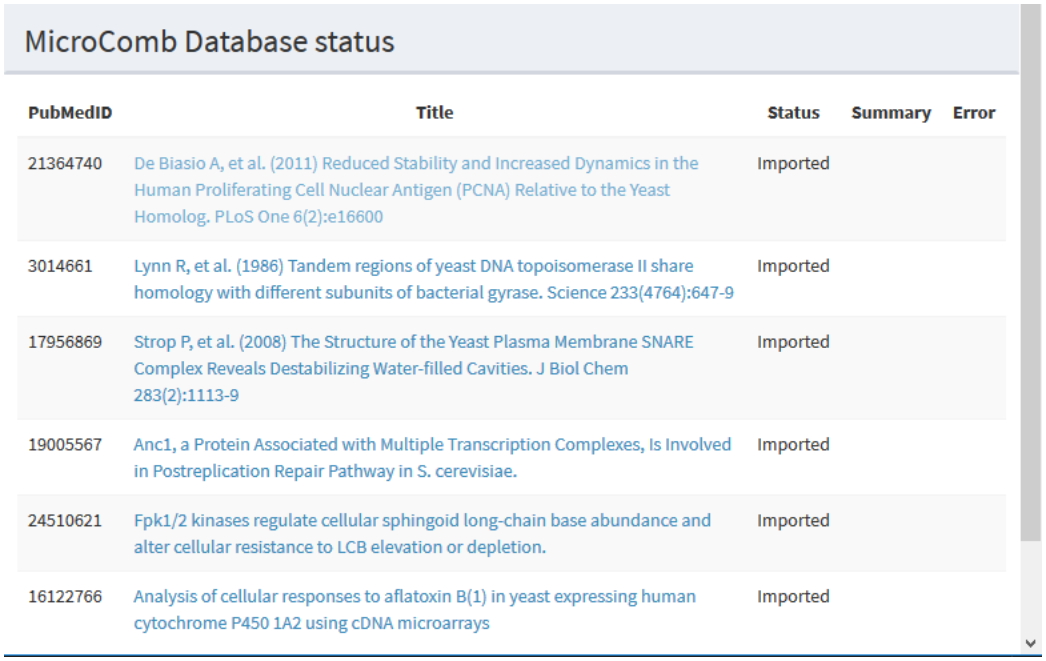
6.5 Prikaz stanja javne baze

Na tej strani si lahko uporabniki ogledajo stanje baze microCOMB-a. Na vrhu strani se izpisujejo splošni podatki o stanju baze: čas zadnje posodobitve, količina publikacij v bazi, število publikacij čakajočih na uvoz, in kdaj se bo izvedla naslednja posodobitev. Uporabnik se na podlagi teh informacij lahko hitro odloči, ali je smiselno ponoviti katero od njegovih analiz. Nekateri uporabniki bodo želeli izvesti analizo z vektorji v točno določeni publikaciji, zato na tej strani prikazujemo stanje vseh sistemu znanih publikacij. Na strani v tabelarični obliki prikazujemo tudi stanje publikacij. Mogoča so štiri stanja:

Slika 6.5: Prikaz rezultatov

- Na čakanju (Queued): aplikacija je zaznala publikacijo, ni pa še prenesla njenih ekspresijskih vektorjev;
- Prenešeno (Retrieved): ekspresijski vektorji publikacije so preneseni in shranjeni v pripadajoči podatkovni datoteki, niso pa integrirani v matriko microCOMB. Analize jih zato ne upoštevajo;
- Uvoženo (Imported): ekspresijski vektorji publikacije so dodani matriki microCOMB, zato lahko uporabnik izvaja analize s to publikacijo;
- Napaka (Error): nekje v postopku obdelave publikacije je prišlo do napake.

Zaradi velike količine publikacij smo na stran dodali iskalno polje, s katerim lahko uporabnik poišče določeno publikacijo na podlagi njene identifikacijske številke PubMed.



Slika 6.6: Prikaz stanja baze javnih ekspresij

Poglavje 7

Sklepne ugotovitve

V okviru diplomske naloge smo razvili aplikacijo, ki uporabniku omogoča enostavno primerjavo eksperimentalno izmerjenih genskih ekspresij z uporabo programa microCOMB. Razvita aplikacija omogoča uporabnikom posredovanje svojih ekspresijskih podatkov v analizo, prikaz rezultatov preteklih in novih analiz, ter javno objavo svojih ekspresijskih podatkov.

Poudarek naloge je bila uporaba sodobnih metod razvoja spletnih aplikacij, modularnost sistema in preprostosti vzdrževanja. V sklopu naloge smo podrobneje opisali vzpostavitev razvojnega okolja, arhitekturo aplikacije in njeno delovanje.

Ugotovili smo, da je sodobnih metod razvoja spletnih aplikacij veliko, in da se le-te med seboj zelo razlikujejo. Zaradi te raznolikosti smo se odločili za React, ki je popularen in podprt s strani tehnološkega giganta kot je Facebook. Z izbiro arhitekture smo imeli manj težav, saj je konfiguracija spletnih storitev REST in tip aplikacije SPA v brskalniku postala skoraj standard razvoja spletnih aplikacij.

Možnih izboljšav, ki so nam prišle na misel med razvojem je veliko. Med najbolj perspektivnimi so:

- Uporabiti bazo NoSql za hranjenje ekspresijskih podatkov. Trenutno aplikacija hrani ekspresije v datotečni obliki, ker je sama količina podatkov in način njihove obdelave v nasprotju s hrambo v RDBMS bazi

kot je PostgreSQL

- Izboljšati uporabniški vmesnik glede na odziv uporabnikov.
- Vzpostaviti sistem sprotne integracije in izboljšati pokritost kode s testiranjem enot. Na ta način bi pohitrili kontinuiran razvoj in zmanjšali količino napak v končnemu izdelku.

Literatura

[1] microCOMB za določanje komponent genske ekspresije [Online]. Dosegljivo:
http://bubble.fri.uni-lj.si/microCOMB/. [Dostopano 2016-08-29].

[2] PIP The PyPA recommended tool for installing Python packages. [Online]. Dosegljivo:
https://pypi.python.org/pypi/pip [Dostopano 2016-08-29].

[3] PyCharm Python IDE for Professional Developers [Online]. Dosegljivo:
https://www.jetbrains.com/pycharm/ [Dostopano 2016-08-29].

[4] Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. [Online]. Dosegljivo:
https://nodejs.org [Dostopano 2016-08-29].

[5] NPM the package manager for the JavaScript runtime environment Node.js. [Online]. Dosegljivo:
https://www.npmjs.com [Dostopano 2016-08-29].

[6] WebPack module bundler. [Online]. Dosegljivo:
https://webpack.github.io [Dostopano 2016-08-29].

[7] WebStorm The smartest JavaScript IDE [Online]. Dosegljivo:
https://www.jetbrains.com/webstorm/ [Dostopano 2016-08-29].

[8] React Developer Tools [Online]. Dosegljivo:
https://github.com/facebook/react-devtools [Dostopano 2016-08-29].

[9] Redux DevTools Extension [Online]. Dosegljivo: https://github.com/zalmoxisus/redux-devtools-extension [Dostopano 2016-08-29].

[10] Ecma International [Online]. Dosegljivo: http://www.ecma-international.org. [Dostopano 2016-08-29].

[11] Babel The compiler for writing next generation JavaScript [Online]. Dosegljivo: https://babeljs.io/. [Dostopano 2016-08-29].

[12] Docker Build, Ship, and Run Any App, Anywhere [Online]. Dosegljivo: https://www.docker.com. [Dostopano 2016-08-29].

[13] Docker Compose [Online]. Dosegljivo: https://docs.docker.com/compose. [Dostopano 2016-08-29].

[14] React A JavaScript library for building user interfaces [Online]. Dosegljivo: https://facebook.github.io/react. [Dostopano 2016-08-29].

[15] Redux is a predictable state container for JavaScript apps [Online]. Dosegljivo: http://redux.js.org [Dostopano 2016-08-29].

[16] React Slingshot is a comprehensive starter kit for rapid application development using React. [Online]. Dosegljivo: https://github.com/coryhouse/react-slingshot [Dostopano 2016-08-29].

[17] Password Hashing Competition [Online]. Dosegljivo: https://password-hashing.net/. [Dostopano 2016-08-29].

[18] Alembic database migration tool for SQLAlchemy. [Online]. Dosegljivo: https://pypi.python.org/pypi/alembic. [Dostopano 2016-08-29].

[19] PyJWT JSON Web Token implementation in Python [Online]. Dosegljivo: https://pypi.python.org/pypi/PyJWT/1.4.2. [Dostopano 2016-08-29].

[20] Bcrypt 2.0.0: Python Package Index [Online]. Dosegljivo: https://pypi.python.org/pypi/bcrypt/2.0.0. [Dostopano 2016-08-29].

[21] Cornice A REST framework for Pyramid [Online]. Dosegljivo: https://cornice.readthedocs.io/. [Dostopano 2016-08-29].

[22] Bootstrap 3 [Online]. Dosegljivo: https://getbootstrap.com/. [Dostopano 2016-08-29].

[23] PostgreSQL The world's most advanced open source database [Online]. Dosegljivo: https://www.postgresql.org. [Dostopano 2016-08-29].

[24] NGINX High Performance Load Balancer, Web Server, Reverse Proxy [Online]. Dosegljivo: https://www.nginx.com. [Dostopano 2016-08-29].

[25] Pyramid [Online]. Dosegljivo: http://www.pylonsproject.org. [Dostopano 2016-08-29].

[26] SQLAlchemy The Database Toolkit for Python [Online]. Dosegljivo: http://www.sqlalchemy.org. [Dostopano 2016-08-29].

[27] PostgreSQL + Python — Psycopg [Online]. Dosegljivo: http://initd.org/psycopg. [Dostopano 2016-08-29].

[28] Colander 1.3.1 documentation [Online]. Dosegljivo: http://docs.pylonsproject.org/projects/colander. [Dostopano 2016-08-29].

[29] InterMine — University of Cambridge [Online]. Dosegljivo: http://intermine.org. [Dostopano 2016-08-29].

[30] Advanced Python Scheduler [Online]. Dosegljivo: https://apscheduler.readthedocs.io. [Dostopano 2016-08-29].

[31] Redux-Thunk Thunk middleware for Redux. [Online]. Dosegljivo: https://github.com/gaearon/redux-thunk. [Dostopano 2016-08-29].

[32] Git is a free and open source distributed version control system [Online]. Dosegljivo: https://git-scm.com. [Dostopano 2016-08-29].

[33] Python 3.5.2 [Online]. Dosegljivo: https://www.python.org. [Dostopano 2016-08-29].

[34] Using PyCharm with Pyramid - The Pyramid Community Cookbook v0.2 [Online]. Dosegljivo: http://docs.pylonsproject.org/projects/pyramid-cookbook/en/latest/development_tools/pycharm.html#using-pycharm [Dostopano 2016-08-29].

[35] JetBrains IDE Support [Online]. Dosegljivo: https://chrome.google.com/webstore/detail/jetbrains-ide-support/hmhgeddbohgjknpmjagkdomcpobmllji?utm_source=chrome-app-launcher-info-dialog [Dostopano 2016-08-29].

[36] HMAC: Keyed-Hashing for Message Authentication [Online]. Dosegljivo: https://www.ietf.org/rfc/rfc2104.txt [Dostopano 2016-08-29].

[37] Using SHA2 Algorithms with Cryptographic Message Syntax [Online]. Dosegljivo: https://tools.ietf.org/html/rfc5754 [Dostopano 2016-08-29].

[38] AdminLTE fully responsive admin template. Based on Bootstrap 3 framework. [Online]. Dosegljivo: https://github.com/almasaeed2010/AdminLTE [Dostopano 2016-08-29].

[39] Almsaeed Studio [Online]. Dosegljivo:
https://almsaeedstudio.com/ [Dostopano 2016-08-29].

[40] JSON Web Token (JWT) [Online]. Dosegljivo:
https://tools.ietf.org/html/rfc7519 [Dostopano 2016-08-29].

[41] PubMed - NCBI [Online]. Dosegljivo:
https://www.ncbi.nlm.nih.gov/pubmed [Dostopano 2016-08-29].