VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
MATEMATINĖS INFORMATIKOS KATEDRA

Biologinių aplikacijų kūrimas su ’DJango’ karkasu

Development of biological web applications using ’DJango’ frameworks

Kursinis darbas

Atliko: 3 kurso Bioinformatikos studentė
Ugnė Antanaitytė

Darbo vadovas: lekt. Irus Grinis

Vilnius 2022

TURINYS

Įvadas

Žiniatinklio programa (aplikacija) - yra taikomoji programa, kuri saugoma nuotoliniame serveryje ir pateikiama internetu per naršyklės sąsają [Con10]. Žiniatinklio programos gali turėti įvairių paskirčių, jomis gali naudotis tiek organizacijos darbo tikslams įgyvendinti, tiek žmonės asmeninėms reikmėms. Biologinės aplikacijos dažnai yra skirtos skaičiavimo algoritmų, kurie paremti biologiniais mechanizmais, naudojimuisi mokslinių tyrimu metu arba edukaciniais tikslais. Tokios žiniatinklio programos yra naudojamos įvairių sričių gamtos mokslų specialistų ar tam tikrų įmonių darbuotojų. Jos yra ganėtinai populiarios, nes yra universalios, turi lengvai naudojamas vartotojo sąsajas ir leidžia naudotis biologiniais įrankiais jų neinstaliavus ir neapkraunant savo kompiuterio atminties.

Biologinių aplikacijų kūrimui dažniausiai naudojami įvairūs karkasai – įrankių rinkiniai, kurie leidžia žiniatinklio programos vystytojams lengviau integruoti funkcijas, bei kurti ir valdyti žiniatinklio programas. Plečiantis biologinei aplikacijai, programos kūrėjams darosi sunku palaikyti tvarkingą ir vientisą programavimo architektūrą. Šią problemą nesunku išspręsti su žiniatinklio karkasu, kuris suteikia programavimo infrastruktūrą, leidžiančią sutelkti dėmesį į švaraus kodo rašymą, kurį būtų nesunku atnaujinti ir pildyti ateinančiose produkto versijose. Šiame darbe nagrinėjamas „Django” - aukšto lygio „Python” žiniatinklio karkasas, leidžiantis kurti biologines aplikacijas [Fou21].

Šio darbo tikslas yra susipažinti su bioinformatikiniais žiniatinklio programų konstravimo metodais ir su „Django” karkaso veikimo principais, bei jame įdiegtomis funkcijomis.

Tam, kad įgyvendinti šiuos tikslus, išsikelti uždaviniai:

1. Išnagrinėti literatūrą, aprašančią „Django” karkaso suteikiamą programavimo infrastruktūrą, bei jo principus, kurie padeda spręsti tinklapių konstravimo problemas.

2. Aprašyti terminalo emuliatoriaus reikšmę ir privalumus, bei surasti įrankį, kurį būtų galima pritaikyti terminalo emuliatoriaus implementacijai naršyklėje.

3. Aptarti „Unix” komandų ir E-įrankių konvejerių reikšmę bioinformatikoje.

4. Intuityviai išsikelti reikalavimus savo kuriamai biologiniai žiniatinklio aplikacijai.

5. Remiantis išsikeltais tikslais, bei įgytomis žiniomis nagrinėjant literatūrą, sukurti biologinę tinklapio aplikaciją.

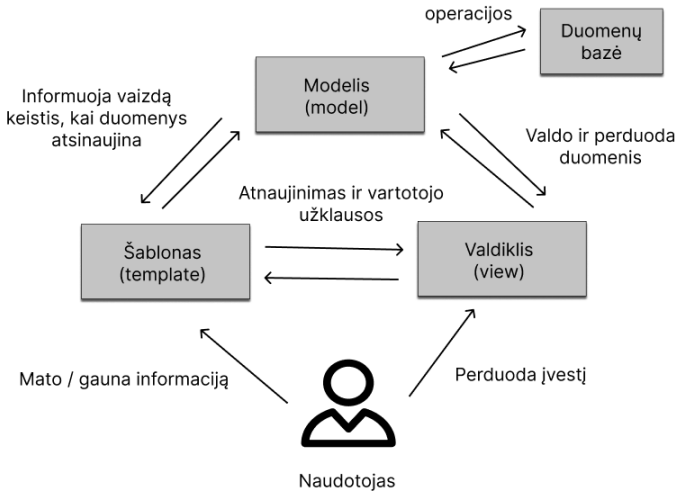
6. Atsižvelgiant į išnagrinėtą literatūrą ir praktinę darbo dalį, apibendrinti projekto rezultatus, bei suformuluoti išvadas apie „Django" karkasą ir kitus naudotus įrankius.

1. „Django” karkaso apžvalga

Karkasai laikomi svarbia programų kūrimo dalimi, kadangi kylantys taikomųjų programų standartai ir technologijų sudėtingumas, reikalauja turtingesnių ir interaktyvesnių žiniatinklio programų. Karkasai gali apimti serverio pusės (*angl. Backend*) ir / arba kliento pusės (*angl. Frontend*) kūrimo sistemas. „Django” - aukšto lygio serverio pusės „Python” žiniatinklio karkasas, kuris paremtas pagrindinių funkcijų įtraukimo (*angl. batteries-included*) filosofija. Remiantis šia filosofija, „Django” karkasas sukonstruotas taip, kad dažniausiai naudojamos funkcijos tinklapių kūrimui suteikiamos su pačia karkaso sistema, o ne kaip atskiri moduliai . Taip pat šiame karkase realizuotas Modelis-Vaizdas-Šablonas (*angl. Model-View-Template*) programinės įrangos projektavimo modelis, kuris leidžia lengviau kurti objektų atvaizdavimus žiniatinklyje [Adr22].

1.1. Modelis-Vaizdas-Šablonas dizaino modelis

Modelis-Vaizdas-Šablonas (*angl. Model-View-Template*) apibrėžia programinės įrangos kūrimo būdą. Vadovaujantis šio būdo principais programos kodas rašomas taip, kad duomenų apibrėžimo ir prieigos dalis (modelis) būtų atskirta nuo užklausų maršruto parinkimo logikos (vaizdo), kuri savo ruožtu yra atskirta nuo vartotojo sąsajos (šablono). Pagrindinis tokio metodo privalumas yra tas, kad komponentai yra laisvai sujungti, kiekviena atskira dalis turi vieną pagrindinį tikslą ir gali būti keičiama atskirai, nepaveikdama kitos dalies [Adr08d].



1 pav. Modelis-Šablonas-Vaizdas struktūra

Vaizdo (*angl. view*) paskirtis „DJango" sistemoje užtikrinama „Python" funkcijomis, kurios priima žiniatinklio užklausą (*angl. Web request*) ir pateikia žiniatinklio atsakymą (*angl. Web response*). Šis atsakymas gali būti HTML turinys, tinklalapis, peradresavimas, 404 klaida, XML dokumentas ir pan. Vaizdo pagrindinė funkcija - užtikrinti logiką, reikalingą atsakymui tinklapio naudotojui grąžinti [Adr08e].

Svetainėms yra būdingas dažnesnis dizaino keitimasis nei kodo, taip pat dizaino kūrimas ir programavimas yra laikomi skirtingomis disciplinomis, todėl didelis privalumas yra tai, kad dizainas gali pasikeisti nekeičiant pačio kodo. Tai galima padaryti naudojant „DJango" šablonų (*angl. template*) sistemą. Šablonas yra teksto eilutė, skirta atskirti dokumento pateikimą nuo jo logikos. Toks komponentas apibrėžia vietos rezervavimo ženklus ir įvairius loginius veiksmus, kurie reguliuoja, kaip dokumentas turi būti rodomas. Šio karkaso sistema leidžia pasinaudoti šablonų paveldimumo ar įtraukimo strategijomis, tokiu būdu užtikrinant kodo tvarkingumą ir retesnį pasikartojimą (*angl. duplicate code*) [Adr08f].

Šiuolaikinėse žiniatinklio programose logika apima sąveiką su duomenų baze. Duomenų baze valdoma svetainė prisijungia prie duomenų bazės serverio, nuskaito iš jo kai kuriuos duomenis ir tuos duomenis pateikia suformatuotus tinklalapyje. Taip pat teikiamos funkcijos, leidžiančios svetainės lankytojams savarankiškai užpildyti duomenų bazę. Duomenų prieigos sluoksnyje yra viskas apie duomenis: kaip prieiti prie jų, kaip juos patvirtinti, kokia elgsena jie vadovaujasi ir kokiais ryšiais jie siejami. „DJango" modelis yra duomenų aprašymas jūsų duomenų bazėje, vaizduojamas kaip „Python" kodas. Modelis gali naudotis įvairiais duomenų bazių varikliais: „PostgreSQL", „MySQL", „SQLite", „Oracle" and „Microsoft SQL Server". Numatytasis duomenų bazių variklis - „SQLite" [Adr08g].

1.2. Administratoriaus sąsaja

Aplikacijoms administratoriaus sąsaja gali būti svarbi infrastruktūros dalis. Administratoriaus sąsaja - žiniatinklio sąsaja, skirta tik tam tikriems vartotojams, leidžianti pridėti, redaguoti ir ištrinti svetainės turinį, bei valdyti naudotojų prieigos leidimus (*angl. user access permission*). „DJango" suteikia galimybę naudotis automatine administratoriaus sąsaja, ši funkcija veikia nuskaitant sukurtų modelių metaduomenis ir sukuriant sąsaja, kuri gali tuos duomenis valdyti ir re-

daguoti. Be pagrindinių duomenų redagavimo užduočių, administratoriaus sąsaja yra naudinga duomenų modelių tikrinimui (sukūrus naują modelį, sekanti užduotis jį iškviesti administratoriaus sąsaja ir įvesti netikrus duomenis, kad patikrinti, ar modelis veikia teisingai), gaunamų duomenų tvarkymui (kilus problemai su automatiškai įvedamais duomenimis, palanku turėti galimybę juos redaguoti) [Adr08c].

1.3. Sesijos, vartotojai ir registracijos

HTTP protokolas sukurtas taip, kad nebūtų ryšio tarp užklausų, todėl negalima identifikuoti asmens, kuris jas siunčia. Šią problemą galima spręsti įvairiais būdais: slapukais ar įrankiais, kurie valdo sesijas, vartotojų prisijungimus ir registracijas. Slapukai (*angl. cookies*) yra nedidelis kiekis informacijos, dažnai apimanti unikalius identifikatorius, kuriuos žiniatinklio serveriai siunčia naršyklėms. Tada šie slapukai gali būti siunčiami atgal į serverį kai naršyklė paprašo naujo puslapio – tokiu būdu yra identifikuojamas naudotojas. Tačiau slapukai turi nemažai veikimo spragų: jų saugojimas naršyklėje yra savanoriškas, jie yra nesaugūs (ypač jei naudojamas ne HTTPS protokolas), juos galima redaguoti. „DJango" karkaso suteikiamas sprendimas šioms problemoms – sesijos ir vartotojo autentifikacijos ir autorizacijos (*angl. auth/auth*) sistemos.

Sesijos yra „DJango" naudojamas mechanizmas, skirtas stebėti būseną tarp svetainės ir konkrečios naršyklės. Tai panašus metodas kaip slapukai, kai saugomi duomenis kiekvienoje naršyklėje ir gaunami svetainėje kai naršyklė prisijungia. „DJango" naudoja slapuką su specialiu seanso ID, kad nustatytų naršyklę ir susietą seansą su svetaine. Tikrieji seanso duomenys pagal numatytuosius nustatymus saugomi svetainės duomenų bazėje, todėl tai yra saugesnis metodas nei duomenų saugojimas slapuke.

Autentifikacijos ir autorizacijos sistema vyksta keliais etapais: patvirtinama, kad vartotojas yra tas, kas jis teigia, kad yra, patikrinama, ar vartotojas įgaliotas atlikti tam tikrą operaciją. Šios sistemos sudedamosios dalys: vartotojai – asmenys užsiregistravę svetainėje, leidimai - dvejetainės (taip/ne) vėliavėlės, nurodančios, ar vartotojas gali atlikti tam tikrą užduotį, grupės – vartotojų grupės su pritaikytomis tokiomis pačiomis vėliavėlėmis, pranešimai - būdas rodyti sistemos pranešimus vartotojams, profiliai – mechanizmas priskiriantis vartotojams pritaikytą svetainės informaciją. „DJango" suteikia nemažai sesijos sistemos funkcijų: autentifikacijai naudojama funkcija

„authenticate()", kuri kaip parametrus naudoja slaptažodį ir vartotojo vardą ir grąžina vartotojo objektą, prisijungimui naudojama funkcija „login()", kuri kaip parametrus naudoja „HttpRequest" ir vartojo objektus, kad išsaugotų vartotojo ID seanso metu [Adr08b].

1.4. Saugumas

Žiniatinklio programų apsauga yra pagrindinis bet kurio žiniatinklio komponentas. Interneto globalumas nulemia tai, kad atakos, išnaudojančios saugumo pažeidžiamumus, gali būti vykdomos iš skirtingų vietų, bei apimti įvairų mastą ar sudėtingumą. „DJango" sistema užtikrina automatišką apsaugą nuo daugelio klaidų, nulemiančių saugumo trūkumus, kurias daro nepatyrę žiniatinklio programuotojai [Adr08a].

SQL injekcija yra dažnai pasitaikanti ataka, kurios metu užpuolikas pakeičia tinklalapio parametrus, tam kad įterptų savavališkus SQL fragmentus, kuriuos neapsaugota žiniatinklio programa vykdo tiesiogiai į savo duomenų bazę. Tam, kad išvengti galimo duomenų praradimo ar kitų piktavališkų veiksmų su duomenimis, „DJango" sistema pagal naudojamos duomenų bazės citavimo taisykles automatiškai pašalina visus specialius SQL parametrus.

Kryžminis skriptas (*angl. Cross-site scripting*) randamas žiniatinklio programose, kurioms nepavyksta išvengti netinkamai pateikto vartotojo turinio prieš pateikiant jį HTML pavidalu. Tai leidžia užpuolikui į tinklalapį įterpti savavališką HTML kodo dalį, kurioje gali reikalauti vartotojų privačios informacijos ar kitais būdais pakenkti tinklapiui ar jo reputacijai. Norėdami išvengti šių problemų, tinklapio programuotojai remiantis „DJango" rekomendacijomis, privalo visada šablonuose naudoti išvengimo žymę (*angl. escape tag*) arba nustatyti automatišką išvengimą.

Kryžminis svetainių užklausų klastojimas arba „CSRF" (*angl. Cross-site request forgery*) yra žiniatinklio saugos spraga, leidžianti užpuolikui paskatinti vartotojus atlikti veiksmus, kurių jie neketina atlikti (pvz., pasikeisti slaptažodį). Tai leidžia užpuolikui apeiti kilmės politiką, kuri skirta neleisti skirtingoms svetainėms trukdyti viena kitai. „Django" sistema suteikia galimybę išvengti tokių atakų su „CSRF" tarpine programine įranga, kuri leidžia apsaugoti nesaugius užklausos metodus - POST, PUT, DELETE. Tam naudojama „csrftoken" žymė, kuri nurodo, kad serveriui pateikiamos formos priimamos, jei jos siunčiamoje užklausoje yra „CSRF" raktas, atitinkantis tą, kurį atpažįsta serveris [Adr22].

2. Terminalo emuliatoriai

2.1. Reikšmė ir privalumai

Terminalo emuliatorius yra kompiuterinė programa, kuri imituoja arba veikia kaip fizinis terminalas, sudarytas iš klaviatūros ir monitoriaus. Fiziniai terminalai suteikia prieigą prie duomenų ir programinės įrangos programų, įdiegtų arba saugomų centralizuotuose kompiuteriuose. Šie specializuoti kompiuteriai paprastai yra serveriai, kuriuose yra didelė atminties talpa, galingi procesoriai, galintys apdoroti didelį kiekį klientų užklausų vienu metu. Tuo tarpu, terminalo emuliatoriai skirti tam, kad pakeistų fizinius terminalus. Privalumai, kuriuos suteikia terminalo emuliatoriai vartotojams, dažniausiai yra susiję su užduočių automatizavimu ir patogia prieiga prie centrinių kompiuterių. Terminalo emuliatoriai neretai naudojami tam ,kad grupė žmonių, tarkim įmonės darbuotojų, galėtų naudotis savo asmeniniais kompiuteriais ir tuo pačiu pasiektų informaciją įmonės centriniuose kompiuteriuose ar automatizuotų ilgai trunkančius procesus. Šiuo principu vadovaujasi daugelis biologinės srities įmonių, kadangi apdorojami duomenų kiekiai dažniausiai būna itin dideli, bei užtrunka ilgą laiko tarpą, bei reikalauja tam tikro automatizavimo [Har22].

2.2. Terminalo emuliatoriaus implementacija

Šiame darbe buvo siekiama implementuoti terminalo emuliatorių, kuris būtų pateikiamas vartotojams naršyklėje, tam buvo panaudotas „Xterm.js“ - kliento pusės programų kūrimo įrankis, parašytas „TypeScript“ ir „JavaScript“ - objektiškai orientuota skriptų programavimo kalba. Šio įrankio pagalba buvo inicijuotas ir atvaizduojamas naršyklėje jo suteikiamas terminalo objektas. Terminalo ekrano skaitymo režimas ir skaitiniai raktai identifikuojantys tam tikro paspausto klavišo reikšmę suteikė galimybę kontroliuoti momentą, kurio metu įvedamos komandos siunčiamos serveriui. Kadangi terminalo emuliatorius turi veikti realaus laiko programos principu, buvo naudojamasi „socket.io“ - „JavaScript“ biblioteka skirta dvipusei komunikacijai tarp serverio ir kliento. Pasinaudojant šiais „Xterm.js“ ir „JavaScript“ teikiamais įrankiais buvo galima realizuoti įvairias komandas reikalingas šiai realaus laiko žiniatinklio aplikacijai [IKR+21].

3. Unix apvalkalas ir E-įrankiai bioinformatikoje

3.1. Unix filosofija

„Unix“ apvalkalas yra viena iš dažniausiai naudojamų bioinformatikos skaičiavimo aplinkų - tai sąsaja su programomis, kuri veikia kaip interaktyvi konsolė, skirta duomenų apdorojimui. Nesinaudojant šia aplinka, bioinformatikams tektų rašyti sudėtingas ir ilgas programas, kurios kaip įvestį naudotų neapdorotus failus ir grąžintų apdorotus duomenis, kaip rezultatus. Atsižvelgiant į didelį įvesties duomenų kiekį, toks procesas galėtų užtrukti kelias valandas ar ilgiau, taip pat programa turėtų apimti daug skirtingų procesų paeiliui (pvz., duomenų nuskaitymas, kokybės užtikrinimas, netinkamų duomenų pašalinimas, lygiavimas su tam tikru genomu ir t.t.). Šią programą būtų sunku pritaikyti skirtingiems projektams ir identifikuoti klaidas vykdymo metu. Taigi, „Unix“ apvalkalas itin gerai pritaikytas bioinformatikinėms programoms, kadangi sukurtas taip, kad vartotojai galėtų rašyti ilgas ir sudėtingas programas, apjungdami nesudėtingus programų modulius į konvejerius. Anot „Unix“ įrankių kūrėjo Doug McIlory (2003), šios sistemos filosofija yra ta, kad programos turi atlikti vieną nesudėtingą veiksmą, bet itin gerai, jas būtų galima jungti, o įvestis turėtų būti teksto srautai, kadangi tai užtikrintų programų universalumą [Buf15a].Taigi, „Unix“ sistema, sukurta tokiu principu, gali užtikrinti bioinformatikinių programų efektyvumą.

3.2. Unix konvejeriai

Unix konvejeriai - patogus įrankis bioinformatinėms užduotims, leidžiantis programų moduliams sąveikauti tarpusavyje, nukreipiant vienos programos standartinės išvesties srautą į kitos komandos standartinės įvesties srautą, tuo tarpu klaidas pateikiant tik terminalo ekrane. Tokiu būdu nėra sukuriamų tarpinių failų, kurie galėtų būti naudingi klaidų identifikavimui, tačiau yra paspartinamas procesas, kadangi atsisakoma nereikalingų failų rašymų. Tokiu būdu dirbant su dideliais naujos-kartos sekoskaitos duomenims, „Unix“ filtras galėtų atlikti svarbų vaidmenį proceso paspartinimo atžvilgiu. Taigi, vienos programos išvesties perdavimas tiesiai į kitos programos įvestį yra skaičiavimo požiūriu efektyvus ir paprastas būdas susieti Unix programas ir komandas, tai suteikia galimybę, kurti sudėtingesnius įrankius iš mažesnių modulinių dalių nesvarbu kokia kalba parašytos programos, tol kol susietos programos gali suprasti viena kitos išvestis [Buf15b].

Šiame darbe buvo siekiama sukurti žiniatinklio bioinformatinį įrankį, kuris veiktų „Unix“ filtro principu – terminalo emuliatoriaus pagalba būtų iškviečiamos „Unix“ komandos, kurias būtų galima jungti į filtrus. Toks įrankis vartotojams suteiktų galimybę naudotis kai kuriais „Unix“ operacinės sistemos privalumais neinstaliavus jos savo kompiuteryje, o tiesiog prisijungdami prie tinklapio programos.

3.3. E-įrankiai

E-įrankiai (*angl. The Entrez Programming Utilities*) yra aštuonių serverio programų rinkinys, užtikrinantis stabilią sąsają su „Entrez“ užklausų ir duomenų bazių sistema NCBI. Šie įrankiai naudoja fiksuotą URL sintaksę, kuri paverčia įvesties parametrų rinkinį į reikšmes, reikalingas įvairiems NCBI programinės įrangos komponentams, norint ieškoti ir gauti prašomų įvairių biologinių duomenų: baltymų sekų, genų įrašų, trimačių molekulinių struktūrų ar biomedicininės literatūros.

Kiekviena „Entrez“ duomenų bazė nurodo joje esančius duomenų įrašus unikaliais sveikųjų skaičių identifikatoriais - UID. E-įrankių naudojimas, taip pat kaip ir „Unix“ komandų, galimas konvejerio principu. Kadangi UID naudojami tiek duomenims įvesti, tiek išvesti, tai svarbus komponentas konvejerio konstravimui tarp šių įrankių (pvz, norint parsisiųsti duomenis naudojamas „Efetch” įrankis, kuris kaip įvestį priima UID sąrašus, šiuos sąrašus galima gauti pasinaudojus „Esearch” įrankiu, kuris pagal tekstinę paiešką grąžina šiuos sąrašus). Taigi, duomenų paieškos patogumą ir efektyvumą gali užtikrinti tai, kad „Entrez“ duomenys turi savo unikalius identifikatorius ir tai, kad E-įrankiai grąžina arba priima kaip įvesti UID sąrašus, tam kad būtų galima konstruoti konvejerius tarp jų [fBio10].

Kadangi bioinformatikų darbas neapsieina be biologinių duomenų, kurie dažnai yra parsisiunčiami būtent iš „Entrez“ duomenų bazės, šioje tinklapio programoje buvo siekiama vartotojams suteikti galimybę terminalo emuliatoriaus pagalba naudotis E-įrankiais ir jungti juos konvejerio principu. Toks funkcionalumas svarbus norint suteikti galimybę efektyviau ir sparčiau naudotis aplikacija – tam, kad nereikėtų atskirai siųstis duomenų ir tik tuomet įkelti į savo paskyrą.

4. Python subprocesas

4.1. Procesai ir subprocesai

Procesas yra operacinės sistemos vykdomos programos abstrakcija. Operacinė sistema seka procesus procesų lentelėje arba proceso valdymo bloke, kuriame yra saugomi atvertų failų identifikatoriai, saugos kontekstas, nuorodos į jo adresų erdves ir pan. Ši lentelė leidžia operacinei sistemai savo nuožiūra atsisakyti tam tikro proceso, kadangi saugoma visa informacija, kurios reikia norint grįžti ir tęsti procesą. Tokiu būdu procesas gali būti pertrauktas daugybe kartų, tačiau pratęsiamas vykdyti nuo taško, kuriame buvo sustabdytas.

Procesas, kuris pradeda kitą procesą, vadinamas tėviniu, o jo pradedamas – dukteriniu. Šie procesai dažniausiai vyksta nepriklausomai, kartais dukterinis procesas paveldi konkrečius išteklius ar kontekstus.

„Python“ suteikia galimybę inicijuoti dukterinius procesus pasinaudojant subprocesų moduliu. Naudoti šį modulį galima tiek „Unix“ apvalkalo komandų vykdymui, tiek bet kurios kitos programos paleidimui, jei žinomas tikslus jos pavadinimas ar kelias iki jos. Komandų ar programų paleidimą kontroliuoja tam tikri paleidimo procesai, kurie manipuliuodami operacinės sistemos procesų medžiu perskirsto tėvinių dukterinių procesų ryšius.

Šiame darbe „Python“ subprocesų modulis buvo naudojamas, „Unix“ komandų ir E-įrankių dukterinių procesų iškvietimui. Tam buvo naudojamasi „Popen” klase, kuri suteikia galimybe procesus vykdyti paraleliai, tokiu būdu suteikiant vartotojams patogesnį funkcionalumą.

5. Biologinės žiniatinklio aplikacijos kūrimas

5.1. Tikslai ir reikalavimai

Praktinei šio darbo daliai atlikti pasirinkta sukurti biologinę žiniatinklio programą, kuri leistų vartotojams naudotis terminalo emuliatoriumi tam, kad galėtų konstruoti „Unix“ apvalkalo komandų ir E-įrankių konvejerius. Žiniatinklio programos funkcionalumas turėtų būti prieinamas tik prisijungusiems vartotojams. Kiekvienas vartotojas turėtų galimybę įsikelti savo asmeninius failus ar atsisiųsti iš „Entrez“ duomenų bazės tam, kad galėtų juos apdoroti įvairiais įrankiais ir komandomis. Komandų išvestis turėtų būti pateikiama vartotojui arba išsaugojama duomenų bazėje taip, kad būtų prieinama tik tam vartotojui, kuris inicijavo komandą. Siekiant suteikti vartotojui patogesnę darbo aplinką, tam tikras funkcionalumas turėtų būti užtikrintas ne vien komandinės eilutės sąsaja. Ši žiniatinklio aplikacija be pagrindinio savo funkcionalumo (komandų ir įrankių konvejerių konstravimo biologiniams duomenims apdoroti) turėtų suteikti vartotojui informatyvią sąsają, bei suprantamą programos naudojimą: galimybę naršyti tarp žiniatinklio puslapių, matyti savo prisijungimo statusą ir įkeltus failus, atlikti failų paiešką, bei rasti pagalbą suteikiančią informaciją.

Remiantis šiais tikslais, buvo suformuluoti kuriamos sistemos reikalavimai:

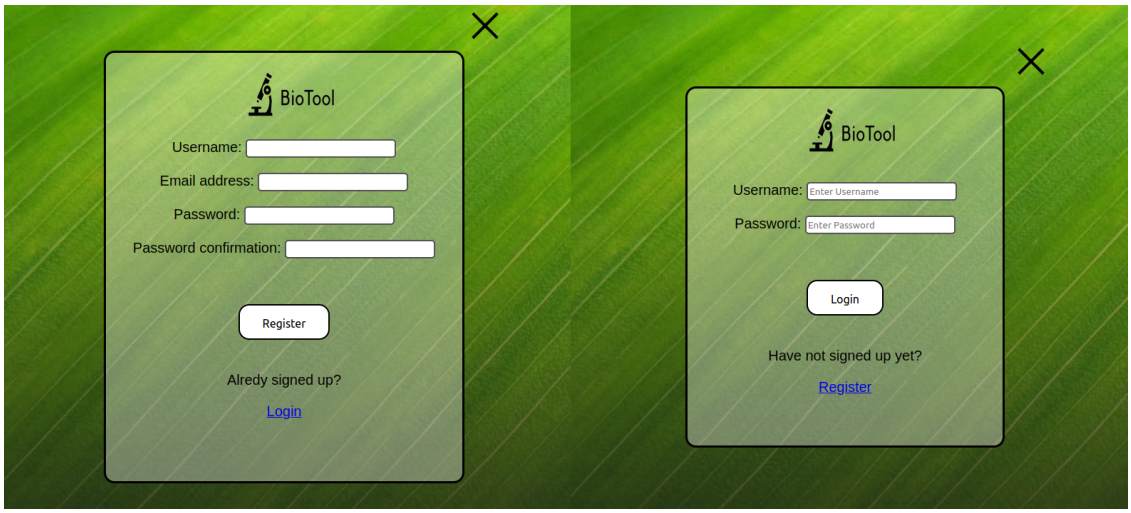
1. Vartotojas gali prisiregistruoti ir prisijungti prie sistemos, naudodamas savo sukurtą vartotojo vardą, slaptažodį, bei elektroninį paštą
2. Vartotojas gali matyti prisijungusio vartotojo vardą
3. Vartotojas gali atsijungti nuo sistemos
4. Prisijungęs vartotojas gali įkelti failus į savo paskyrą
5. Prisijungęs vartotojas gali matyti savo įkeltus failus
6. Prisijungęs vartotojas gali ieškoti tam tikrų failų pagal jų pavadinimus
7. Prisijungęs vartotojas gali matyti savo komandų istoriją
8. Prisijungęs vartotojas gali naudotis terminalo emuliatoriumi komandoms vykdyti

9. Terminalo emuliatoriaus pagalba gali būti konstruojami „Unix“ komandų ir E-įrankių konvejeriai, kurių rezultatai terminale grąžinami vartotojui

5.2. Tinklapio aplikacija „BioTool”

Remiantis išsikeltais reikalavimais sistemai, buvo sukurta žiniatinklio programa „BioTool”, sudaryta iš trijų puslapių: registracijos, prisijungimo ir pagrindinio puslapio, kuriame elementai pateikiami pagal vartotojo statusą.

Registracijos ir prisijungimo puslapius sudaro „Django” formos, kuriomis vartotojai gali kurti savo paskyras arba prisijungti prie jų.



2 pav. Prisijungimo ir registracijos puslapiai

Pagrindinio puslapio atvaizdavimas priklauso nuo vartotojo statuso – neprisijungusiam vartotojui neatvaizduojami failų įkėlimo ir vartotojo komandų istorijos laukai. Prisijungusiam vartotojui pateikiamos puslapio sritys: antraštėje pateikiamas dabartinio prisijungusi vartotojo vardas, pagalbos ir atsijungimo mygtukai, dešinėje atvaizduojamas komandų sąrašas su paieškos laukeliu, centre pateikiamas terminalo emuliatorius ir trumpas programos aprašymas, kairėje vartotojo komandų istorija , vartotojo įkelti failai, failų paieškos laukas, bei failų įkėlimo mygtukai.

Terminalo emuliatorius taip pat veikia priklausomai nuo vartotojo prisijungimo statuso – neprisijungusiam vartotojui terminale išvedamas pranešimas, kuriame prašoma vartotojo prisijungti, komandų įvestis yra negalima, prisijungusiam vartotojui terminale išvedamas pranešimas su vartotojo prisijungimo vardu, visos komandos vartotojui skirtoje direktorijoje yra galimos.

Prisijungęs vartotojas visus veiksmus gali atlikti tiek grafine sąsaja, tiek įvedant komandas terminalo emuliatoriaus ekrane. Dešinėje pateikti įrankiai inicijuoja komandas terminalo ekrane, tačiau vartotojas turi pabaigti jų įvesti, kitaip komandų vykdymo metu bus grąžinamos klaidos. Komandų aprašymai pateikiami paspaudus „Help" mygtuką antraštės dešinėje.

3 pav. Pagrindiniai puslapiai neprisijungusiam ir prisijungusiam vartotojui

5.3. Vartotojo autentifikavimas

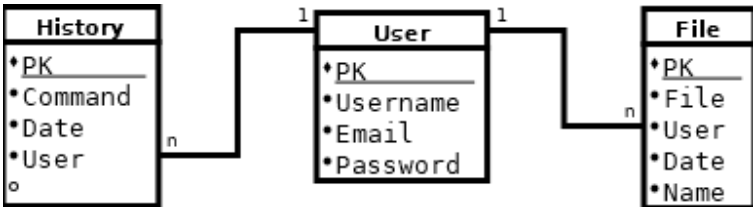
Vartotojo prisijungimo ir registracijos funkcijoms realizuoti buvo naudojamas „Django" autentifikacijos sistemos vartotojo modelis, aprašantis vartotojo klasės atvaizdavimą duomenų bazėje. Šis modelis suteikė galimybę kurti vartotojo objektus, kurie atspindi realius žmones, prisijungiančius prie svetainės, taip pat apriboti tam tikras prieigas, realizuoti registracijos ir prisijungimo formas.

Autentifikacija šiame darbe buvo kuriama pagal MVT dizaino modelį. Panaudoti vartotojo

modelio atributai buvo: vartotojo vardas, elektroninis paštas, bei slaptažodis. Realizuotose vartotojo registracijos ir prisijungimo formose įvedama informacija buvo surenkama į HTTP POST užklausą ir siunčiama serveriui. Kadangi POST metodas nėra apsaugotas nuo kryžminio skripto atakų, naudota „CSRFToken“ žymė. Užklausų apdorojimui buvo naudojamos registracijos, prisijungimo ir atsijungimo funkcijos, parašytos parinkimo logikos – vaizdo dalyje. Šiose funkcijose buvo naudojami „Django“ suteikiami autentifikacijos metodai, skirti vartotojo ID pridėjimui prie sesijos (login), vartotojo objekto grąžinimui (autentificate) ar sesijos nutraukimui (logout). Po šių autentifikacijos metodų buvo grąžinama funkcija, kurioje realizuotas pagrindinis programos funkcionalumas kartu su pagrindinio puslapio atvaizdavimu arba grąžinamas vienkartinis klaidos pranešimas apie klaidingus formos įvesties tipus. Prieigos prie tam tikrų funkcijų, kaip failų įkėlimo ar komandų vykdymo, neprisijungusiems vartotojams buvo apribotos parinkimo logikos dalyje, remiantis autentifikavimo tikrinimo metodo rezultatais (isautentificated).

5.4. Failų sistema

Šioje žiniatinklio programoje buvo siekiama sukurti failų įkėlimo sistemą, kuri leistų kiekvienam autentifikuotam vartotojui pasiekti tik tam tikrus failus. Šiam tikslui buvo sukurtas failo modelis su atributais: pavadinimas, sukūrimo data, vartotojas, bei failas. Vartotojo atributas žymi išorinį raktą, kuris sieja failą su jo savininku, failas ,talpinamas direktorijoje, kurios pavadinimas savininko pirminis raktas. Failų kūrimas realizuotas panašiu principu kaip ir vartotojo registracijos ar prisijungimo: informacija surenkama formose (tik autentifikuotų vartotojų), tuomet siunčiamos HTTP POST užklausa serveriui užkoduota „multipart/form-data“ metodu, kur yra apdorojama atitinkamos parinkimo logikos dalies funkcijos.



4 pav. Duomenų bazės ER

5.5. Pagrindinių funkcijų implementacija

Šios tinklapio programos pagrindinė funkcija apdoroti komandų ir įrankių konvejerius, pateikiamus terminalo emuliatoriuje, ir grąžinti vartotojui išvestį. Tokiam funkcionalumui realizuoti buvo naudojami anksčiau minėtas „Xterm.js“ įrankis, „JavaScript“ – programavimo kalba ir „Python“ dukterinio proceso modulis. Terminalo funkcionalumas realizuotas taip, kad būtų kontroliuojamas įvedamų klavišų skaitinių identifikatorių. Kodas 13, identifikuojantis „enter“ klavišo paspaudimą, inicijuoja komandos siuntimą serverio pusei; kodas 8, identifikuojantis „backspace“, inicijuoja komandų trynimą terminalo ekrane, visi likę kodai skirti komandos rašymui – skaitinių simbolių pridėjimui prie komandos kintamojo.

Komandos terminale kuriamos klavišų įvestimis arba pateiktų įrankių mygtukų sekcija. Kadangi būtina išsaugoti terminale įvestas komandas, komandų siuntimas serveriui vyksta asinchroniškai nurodytai funkcijai pagal jos URL. Funkcijai, kuri atsakinga už pagrindinį programos funkcionalumą, siunčiama ne viena POST užklausa, todėl kartu su duomenimis pridedamas POST užklausos identifikatorius, pagal kurį nustatoma, kokia tai užklausa. Jei identifikatorius nurodo, kad tai komandos vykdymo užklausa, nurodytu keliu iki užklausos vartotojo direktorijos iškviečiama komanda kaip dukterinis procesas. Dukteriniai procesai realizuoti su „Popen“ todėl vyksta lygiagrečiai. Toks vykdymas realizuotas E-įrankių konvejeriams, kurių vykdymas gali užtrukti ilgą laiko tarpą.

Kadangi kai kurios neteisingai įvestos ar nebaigtos „Unix“ komandos ne visada grąžina klaidą (pvz., cat), buvo nustatytas laiko limitas, ties kuriuo užbaigiamas nepasibaigęs dukterinis procesas. Šis sprendimas nėra optimalus visais atvejais, nes sudėtingų „Unix“ konvejerių vykdymas gali viršyti nurodytą laiko limitą, bet gali veikti nesudėtingų konvejerių atvejais. Tolimesnis „Unix“ komandų klaidų apdorojimas bus nagrinėjamas tolimesniame darbe.

Dukteriniui procesui pasibaigus jo standartinė ir klaidų išvestis dešifruojama į ASCII ir grąžinama kaip JSON atsakymas, asinchroninei funkcijai inicijavusiai komandos perdavimą. Toliau gautas atsakymas pateikiamas terminalo emuliatoriumi vartotojui.

Šiek tiek kitoks funkcionalumo realizavimas pritaikytas komandai „upload“, kuri nėra nei „Unix“ komanda, nei E-įrankis. Šios komandos įvedimas inicijuoja failo įkėlimą, kuris asinchroniškai su užklausos identifikatoriumi siunčiamas funkcijai pagal nurodytą URL. Kadangi failai gali

būti koduojami tik „multipart/form-data“ metodu, duomenys buvo siunčiami kaip „formData“ objektas „fetch“ metodu, kuris grąžina „promise“ objektą. Serverio pusėje pagal nurodyta užklausos identifikatorių ir vartotoją inicijavusi užklausą, realizuotas failo objekto sukūrimas ir jo patalpinimas atitinkamo vartotojo direktorijoje.

5.6. Antrinių funkcijų implementacija

Antrinis funkcionalumas šioje programoje – visos funkcijos, nesiejamos su terminalo emuliatoriaus vykdomais antriniais procesais: failų paieška, naršymas tarp nuorodų. Paieška realizuota tokiu pačiu principu, kaip ir failų įkėlimas „upload“ komanda – vykdomas asinchroniškas paieškos teksto siuntimas serveriui, kur tam tikra funkcija apdoroja užklausa pagal identifikatorių ir grąžina tam tikrą rezultatą kaip „JSON” atsakymą. Tuomet naudojama „Jquery“ biblioteka html elementų selekcijai ir manipuliacijai su jų vertėmis, šiuo atveju, stulpelio išrinkimui pagal jo identifikatorių ir paragrafų su failų pavadinimais įterpimui. Naršymas tarp nuorodų užtikrintas „Django” šablonų sistema, ledžiančia įterpti nukreipimo URL į html kodą, bei parinkimo logikos funkcijomis, kurios nukreipia vartotojus į tam tikrus žiniatinklio puslapius.

5.7. Dar nerealizuotas funkcionalumas

Atsižvelgiant į jau realizuotą žiniatinklio programos funkcionalumą, pastebėtus implementacijų ir funkcionalumo trūkumus, galima spręsti, kokie patobulinimai turėtų būti atlikti ateityje.

Šiame darbe pastebėti funkcionalumo trūkumai: komandų procesų užbaigimas po tam tikro termino, pastovus lygiagrečių procesų vykdymas, neoptimalus komandų apribojimas. Komandų nutraukimas pasiekus tam tikrą vykdymo laiką nėra pats optimaliausias sprendimo variantas, kadangi tokiu būdu yra apribojamas komandų konvejerių sudėtingumas ir vartotojas neturi galimybės atlikti sudėtingesnes operacijas. Toliau atliekant šį darbą reiktų realizuoti neteisingų komandų nutraukimą kitokiu principu (atsižvelgiant ne į vykdymo laiką). Lygiagretūs procesų vykdymai šioje programoje reikalingi norint suteikti vartotojams galimybę vienu metu vykdyti daugiau procesų, kurie užtrunka ilgą laiko tarpą, tačiau paprastiems procesams kaip „cat” ar „ls” tai tikrai nereikalinga. Todėl tęsiant šį projektą reiktų atskirti lygiagrečius procesus nuo paeiliui vykstančių arba suteikti galimybę vartotojui pasirinkti proceso vykdymo procedūrą pačiam. Kadangi komandos

dukteriniam procesui pateikiamos kaip vienas tekstinis kintamasis, vartotojas turi neapribotas galimybes, kas sukelia dideles saugumo spragas. Dabartiniame projekte siekiant išvengti vartotojo įsibrovimų apribotos tam tikros komandos ir galimybės (pvz, cd, rm). Toliau vystant projektą, reiktų surasti optimalesnių būdų kontroliuoti vartotojo vykdomas komandas, neapribojant tam tikrų galimybių, kurios gali būti naudingos vartotojams.

Papildomos funkcijos, kurios galėtų būti realizuotos šioje tinklapio programoje: komandinis prisijungimas, prisijungimas prie aplankalų, aplankalų kūrimo galimybė, sudėtingesni bioinformatiniai įrankiai. Kadangi tokio pobūdžio programa skirta naudotis tam tikrai organizacijai, kuri dirba su biologinių duomenų apdorojimu, būtų naudinga, kad vartotojai galėtų sudaryti tam tikras komandas, darbuotojų grupes, tarp kurių galėtų dalintis failais ar kita informacija. Komandose galėtų egzistuoti atskira autentifikavimo sistema, leidžianti prisijungti prie tam tikrų direktorijų tik autentifikuotiems vartotojams. Norint palaikyti tvarkingumą vartotojo paskyroje, tolesniame projekte reiktų sukurti galimybę, kurti ne tik failus bet ir jų aplankalus. Norint suteikti platesnes galimybes vartotojams, taip pat reiktų praplėsti naudojamų įrankių sąrašą tam, kad galėtų būti atliekami darbai tiesiogiai susiję su biologinių duomenų analize.

Rezultatai ir išvados

Šiame darbe buvo siekiama susipažinti su biologinių žiniatinklio programų ypatybėmis, panagrinėti „Django" karkaso veikimo principus ir juo kuriamos programos dizaino architektūrą, bei remiantis įgytomis žiniomis sukurti realaus laiko tinklapio aplikaciją, kuri suteiktų vartotojams galimybę konstruoti „Unix" komandų ir E-įrankių konvejerius biologinių duomenų apdorojimui.

Rezultatai:

1. Darbe apžvelgti kai kurie tinklapių kūrimo principai ir problemos, su kuriomis gali susidurti jų kūrėjai. Šioms žiniatinklio aplikacijų kūrimo problemoms aprašyti sprendimai, kuriuos suteikia „Django" karkaso funkcijos.

2. Apžvelgti „Unix" apvalkalo komandų ir E-įrankių konvejerių principai ir jų svarba bioinformatikoje, aprašyta terminalo emuliatoriaus, bei dukterinio proceso reikšmė šiame projekte.

3. Išnagrinėtas „Xterm.js" įrankio veikimo principas ir pateiktas metodas kaip šis įrankis gali būti implementuojamas į žiniatinklio aplikacijas, konstruojamas su „Django" karkasu.

4. Remiantis įgytomis teorinėmis žiniomis, buvo sukurta biologinė realaus laiko tinklapio aplikacija, suteikianti galimybę prisijungusiems vartotojams terminalo emuliatoriaus pagalba apdoroti įsikeltus arba atsisiųstus biologinius duomenis.

5. Darbe pateiktas metodas, kaip su „Django" karkasu ir „JavaScript" asinchroninėmis funkcijomis galima sukurti realaus laiko tinklapio programą.

Taigi, šio darbo metu sukurta žiniatinklio aplikacija yra tinkamas įrankis prisijungusiems vartotojams terminalo emuliatoriaus pagalba konstruoti „Unix" komandų ir E-įrankių konvejerius biologinių duomenų apdorojimui. Tokio tipo programa, atlikus tam tikrus patobulinimus ir pridėjus daugiau funkcionalumo ir įrankių, galėtų būti skirta biologinės srities įmonės bioinformatiniams darbams centralizuoti.

Remiantis darbo rezultatais, galima daryti šias išvadas (kai kuriuose punktuose pateikiama nuomonė):

1. „Django" karkasas - patogi programos kūrimo infrastruktūra, tinkama realaus laiko žiniatinklio programų konstravimui.

2. „Django” sistemos autentifikacijos modulis yra geras įrankis žiniatinklio programų prisijungimo ir registracijų funkcijų realizacijai.

3. Pasinaudojus „Xterm.js” įrankiu kartu su „Django” karkasu ir „Python” subproceso moduliu galima sukonstruoti terminalo emuliatorių, pateikiamą per naršykle, bei realizuoti jame tam tikrų komandų vykdymą.

4. „Django” karkasas kartu su „JavaScript” asinchroninėmis funkcijomis pakankami įrankiai realizuoti realaus laiko žiniatinklio aplikaciją su autentifikacijos sistema.

5. Remiantis išnagrinėta literatūra, „Django” karkasas užtikrina daug automatinių priemonių spręsti dažnai pasitaikančioms tinklapių kūrimo problemoms ir saugumo spragoms: MVT dizainas, apsauga nuo SQL injekcijų, CSRF žymės ir t.t. Šios priemonės palengvina ir pagreitina žiniatinklių konstravimo procesą.

Literatūra

[Adr08a] Jacob K. Moss Adrian Holovaty. *Security*. *The Definitive Guide to Django: Web Development Done Right*. Apress, 2008, p. 219–222.

[Adr08b] Jacob K. Moss Adrian Holovaty. *Sessions, users, registrations*. *The Definitive Guide to Django: Web Development Done Right*. Apress, 2008, p. 144–151.

[Adr08c] Jacob K. Moss Adrian Holovaty. *The Django administration site*. *The Definitive Guide to Django: Web Development Done Right*. Apress, 2008, p. 68–80.

[Adr08d] Jacob K. Moss Adrian Holovaty. *The MVC Design Pattern*. *The Definitive Guide to Django: Web Development Done Right*. Apress, 2008, p. 2–3.

[Adr08e] Jacob K. Moss Adrian Holovaty. *The MVC Design Pattern*. *The Definitive Guide to Django: Web Development Done Right*. Apress, 2008, p. 26.

[Adr08f] Jacob K. Moss Adrian Holovaty. *The MVC Design Pattern*. *The Definitive Guide to Django: Web Development Done Right*. Apress, 2008, p. 24–47.

[Adr08g] Jacob K. Moss Adrian Holovaty. *The MVC Design Pattern*. *The Definitive Guide to Django: Web Development Done Right*. Apress, 2008, p. 48–53.

[Adr22] Simon Willison Adrian Holovaty. DJango documentation 4.0, 2022. URL: `https://docs.djangoproject.com/en/4.0/ref/csrf/`.

[Buf15a] Vince Buffalo. *Remedial Unix Shell*. *Bioinformatics Data Skills*. O'Reilly, 2015, p. 37–39.

[Buf15b] Vince Buffalo. *The Almighty Unix Pipe: Speed and Beauty in One*. *Bioinformatics Data Skills*. O'Reilly, 2015, p. 45–46.

[Con10] TechTarget Contributor. Web application (Web app), 2010. URL: `https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app`.

[fBio10] National Center for Biotechnology Information (US). Entrez Programming Utilities Help, 2010. URL: `https://www.ncbi.nlm.nih.gov/books/NBK25501/`.

[Fou21] Django Software Foundation. Django (Version 4.0), 2021. URL: https://www.djangoproject.com/.

[Har22] Vanessa Harvey. What is a Terminal Emulator?, 2022. URL: https://www.easytechjunkie.com/what-is-a-terminal-emulator.htm. Last Modified Date: May 27, 2022.

[IKR+21] Daniel Imms, Paris Kasidiaris, Megan Rogge, Nick Pezza, Christopher Jeffrey ir Felipe Gasper. Xterm.js documentation 4.14, 2021. URL: https://xtermjs.org/docs/.